Rethinking paradigmatic collaboration: new metrics for inter-language synergy in software engineering

Mikel Vandeloise[1], Gonzague Yernaux[2] and Wim Vanhoof[3]

Faculty of Computer Science, University of Namur, Rue Grandgagnage 21, 5000 Namur, Belgium

Abstract

This paper introduces two novel metrics for assessing collaboration potential between programming paradigms and languages, offering a fresh approach to evaluating inter-paradigm interactions. The quantitative framework introduced challenges traditional methods by highlighting both the opportunities and the challenges of integrating paradigms with varying characteristics. Initial findings suggest that the metric effectively captures collaboration within similar paradigms, with functional paradigms demonstrating particularly strong collaboration potential. However, it also underscores the complexity of unifying divergent paradigms. The goal of this paper is to provoke discussion and foster further research, such as the refinement of metrics to obtain useful measurements in technology selection steps and interdisciplinary software development.

Keywords

Programming paradigms, Collaboration metrics, Interoperability, Languages integration

1. Introduction

Programming paradigms are fundamental frameworks that guide the conceptualisation, implementation, and evolution of software systems. These paradigms, ranging from simple declarative structures to complex procedural and concurrent constructs, must be carefully selected to meet the evolving challenges of software projects [1]. Often, a hybrid approach becomes necessary. However, integrating multiple paradigms increases complexity, particularly in terms of interoperability and maintainability, which are crucial for long-term software evolution [2, 3]. Effectively managing these complexities is key to the success and sustainability of evolving software systems. To clarify our approach, we define *inter-paradigmatic collaboration* as follows.

Definition 1.1 *Collaboration between programming paradigms* (or *inter-paradigmatic collaboration*) refers to the complementary potential of two paradigms to work together effectively at the abstract and conceptual level. This potential is considered the inverse of the effort required to align and integrate their conceptual frameworks. The collaboration relies on the balance of two key dimensions, each displaying an inherent tension:

- *Conceptual complementarity* is the extent to which the core concepts and strengths of one paradigm complement and reinforce those of another, mitigating each other's weaknesses and resulting in a more robust and versatile combined framework.
- *Abstract interoperability* is the ease with which different paradigms can conceptually interoperate and integrate their theoretical constructs, rewarding seamless integration while penalising the cognitive and structural effort required to combine them.

With most existing research on interoperability between programming languages focusing on the technical and syntactic aspects of the question [4, 5], there is still a lack of *quantitative* tools capable of encapsulating the notion at the level of programming paradigms. Our research aims to address this gap. The key contributions of this paper are threefold.

BENEVOL24: The 23rd Belgium-Netherlands Software Evolution Workshop, November 21-22, Namur, Belgium

mikel.vandeloise@unamur.be (M. Vandeloise); gonzague.yernaux@unamur.be (G. Yernaux); wim.vanhoof@unamur.be (W. Vanhoof)

0009-0002-0858-471X (M. Vandeloise); 0000-0001-6430-8168 (G. Yernaux); 0000-0003-3769-6294 (W. Vanhoof)

1. We adapt an existing taxonomy of paradigms to allow for a rigorous formulation of the concept of *paradigm*.
2. We develop a *metric* tailored to assess the potential for collaboration between different paradigms. The metric is designed to allow for a certain (parametric) flexibility, that can be fine-tuned based on research objectives.
3. We extend this metric to evaluate the potential for collaboration between different *languages*, focusing on their underlying paradigms.

2. Towards a refined taxonomy of programming paradigms

In 2009, Van Roy's foundational work [6] set the stage for a comprehensive classification of programming paradigms. In this section, we extend and refine his seminal taxonomy through the somewhat specific lens of inter-language synergies and collaborations. The theoretical foundations of interest in this pursuit have, in fact, been little investigated in the past; the main sources that we have collected and selected through an extensive literature review are [7, 1, 8, 9].

