Report from Dagstuhl Seminar 17342

SLEBOK: The Software Language Engineering Body of Knowledge

Edited by

Benoît Combemale[1], Ralf Lämmel[2], and Eric Van Wyk[3]

1 **IRISA – Rennes, FR,** benoit.combemale@irisa.fr
2 **Universität Koblenz-Landau, DE,** laemmel@uni-koblenz.de
3 **University of Minnesota – Minneapolis, US,** evw@umn.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 17342 "SLEBOK: The Software Language Engineering Body of Knowledge". **Software Language Engineering** (SLE) has emerged as a scientific field, with a strong motivation to connect and integrate different research disciplines such as compiler construction, reverse engineering, software transformation, model-driven engineering, and ontologies. This seminar supported further integration of said communities with the clear objective of assembling a **Body of Knowledge** on SLE (SLEBoK). The BoK features artifacts, definitions, methods, techniques, best practices, open challenges, case studies, teaching material, and other components that will afterwards help students, researchers, teachers, and practitioners to learn from, to better leverage, to better contribute to, and to better disseminate the intellectual contributions and practical tools and techniques coming from the SLE field.

Seminar August 20–25, 2017 – http://www.dagstuhl.de/17342
1998 ACM Subject Classification D2.0 Software Engineering: General – Standards, D2.1 Software Engineering: Requirements/Specifications – Languages, D2.3 Software Engineering: Coding Tools and Techniques – Standards
Keywords and phrases body of knowledge, language design and implementation, metaprogramming, software languages
Digital Object Identifier 10.4230/DagRep.7.8.45
Edited in cooperation with Manuel Leduc

1 Executive Summary

Benoît Combemale
Ralf Lämmel
Eric Van Wyk

License Creative Commons BY 3.0 Unported license
© Benoît Combemale, Ralf Lämmel, and Eric Van Wyk

Overview and Motivation

Over the last 10 years, the field of **Software Language Engineering** (SLE) has emerged based on a strong motivation to connect and integrate different research disciplines such as compiler construction, reverse engineering, software transformation, model-driven engineering, and ontologies. This seminar strives for directly promoting the further integration of said communities with the clear objective of assembling a **Body of Knowledge** on SLE (SLEBoK). The BoK features artefacts, definitions, methods, techniques, best practices, open

Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

challenges, case studies, teaching material, and other components that will afterwards help students, researchers, teachers, and practitioners to learn from, to better leverage, to better contribute to, and to better disseminate the intellectual contributions and practical tools and techniques coming from the SLE field.

The following questions and issues provided the guiding principles for the seminar. The first two categories reflect on the result of the past decade and the last category looks forward to the next decade; these categories has been addressed by the seminar attendees in breakout groups.

- **Conceptual model of the SLE field**: What is a comprehensive and objective (validated) classification of SLE approaches? What are appropriate dimensions of such a classification? How to otherwise ontologically organize software language engineering, e.g., in terms of application areas, relationships to other software engineering areas, and fundamental SLE concepts?
- **Curriculum contributions by the SLE field**: What is the suite of formal notions and engineering methods, that one could want to see introduced in a computer science curriculum so that SLE is properly represented? What is a reference curriculum for SLE? What is an appropriate combination of timeless foundations and relevant (current) applications and technologies? How to contribute to or otherwise support a computer science curriculum?
- **Open SLE challenges**: What are the open challenges in SLE and how to lay out a larger research agenda that the community can refer to in the next 10 years? How to connect to important developments such as AI and IoT? How to measure the relevance of the research priorities proposed?

With the SLE field approximately 10 years old, there is a strong support by the community to analyse the situation and to move to the next level of maturity. This Dagstuhl seminar provided the ideal format for such a critical analysis and further development of SLE's foundation. As a result of the work on the above three pillars, the seminar attendees initiated the SLEBOK: **https://github.com/slebok/slebok**.

2 Table of Contents

3 Working Groups

3.1 Reuse and modularity in specifications of software languages

Peter D. Mosses (TU Delft, NL)

License Creative Commons BY 3.0 Unported license
© Peter D. Mosses

Precise and formal specifications of software languages can be used not only as definitions and documentation, but also for generation of language processing tools. For example, a context-free grammar can define the syntax of a programming language, and can be used to generate a parser for the specified language. Similarly, interpreters or compilers can be generated from semantic specifications.

However, giving and ensuring the correctness of a complete specification of a software language generally requires a significant effort. The required effort can be greatly reduced by reusing (parts of) previous language specifications. For example, one language can embed another one, or languages can extend a base language. In addition, languages evolve, and their specifications need to co-evolve; the new version might reuse parts of the specifications of the old version.

