Polyglot Software Development: Wait, What?

Gunter Mussbacher, Benoit Combemale, Jörg Kienzle, Lola Burgueño, Antonio Garcia-Dominguez, Jean-Marc Jézéquel, Gwendal Jouneaux, Djamel-Eddine Khelladi, Sébastien Mosser, Corinne Pulgar, et al.

Polyglot Software Development: Wait, What?

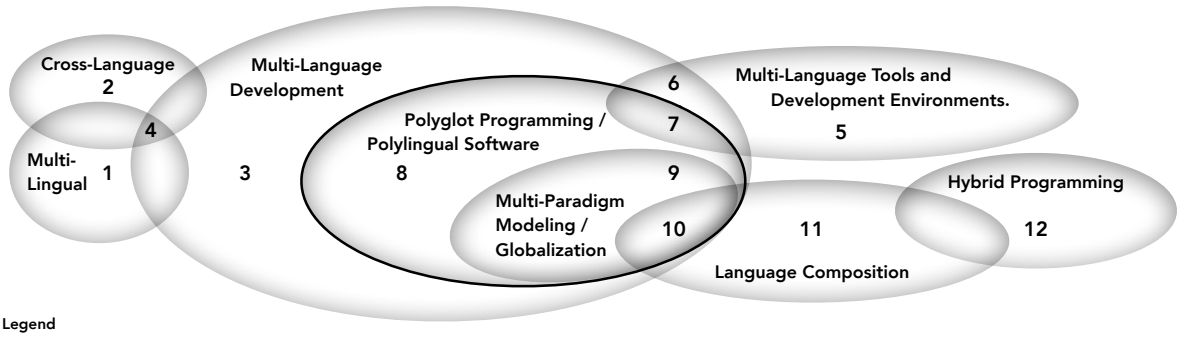
Gunter Mussbacher[1,2], Benoit Combemale[3], Jörg Kienzle[4,1], Lola Burgueño[4], Antonio Garcia-Dominguez[5], Jean-Marc Jézéquel[3], Gwendal Jouneaux[3], Djamel-Eddine Khelladi[3], Sébastien Mosser[6], Corinne Pulgar[7], Houari Sahraoui[8], Maximilian Schiedermeier[1], and Tijs van der Storm[9]

[1]McGill University, Montreal, Canada
[2]INRIA, France
[3]Université de Rennes, Rennes, France
[4]ITIS Software, Universidad de Málaga, Málaga, Spain
[5]University of York, York, UK
[6]McMaster University, Hamilton, Canada
[7]École de Technologie Supérieure, Université du Québec, Montreal, Canada
[8]Université de Montréal, Montreal, Canada
[9]Centrum Wiskunde & Informatica / Rijksuniversiteit Groningen, The Netherlands

Abstract

The notion of polyglot software development refers to the fact that most software projects nowadays rely on multiple languages to deal with widely different concerns, from core business concerns to user interface, security, and deployment concerns among many others. Many different wordings around this notion have been proposed in the literature, with little understanding of their differences. In this article, we propose a concise and unambiguous definition of polyglot software development including a conceptual model and its illustration on a well-known, open-source project. We further characterize the techniques used for the specification and operationalization of polyglot software development with a feature model, concentrating on polyglot programming. We conclude the article outlining the many challenges and perspectives raised by polyglot software development.

Modern software development commonly requires the use of several languages in almost all activities, whether it is requirements engineering, programming in one or more languages, or continuous integration and delivery. For example, requirements may be specified using templates for use cases or user stories and Gherkin scenarios [1]. Continuous integration and delivery may be specified with GitHub Actions and build languages such as Maven or Gradle [2]. The proliferation of domain-specific languages further adds to the incentive to use different languages for an activity [3]. Even a so-called Ruby project such as Mastodon, an open-source, distributed social-media platform, in fact already uses many languages [4]. Besides Ruby, specifications in Docker Compose, Dockerfile, GitHub Actions, Haml, HTML, JavaScript, package.json, Rakefile, SCSS, and SQL are used to handle UI, persistence, and build issues. Mastodon is not an isolated example. In 2017, Mayer et al. con-