Our taxonomy is essentially embodied by an acyclic directed graph where each node corresponds to a distinct paradigm (Figure 1a), and each edge signifies the acquisition of a concept (Figure 1b) that makes a paradigm evolve into a more specialised form. We call the acquisition of specific concepts through such evolutionary paths **conceptual hierarchy**. The resulting structure allows incorporating the paradigms' inherent concepts as well as their computable attributes such as Turing completeness (Figure 1c) and non-determinism observability (Figure 1d). These attributes allow categorising the paradigms into broader groups known as *meta-paradigms* (Figure 1e), which help to provide a macroscopic view of their relationships, according to the following definition.

Definition 2.1 A *meta-paradigm* is a collection of core concepts, attributes, and principles shared by multiple paradigms. It conceptually regroups features that influence the models, design, and implementation of groups of paradigms. Transitions between meta-paradigms result from acquiring specific concepts, leading to shifts in focus and application.

An interesting case arises when distinct paradigms present similar (yet separated) conceptual hierarchies, which can in some cases influence their compatibility predispositions. We call this specific relation *propagation*; see Figure 1f for concrete examples.

Definition 2.2 *Propagation* is a relationship that occurs between two paradigms that share a same set of concepts, but acquired in different orders.

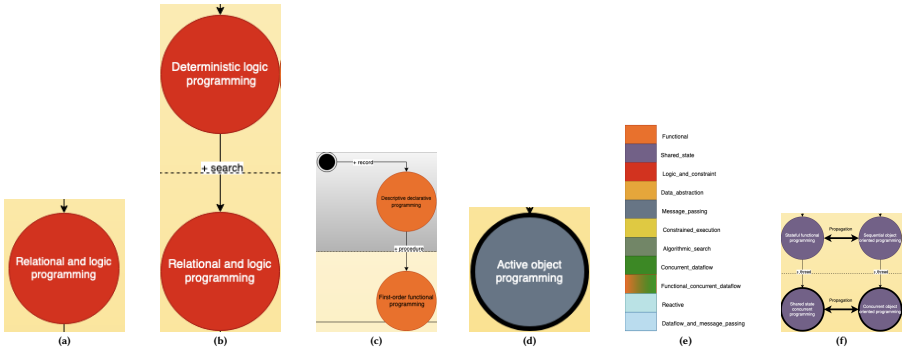


Figure 1: Fragments of the taxonomy. The grey-to-yellow gradient in (Figure 1c) indicates the shift from paradigms limited to data structures to those enabling Turing completeness.

In total, our proposed taxonomy includes 28 distinct paradigms defined by 17 key concepts and 2 observable binary characteristics and organised into 11 meta-paradigms. The whole graph is depicted

in Appendix A's Figure 4, along with an exhaustive enumeration of the different constitutive sets and an illustration of our approach (Table 1) in which we instantiate our framework on the so-called "deterministic logic" paradigm. More information on the array of paradigms considered can also be found online in the form of a poster [10].

3. A measure for paradigmatic collaboration

Based on our taxonomy, we can formulate a definition of the very notion of a paradigm.

Definition 3.1 A *programming paradigm*, often denoted P, is formally defined as a 6-tuple, i.e. $P = (C, F, S, T, O, M)$, where each element represents a specific dimension of the paradigm:

- C ($\subseteq$ Concepts) is the non-empty set of key concepts that define the paradigm.
- F ($\subseteq$ Paradigms) is the *parent* paradigms from which P evolved.
- S ($\subseteq$ Paradigms) is the *child* paradigms descended from P.
- T ($\in \{0, 1\}$) is a Boolean indicating if all languages using P can be Turing-complete.
- O ($\in \{0, 1\}$) is a Boolean indicating if P permits observable non-determinism.
- M ($\in$ Meta-paradigms) is the associated meta-paradigm.

Now, to assess the compatibility and collaboration potential between paradigms, we developed a metric that considers their structure, relationships, and key properties.

