An Ontology to Understand Programming Cocktails

Alvaro Costa Neto
0000-0003-1861-3545
Research Centre in Digitalization and Intelligent Robotics (CeDRI)
Laboratório para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC)
Instituto Politécnico de Bragança
Campus de Santa Apolónia, 5300-253 Bragança, Portugal
ALGORITMI Research Centre / LASI, DI
University of Minho, Braga, Portugal
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Barretos, Brazil
Email: alvaro@ifsp.edu.br

Maria João Varanda Pereira
0000-0001-6323-0071
Research Centre in Digitalization and Intelligent Robotics (CeDRI)
Laboratório para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC)
Instituto Politécnico de Bragança
Campus de Santa Apolónia, 5300-253 Bragança, Portugal
Email: mjoao@ipb.pt

Pedro Rangel Henriques
0000-0002-3208-0207
ALGORITMI Research Centre / LASI, DI
University of Minho, Braga, Portugal
Email: prh@di.uminho.pt

***Abstract*—An ever-growing landscape of programming technologies (tools, languages, libraries and frameworks) has rapidly become the norm in many domains of computer programming—Web Development being the most noticeable example. The concurrent use of many compartmentalised technologies has advantages: it allows for flexibility in implementation, while also improving reusability. On the other hand, this proliferation tends to create convoluted development workflows that must be (painstakingly) planned, managed and maintained. The combination of multiple languages, libraries, frameworks and tools (*Ingredients*) in a single project effectively forms a *Programming Cocktail*, that can rapidly become cognitive and financially onerous. Aiming at understanding these complex situations, an ontology was created to provide a formal and structured analysis of these cocktails. It emerged from a survey of technologies that several companies are currently using to develop their systems, and aims to provide support for better understanding, classifying and characterising *Programming Cocktails*. This paper presents not only the ontology itself, but also the consequent knowledge that was constructed and structured through its development.**

***Index Terms*—Ontology, Programming Cocktails, Software Development, Programming Technologies, Knowledge Construction**

I. Introduction

THE DEVELOPMENT process of an application invariably requires the use of certain technologies, such as languages (for programming, specification *etc.*), libraries, frameworks and tools. This process may be monotonic, requiring no more than a base language and, occasionally, a handful of libraries. On the other hand, it may also be plural and polyglot, with several frameworks, libraries, tools and components (modern Web applications being the most prominent examples).

As is typical in the latter case, whenever a project demands—or is propelled by—the presence of multiple technologies, those involved in the construction of the application must learn, use and manage them. The epistemological challenges that arise in these situations resemble the ones that have been studied in Computer Programming Education for decades [1]. These studies range from tools to aid students and teachers [2]–[5], educational methodologies [6]–[8], success and failure factors [9], [10] to more psychological endeavours [11]–[14]. Technologies that are unfamiliar to programmers must be learnt and understood [9], [15] during the entire life cycle of an application. Either in the initial development phases of a project, or when technology adoption changes, knowledge must be constructed for the learning process to happen. In the presence of several technologies, a new caveat appears: beyond understanding each one, programmers must also manage a surge in cognitive burden as their brains are required to cope with alternating mental models.

Research into dealing with these challenges usually present themselves as comparative surveys [16]–[18] that list different programming technologies and their main characteristics. They usually aim to establish a clear landscape and support decisions on which technologies are best suited to specific contexts based solely on individual properties of each technology. Inherently, these studies fail to take into account any possible combination thereof, focusing their efforts in relating and comparing pre-determined aspects. It then becomes clear that a comparative study is not enough to understand how these programming technologies relate to each other in real-life scenarios. The concepts that relate to these technologies, and their interconnections must be formally and structurally

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 and Alvaro 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).

mapped. Knowledge must be constructed to cover not only each Ingredient (technology), but the Cocktail (combination) itself. A possible answer to this challenge relies in the use of ontologies [19], [20], a formal method to structure knowledge, to conceptualize and instantiate information from Cocktails, establishing reasonable inferences on its landscape of programming technologies.