The goal of this working group is to define, distinguish and illustrate the approaches to reuse found in practice. To this end, we investigate examples of reuse, analyse how it is achieved and extract general patterns and practices. In particular, we consider the role of modular structure in connection with reuse.

The working group discussions during the seminar focused on clarifying the relevant concepts and terminology. Oscar Nierstrasz produced a MindMap that reflected the outcome of the discussions. Many of the working group participants contributed references to frameworks and tools in a collaborative WriteMe document during the seminar. The moderator drafted an outline of a more structured document (based on a previously proposed classification of reuse scenarios) at the end of the seminar, and solicited details of examples of reuse in software language specification.

After the seminar, participants of the working group contributed brief (1-page) descriptions of examples of software language specification reuse to the WriteMe document, including references to publications and websites. These examples have subsequently been grouped according to a revised (but still tentative) classification of reuse scenarios. They exploit a wide range of language specification frameworks, including Ecore+ALE, JastAdd, Kiama, Melange, MPS, Neverlang, Object Algebras, Rascal, Silver, and SugarJ. Analysis and discussion of these examples should provide a basis for achieving the stated goal of the working group.

3.2 Attribute Grammar working group

Anthony Sloane (Macquarie University – Sydney, AU)

License Creative Commons BY 3.0 Unported license
© Anthony Sloane

Attribute grammars were extensively studied from their introduction by Knuth in the late 1960s through a "golden age" particularly in the 1980s and early 1990s. In this period, attribute grammars were applied successfully to problems ranging from programming language

semantics to natural language processing. Many different grammar analyses and evaluation strategies were developed.

Since the golden age, interest in attribute grammars in computer science has waned somewhat, yet research progress has continued. Notable since that period are a stronger focus on higher-order attributes, addition of cross-tree references and the use of implicit forwarding for language extension. On-demand evaluation of attributes has taken a prominent position in modern attribute grammar systems. Current attribute grammar research tends to be less about novel notations or evaluation approaches and more about applications.

The aim of this working group is to make recent developments in attribute grammar research accessible to the field as a whole. Within the context of the SLEBOK we aim to produce materials that give an accurate picture of the key attribute grammar results and systems particularly since the golden age. During the seminar we focused on discussions to identify broad themes. Preliminary SLEBOK contributions were developed for attribute grammar terminology and a reader article on the main attribute grammar literature was begun. On-going work will finalise these contributions and work toward a more comprehensive survey article.

3.3 Software language Survey

Eugene Syriani (Uniersity of Montral, CA)

License Creative Commons BY 3.0 Unported license
© Eugene Syriani

Introduction

During the Dagstuhl seminar 17342 on a software language engineering body of knowledge (SLEBOK), our working group has focused on answering the question: "What is a software language?". After informally discussing within our group and other participants, we could not sense a common agreement. Therefore, we decided that a formal survey needs to be conducted to better answer this question. During the seminar, we conducted a pilot survey to better formulate hypothesis and lead to a formal survey that will eventually be distributed to the SLE community. In the following, we report on the survey conducted during the SLEBOK seminar at Dagstuhl.

Research method

We prepared an online survey using Google Forms. All 25 participants of the seminar participated in the survey. However, the members of our working group were excluded from the final results. Therefore, the total number of participants was 21. The survey is divided into four sections. The first section identified the participant with only his email. The following set of questions served to collect background information on each participant, e.g., expertise related to SLE, community affiliation, and seniority status. The third section presents 64 candidate languages and asks the participant to decide whether it is a software language. The possible answers are yes, no, I don't know, and could be either. The only information available to the participant is either the name of a language, when it is well-known (e.g., XMI, SQL, English), or a one line description of its purpose (e.g., Student course feedback form). These questions were randomized. The final section had one open question for participants to leave any comment. The language questions spanned various categories,

such as: API, Encoding formats, Forms and UI, human-oriented languages, metalanguages, modeling languages, programming languages, storage formats, technical domain-specific languages (DSL), etc. We noted what was the most dominant expected answer and the expected agreement rate (50% to 100%). We relied on two metrics. First, we used the raw answer to compute the agreement among participants. We ignore "I don't know" and could be either counted as positive. Second, we assigned a score per answer on a scale of 0 to 2 to compute the deviation from the answers are working group agreed upon.

Results

Overall, there are 76% agreement among participants, 75% agreement per language, and 71% agreement per category. This is considered acceptable according to Cohen's Kappa. We found there was very good consistency among languages within the same category in most cases. This survey identified clearly the following as software languages: domain-specific, programming, description, meta-, and modeling languages. It also identified clearly the following as not being software languages: artificial human, natural, and physics languages. Some language categories could be either but needed more context information to decide: API, constrained strings, Forms, UI, and spreadsheets. There was however no consensus for storage formats, encodings, structured text mechanical, and ontology languages. They depend on the background of the participant. We therefore conclude that a language may be considered a software language depending on the context in which it is used.