Definition 3.2 The *Inter-Paradigm Collaboration Metric* (PCM) : Paradigms $\times$ Paradigms $\mapsto [0, 1]$ is defined, for $P_1 = (C_1, F_1, S_1, T_1, O_1, M_1)$ and $P_2 = (C_2, F_2, S_2, T_2, O_2, M_2)$, as

$$\begin{aligned} PCM(P_1, P_2) \quad = \quad & \alpha \times IC(C_1, C_2) + \beta \times KR(F_1, S_2) + \gamma \times KR(F_2, S_1) \\ & + \delta \times MP(M_1, M_2) + \epsilon \times TC(T_1, T_2) + \zeta \times OND(O_1, O_2) \end{aligned}$$

$$\text{where } \alpha, \beta, \gamma, \delta, \epsilon, \zeta \in [0, 1] \wedge \alpha + \beta + \gamma + \delta + \epsilon + \zeta = 1 \wedge \beta = \gamma$$

The parametric coefficients α, β, γ, δ, ϵ, and ζ are called the *collaboration weights*. In this initial study, each weight is set to $\frac{1}{6}$ to ensure equal contribution from the six factors, although future research may adjust these values based on specific requirements.

The first component of the metric is the *intersection of concepts*, calculated as the ratio of shared concepts to the total number of concepts, and incarnated in the definition by the function $IC : \mathscr{P}(\text{Concepts}) \times \mathscr{P}(\text{Concepts}) \to \mathbb{R}$ defined by $IC(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$. This ratio helps determine how well paradigms complement each other.

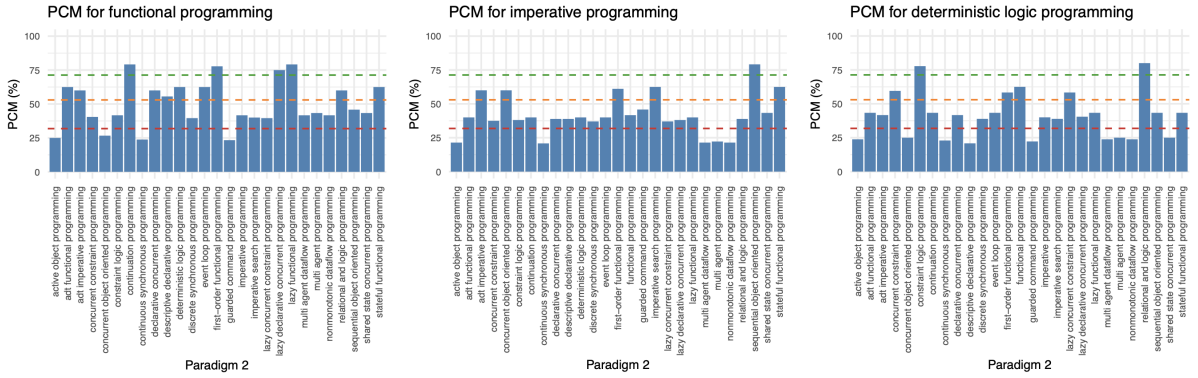
Next, we check for *kinship relationships* by evaluating whether there is overlap between the parent paradigms of one and the child paradigms of the other. The notion is formalised by the function $KR : \mathscr{P}(\text{Paradigms}) \times \mathscr{P}(\text{Paradigms}) \mapsto \{0, 1\}$ where $KR(F, S) = 1$ if and only if $F \cap S \neq \emptyset$. Although KR is asymmetric, using both $KR(F_1, S_2)$ and $KR(F_2, S_1)$ in PCM ensures symmetry in the overall metric, thus preventing PCM from requiring symmetry in each of its components.

The *compatibility of meta-paradigms* is determined by a function MP : Meta-paradigms $\times$ Meta-paradigms $\mapsto \{0, 1\}$ such that $MP(M_1, M_2) = 1 \Leftrightarrow M_1 = M_2$.

We also check for *Turing completeness* with $TC : \{0, 1\}^2 \mapsto \{0, 1\}$ where $TC(T_1, T_2) = T_1 \wedge T_2$.

Finally, we similarly analyse *observable non-determinism* by determining whether both paradigms permit it, formally using the function $OND : \{0, 1\}^2 \mapsto \{0, 1\}$ such that $OND(O_1, O_2) = O_1 \wedge O_2$.

