Empirical language analysis in software linguistics

Jean-Marie Favre[1], Dragan Gasevic[2], Ralf Lämmel[3], and Ekaterina Pek[3]

[1] OneTree Technologies, Luxembourg
[2] Athabasca University, Canada
[3] Universität Koblenz-Landau, Germany

Abstract. Software linguistics is the science of software languages. In this short paper, we sketch the general discipline of software linguistics, but our focus is on one part of it: empirical analysis of software languages. Such analysis is concerned with understanding language usage on the grounds of a corpus. In this short paper, we sketch a survey on empirical language analysis, and we argue that the research method of content analysis is needed for a thorough survey.

1 Introduction

Software Language Engineering (SLE) or "Software Languages are Software too". Software language descriptions and processors are pieces of software. Hence, all kinds of Software Engineering concepts and techniques can be adopted to software languages. SLE is about the systematic design, implementation, deployment, and evolution of software languages [14]. Clearly, software language descriptions have particular properties, when compared to other kinds of artifacts in software engineering. Hence, traditional software engineering life-cycles, methods, set of qualities and constraints must be genuinely adapted. If we think about the distinction of software languages vs. natural languages, then Software Language Engineering can be compared to the established field of *Natural Language Engineering* [6, 10].

Software Linguistics (SL) or "Software Languages are Language too". SLE practices should be informed by scientific knowledge. In the case of natural languages, linguistics is the science of languages [5]. Hence, it is worth to see which concepts, research methods, perhaps even techniques or results from the field of linguistics could be adopted to the study of software languages. In this manner, we obtain "Software Linguistics". The term *Software Linguistics* was introduced by Misek-Falkoff in 1982 [18]. This term and the whole concept of adopting linguistics for software (programming) languages has not seen much interest. We refer to [8, 17] for some controversy. We note that Software Linguistics should not be confused with Computational Linguistics—the former is "linguistics for software languages"; the latter is (simply speaking) "computing for linguistics" (for natural languages).

Towards a survey on empirical software language analysis. A common form of software linguistics is empirical analysis of software language where one is concerned with understanding usage of software languages on the grounds of a corpus. We are interested in any kind of software language. Such analysis has been carried out for

many software languages and many aspects of language usage. In this short paper, we begin a corresponding survey of a collection of publications on the subject of empirical software language analysis. This is work in progress, and we discuss how we expect to use the research method of content analysis [15] for the continuation of the project.[1]

Road-map In Sec. 2, we contextualize the present paper in reference to linguistics for natural and software languages. In Sec. 3, we begin the survey of a collection of papers on empirical language analysis. In Sec. 4, we discuss the employment of the research method of content analysis for completing the current efforts on surveying empirical language analysis in a systematic manner. In Sec. 5, we conclude the paper.

2 Broader research context: software linguistics

Empirical software language analysis is an important part of Software Linguistics. A survey on such work helps understanding and improving the state of the art in empirical software language analysis—also specifically with regard to scientific methodology. Such a surveying effort also exercises the adoption of regular linguistics to software.

In this section, we would like to contribute to a broader understanding and definition of Software Linguistics—through references to (Natural) Linguistics. We have found that the mature, scientific framework provided by linguistics can be reused for software languages—even though many techniques related to natural languages may not be (directly) applicable. Much can be reused beyond the classical separation of different levels such as syntax, semantics and pragmatics. In fact, one can systematically mine Software Linguistics from resources such as "The Cambridge encyclopedia of language" [5]. A few examples are given below.

Comparative linguistics studies, compares and classifies languages according to their features using either a quantitative or qualitative approach. It aims at identifying patterns that are recurrent in different languages, but also differences and relationships between languages. Comparative linguistics may also apply to software languages. For instance, "Programming Linguistics" [7] compares programming languages in terms of commonalities, relationships, and differences while discussing basic matters of syntax, semantics and styles. "The comparison of programming languages: A linguistic approach" [12] goes deeper into linguistic aspects. We also refer to [2, 19].