3.4 Practical Guide to Parsing

Jurgen Vinju (CWI, NL)

License Creative Commons BY 3.0 Unported license
© Jurgen Vinju

While the books on the topic of parsing cover mostly the theory of parsing algorithms and not the pragmatics of applying these algorithms, at the same time the manuals of particular parsing technologies cover only basic usage patterns and configuration options of the given tools. In between these books and manual extremes lies a knowledge and skill gap:

- which (types of) parsing tool fits the language and language processing task best?
- what design decisions must a language engineer consider when designing a parser?
- how to balance trade-offs between quality and getting-it-done?
- how to balance accuracy and correctness with efficiency and practicality?

The SLE community-at-large represents a considerably body of knowledge on obtaining parsers for software languages; together we have built parsing technologies, used them, evaluated them, improved them and applied them in a big number of projects. Therefore we seem to be in a unique position to fill this perceived gap in parsing literature. It is also noted that Wikipedia is neither complete nor up-to-date in this topic, and we envision contributing to Wikipedia where possible as a side-effect.

The goal of the break-out group at the seminar was to kick-off a concerted effort in creating "A Practical Guide to Parsing" as one of the documents in the wider SLEBOK initiative. The goal of "A Practical Guide to Parsing" is to enable newcomers and experts to get an overview of their own tasks, to enable them to make well-founded design decisions and to build better parsers for the job at hand. To start this process the break-out group brainstormed about the following topics:

- a short list of "personas"; models of people who would use the Guide. The personas serve to guide the writing team into producing actionable descriptions with practical value to the audience.
- a list of relevant parsing terms
- a list of quality dimensions for parsing
- a list of trade-offs between two or more of said dimensions with short descriptions

The results of these brainstorms sessions were captured in a raw document as notes. The notes will be used later to structure the document and the work. Later at the SPLASH conference be completed this brainstorm with a different group of members from the community. The current plan is to rework the large set of notes into an overall structure for the Guide and to assign writing tasks to several volunteering members of the community. It is noted that anybody who contributed will be listed as a co-author of the Guide, where author names are annotated with their specific role in the writing process. The roles are yet to be decided.

3.5 Curriculum WG

Ralf Lämmel (University of Koblenz-Landau)

License Creative Commons BY 3.0 Unported license
© Ralf Lämmel

Summary

Even during the preparation of the Dagstuhl seminar 17342 on the software language engineering body of knowledge (SLEBOK), it was clear that the community needs to take an inventory of the curriculum situation of the SLE field. At the seminar, we formed a WG on the SLE curriculum which focused on analyzing existing courses with SLE relevance on the grounds of semi-structured interviews. The interviewing and coding as well as interpretation or recommendation work is still ongoing, but we summarize the interviewing structure here and hint at some findings.

Assumptions

Programming languages and compiler construction are losing their prominent position in CS/SE/MDE curricula. SLE may combine core aspects from these areas and connect them further with a software engineering orientation and thus may fit well into modern curricula. For instance, SLE has many more traditional and modern use cases other than just a classic compiler. Analyzing actual SLE courses is helping in understanding the SLE body of knowledge, as SLE courses should be linear projections of a de-facto BOK. There is much variation to be expected across different courses because SLE is a developing area and due to preferences of teachers in terms of languages and tools as well as locally imposed constraints.

Interview

We identified 15 courses among the participants of the SLEBOK Dagstuhl. We already performed 7 interviews. The interview structure is based on questions regarding learning objectives, course topics, technological spaces, technologies, languages, actual versus "ideal" title, course workload and relevance of lectures, labs, homework, exam, etc., history and evolution of the course, similarity of the course to other SLE courses, dependencies of this

course on others and vice versa, the status of the course to be a choice subject or mandatory, and the use of teaching material. Each interview takes about 10 minutes for the prepared questions. We would typically take another 10 minutes for discussion, also prepared by the general question "What questions had you expected us to ask?".

Preliminary findings

Courses with reliance on model-driven engineering are relatively common. Alternatively, mainstream programming setups or specialized (more or less academic) metaprogramming systems are leveraged in some cases. The full spectrum from Bachelor courses with a larger number of participants and relatively standardized assignments and exams up to research-oriented Master courses with a smaller number of participants and personalized project work exist. The typical course appeals to a software engineering direction and it may be driven by another major theme in software engineering (other than compiler construction, domain-specific languages, or SLE broadly), e.g., software construction or software evolution. In fact, a strong link to compiler construction is not common. Most of the identified courses are already established for 5 years or more. There is enough overlap across all the encountered courses so that some exchange of artifacts (e.g., homework assignments, project topics, and source code) should be worthwhile. The bag of learning objectives and course topics is so diverse and incomparable that some coding would be useful to facilitate better understanding the relationships between the different courses.

