Application of Programming Cocktails Identity Cards to Development Complexity Analysis

Alvaro Costa Neto[1,2,3], Maria João Varanda Pereira[2] and Pedro Rangel Henriques[3]

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Avenida C-1, 250, 14781-502 Barretos, SP, Brasil
[2]Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal
[3]ALGORITMI Research Centre/LASI, DI - University of Minho, Campus de Gualtar, Rua da Universidade, 4710-057 Braga, Portugal

Abstract

Complexity in software projects tends to grow considerably as resources and stakeholders raise in numbers. Several factors may contribute to this, ranging from miscommunication between developers, to excessive dependency on external libraries and frameworks. Consequently, managing both developers and the assets they use becomes increasingly hard as features are implemented and changes in linguistic characteristics and coding styles become necessary. This position paper presents Programming Cocktails, their Ingredients, and Resources, three basic software development management concepts. These three main concepts culminate in Programming Cocktails Identity Cards, an ontology-based modelling technique to aid in assessing, planning, and understanding how each development technology contributes—both positively and negatively—to several aspects of software development, such as cognitive burden, risk and cost.

Keywords

programming cocktails, programming languages, development complexity, cognitive analysis

1. Introduction

It is well established that as development projects grow in size and requirements multiply, complexity also increases [1, 2, 3, 4]. From the many factors that contribute to this growth, the interconnection between programming technologies (such as languages, libraries, tools and frameworks) poses challenges that may negatively affect cognitive effectiveness, raise risks and generate costs.

In this context, analysis should be made on the combined uses and interconnections of these technologies, reaching beyond comparison of individual metrics (as it is common [5, 6, 7]). This point-of-view highlights several factors that may influence success in choosing and managing which programming technologies should be used in a certain project. Which language is a better fit for a substitution mid-development? What cognitive strains would it impose based on its differences to other languages, libraries and frameworks already in use? Do these differences raise security risks and vulnerabilities, or do they enforce behaviors that may nullify each other weaknesses?

The answers to these questions rely on the knowledge behind these combinations and how to better use and choose which technologies should be integrated into a development project. Naturally, it is of utmost importance to structure and analyse this knowledge in a productive way, considering, first of all, which factor is under evaluation (risk, cost, cognitive burden *etc.*), and secondly, how each technology relates to each other in this analysis. This paper presents Programming Cocktails as a formal, structured and novel concept that should aid developers, managers and their peers in reaching further conclusions about the intrinsic relationships between languages, libraries, frameworks and tools in their projects.

This position paper is divided into five sections. After the introduction and contextualization in Section 1, Section 2 introduces the basic concepts of Programming Cocktails, Ingredients, and Resources. Section 3 presents Cocktail Identity Cards as an ontological form of identification for Programming Cocktails. Section 4 proposes augmentations to and uses for the Cocktail Identity Cards that could

BENEVOL'24: 23rd Belgium-Netherlands Software Evolution Workshop, November 21–22, 2024, Namur, Belgium
alvaro@ifsp.edu.br (A. Costa Neto); mjoao@ipb.pt (M. J. V. Pereira); prh@di.uminho.pt (P. R. Henriques)
http://www.ipb.pt/~mjoao/ (M. J. V. Pereira); https://www.di.uminho.pt/~prh/ (P. R. Henriques)
0000-0003-1861-3545 (A. Costa Neto); 0000-0001-6323-0071 (M. J. V. Pereira); 0000-0002-3208-0207 (P. R. Henriques)

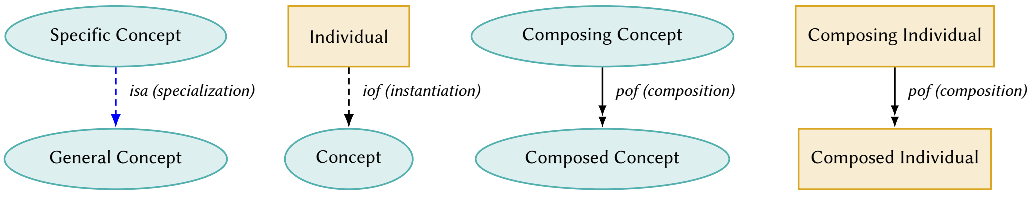

Figure 1: Programming Cocktails ontology's graphical notation.

improve planning of several aspects in development life-cycles, both technical, such as risk and cost, and psychological, such as cognitive burden and affinity attrition. Finally, Section 5 concludes the article with final remarks of what has already been modelled via Cocktail Identity Cards, and what is planned to be researched and implemented in the near future.