These metrics provide a systematic evaluation of compatibility and collaboration potential, focusing on conceptual and computational alignment [11, 12]. We applied this approach to 28 paradigms, with notable examples including the functional, imperative, and deterministic logic paradigms. The complete results are available in [13], and a focus on three paradigms is presented in Figure 2. Interestingly, the functional paradigm (Figure 2a) demonstrates a high level of collaboration potential with many pairs, suggesting strong compatibility and deep integration possibilities. In contrast, the imperative paradigm (Figure 2b) shows more variability, with a balanced distribution between low and moderate collaboration potential, this time hinting that specialised approaches may be needed to achieve effective integration. As a last remark in this non-exhaustive analysis, we noticed that the deterministic logic paradigm (Figure 2c) shows strong collaboration potential within its own meta-paradigm ("constraints and logic"), but generally moderate compatibility outside of it, indicating a more selective capacity for integration. The whole dataset, modelled using Haskell, is accessible in an online repository[1].



(a) *PCM* for Functional (b) *PCM* for Imperative (c) *PCM* for Deterministic Logic

Figure 2: *PCM* comparisons for three different programming paradigms

4. Extending the collaboration metric to programming languages

To integrate the results obtained for each constitutive paradigm of a language, we can now define a programming language as follows.

Definition 4.1 A *programming language* is a set of multiple programming paradigms that collectively define its capabilities and behavioural characteristics. For a language named L, the corresponding set is denoted by $P_L = \{P_1, P_2, \dots, P_n\}$, where each P_i ($i \in 1..n$) is a distinct paradigm (represented by its tuple) contributing to the language's comprehensive functionality.

Detailed descriptions of the paradigmatic structures of individual languages are available in [10]. Given the definition above, we can now define a new metric *LCM*, which captures paradigmatic compatibility as well, but this time across languages. This metric operates on the combined paradigm sets of the languages while preserving the integrity of each paradigm, even if it appears in both sets. Definition 4.2 below formalises this approach. It uses an ordering relation, denoted by $>$, to compare the paradigms that are represented within the taxonomy. The first incarnation of the relation can be defined as follows. For two paradigms c and c', represented as tuples $c = (x_0, x_1, \dots, x_{m-1})$ and $c' = (y_0, y_1, \dots, y_{n-1})$, we define the order as

$$c' > c \Leftrightarrow (\exists k \in \{0, 1, \dots, \min(m, n) - 1\} \,|\, x_i = y_i \,\forall i < k \wedge x_k < y_k) \vee (m < n \wedge x_i = y_i \,\forall i < m)$$

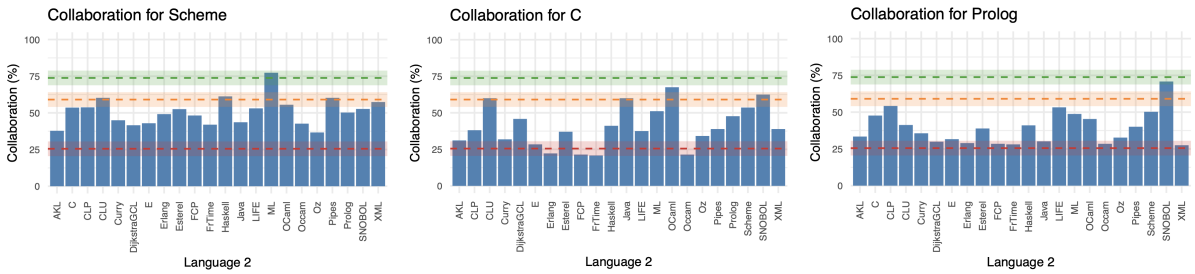
[1]See https://github.com/Vdloisem/MCC.