4 Opinion Pieces

4.1 On the need for a SLEBoK

Benoit Combemale (IRISA – Rennes, FR)

License Creative Commons BY 3.0 Unported license
© Benoit Combemale

To address the complexity of modern software-intensive systems, the software engineering community is starting a technology revolution in software development, and the shape of this revolution is becoming more and more clear. Little, domain-specific, software languages (aka. Domain Specific Languages, DSLs) are increasingly being developed to continuously capitalize the domain expertise of various stakeholders, and then used as formalisations of the domains to define relationships among them and support the required integration activities.

This new language-oriented development paradigm is emerging in various guises (e.g., metamodelling, model transformation, generative programming, compilers, etc.), and in various shapes (from API or fluent API, to internal or external DSLs). All these approaches belong to the emerging field of Software Language Engineering (SLE).

The next generation of engineers will have to be fluent in many of these approaches to build the languages needed to implement large-scale and complex software-intensive systems. Software engineers for technical domains, or domain experts for business domains should be able to directly leverage their own experience to improve the efficiency and the quality of the produced software-intensive systems, as well as to support the integration of the various concerns.

While SLE is becoming an everyday reality for software engineers, it is time to ensure that the next generation of engineers has the training and knowledge necessary to synergistically apply all SLE approaches. An SLEBoK is key for collecting and disseminating this knowledge in education, in industry and in academia, and should have a huge impact in the future.

4.2 Why SLE

Friedrich Steimann (Fernuniversität in Hagen, DE)

License Creative Commons BY 3.0 Unported license
© Friedrich Steimann

Paraphrasing James Noble on why programming languages matter [1], I say:

1. Today, software is the most important infrastructure for everything.
2. Software is totally dependent on software languages. Ergo:
3. Software languages are the most important pieces of infrastructure for writing software, and thus the most important meta-infrastructure for everything!

This hijacking of James's thesis I justify with the equation

$$\textit{software languages} = \textit{programming languages} + x$$

which makes my thesis a generalization of James's. However, what is x? A good answer will be needed in order not to be subsumed (or looked down at) by the PL community, which is certainly a role model in terms of scientific standards. In particular, x must be non-negligible, even if viewed from outside the SLE community.

Software Language Engineering adds the engineering perspective to the field, which means we must be able to design software languages that we can guarantee to have certain desired properties. Failing such a guarantee means that someone can be held liable, and will face consequences. If we do not take it that serious, SLE will never be engineering.

References

1 James Noble, "Why Programming Languages Matter", comment to IFIP WG2.16 Programming Language Design mailing list, 17 October 2015 (list accessible at https://lists.csail.mit.edu/mailman/listinfo/pldesign); also paraphrased in Andrew Black's talk "Why Programming Languages Matter" given at SPLASH 2015 and 2016 (which brought it to my attention).

Participants

- Mathieu Acher
University of Rennes, FR
- Anya Helene Bagge
University of Bergen, NO
- Walter Cazzola
University of Milan, IT
- Andrei Chis
feenk – Wabern, CH
- Benoit Combemale
IRISA – Rennes, FR
- Thomas Degueule
CWI – Amsterdam, NL
- Sebastian Erdweg
TU Delft, NL
- Johannes Härtel
Universität Koblenz-Landau, DE
- Görel Hedin
Lund University, SE
- Marcel Heinz
Universität Koblenz-Landau, DE
- Ralf Lämmel
Universität Koblenz-Landau, DE
- Manuel Leduc
IRISA – Rennes, FR
- Tanja Mayerhofer
TU Wien, AT
- Peter D. Mosses
TU Delft, NL
- Gunter Mussbacher
McGill University – Montreal, CA
- Oscar M. Nierstrasz
Universität Bern, CH
- Anthony Sloane
Macquarie University – Sydney, AU
- Friedrich Steimann
Fernuniversität in Hagen, DE
- Eugene Syriani
University of Montréal, CA
- Tijs van der Storm
CWI – Amsterdam, NL
- Eric Van Wyk
University of Minnesota – Minneapolis, US
- Hans Vangheluwe
University of Antwerp, BE
- Jurgen J. Vinju
CWI – Amsterdam, NL
- Markus Völter
Völter Ingenieurbüro – Stuttgart, DE
- Vadim Zaytsev
RainCode – Brussels, BE