Historical linguistics studies the history and evolution of languages—often with the goal of identifying language families, that is, languages that derive from a common ancestor. Part of this research compensates for the lost origin of natural languages. In the case of software languages, history is often well documented. Consider, for example, the History of Programming Languages (HOPL) conference series. HOPL focuses on programming languages rather than software languages in general. Also, HOPL does

[1] There is the SourceForge project `http://toknow.sourceforge.net/` that is designated to this surveying effort on empirical analysis of software languages. The project hosts (identifies) a paper collection, in particular. In the present paper, references to the papers of the collection use angle brackets as in "⟨Knuth71⟩".

not stipulate systematic linguistics studies; a typical paper relies on historical accounts by the primary language designers. Some reports, though, provide a large set of qualitative information regarding the evolution of dialects of languages, e.g., [1]. The *impact* of the evolution of software languages on software evolution, though real, is not well understood [9], and deserves more research inspired by linguistics.

Geo-linguistics studies the intersection of geography and linguistics. Studies can, for example, take into account the distribution of languages or dialects over countries, continents, and regions. We encounter a related "clustering" dimension for software languages in Sec. 3. Also, in [19], the emergence of dialects of Lisp are considered in terms of geographical zones.

Socio-linguistics studies languages as social and societal phenomena. The design of software languages and dialects is often directly linked to such phenomena, too. For instance, in [19] Steele and Gabriel conclude "Overall, the evolution of Lisp has been guided more by institutional rivalry, one-upsmanship, and the glee born of technical cleverness that is characteristic of the hacker culture than by sober assessments of technical requirements". The field of socio-linguistics for software remains largely unexplored, but see [3] for a related account.

Corpus linguistics deals with all aspects of designing, producing, annotating, analyzing and sharing corpora. Producing a useful, natural linguistics corpus could be an effort that goes far beyond what individual researchers or teams can do. There are international associations who support sharing, e.g., the European Language Resources Association (ELRA).[2] One should assume that empirical research on the usage of software languages also involves efforts on 'software corpus engineering/linguistics'. Such software corpus linguistics should be simplified by the fact that software language artifacts are inherently digitally stored. However, the survey of Sec. 3 shows that corpora are too often unavailable or unreproducible.

3 Towards a survey on empirical language analysis

In the following, we describe the beginning of a survey on empirical language analysis. In particular, we identify first research questions, and we provide a corresponding coding scheme à la content analysis [15].

3.1 Paper collection

At the time of writing, we have accumulated 52 papers on empirical analysis of software languages. As an illustration, Fig. 1 shows the language distribution for the full collection. For reasons of brevity and maturity of all metadata, all of the subsequent tables will focus on a *selective collection* of 17 papers. Both the full and the selective collections are described online; see footnote 1.

[2] ELRA website: `http://www.elra.info/`

Fig. 1. Tag cloud for the language distribution for the underlying paper collection.

3.2 Research questions

i) Each paper in the collection involves a corpus of a chosen software language. What are the characteristics of those corpora; refer to Sec. 3.4? ii) Each empirical analysis can be expected to serve some objective. What are those objectives for the collection of papers; refer to Sec. 3.5? iii) Each empirical analysis can be expected to leverage some actual (typically automated) analyses on the corpus. What are those analyses for the collection of papers; refer to Sec. 3.6?

3.3 Terminology

We use the term (software) *corpus* to refer to a collection of *items* that are expressed in the language at hand. (These items may be valid or invalid elements of the language in a formal sense.) Items originate from possibly several *sources*. We use the term *source* to refer to different kinds of physical or virtual sources that contribute items. For instance, a corpus may leverage an open-source repository as a source to make available, or to retrieve the items—based on an appropriate search strategy. A paper may provide a *corpus description* that identifies sources and explains the derivation of the actual corpus (say, item set) from the sources.