Legend

1. "Lightweight Multilingual Software Analysis" doi.org/10.48550/arXiv.1808.01210
2. "CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation" doi.org/10.1109/ASE.2019.00099
3. "On Multi-Language Software Development, Cross-Language Links and Accompanying Tools: a Survey of Professional Software Developers" doi.org/10.1186/s40411-017-0035-z
4. "Cross-Language Interoperability in a Multi-Language Runtime" doi.org/10.1145/3201898
5. "The Design Space of Multi-Language Development Environments" doi.org/10.1007/s10270-013-0376-y
6. "Multilanguage Debugger Architecture" doi.org/10.1007/978-3-642-11266-9_61
7. "Debug All Your Code: Portable Mixed-environment Debugging" doi.org/10.1145/1640089.1640105
   "Code Smells for Multi-language Systems" doi.org/10.1145/3361149.3361161
8. "An Empirical Assessment of Polyglot-ism in GitHub" doi.org/10.1145/2601248.2601269
   "Investigating the Effect of Polyglot Programming on Developers" doi.org/10.1109/VL/HCC51201.2021.9576404
   "Automated Support for Seamless Interoperability in Polylingual Software Systems" doi.org/10.1145/250707.239123
9. "Multi-paradigm modeling for cyber–physical systems: A systematic mapping review" doi.org/10.1016/j.jss.2021.111081
10. "Globalizing Modeling Languages" doi.org/10.1109/MC.2014.147
11. "Language Composition Untangled" doi.org/10.1145/2427048.2427055
12. "A Hybrid Synchronous Language with Hierarchical Automata: Static Typing and Translation to Synchronous Code" doi.org/10.1145/2038642.2038664

Figure 1: Ambiguous Terms Related to Polyglot Software Development

ducted a survey to gather responses from 139 professional software developers who reported an average of 7 languages per project, with over 90% of developers reporting problems related to language interactions [5].

There are many reasons for why several languages are used in combination: socio-technical reasons such as practitioners's expertise/preferences and best practices; conceptual reasons such as separation of concerns, design decisions, and variability management; technical reasons such as availability of libraries/functionality, efficiency, automation/reproduction, reasoning/analysis, and quality assurance; and business reasons such as coping with legacy applications/systems, technological debt, and vendor-lock in.

It is therefore no surprise that many communities are investigating the combination of several languages [6]. Yet, a long and ambiguous list of terms exists for polyglot software development from different communities. We illustrated all the terms we discovered in Figure 1, and also provide references to representative papers in the scientific literature that use that terminology. While by no means exhaustive, this list already showcases the lack of a common view, that is to say: different communities often use the same term with different meanings, or use different terms for the same meaning. The effect is a vastly ambiguous picture of the term *polyglot*, as well as a merely blurry sketch of common associated implications for a development process. Our goal is to clarify this fuzziness by providing a clear definition of polyglot software development. In turn, this may qualify as a common denominator for individual domain experts, to leverage an anti-silo effect that facilitates the exchange of contained knowledge.

In the remainder of this article, we first introduce a conceptual model for polyglot software development that allows us to clearly define polyglot software development, its polyglot processes and tasks, and whether polyglot

stakeholders are required. We exemplify the conceptual model with Mastodon and other examples. We further characterize polyglot software development and elaborate on polyglot programming, before concluding with open challenges and perspectives.

Conceptual Model

To unify the large variety of terms related to the use of languages, this section proposes a conceptual model for software development with multiple languages in Figure 2. Note that we focus only on those development concepts that directly involve or somehow relate to languages.

At the heart of our conceptual model is the **Task**, which is a unit of work (e.g., "specify web views") that involves a set of **StakeholderRole**s (e.g., "developer"). One **Stakeholder** may play one or more stakeholder roles. A task requires the use of one or several **Artifact**s expressed in one or more **Language**s, because the artifacts are either consumed as **input** or produced as **output** by the task. Some artifacts may be integrated with each other using one or several **IntegrationTechnique**s. A language offers one or more **Paradigm**s in which to formulate the intended properties or behavior of the system under development (e.g., "object-oriented programming", "functional programming", and "procedural programming" for Ruby).

An important distinction for a stakeholder role to be associated with an artifact of a language is that the role needs to actively **edit** something in the artifact (e.g., write code, add a model element). If this is not the case, then the stakeholder does not **use** the language. Simply viewing or executing an artifact does not qualify (e.g., the result of a model generation or compilation, respectively). For example, while the task of compiling code will require an input artifact and will output bytecode/machine code, most stakeholders will not directly engage with the compilation results. Hence, the stakeholders do not use the bytecode/machine code language nor do they use the language of the input artifact since they do not edit it.