2. Programming Cocktails

At its basis, *Programming Cocktails* are the sets of programming technologies that are directly used to develop software systems [8]. They may be composed of four types of *Ingredients*:

- *Languages* for programming, communication, configuration *etc.*
- *Libraries*, both in source and pre-compiled varieties;
- *Frameworks*, as complementary code that requires the use or adoption of specific protocols, behavior or paradigms;
- *Tools* that are directly used in the development tasks, such as source code editors, and IDEs.

Finally, technologies that indirectly participate in the development process, such as services and runtime support systems (e.g. Database Management Systems, DBMS) are considered *Resources*. While some of them might have components that are directly used in the development process (such as a connection library for a DBMS), the Resource itself is intended to support execution, not development.

These concepts were defined in an ontology that was initially created to structure knowledge about data that was gathered in a survey [8]. It formed the basis on which the entire model of Programming Cocktails was created, including the Cocktail Identity Cards (explained in Section 3). Figure 2 shows the open conceptual model for the ontology (see Figure 1 for the graphical notation), with all its basic definitions, such as the previously mentioned concepts, the core concepts of *System*, *Development*, and *Cocktail*, as well as *Tasks*, which correspond to actual development tasks, such as *Front-end Development*, *Database Communication* and equivalent. There are two more important points to be made about the ontology:

- Despite the fact that the original intention of the ontology was to structure the data gathered in the survey [8], the results have become a foundation for modelling Cocktails via instantiation of the open conceptual model. This should aid stakeholders in the development team to visualize, document, analyse and decide which programming technologies should be used and why;
- The conceptual model is considered *open* because the definition of the Task specializations (*TaskA* to *TaskZ*) is deferred to the individuals that will model Programming Cocktails using it. This decision aims to avoid pre-defining tasks that would eventually be either too generic for any kind of useful representation, or too granular and specific, which would inevitably lead to obsolescence.

The conceptual model is the starting point to the actual modelling of the Programming Cocktails, leaving practical application to its instantiation: the Cocktail Identity Cards.

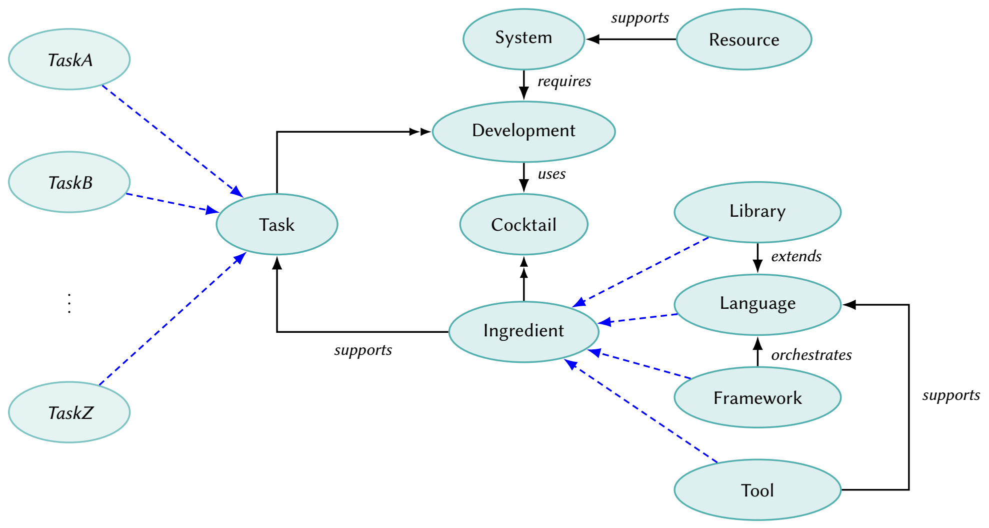

Figure 2: Open conceptual model for the Programming Cocktails ontology.

3. Cocktail Identity Cards

The conceptual model forms the basis for the modelling process, providing guidelines akin to schematics for the actual model. Its instantiation, consequently, renders a series of interconnected *individuals*[1] that provides an instant overview of the Cocktail's structure. If carefully drawn and *pruned*[2], the resulting diagram is easily recognizable and most of the structural information about the Cocktail is immediately observable, such as ingredient quantity and diversity, different branches of development, which languages are more dependant of complementary ingredients, *etc.*

Figures 3 and 4 show two Cocktail Identity Cards, for two different software systems. The first is a Web application for an in-house Questions and Answers (Q&A) system. As can be seen, not all relations are shown, as the *System*, *Development* and *Cocktail* concepts are hidden to improve legibility. The tasks were defined in a higher level of abstraction, including only general areas of development (markup, styling and scripting). This might not be the case with a different Cocktail, or even at another situation for the same Cocktail that requires a more detailed representation of each Ingredient's role in the project implementation. In the end, the definition of the granularity (or abstraction level) of the tasks is highly situational. The second Identity Card (shown in Figure 4) models a Covid-19 pass management mobile application. It has two branches of development, one for each target platform: *iOS* and *Android*. This Identity Card also shows that both languages, *Swift* and *Kotlin*, are less dependant on complementary code, as no instances of *Library* and *Framework* are connected to them[3]. This may lead to a lower complexity, as linguistic and psychological features (such as affinity [9, 10] and cognitive load [11]) are constant in each branch.