This article is divided into four more sections. Section II presents the definitions for Programming Cocktails, their Ingredients and other related concepts. Section III details a survey of real-life Cocktails conducted with several Software Companies, and the overall results that have been observed. Section IV presents the ontology that was created to formally analyse and understand Programming Cocktails, the concept of a Cocktail Identity Card, and what knowledge was constructed around and through them. Finally, Section V concludes the paper with with a summary of lessons learned, and presents the next steps in the research of Programming Cocktails.

II. Programming Cocktails

Before delving into the intricacies of application development and the complex relations between the components that are used to build them, it is of good measure to define what *Programming Cocktails* and *Ingredients* actually mean.

It is comprehensible that the use of such relaxed terms to describe logical and structured concepts might seem as a stretch (or even sarcastic) at first sight. Maybe just an analogy, that is furiously gripping itself on the edge of an undeniably sharp, sleek and mathematically sound cliff. Nevertheless, it is in fact very meaningful to this article's context and objectives. Anyone who has ever tried to concoct actual cocktails should be able to described them by more than a list of components. The results are sometimes clean and homogeneous, with strong and decisive tastes. In other cases, the components barely mix together, presenting fuzzy (even chunky) separations that stubbornly remain. In the worst scenarios, when the list of ingredients, their measures, and combinations are poorly chosen, the final result may become undrinkable.

Analogously, the term *Programming Cocktail* defines a combination of computer programming technologies—such as programming languages, libraries and frameworks—that is used to develop specific software applications. *Ingredients* are the components of a *Cocktail*. It is important to note that a Cocktail is associated with a specific application or service, and its Ingredients may also appear in the Cocktail for other application under the same development context[1]. Suppose a company develops three applications:

- **Application A:** HTML, CSS, JavaScript, and ReactJS;
- **Application B:** HTML, CSS, JavaScript, MySQL, and PHP;
- **Application C:** C++, and Unity.

It might seem that, as a whole, there is one Cocktail for the company: the union of the sets formed by each application's Cocktail. Nonetheless, for the purposes of this study, each Cocktail is taken independently, even if it means to consider Ingredients more than once in the same development context. In short, there are three Programming Cocktails in the previous example, one for each application (A, B, and C).

As is expected, a few decisions had to be made while defining these terms. The first and foremost was: which development technologies should be considered Ingredients? At first sight, there are countless technologies that are involved in the development of an application. From standard and well-known programming languages, through Domain-Specific Languages (DSL) for diverse specifications, configuration and communication; to niche libraries, full-stack frameworks, editors and debuggers, the list of candidates to be identified as Ingredients is varied and long. A qualitative threshold was defined to separate what would be considered part of a Cocktail. A programming technology was identified as an Ingredient only if it is *directly applied to the development*[2] *process of an application.*

On the other hand, several technologies are commonly used during deployment or execution of an application, such as Database Management Systems (DBMS), queue coordinators, *etc*. Despite their influence on the design and implementation of an application, these technologies are not considered Ingredients, they are *Resources*. Examples include: Apache Web Server [21], ActiveMQ [22], MySQL [23], and memcached [24].

The second decision concerned the definition of categories for the Ingredients. For the purposes of this study, an Ingredient may be categorized as one of four possibilities:

- **Language:** encompasses any kind of text or graphics-based language. May be used for programming, specification, description, communication, scripting, so on and so forth. Examples: C [25], Python [26], HTML [27], CSS [28], SQL [29], *etc.*
- **Library:** a portion of code (either in source form or pre-compiled) that augments programming languages and their standard libraries with extra functionality. Examples: LibSSH [30], RayLib [31], *etc.*
- **Framework:** scaffolding augmentations to programming languages. Albeit similar to libraries, frameworks add functionality while imposing some form of structure to the source code (syntactic, semantic, or paradigmatic) or the use of pre-defined components[3]. Examples: SwiftUI [32], React Native [33], *etc.*
- **Tool:** specifies any tool that is directly used for development, such as editors, Integrated Development Environments (IDE), debuggers *etc.*

There might be cases in which the borders between these categories become tenuous. In these situations, an Ingredient that has multiple roles in the development process might need

[1]*Development context* represents the set of factors that influence the actual construction of an application, including, but not limited to, its team, technologies, tools, organization, and requirements

[2]*Development* here indicates a generalised concept which includes, but is not limited to, programming tasks.