A ternary association is required since an artifact may be expressed in several languages and a stakeholder role may only use some of those languages. For example, a performance specialist may edit only the MARTE annotations in a UML class diagram.

To bring artifacts of languages together for a task, a certain **IntegrationTechnique** is used, where each artifact and its language(s) play a role, captured in the conceptual model by the qualified associations between integration technique and artifact and between integration technique and language.

For example, the "specify web views" task in Mastodon involves the creation of a "Haml" output artifact for the front-end developer and a "Ruby" output artifact for the back-end developer. These developers may in fact be the same person, as a stakeholder may play multiple roles. Since this is a task that requires integrating two or more languages, the task uses an integration technique where Haml plays the role of "template" and Ruby is the "interpreter". The follow-up runtime task "generate web views" that produces artifacts in "HTML" from the integrated Haml+Ruby specifications is a task that involves no editing stakeholders, but has two input artifacts and one output artifact.

Finally, during software development, tasks are typically performed in some order. For this purpose our conceptual model contains the **Process** concept which groups a set of tasks and a set of stakeholders. For the sake of practicality we also allow processes to contain subprocesses, i.e., to form hierarchies. We are not explicitly modelling the partial ordering

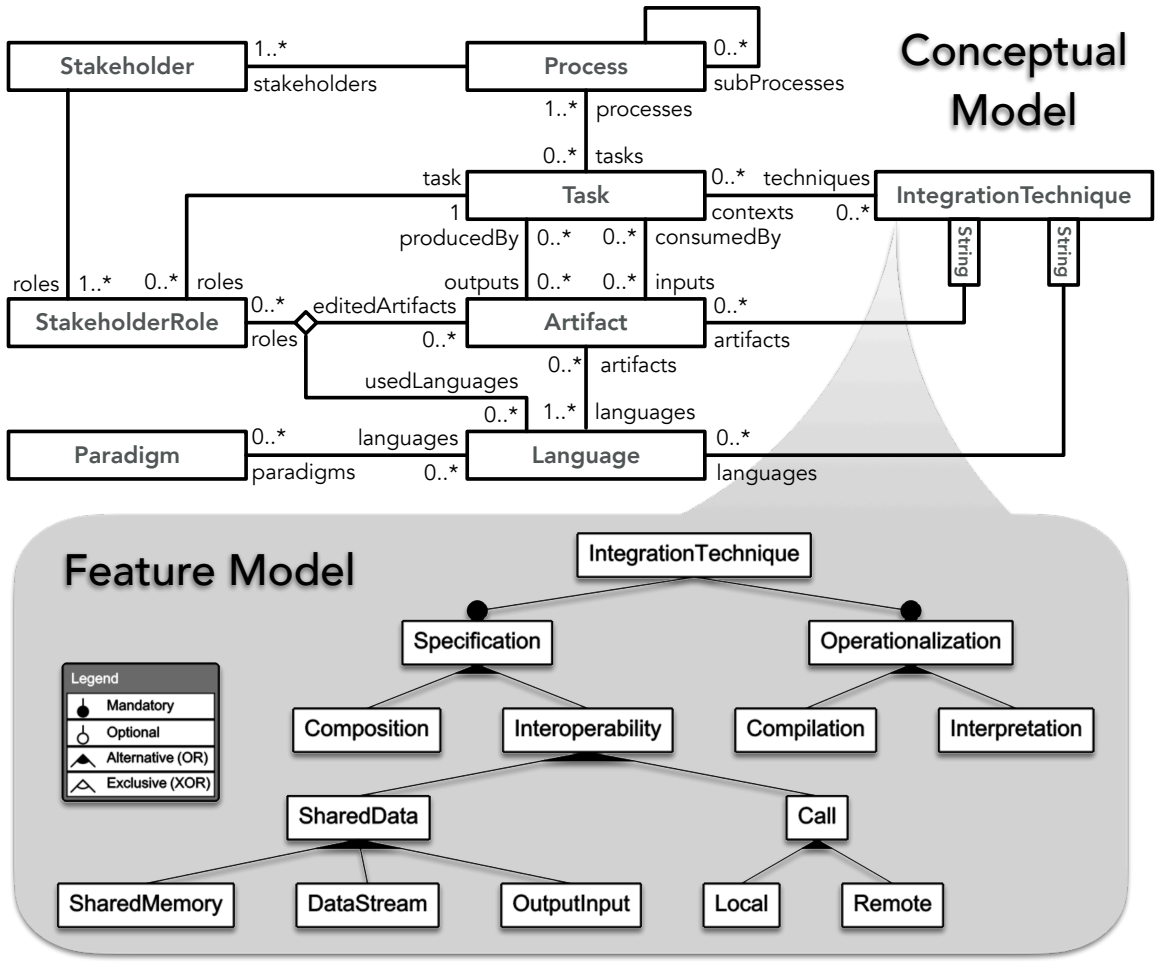


Figure 2: Conceptual Model for Polyglot Software Development and Feature Model Illustrating Different Integration Techniques

of tasks within a process, though, as it is of no relevance regarding our discussion on polyglotism. Implicitly a partial ordering is established nevertheless, because tasks that require input artifacts can only be performed once the artifacts have been output by a preceding task in the process.

To finalize, we need to make the definition of a task more precise to avoid confusion between process, subprocesses, and tasks. A task is supposed to be the smallest unit of work, i.e., it should not arbitrarily consist of artifacts with many languages that are not directly related to each other (e.g., one task is defined for a whole process instead of splitting the process into several atomic tasks). We can do this by adding a constraint to the conceptual model.

A task may only contain artifact(s) of more than one language if the languages are integrated by a technique.

context Task:
inv: roles.usedLanguages→asSet()→size() ≥ 2 **implies**
techniques.artifacts→includesAll(roles.editedArtifacts)
and
techniques.languages→includesAll(roles.usedLanguages)

In the Mastodon project, for example, an activity such as "specify web views and build script" that includes Ruby, Haml, and Dockerfile, would have to be modelled as two tasks.