Definition 4.2 Let P_{L_A} and P_{L_B} be the sets of programming paradigms associated, respectively, with the programming languages named L_A and L_B. Let C be the multiset $P_{L_A} \oplus P_{L_B}$, i.e. the ordered concatenation of P_{L_A} and P_{L_B} using the relation $>$ defined above. The *Inter-Language Collaboration Metric* (*LCM*) between L_A and L_B is then calculated as

$$LCM(P_{L_A}, P_{L_B}) = \frac{1}{\binom{n}{2}} \sum_{(c,c')\in R} PCM(c, c')$$

$$\text{where } R = \{(c, c') | c, c' \in C, c' > c\} \text{ and } n = |C|.$$

The process described in Definition 4.2 provides a structured framework to analyse collaboration between L_A and L_B, offering preliminary insights into language selection, comparative analysis, and paradigm interaction (see Figure 3). The dotted lines (also present in Figure 2) mark a notion of "collaboration levels", based on confidence intervals derived from the distribution of collaboration scores. For a more comprehensive overview, including a more thorough explanation regarding the collaboration levels, see [14].



(a) Fuctional language (b) Imperative language (c) Deterministic Logic Language

Figure 3: *LCM* comparisons for three different programming languages

5. Conclusions and future work

This research has laid the groundwork for evaluating collaboration between programming paradigms based on the adaptation of an existing taxonomy in that context. Our findings suggest that functional paradigms and, by extension, the languages that incorporate them, show stronger indicators of collaboration potential. This kind of insight is especially valuable for software engineers that need to make informed decisions during the architecture and design phases of system development, as it offers a quantitative basis for selecting paradigms that can work together more effectively. Although the metrics show promise, for now, they primarily focus on static characteristics and do not fully capture the broader concerns of language interoperability as (complex) systems grow and evolve. Future work will aim to explore deeper inter-paradigmatic interactions and refine the metrics, with a focus on expanding the framework to support more languages and paradigms while assessing its practical relevance through qualitative studies. These will examine developers' language selection based on familiarity, community support, and domain-specific constraints. Additionally, we plan to develop tools for multi-language integration in development environments, potentially automating early-stage software decisions. To enhance comprehensiveness, we will incorporate factors such as typing, compilation, and memory management into our metrics. Real-world case studies will further refine these metrics, formalize interoperability and collaboration, and help determine optimal collaboration weights depending on application contexts. Finally, we will explore assigning weights to each paradigm within a language and integrating them into *LCM* to better capture their relative influence. These efforts should refine our proof of concept and support more complete and effective analyses of software development strategies in various programming environments.

References

[1] P. V. Roy, S. Haridi, Concepts, Techniques, and Models of Computer Programming, The MIT Press, Cambridge, Massachusetts, 2004.

[2] M. Cimadamore, M. Viroli, Integrating java and prolog using java 5.0 generics and annotations, in: Proceedings of the 6th International Workshop on Multiparadigm Programming with Object-Oriented Languages (MPOOL), 2007, pp. 1 – 21.

[3] L. Ostermayer, Seamless cooperation of java and prolog for rule-based software development, in: Proceedings of the RuleML 2015 Challenge and the RuleML 2015 Doctoral Consortium, hosted by the 9th International Web Rule Symposium (RuleML 2015), volume CEUR Workshop Proceedings 1417, Berlin, Germany, 2015, pp. 1–8.

[4] M. Banbara, N. Tamura, K. Inoue, Prolog cafe: A prolog to java translator system, in: Proceedings of the 16th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2005), Springer, Japan, 2006, pp. 1–11.

[5] M. Eichberg, Compiling Prolog to Idiomatic Java, in: J. P. Gallagher, M. Gelfond (Eds.), Technical Communications of the 27th International Conference on Logic Programming (ICLP'11), volume 11 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2011, pp. 84–94. doi:10.4230/LIPIcs.ICLP.2011.84.

[6] P. V. Roy, et al., Programming paradigms for dummies: What every programmer should know, New Computational Paradigms for Computer Music 104 (2009) 616–621.

[7] J. Reynolds, Theories of Programming Languages, Cambridge University Press, 1998. doi:10.1017/cbo9780511626364.

[8] M. Gabbrielli, S. Martini, Programming Languages: Principles and Paradigms, Undergraduate Topics in Computer Science, 1 ed., Springer London, 2010. doi:10.1007/978-1-84882-914-5.