[3]Given that there is no standard for distinguishing between libraries and frameworks, this definition may collide with others'.

to be either sliced into its constituent parts, or included in more than one category. As an example, testing frameworks, such as JUnit [34], usually include both libraries and servers to allow for concurrent testing. These Ingredients could possibly be separated into their individual roles (*JUnitLibrary* and *JUnitServer*, per example) or included in both categories (Library and Tool).

The reasoning behind these decisions became clear through the construction of the ontology (explained in Section IV), with its foundational rationale extracted from real-world Cocktails, surveyed from several multi-national companies in Portugal.

III. Cocktails Assemblage

As previously stated, obtaining the current uses of Programming Cocktails was paramount to establishing an overall picture of computer programming technologies. To this intent, several companies were contacted in a survey for information about which Programming Cocktails they have used. Their feedback allowed for the construction of the ontology's main concepts (presented in Section IV) and the consequent structuring of the knowledge surrounding Programming Cocktails.

A. Survey

Starting in October 2023, several companies that have offices in Portugal were contacted via email for a survey of Programming Cocktails. The email (shown in the Appendix) described the context of the study and asked for the programming technologies each company has used, divided by applications in which each Cocktail was used. Companies were specifically asked to answer informally via email, in order to stimulate participation and consequently obtain faster and more numerous responses. Given previous experiences, online survey questionnaires, such as those created via platforms akin to Google Forms[4] tend to be postponed, resulting in fewer answers. While the amount of companies that responded was far from ideal, this number would possibly be even lower if a formal system was used.

Up to the time of this paper's submission[5], 213 companies were contacted and of those, 15 responded with several Cocktails they have used in the past, or still use in the present.

A few important considerations:

- Given the informal nature of the survey, some answers had to be either supplemented (in the case of obvious missing elements, such as Cocktails with React that missed JavaScript) or followed up with further communication;
- Some answers pointed to the fact that the *borders between some systems are a bit fuzzy*, and their Cocktails represent overall divisions that are shared between groups of applications. Such is the case with systems that are heavily structured around micro-services, as example;
- Despite the fact that only Portuguese offices of the surveyed companies were contacted, the majority of them have international endeavours or are multinational themselves, which reduces the locality bias of the answers;
- Exhaustiveness was never the goal for the survey. Given that the programming technologies landscape is ever changing in a fast pace, the survey was designed to support the construction of knowledge about Programming Cocktails, which in turn, may eventually be applied to future works and studies.

Currently, 49 Programming Cocktails have been obtained, spanning a total of 124 different Ingredients and Resources, that range from programming languages and frameworks, to database management systems and resource cache management applications.

B. Data Overview

As previously mentioned, statistical data analysis is not the main goal of this study. Nonetheless, a few statistical facts can be extracted from the Programming Cocktails that were gathered in the survey.

In order to better organize the survey results, an online spreadsheet[6] was created, listing Ingredients on the lines and Cocktails on the columns (Table I represents a summarised example of the actual spreadsheet). Column *A* contains the names of the Ingredients. The columns from *B* to *D* categorize each Ingredient into, respectively, its type[7] (*Language*, *Library*, *Framework* or *Tool*), the Language with which it was used, and the Task it was applied to (Tasks will be further explained in Subsection IV-B). In the eventual case of an Ingredient either being used with more than one Language, or applied to more than one Task, its line would be duplicated and its categories adapted as needed. An hypothetical example would be the .NET Framework, which can be used with several different programming languages, and would require such treatment. Finally, from *E* onwards, each Cocktail was listed in its own column, with their Ingredients' rows marked to represent their inclusion. As an example, in Table I, the column *App2* represents the second Cocktail that was gathered and includes both C# and YAML.