Polyglotism

Since the production of software always involves translation from human readable languages to machine languages, all software development can be seen as polyglot. However, we are going to give a more nuanced definition of polyglot based on the *use of languages for a task* as explained earlier.

The conceptual model introduced allows for thinking about polyglotism at multiple levels, i.e., at the task and the process level, and also with respect to stakeholder roles and stakeholders.

A task is polyglot if the stakeholder roles of the task edit artifact(s) in more than one language.

context Task **def** isPolyglot(): Boolean =
roles.usedLanguages→asSet()→size() ≥ 2

For example, consider a task "specify web page" with an output artifact in two languages, i.e., HTML and CSS. The task could require two stakeholder roles, one for HTML and one for CSS, or the same stakeholder role for both languages. In both cases the task is polyglot, and an integration technique is required because two languages are used in an edited artifact. Another common situation occurs when a low-level language is embedded within a high-level programming language. For example, it is common to embed C code in Python for increasing performance of computationally expensive algorithms, and therefore any programming task with such a setup is polyglot. However, if the task is fully automated, i.e., there is no stakeholder role, then the task is not polyglot. A polyglot task requires *active stakeholder involvement with multiple languages.*

This distinction is also exemplified by the tasks "write model transformation" and "run model transformation". Both tasks are not polyglot. The former is not polyglot because it involves a stakeholder role that edits the output artifact in only a single language, e.g., an ATL script for the model transformation, based on two input artifacts, i.e., the metamodels for the source and target languages of the transformation. The latter is not polyglot because it is automated and does not involve an active stakeholder role but three input artifacts (e.g., the ATL script and two models corresponding to the source and target metamodels) and an automatically created output artifact in the target language.

Similarly, the specification of a consistency rule or an analysis script (e.g., energy consumption for web pages) are tasks that are not polyglot unless the specification itself requires multiple languages. The metamodels of the languages for which a consistency rule is specified are the input artifacts and not edited. Likewise, the web pages that are analyzed are also input artifacts that are not edited. The execution of the consistency rule (which may perform changes to the input models) and the running of the analysis are automated, and hence they are not polyglot because no stakeholder is actively involved.

Based on the definition of a polyglot task, similar definitions for stakeholder roles, stakeholders, and processes can be formulated.