[9] K. C. Louden, K. A. Lambert, Programming Languages: Principles and Practices, Cengage Learning, 2011.

[10] M. Vandeloise, Programming paradigms: Taxonomy poster (supplementary material for "rethinking paradigmatic collaboration: new metrics for inter-paradigm synergy in software engineering"), 2024. https://doi.org/10.5281/zenodo.11079624.

[11] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, Prentice Hall, 2003.

[12] N. U. Eisty, D. Perez, J. C. Carver, J. D. Moulton, H. A. Nam, Testing research software: A case study, in: Computational Science – ICCS 2020, volume 12143 of *Lecture Notes in Computer Science*, Springer, Cham, 2020, pp. 444–456. doi:10.1007/978-3-030-50436-6_33.

[13] M. Vandeloise, Metrics of collaboration across programming paradigms and languages: Comprehensive dataset (supplementary material for "rethinking paradigmatic collaboration: new metrics for inter-paradigm synergy in software engineering"), 2024. https://doi.org/10.5281/zenodo.11077187.

[14] M. Vandeloise, Towards a quantitative evaluation of paradigmatic collaboration: insights from java-prolog and beyond (supplementary material for "rethinking paradigmatic collaboration: new metrics for inter-paradigm synergy in software engineering"), 2024. https://doi.org/10.5281/zenodo.13381795.

A. Appendix A

The following sets provide a detailed view of the refined taxonomy developed in this research. This taxonomy categorises 28 distinct paradigms based on 17 key concepts and two observable binary characteristics. These paradigms are further organised into 11 meta-paradigms, offering a comprehensive framework for evaluating collaboration potential within and across programming languages.

$$\text{Paradigms} = \left\{ \begin{array}{l} \text{descriptive declarative, first-order functional, functional, imperative, deterministic logic,} \\ \text{lazy functional, continuation, adt functional, event loop, stateful functional,} \\ \text{guarded command, imperative search, sequential object oriented, relational and logic,} \\ \text{declarative concurrent, adt imperative, multi agent, active object,} \\ \text{shared state concurrent, concurrent object oriented, constraint logic,} \\ \text{concurrent constraint, lazy declarative concurrent, nonmonotonic dataflow,} \\ \text{multi agent dataflow, continuous synchronous, lazy concurrent constraint,} \\ \text{discrete synchronous} \end{array} \right\}$$

$$\text{Concepts} = \left\{ \begin{array}{l} \text{record, procedure, closure, unification, cell, search, solver, thread, by-need sync,} \\ \text{continuation, unforgeable constant, channel, single assign, nondet choice,} \\ \text{sync on partial termination, clocked computation, local cell} \end{array} \right\}$$

$$\text{Binary characteristics} = \left\{ \text{ Turing completeness, observable non-determinism } \right\}$$

$$\text{Meta-paradigms} = \left\{ \begin{array}{l} \text{functional, shared state, logic and constraint, data abstraction, message passing,} \\ \text{concurrent dataflow, algorithmic search, constrained execution, reactive,} \\ \text{dataflow and message passing, functional concurrent dataflow} \end{array} \right\}$$

The table below illustrates an example of the structure used in our taxonomy, specifically applied to the deterministic logic paradigm. Details the core concepts, computational properties, and position of the paradigm within the evolutionary path of the taxonomy, highlighting its relationships with related paradigms.

Table 1
Example with the deterministic logic paradigm

Category	Actual Value	Figure
Paradigm	deterministic logic programming	(Figure 1a)
Concepts	record, procedure, closure, unification	(Figure 1b)
Turing completeness	True	(Figure 1c)
Non-determinism observability	False	(Figure 1d)
Meta-paradigm	logic and constraint	(Figure 1e)
Evolutionary path: parent	functional programming	(Figure 1b)
Evolutionary path: child	relational and logic programming	(Figure 1b)
Propagation	False	(Figure 1f)

Figure 4 shows an overview of all the paradigms that are (for now) included in our framework, in the form of an acyclic directed graph as described in the paper.