3.4 Corpus characteristics

Fig. 2 provides metadata that we inferred for the corpora of the selective paper collection.[3] We capture the following characteristics of the software corpora: the software language of the corpus, numbers of sources and items (with a suitable unit), the online *accessibility* of the sources (on a scale of *none*, *partial*, and *full*), and the *reproducibility* of the corpus (on a scale of *none*, *approximate*, *precise*, and *trivial*). We say that reproducibility is *trivial*, if the corpus is available online—in one piece; reproducibility is *precise*, if the sources and the corpus description suffice to reproduce the corpus precisely by essentially executing the corpus description. Otherwise, we apply a judgement call, and use the tags *approximate* or *none*. For instance, the inability to reproduce a ranking list of a past web search *may* be compensated for by a new web search, and hence, reproducibility can be retained at an approximate level. In future work, we would like to go beyond the characteristics that we have sketched here.

3.5 Objectives of the papers

Based on our (preliminary) analysis of the paper collection, we propose the following (preliminary) list of objectives for empirical language analysis; see Fig. 3 for corresponding metadata for the selective paper collection.

[3] A cell with content "?" means that the relevant data could not be determined.

	language	sources	items	unit	accessibility	reproducibility
⟨AlvesV05⟩	SDF	8	27	grammars	partial	approximate
⟨BaxterFNRSVMT06⟩	Java	17	56	projects	full	precise
⟨ChevanceH78⟩	COBOL	1	50	programs	none	none
⟨CollbergMS04⟩	Java	1	1132	JAR files	full	approximate
⟨CookL82⟩	Pascal	1	264	programs	none	none
⟨CranorESMC08⟩	P3P	3	?	policies	full	approximate
⟨GilM05⟩	Java	4	14	projects	full	precise
⟨HageK08⟩	Haskell	1	68000	compilations	none	approximate
⟨Hahsler04⟩	Java	1	988	projects	full	approximate
⟨Hautus02⟩	Java	?	9	packages	full	approximate
⟨KimSNM05⟩	Java	2	2	programs	full	approximate
⟨Knuth71⟩	Fortran	7	440	programs	none	none
⟨LaemmelKR05⟩	XSD	2	63	schemas	partial	none
⟨LaemmelP10⟩	P3P	1	3227	policies	full	trivial
⟨ReayDM09⟩	P3P	1	2287	policies	full	approximate
⟨SaalW77⟩	APL	6	32	workspaces	none	none
⟨Visser06⟩	XSD	9	9	schemas	full	approximate

Fig. 2. Corpus characteristics for selective paper collection

Language adoption The objective is to determine whether the language is used, and with what frequency. Typically, some scope applies. For instance, we may limit the scope geographically, or on the time-line.

Language habits The objective is to understand the usage of the language in terms of syntactically or semantically defined terms. For instance, we may study the coverage of the language's diverse constructs, or any other, well-defined metrics for that matter. This objective may be addressed with substantial measurements and statistical analysis.

Language taming The objective is to impose extra structure on language usage so that habits can be categorized in new ways. For instance, we may equip the language with patterns or metrics that are newly introduced or adopted from other languages. In some cases, the empirical effort towards addressing the objective of language taming may also qualify as effort that attests to the objective of language habits.

User feedback The objective is to compile data of any kind that helps the language user to better understand or improve programs. For instance, we may carry out an analysis to support the proposal of a new pattern that should help with using the language more effectively. This objective could be seen as a more specific kind of language taming.

Language evolution The objective is to gather input for design work on the next version of the language. For instance, we may try to detect indications for missing constructs.

User behavior The objective is to understand the usage of the language and its tools in a way that involves users or user experiences directly—thereby going beyond the narrow notion of corpus consisting only of "programs". For instance, we may analyse instances of compiler invocations with regard to problems of getting programs to compile eventually.

Implementor feedback The objective is to understand parameters of language usage that help language implementors to improve compilers and other language tools. For instance, we may carry out an analysis to suggest compiler optimizations.