A stakeholder role is polyglot if it requires to edit artifact(s) in more than one language.

context StakeholderRole **def**: isPolyglot(): Boolean =
usedLanguages→asSet()→size() ≥ 2

A stakeholder needs to be polyglot if the union of roles they play edit artifact(s) in more than one language.

context Stakeholder **def**: isPolyglot(): Boolean =
roles.usedLanguages→asSet()→size() ≥ 2

A process is polyglot if the stakeholder roles of the tasks that it or any of its subprocesses contains edit artifact(s) in more than one language.

context Process **def**: isPolyglot(): Boolean =
self.closure(subprocesses).tasks.roles.usedLanguages
→asSet()→size() ≥ 2

For example, the earlier Ruby+Haml "specify web views" task has task-level polyglotism, but some other systems may exhibit process-level polyglotism. For example, in a "data visualization" process, one task may use Python to transform data, and another task may use R to visualize the transformed data. At the uppermost process level, many modern systems will exhibit polyglotism (e.g., using a formal requirements language and an implementation language).

On the other hand, there are still many projects which are not polyglot. For instance, there are numerous domains such as data science, biology, or finance whose projects use a single language (such as Python) for all tasks (e.g., data curation, analysis, computation, visualization, etc.). Such a task is represented in the conceptual model by a task that produces an output artifact edited by a stakeholder role but only in the Python language and without any integration technique.

In literature and practice, different communities refer to the concepts in our conceptual model differently. This existing terminology (cf. Figure 1) can be mapped to our conceptual model as follows. "Polyglot development/programming" is in line with our definition of polyglotism. Within it, "multi-paradigm modeling/globalization" are seminal approaches with an explicit focus on language integration (or *composition*) techniques. "Polyglot programming" and "Polylingual software", as well as "multi-language development" refer to a development process with tasks that span more than one language, but multi-language development is more general and refers to approaches without language integration technique. These terms should not be confused with "multi-lingual" software development tools, which include all language-agnostic tools that can be reused across a well-defined range of existing languages. "Cross-language" refers to tools that can operate across multiple languages while relating them (e.g., when performing clone detection across Java and Python programs, the tool not only has to work on both Java and Python programs, but also has to relate them). "Multi-language tools and development environments" focus on the tooling aspect, but do not contribute to the underlying foundations of software development with multiple languages. In contrast, "language composition" techniques refers to work on the foundations for dealing with multiple languages, which may involve polygot development, but also language design and implementation for hybrid programming languages, i.e., with multiple paradigms, but without language integration techniques. Finally, "hybrid programming" refers to a single language that combines more than one paradigm (e.g., continuous and discrete programming).

All communities depicted in Figure 1 build on the foundations of model-driven engineering (MDE) as well as language-oriented programming (LOP). In MDE, models play a central role during software developent, as the whole software life cycle is seen as a process of model production, refinement, and integration [7]. Similary, in LOP a language is treated like any other development artifact and instead of using general-purpose languages, the creation and implementation of domain-specific languages for solving problems is preferred [8].

Integration Techniques

In this section, we provide more details on existing language integration techniques mentioned in the conceptual model by focusing on polyglot programming and hence executable artifacts. Figure 2 depicts the possible choices for the integration technique of executable artifacts as a feature model. Each feature represents a choice.

Each integration technique requires at least one choice for its **Specification** and one for its **Operationalization**. The former handles how we define the interaction between languages at design time, and the latter specifies how the interaction is realized during execution. The specification can be implemented with a **Composition** solution [9] and/or **Interoperability** solution [10]. Composition covers all various techniques from embedding of a language into another to unifying two languages at the syntax and/or semantic levels. We do not provide further details on the many existing composition techniques and their classification, but the interested reader is referred to this survey paper [11].

Interoperability covers the communication between different languages. Interoperability needs to deal with two important aspects, namely how data sharing (**SharedData**) and **Call**s are handled. The calls between languages can either be **Remote**, when the call goes through a network, or else **Local**. The shared data can either be implemented with a **SharedMemory**, a data streaming mechanism (**DataStream**), or simply by one language writing some output that another language consumes as an input, for example through a file on disk (**OutputInput**).

Operationalization represents how the specification will be realized during execution. This can either be achieved through **Compilation** and/or **Interpretation**, i.e., either by executing the relationships between the two languages at compile-time, e.g., Melange [12], or by interpreting the specified relationships at runtime, e.g., BCOoL [13].