The Cocktail Identity Cards are useful as-is for quick identification of several characteristics of the Cocktails they represent. Nonetheless, if augmented with further information, it may serve as a foundation for other, more complex analysis.

4. Possible Uses and Augmentation for Cocktail Identity Cards

Despite the immediate structural feedback obtained from the Identity Cards, more can be represented and documented if the relationships in the model are augmented to represent specific metrics. These metrics may relate to technical aspects, such as cost of maintenance, size needed for cloud hosting, or

[1]The instance of an ontological concept may also be called an *individual*.
[2]Some instantiation relations are hidden in order to avoid cluttering the diagram.
[3]That is not to say that these languages are inherently less dependent on other components, only that in this case, the model indicates a lower level of dependency.

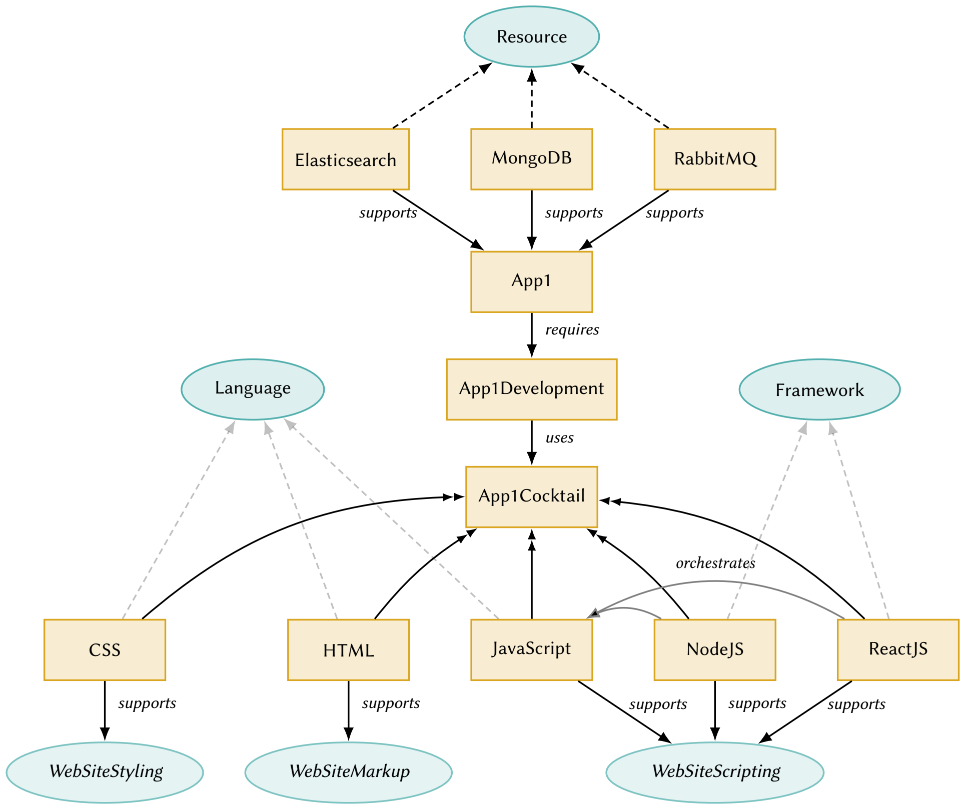

Figure 3: Cocktail Identity Card for a Q&A system.