	⟨AlvesV05⟩	⟨BaxterFNRSVMT06⟩	⟨ChevanceH78⟩	⟨CollbergMS04⟩	⟨CookL82⟩	⟨CranorESMC08⟩	⟨GilM05⟩	⟨HageK08⟩	⟨Hahsler04⟩	⟨Hautus02⟩	⟨KimSNM05⟩	⟨Knuth71⟩	⟨LaemmelKR05⟩	⟨LaemmelP10⟩	⟨ReayDM09⟩	⟨SaalW77⟩	⟨Visser06⟩
language adoption						●			●						●		
language habits		●		●	●	●			●					●	●	●	
language taming	●						●			●	●		●				●
user feedback										●							
language evolution				●			●										
user behavior								●									
implementor feedback			●	●	●							●					

Fig. 3. Objectives of the selected publications

Without going into detail here, the available data caters for various observations. For instance, we realize that research on language adoption is generally not exercised for programming languages. It appears that online communications but not scientific publications are concerned with such adoption.[4,5,6]

3.6 Analyses of the papers

Based on our (preliminary) analysis of the paper collection, we have come up with a simple hierarchical classification of (typically automated) analyses that are leveraged in the empirical research projects; see Fig. 4 for the classification; see Fig. 5 for corresponding metadata for the selective paper collection.

The presented classification focuses on prominent forms of static and dynamic analysis. In our paper collection, static analysis is considerably more common, and there is a substantial variety of different analyses. We also indicate two additional dimensions for analyses. An analysis is concerned with *evolution*, when different versions of items, sources, or languages are considered. The dimension of *clustering* generalizes geo-linguistics of Sec. 2. By no means, our classification scheme is complete. For instance, we currently miss characteristics regarding data analysis (e.g., in terms of the involved statistical methods), and the presentation of research results (e.g., in terms of the leveraged tables, charts, etc.).

4 Outlook on research methodology

The survey is work in progress. We are in the process of assessing the feasibility of such a survey, discovering research questions, and studying applicable research methodol-

[4] The TIOBE Index of language popularity: `http://www.tiobe.com/tpci.htm`

[5] Another web site on language popularity: `http://langpop.com/`

[6] Language Popularity Index tool: `http://lang-index.sourceforge.net/`

Static analysis	Source code or other static entities are analyzed.
Validity	The validity of items in terms of syntax or type system is analyzed.
Metrics	Metrics are analyzed.
Size	The size of items is analyzed, e.g., in terms of lines of code.
Complexity	The complexity of items is analyzed, e.g., the McCabe complexity.
Structural properties	Example: the depth of inheritance hierarchy in OO programs.
Coverage	The coverage of language constructs is analyzed.
Styles	The usage of coding styles is analyzed.
Patterns	The usage of patterns, e.g., design patterns, is analyzed.
Cloning	Cloning across items of the corpus is analyzed.
Bugs	The items are analyzed w.r.t. bugs that go beyond syntax and type errors.
Dynamic analysis	Actual program runs are analyzed.
Profiles	Execution frequencies of methods, for example, are analyzed.
Traces	Execution traces of method calls, for example, are analyzed.
Dimensions of analysis	Orthogonal dimensions applicable to analyses.
Evolution	An analysis is carried out comparatively for multiple versions.
Clustering	The corpus is clustered by metadata such as country, team size, or others.

Fig. 4. Classification of analyses

	⟨AlvesV05⟩	⟨BaxterFNRSVMT06⟩	⟨ChevanceH78⟩	⟨CollbergMS04⟩	⟨CookL82⟩	⟨CranorESMC08⟩	⟨GilM05⟩	⟨HageK08⟩	⟨Hahsler04⟩	⟨Hautus02⟩	⟨KimSNM05⟩	⟨Knuth71⟩	⟨LaemmelKR05⟩	⟨LaemmelP10⟩	⟨ReayDM09⟩	⟨SaalW77⟩	⟨Visser06⟩
validity						●		●						●	●		
metrics	●	●		●	●			●	●	●			●	●		●	●
coverage			●	●	●	●	●					●	●	●		●	
styles				●			●			●			●	●			
patterns							●		●								
cloning											●			●			
bugs						●								●	●		
profiles			●					●				●					
traces																	
evolution						●		●					●				
clustering						●			●						●		

Fig. 5. Analyses of the selected publications

ogy. Ultimately, we aim at reproducible research results and assessment of validity and limitations. In this discussion section, we briefly hint at the utility of content analysis for the continuation of this research.

