The Hitchhiker’s Guide to Software Languages

GPCE’10 Keynote

Ralf Lämmel
Software Languages Team, Universität Koblenz-Landau

Abstract

There is only that much space in the CS curriculum, and there are always new subjects that should be accommodated by the curriculum. For instance, in our community, we would want all graduates to leave university with a modest background in technical spaces, software languages, and meta-programming; also, with conceptually informed and reasonably timeless skills to efficiently master related programming techniques and technologies. In reality, the curricula of few CS departments meet this expectation.

In this talk, I will discuss such curricula-related expectations of our community and the suboptimal situation at CS departments— as I perceive them. More importantly, I will allude to a revision of the CS curriculum that could optimize matters and may stand a chance for mid-term adoption.

Categories and Subject Descriptors D.2.2 [SOFTWARE ENGINEERING]: Design Tools and Techniques; D.2.3 [SOFTWARE ENGINEERING]: Coding Tools and Techniques; D.2.12 [SOFTWARE ENGINEERING]: Interoperability; D.3.3 [PROGRAMMING LANGUAGES]: Language Constructs and Features; D.3.4 [PROGRAMMING LANGUAGES]: Processors; F.3.3 [LOGICS AND MEANINGS OF PROGRAMS]: Studies of Program Constructs

General Terms Design, Documentation, Experimentation, Languages

Keywords Technical Spaces, Software Languages, Metaprogramming, X/O/R Mapping, Programming Techniques, Programming Technologies

Acknowledgments

I gratefully acknowledge collaboration with Jean-Marie Favre (OneTree Technologies, Luxembourg) and Dragan Gasevic (Athabasca University, Canada) on the subject of this talk.

1. 101 ways to run a company

Think of a tiny MIS (a management information system) for a company. In terms of data, there are departments, employees, managers, etc. with appropriate relationships and constraints (invariants). In terms of operations, there is functionality to total some or all salaries, to increase the salaries by a factor, or to reorganize the company. All such operations may be subject to constraints (pre- and post-conditions). The tiny MIS is potentially subject to further requirements; there may be a need to persist data or applications of operations, or to surface data and operations through a domain-specific language (DSL), a user interface, or as a service.

Now let us implement the tiny MIS in many variations while exercising the full range of technical spaces (XML, objects, relational, etc.), programming techniques and technologies (tools, APLs, programming environments, etc.) of the programming of these classes; data structures and algorithms, tal salaries with SAX in streaming manner on XML data, or with a visitor on POJOs; we would increase salaries with Hibernate (O/R mapping) in a relational database, or with XSLT on XML data; we would check constraints with SPARQL on RDF, or with OCL on Ecore models; we would reorganize the company with ATL on KM3 models, or with Haskell on terms of algebraic datatypes; we would implement a DSL for the tiny MIS with OO design patterns, or with the Meta Programming System; we would provide a Web-based GUI for the MIS with Apache Struts or WebDSL.

2. Adding megamodel and metadata

Appropriate structured documentation is needed so that the properties of the implementations, the contributing artifacts, the leveraged programming techniques and technologies as well as the relationships between all these entities are clarified. For instance, we would express whether tree shape or graph shape is used for data modeling; whether a data query relies on streaming or in-memory processing; whether a data transformation is performed in-place or in an immutable style; whether a given set of Java classes is a primary data model or instead derived from a given XML Schema; etc. As a sophistication, some of these properties and relationships (in fact, tools) may be illustrated by appropriate sketches of metaprograms.

3. A class on software languages

In this manner, we have designed a class that covers basic concerns of technical spaces, software languages, metaprogramming, and related programming techniques and technologies. Existing curricula do not achieve the same coverage; in some cases, a combination of advanced courses gets close to this coverage. Given the general conception of ‘software language’, we may provisionally speak of a class on software languages. Once we have a textbook for such a class, we can call it the “The Hitchhiker’s Guide to Software Languages”. The class could be placed in the curriculum somewhere in the proximity of these classes: data structures and algorithms, software engineering, programming paradigms and concepts. In its basic configuration, the class on software languages should be in- deed part of the Bachelor’s curriculum while anticipating the potential continuation in the Master’s. The necessary space for the new class could be obtained by moving, for example, the compiler-construction class to the Master’s.

Copyright is held by the author/owner(s).
ACM 978-1-4503-0154-1/10/10.