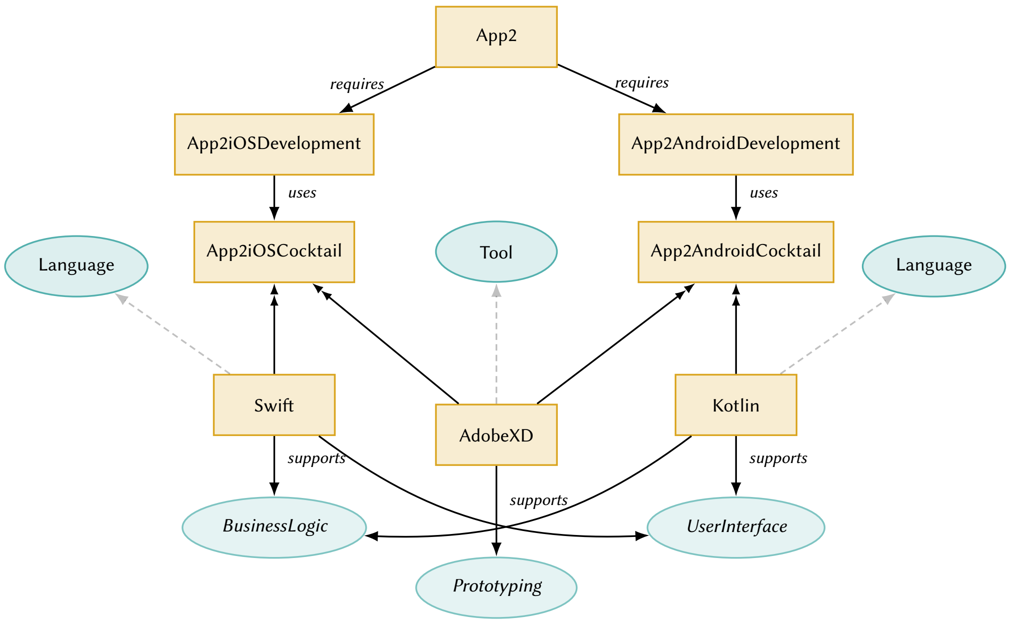

Figure 4: Cocktail Identity Card for a multiplatform Covid-19 pass application.

definitive linguistic metrics, such as number of data types and functions. On the other hand, some more qualitative metrics could also be applied, such as cognitive burden [12], affinity, security risk, *etc.* With these augmentations, the Identity Cards would provide different points-of-view for the structure of the Cocktail. Stakeholders would be able to quickly assess current, former and even possibilities for future changes in the Cocktail's structure and how these changes would affect each of the augmented metrics. This use would form a historic account of how the programming technologies in the project have

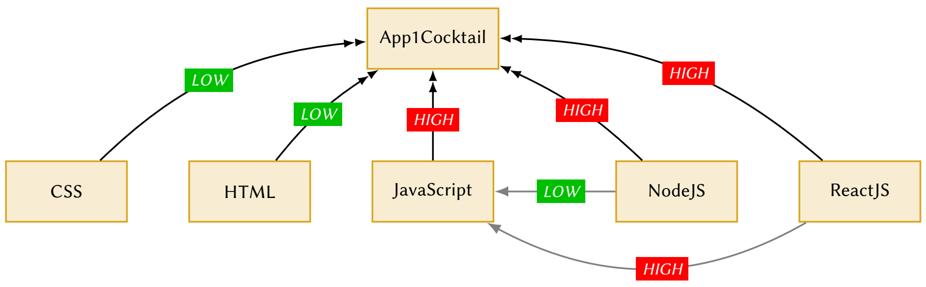

Figure 5: Hypothetical security risk augmentation for *App1*'s Cocktail. This analysis would be recursively applied up to the *System* instance.

changed and evolved, aiding in tracking and predicting software evolution, as the software evolution can be represented by a set of identity cards in a timeline sequence.

A second possible use for the Identity Cards is decision support. By comparing different Identity Cards augmented with metrics such as cost, risk and cognitive burden, decision on which technologies should be included (or substituted) in the project could be taken with an overview of its impact on the project and on the team behind it. Figure 5 presents part of *App1*'s Cocktail Identity Card (see Figure 3) with *hypothetical augmentations* that represent security risks for each Ingredient. The values superimposed on the relations (*HIGH* and *LOW*) could be measured based on specific linguistic features, such as the direct memory access via pointers, manual memory allocation, lack of bounds management in data structures, *etc.* but could also take into consideration historic findings about each technology and the inherent nature of its use. As an example of the latter, a markup language is naturally less vulnerable to exploits as it tends to express only structure instead of behavior and would contribute less to possible vulnerabilities. A simple score-card could be created to establish the level of risk for each Ingredient, and most important of all, for the relations between each other (such as between *NodeJS* and *JavaScript* in Figure 5).