4.1 Content vs. meta-analysis

We consider two research methods commonly used in social and health sciences, namely, *content analysis* [15] and *meta-analysis* [11]. Content analysis is a qualitative research

method commonly used for systematically studying large amounts of communication content such as news, articles, books, videos, or blogs. The key characteristic is that the analyzed content is *categorized* by researchers. Systematic literature surveys [4] often leverage content analysis.

Meta-analysis is another research method used for a systematic analysis of research literature. Unlike content analysis, meta-analysis is a quantitative method, and it is focused on research studies that have highly related research hypotheses. The main goal of a meta-analysis is to aggregate and statistically analyze findings of several research studies. Meta-analysis uses stricter inclusion criteria than content analysis: measured outcomes, and sufficient data to calculate effect sizes. These specifics of meta-analysis challenge its application to empirical software engineering [13]. As a result, we focus on content analysis for the ongoing survey.

4.2 Adoption of content analysis

Let us synthesize the main steps for applying content analysis for studying literature on empirical software language engineering; here we follow the advice of [4]. (Content analysis could serve as a convenient research method for studying not only research publications, but also other types of communications such as online discussions about languages, news articles, or bug report histories.)

1. Formulate research questions. In Sec. 3, we listed research questions related to corpus characteristics, objectives and analyses. We may add questions, for example, about i) the correlation between different factors (e.g., languages vs. objectives), ii) the adopted research method, iii) the statistical techniques that are leveraged, and iv) the presentational tools (figures, charts) etc. that are used in the papers.

2. Formulate inclusion criteria for papers. These criteria are derived from the research questions. So far, we have been using these criteria: i) the paper discusses usage of a software language, and ii) the paper reports empirical results on such usage based on a corpus. These constraints may need to be made more precise to avoid inclusion of "irrelevant" papers.

3. Collect data. That is, we need to discover the relevant literature. Our preliminary collection has been obtained in an ad-hoc manner. We harvested several papers from the literature work that went into our prior, related research: ⟨LaemmelKR05⟩, ⟨LaemmelP10⟩. We examined first-degree and second-degree references. While searching those papers on the web, we encountered a few more "obvious" candidates.

A systematic approach commences as follows. First, we need to determine the *sources* from which the papers will be collected. To this end, we may select digital libraries (e.g., ACM, IEEE, Science Direct, and Springer), and other online sources (DBLP[7]), where the papers will be searched. Next, we need to define the *search strategy*. In particular, we need methodologically defined keywords including constraints on their combination so that the search strategy is reproducible and amenable to assessments. Ultimately, the search strategy needs to be executed so that all those papers are collected that satisfy the inclusion criteria.

[7] `http://www.informatik.uni-trier.de/~ley/db/`

4. Evaluate data. The collected data is evaluated in terms of some classification scheme, which is commonly called *coding scheme*, and which consists of a set of categories that describe the studied domain. In some cases, a (part of a) coding scheme can be defined up-front. For instance, we designed the corpus characteristics of the papers up-front; refer to Sec. 3.4. In general, categories may emerge during evaluation. For instance, we perceived the objectives for empirical language analysis only post-priori; refer to Sec. 3.5. There is a systematic process for developing a coding scheme based on the iteration of certain steps of discovery and testing by other researchers [20].

Each paper in the collection has to be coded eventually. Coding is typically done by two researchers (so-called raters) independently. Two authors of the present paper have acted as raters in some limited manner. For a proper execution, it is inevitable to keep track of original codes of the researchers and the reconciled ones. This is needed for the computation of a measure of accuracy of the coding scheme.

As the process of data collection and coding might be rather labor-intensive, content analysis may leverage some more automated approaches. For example, text mining is one possible approach [16], and off-the-shelf tools—such as AeroText and SPSS—provide corresponding support. It will be interesting to see how well text mining works in our domain.

5. Analyse data. Frequencies of each category are to be calculated and reported. The frequencies describe the level of coverage of a certain phenomenon of interest studied in the domain. As these frequencies are obtained through a systematic research process, they can nicely reflect on the main types of problems the research has been focused on, and probably discover some patterns about missing research. Such analysis also helps giving relatively objective answers to the research questions.