For example, a Scala program calling Java libraries fits the following choices in the feature model of Figure 2: shared memory and local call interoperability, and compilation operationalization. Another example is the case where code in one language invokes code in another language, e.g., the new Foreign Function and Memory (FFM) API in Java allows Java code to invoke low-level code and access data outside the JVM on the same machine. In other cases, interoperability happens through the use of an interface definition language, e.g., OpenAPI, from which client and server stubs are generated. This integration technique would use output/input and remote call interoperability. If, for example, Python talks to compiled C++, then the operationalization would use interpretation on the Python side and compilation on the C++ side.

Taking again the example of Mastodon, different integration techniques are used at various times. For instance, the integration technique between Haml and Ruby uses interoperability as specification through local calls to Haml code as well as shared memory, and is operationalized using the Haml interpreter. A second used integration technique between Ruby and JavaScript relies on interoperability as specification with a data stream using Redis and remote calls, and interpretation as operationalization.

As mentioned in the previous section, not every integration technique is associated with a polyglot task because stakeholder involvement is required. A fully automated task that is not polyglot may still have an integration technique. However, the earlier integration techniques between Haml and Ruby and between Ruby and JavaScript belong indeed to polyglot tasks, since the stakeholders edit artifacts in all languages.

Challenges and Perspectives

As mentioned above, most software development is already polyglot to some extent, and it is not surprising that we see more and more languages appear in modern software projects, e.g., in order to build systems more efficiently or to separate concerns (see sidebar). Polyglot software development, however, faces many technical, process-related, educational, and community challenges. We discuss them and provide related perspectives.

Technical Challenges and Perspectives

Some software development activities that are well understood within a single language become challenging in polyglot software development. For example, we need to develop novel and intuitive tools and techniques for polyglot software comprehension, polyglot software analysis (including, e.g., semantic alignment, debugging, and profiling) and polyglot software documentation. Similarly, whereas testing each language separately is well supported, testing the overall polyglot program and its different interactions remains a challenge. Indeed, a test case would require to integrate the "oracle states" of different programs written in different languages.

Techniques for software security will have to be revisited in the context of polyglot software development. For example, we need to ensure secure communication channels between languages and enable cross-language access control.

When developing polyglot programs, we often have to write the language integration logic from scratch. As a first step, our current code generators should be extended with a layer that automatically exposes the services by system components written in one language to the other languages. Ultimately, the goal is to have full-fledged code generation for polyglot programs that includes the integration logic.

Finally, new opportunities await with the application of AI techniques to polyglot software development. More specifically, we should investigate how to capitalize on multilingual trained LLMs [14].

Process-Related Challenges and Perspectives

We must develop strategies to determine the most appropriate combination of languages to use for a given task, also taking into account the socio-technical context. We might even benefit from identifying anti-patterns of language combinations from unsuccessful projects. We need to develop a theory for tradeoffs between productivity and complexity involved with polyglotism. Adding a language that is well suited to a task can speed up development, but it might also increase the cognitive load for the developer and require a broader range of development skills. Finally, a completely new challenge arises regarding language evolution. As many languages are used and interact with each other, when one evolves, others may be impacted as well. We would need to develop tools and techniques for polyglot impact analysis that can reason over multiple languages simultaneously. Then, when impacts are identified, they must be considered and languages have to co-evolve accordingly.

Educational Challenges and Perspectives

Most software engineering curricula contain courses that teach languages and paradigms, but only rarely students are explicitly exposed to polyglot software development with dedicated support for the coordinated use of multiple languages [15]. We need to find ways to use the presented conceptual model as an edu-

cation tool to convey the real-life complexities to students who are used to "lab" projects, as well as augment our teaching practices with examples of polyglot development activities and techniques to give a more holistic view of real-life software development.

Community Challenges and Perspectives

In this paper, we have identified similarities and variabilities in the terminology related to polyglot development used by various software engineering communities. Traditionally, different communities have been working in relative isolation from each other, and work like the one presented here can help break down the silos that separate them. Yet this work needs to be amended by the plethora of other communities dealing with polyglotism to enable global cross-fertilization.

Sidebar

To make a program it takes a language and a machine.
One language, and a machine. At least in theory.
But practice asks for separation of concerns,
a division of labor between you, and me, and her.
The people demand speed and efficiency, but alas,
a language can compute anything, but is it fast?
So then we invite another and thus transgress
out of paradise with a bite, a sudden kiss of death,
and descend the tarpit of our festished Babylon,
sentenced to tame the Hydra that we have spawned.
Let's study the techniques of our tongues' embrace:
A language alongside another wants to communicate.
A language on top of another is one that generates.
A language within a language, a hatch for my escape.