As for comparison against other modelling options, a few points can be made:

- By their own nature, ontologies are more flexible in their construction than other modelling techniques that have pre-defined semantics, such as Class Diagrams. This fact could lend Cocktail Identity Cards to better fit specific scenarios, as characteristics from different points of view can be assembled in one model. One possible caveat with this argument is that this flexibility could also decrease standardization. It could eventually hinder communication and information exchange between peers that come from different projects, given that each group would adapt the ontology to their particular needs;
- Different levels of abstraction can be expressed in a single model, as new concepts do not have to adhere to any sort of pre-established abstraction level or standard;
- By using only ontological modelling, fragmentation of tools and techniques can be avoided or at least reduced. This lowers development complexity between different peers in a project, such as developers, designers, managers, *etc.*

5. Conclusion

Programming Cocktails provide a novel way to model and observe how different technologies compose and interact in a development setting. This paper presented the foundational ideas and concepts behind them, as well as their immediate result: Cocktail Identity Cards. Through future research into augmenting these Identity Cards, stakeholders in a software development project could evaluate, record, document and analyse different metrics, ranging from the most technical and quantitative, such as points of vulnerability and racing conditions, to more philosophical and qualitative ones, such as quality, affinity, and attrition. Software evolution and maintenance could also be tackled by constructing historic accounts for the changes that occur in Cocktails during its lifetime, establishing a sequence of

Identity Cards that represent the evolution of their development technologies. The Cocktail Identity Cards will be used to assess cognitive burden in Programming Cocktails, culminating in a decision-support methodology that should aid developers and educators in choosing programming languages, frameworks, libraries and tools that lessens the cognitive load requirements for developers and students.

Acknowledgments

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020. The work of Maria João was supported by national funds through FCT/MCTES (PIDDAC): CeDRI, UIDB/05757/2020 (DOI: 10.54499/UIDB/05757/2020) and UIDP/05757/2020 (DOI: 10.54499/UIDP/05757/2020); SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).

References

[1] A. Ghazarian, A theory of software complexity, in: 2015 IEEE/ACM 4th SEMAT Workshop on a General Theory of Software Engineering, 2015, pp. 29–32. doi:10.1109/GTSE.2015.11.

[2] M. Benaroch, K. Lyytinen, How much does software complexity matter for maintenance productivity? the link between team instability and diversity, IEEE Transactions on Software Engineering 49 (2023) 2459–2475. doi:10.1109/TSE.2022.3222119.

[3] T. Honglei, S. Wei, Z. Yanan, The research on software metrics and software complexity metrics, in: 2009 International Forum on Computer Science-Technology and Applications, volume 1, 2009, pp. 131–136. doi:10.1109/IFCSTA.2009.39.

[4] L. Hanyan, W. Shihai, L. Bin, X. Peng, Software complexity measurement based on complex network, in: 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, pp. 262–265. doi:10.1109/ICSESS.2017.8342910.

[5] A. H. Odeh, Analytical and comparison study of main web programming languages: Asp and php, TEM Journal 8 (2019) 1517–1522. URL: http://www.temjournal.com/content/84/TEMJournalNovember2019_1517_1522.pdf. doi:10.18421/TEM84-58.

[6] M. Fourment, M. R. Gillings, A comparison of common programming languages used in bioinformatics, BMC Bioinformatics 82 (2008). URL: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-9-82. doi:10.1186/1471-2105-9-82.

[7] N. Walia, A. Kalia, Programming languages for data mining: a review, International Journal of Computer Trends and Technology 68 (2020) 38–41. URL: https://ijcttjournal.org/archives/ijctt-v68i1p109. doi:10.14445/22312803/IJCTT-V68I1P109.

[8] A. C. Neto, M. J. V. Pereira, P. R. Henriques, An ontology to understand programming cocktails, in: M. Ganzha, L. Maciaszek, M. Paprzycki, D. Ślęzak (Eds.), Proceedings of the 19th Conference on Computer Science and Intelligence Systems, Annals of Computer Science and Information Systems, IEEE, 2024. To be published.

[9] A. Costa Neto, C. Araújo, M. J. V. Pereira, P. R. Henriques, Programmers' affinity to languages, volume 91, Open Access Series in Informatics (OASIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 1–7. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14219. doi:10.4230/OASIcs.ICPEC.2021.3.

[10] A. Costa Neto, C. Araújo, M. J. V. Pereira, P. R. Henriques, Value-focused investigation into programming languages affinity, volume 102, Open Access Series in Informatics (OASIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 1–12. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16605. doi:10.4230/OASIcs.ICPEC.2022.1.

[11] V. Chiew, Y. Wang, A large-scale empirical study on the cognitive complexity of software, in: CCECE 2010, 2010, pp. 1–4. doi:10.1109/CCECE.2010.5575116.

[12] J. Sweller, Cognitive load during problem solving: Effects on learning, Cognitive Science 12 (1988) 257–285. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1202_4. doi:https://doi.org/10.1207/s15516709cog1202_4.