6. Publish analysis. The publication must describe all methodological steps, and include all data needed for reproducibility. For instance, inclusion of the discovered number of papers (per used source), and included papers are mandatory. Similarly, details of data evaluation and data analysis need to be included. For instance, measures of accuracy of the coding scheme are to be included. Further, a qualitative discussion of the surveyed papers must be provided.

5 Concluding remarks

We have described the beginning of a literature survey on empirical language analysis. We have presented first results of the emerging survey and discussed details of the underlying research method. We hope to get other software language engineers and empirical software engineers as well as software linguists or even classical linguists involved in this effort.

In the work on the survey, so far, we have found it inspirational to consult linguistics (for natural languages). In particular, linguistics provides input for the categorization needed in the survey, and it suggests underrepresented areas of empirical language analysis. The other interesting insight has been that a survey on empirical language analysis, in itself, also calls for an empirical method, and we have found that content analysis provides a good fit to produce a first survey on empirical language analysis.

References

1. M. B. Barbara G. Ryder, Mary Lou Soffa. The impact of software engineering research on modern progamming languages. *ACM Transactions on Software Engineering and Methodology*, 14(4):431–477, October 2005.
2. N. S. Baron. *How to Make Your Way through the New Language Maze - Computer Languages: a Guide For The Perplexed*. Doubleday, 1986.
3. L. J. Bertil Ekdahl. The difficulty in communicating with computers. In *Interactive Convergence: Critical Issues in Multimedia*, chapter 2. Inter-Disciplinary Press, 2005.
4. H. M. Cooper. *The integrative research review: A systematic approach*, volume 2 of *Applied social research methods series*. Sage, 1984.
5. D. Crystal. *The Cambridge Encyclopedia of Language, Second Edition*. Cambridge University Press, 2005.
6. H. Cunningham. A definition and short history of language engineering. *Journal of Natural Language Engineering*, 5:1–16, 1999.
7. S. J. David Gelernter. *Programming Linguistics*. MIT Press, 1990.
8. E. W. Dijkstra. On the foolishness of "natural language programming". In *Program Construction, International Summer Schoo*, pages 51–53, London, UK, 1979. Springer-Verlag.
9. J.-M. Favre. Languages evolve too! changing the software time scale. In IEEE, editor, *8th Interntational Workshop on Principles of Software Evolution, IWPSE*, September 2005.
10. H. Georgiev. *Language Engineering*. Continuum, 2007.
11. G. V. Glass, B. McGaw, and M. L. Smith. *Meta-analysis in social research*. Sage, Beverly Hills, CA, 1981.
12. J. Goodenough. The comparison of programming languages: A linguistic approach. *ACM/CSC-ER*, 1968.
13. B. Kitchenham, H. Al-Khilidar, M. A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, 13(1):97–121, 2008.
14. A. Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 2008.
15. K. Krippendorff. *Content Analysis: an Introduction to Its Methodology 2nd edition*. Sage, Thousand Oaks, CA, 2004.
16. N. L. Leech and A. J. Onwuegbuzie. Qualitative data analysis: A compendium of techniques and a framework for selection for school psychology research and beyond. *School Psychology Quarterly*, 23(4):587–604, 2008.
17. M. Mahoney. HOPL II - closing panel: The history of programming: Does our present past have a future? *ACM SIGPLAN Notices*, 31(11):15–37, November 1996.
18. L. D. Misek-Falkoff. The new field of "software linguistics": An early-bird view. In ACM, editor, *ACM SIGMETRICS workshop on Software Metrics*, 1982.
19. G. L. Steele, Jr., and R. P. Gabriel. The evolution of Lisp. In *ACM SIGPLAN Notices*, pages 231–270. Press, 1993.
20. E. Taylor-Powell and M. Renner. Analyzing qualitative data. In *Evaluation in Social Work: The Art and Science of Practice*, chapter 13. University of Wisconsin Extension Program, Development and Evaluation Unit, 2003.