So many tradeoffs at stake
when complexity procreates.

Tijs van der Storm

References

[1] M. S. Murtazina and T. V. Avdeenko, "Ontology-based approach to the requirements engineering in agile environment," in *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*. IEEE, 2018, pp. 496–501.

[2] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[3] T. Kosar, S. Bohra, and M. Mernik, "Domain-specific languages: A systematic mapping study," *Information and Software Technology*, vol. 71, pp. 77–91, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584915001858

[4] A. Raman, S. Joglekar, E. D. Cristofaro, N. Sastry, and G. Tyson, "Challenges in the decentralised web: The mastodon case," in *Proceedings of the internet measurement conference*, 2019, pp. 217–229.

[5] P. Mayer, M. Kirsch, and M. A. Le, "On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers," *Journal of Software Engineering Research and Development*, vol. 5, no. 1, p. 1, 2017. [Online]. Available: https://doi.org/10.1186/s40411-017-0035-z

[6] T. Degueule, B. Combemale, A. Blouin, O. Barais, and J.-M. Jézéquel, "Melange: A meta-language for modular and reusable development of dsls," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 25–36. [Online]. Available: https://doi.org/10.1145/2814251.2814252

[7] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, pp. 41–47, 2006.

[8] R. Pickering, *Language-Oriented Programming*. Berkeley, CA: Apress, 2010, pp. 327–349. [Online]. Available: https://doi.org/10.1007/978-1-4302-2390-0_12

[9] J. Kienzle, G. Mussbacher, B. Combemale, and J. Deantoni, "A unifying framework for homogeneous model composition," *Software & Systems Modeling*, vol. 18, no. 5, pp. 3005–3023, Jan 2019. [Online]. Available: https://doi.org/10.1007/s10270-018-00707-8

[10] M. Grimmer, R. Schatz, C. Seaton, T. Würthinger, M. Luján, and H. Mössenböck, "Cross-language interoperability in a multi-language runtime," *ACM Trans. Program. Lang. Syst.*, vol. 40, no. 2, may 2018. [Online]. Available: https://doi.org/10.1145/3201898

[11] S. Erdweg, P. G. Giarrusso, and T. Rendel, "Language composition untangled," in *Proceedings of the Twelfth Workshop on Language Descriptions, Tools, and Applications*, ser. LDTA '12. New York, NY, USA: Association for Computing Machinery, 2012. [Online]. Available: https://doi.org/10.1145/2427048.2427055

[12] T. Degueule, B. Combemale, A. Blouin, O. Barais, and J.-M. Jézéquel, "Melange: A meta-language for modular and reusable development of dsls," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 25–36. [Online]. Available: https://doi.org/10.1145/2814251.2814252

[13] M. E. Vara Larsen, J. DeAntoni, B. Combemale, and F. Mallet, "A behavioral coordination operator language (bcool)," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 186–195.

[14] T. Ahmed and P. Devanbu, "Multilingual training for software engineering," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2022, pp. 1443–1455. [Online]. Available: https://doi.ieeecomputersociety.org/10.1145/3510003.3510049

[15] M. Ardis, D. Budgen, G. W. Hislop, J. Offutt, M. Sebern, and W. Visser, "Se 2014: Curriculum guidelines for undergraduate degree programs in software engineering," *Computer*, vol. 48, no. 11, pp. 106–109, 2015.

Biographies

Prof. Gunter Mussbacher is an Associate Professor at McGill University. His research interests include Model-Driven Requirements Engineering, Software Language Engineering, Next-Generation Reuse Frameworks, Sustainability, and Human Values. More information about him can be found at http://www.ece.mcgill.ca/~gmussb1/. Contact him at gunter.mussbacher@mcgill.ca.

Prof. Benoit Combemale is Full Professor of Software Engineering at the University of Rennes, France. He is teaching at the engineering school ESIR, and he is co-head of the DiverSE research team, joint between the IRISA and Inria labs. He is also adjunct researcher in the team SM@RT of the IRIT labs, Toulouse, France, scientific advisor of the startup TwinIT, and Editor in Chief of the Springer Nature Journal on Software and Systems Modeling (SoSyM). His research interests include Model-Driven Engineering (MDE), software language engineering (SLE), software validation & verification (V&V) and DevOps. More information at http://combemale.fr/. Contact him at benoit.combemale@irisa.fr.