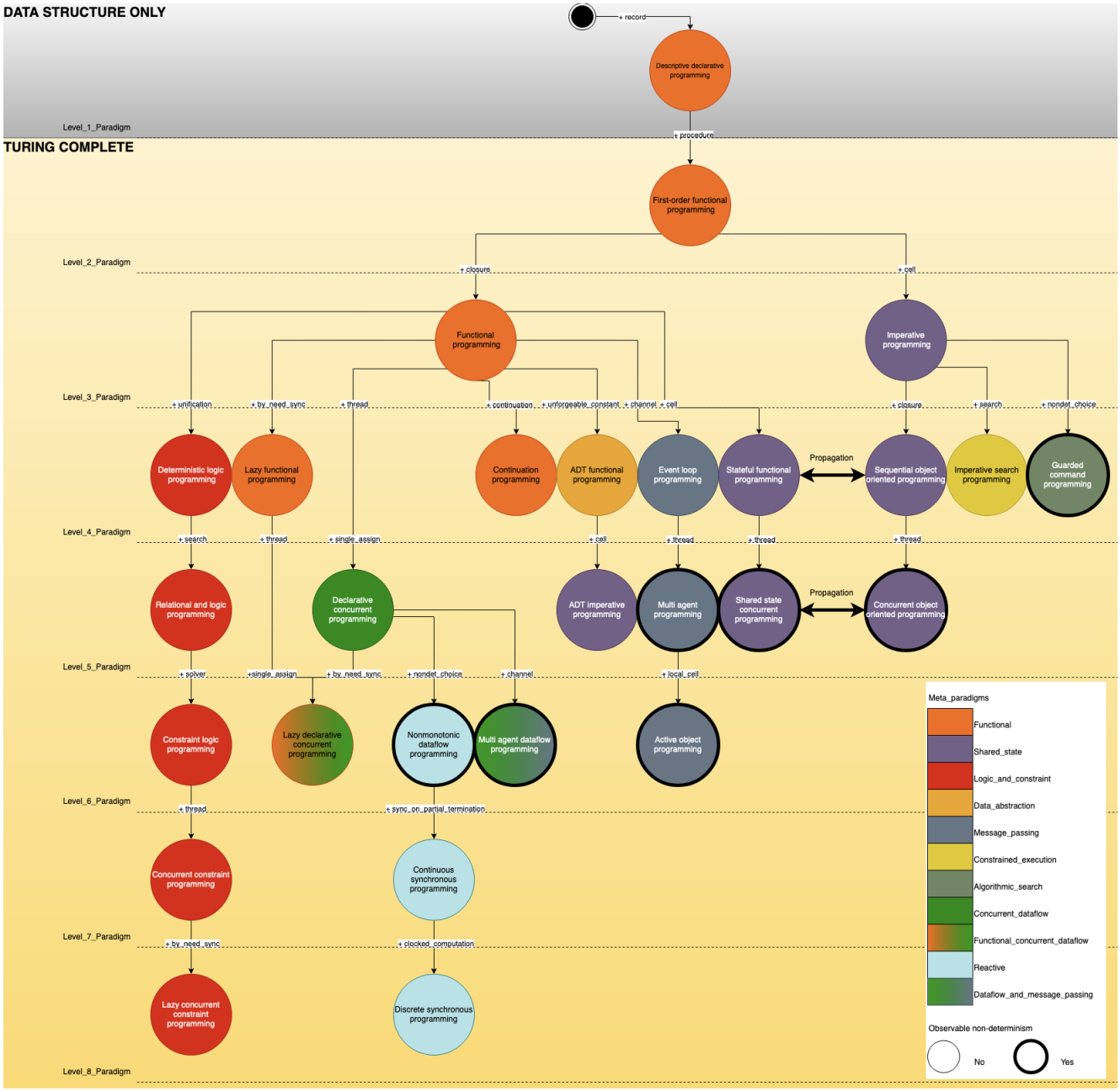


Figure 4: Taxonomy of programming paradigms