Prof. Jörg Kienzle is a researcher at ITIS Software, Universidad de Málaga, Málaga, Spain, and Full Professor at McGill University, Montréal, Québec, Canada, where he leads the Software Composition and Reuse lab (SCORE). His research interests include model-driven software development, software product lines, separation of concerns, reuse, software composition, and modularity. Further information about him can be found at https://djeminy.github.io. Contact him at Joerg.Kienzle@uma.es or Joerg.Kienzle@mcgill.ca.

Prof. Lola Burgueño is an Associate Professor at the University of Málaga, Spain. Her research interests include the application of artificial intelligence to improve software development processes and tools, uncertainty management during the software design phase, and model-based software testing. More information at https://lolaburgueno.github.io. Contact her at lolaburgueno@uma.es.

Dr. Antonio Garcia-Dominguez is a Lecturer in Software Engineering at the Department of Computer Science of the University of York, and a member of the Automated Software Engineering research group. Antonio's main research interests are model-driven software engineering (with lines of work on scalability of MDSE and on runtime models for explainability), and software testing (specifically, search-based testing and metamorphic testing). More information at https://www-users.york.ac.uk/~agd516/. Contact him at a.garcia-dominguez@york.ac.uk.

Prof. Jean-Marc Jézéquel is a Professor at the University of Rennes and a member of the DiverSE team at IRISA/Inria. Since 2024 he is President of Informatics Europe. From 2012 to 2020 he was Director of IRISA, one of the largest public research lab in Informatics in France. Since Sept. 2023, he is a fellow of the Institut Universitaire de France (IUF). His interests include model-driven software engineering for software product lines. More information at http://people.irisa.fr/Jean-Marc.Jezequel. Contact him at jezequel@irisa.fr.

Gwendal Jouneaux is a Ph.D. student in software engineering at University of Rennes (France) and a member of the DiverSE research team. He is interested in model-driven engineering, software language engineering, domain-specific languages, and in particular self-adaptable languages. More information at https://www.gwendal-jouneaux.fr. Contact him at gwendal.jouneaux@irisa.fr.

Dr. Djamel-Eddine Khelladi is a CNRS researcher at the IRISA research lab in the DiverSE team, Université Rennes 1, Rennes, France. His current research interests are Software engineering, Model-Driven Engineering, Software Evolution, Coevolution, Empirical Software Engineering, Incremental Build, Scaling Code Analysis, Software Processes. More information at http://people.irisa.fr/Djamel-Eddine.Khelladi/. Contact him at djamel-eddine.khelladi@irisa.fr.

Prof. Sébastien Mosser is Professor of software engineering at McMaster University (Canada), and a member of the McSCert research centre. His research interests are related to domain-specific modeling and software composition from a language point of view. More information at https://mosser.github.io/. Contact him at mossers@mcmaster.ca.

Corinne Pulgar is a master student at Ecole de Technologie Supérieure, Université du Québec, Montréal, Quebec, Canada. More information at https://www.linkedin.com/in/corinne-pulgar-12a58190/. Contact her at corinne.pulgar.1@ens.etsmtl.ca.

Houari Sahraoui is a professor at the Department of Computer Science and Operations Research of the Université de Montréal at Montréal, Québec, Canada. His research interests include artificial intelligence techniques applied to software engineering, search-based software engineering, and model-driven engineering. Contact him at sahraouh@iro.umontreal.ca.

Maximilian Schiedermeier is a Ph.D. student in Computer Science at McGill University, Montréal (Canada). His research focuses on DSL-based tools for REST API development / security protocol integration and empirical assessments. More information at https://m5c.github.io/. Contact him at max.schiedermeier@mcgill.ca.

Prof. dr. Tijs van der Storm is senior researcher and group leader of the Software Analysis & Transformation (SWAT) group at CWI, and part-time Full Professor of software engineering at the University of Groningen (RUG). His expertise spans programming languages, language engineering, domain-specific languages (DSLs), meta programming, and model-driven engineering. He is the chair of VERSEN, the Dutch national association for software engineering research, chair of IFIP TC2 Working Group on Programming Language Design, and treasurer of the European Association for Programming Languages and Systems (EAPLS). More information can be found here: http://www.cwi.nl/~storm. Contact him at storm@cwi.nl.