In total, 63 different Ingredients have been collected:

- 23 Languages;
- 14 Libraries;
- 22 Frameworks;
- 4 Tools.

As per Resources, 61 have been gathered. Overall, each Cocktail has an average of 8 Ingredients and Resources, which a 4.6 standard deviation. Some other observations include:

- 2 Cocktails are based either on *Low Code* or *No Code* Ingredients;
- Most of the Cocktails belong to Web Development (32 in total);

[4]Available at: https://www.google.com/forms/

[5]All data discussed in this paper should be considered from the same time period, unless stated otherwise.

[6]A read-only version is available at: https://bit.ly/4aFjSjj

[7]Resources have also been included in the spreadsheet and are categorized as such, despite not being Ingredients *per se*.

TABLE I
SUMMARISED EXAMPLE OF THE COCKTAILS SPREADSHEET.

Ingredient	**Type**	**Language**	**Task**	App1	App2	···	AppN
.NET	Framework	C#	Full-stack development			···	X
C#	Language	C#	Server implementation		X	···	
C#	Language	C#	Full-stack development	X		···	
			⋮				
YAML	Language	YAML	Communication		X	···	

- The most frequent Ingredients are (from most to least used):
  - **Language:** HTML, CSS, JavaScript, SQL, C#;
  - **Library:** OData, Bootstrap, (all the other tied in one use);
  - **Framework:** React, Node.js, .NET, Angular, ASP.NET;
  - **Tool:** Visual Studio, Visual Studio Code, PowerPages, Liferay;
- The most frequent combination of a programming language and a framework is JavaScript with React, followed by JavaScript with Node.js, C# with .NET, and JavaScript with Angular;
- 7 Cocktails use only one language;
- At the time of writing, no Cocktails have been collected that directly apply any Artificial Intelligence (AI) support or technology.

As previously mentioned, micro-services architectures presented a challenge in defining borders between systems—and consequently, their Cocktails. In these cases (5 in total), their Cocktails were defined taking into account a group of microservices that implement logical parts of the whole system. The logic behind this definition was dependent on the system itself, and as such, stipulated by the company that provided the Cocktails.

IV. Ontology for Programming Cocktails

The data collected through the survey has a purpose: to allow for better understanding of Programming Cocktails, which entails the construction and structuring of knowledge. Our research group has had several interactions with and has made several contributions to the study of ontologies [35]–[41], both in their construction and definition. Consequently, from several approaches that could be applied to achieve the construction of knowledge about Programming Cocktails, an ontology seemed a straightforward and appropriate choice. It allowed for the formal definition of Programming Cocktails' main concepts, the generation of *Identity Cards*, an ontology-based characterisation mechanism for Cocktails, and the organization of their Ingredients.

Moreover, the ontology will be paramount for future use in coming studies, that will deal with the evaluation of Cocktails in cognitive load metrics. The construction of the ontology was then, in practice, a two-fold endeavour, as it aided in structuring and understanding the data that was collected through the survey (its initial goal), while also providing a foundation on which several studies might surge.

A. OntoDL

The initial version of the ontology was created using a spreadsheet to organize and list its concepts, relations, instances, and connections. While suitable for the beginning phases, when the number of elements was small, as the ontology grew it became evident that other solutions would offer better scaling and future-proofing. A visual representation would be ideal to quickly present the connections between the ontology's elements. Given previous experiences and its simple yet capable syntax, OntoDL [42] was chosen as the main source for the definition and instantiation of the ontology. OntoDL is a Domain-Specific Language (DSL) that was created for modelling ontologies, as an alternative to more verbose options such as the Web Ontology Language (OWL) [43]. It has been used in several projects, including the WebOntoDL application[8], which can interpret ontologies written in OntoDL and translate them to several other formats, such as DOT[9] and OWL. It contains a syntax that is reminiscent of the mathematical formal definition of ontologies. It also allows the use of keywords in Portuguese or in English, which could be a beneficial factor for exchange and contribution from third parties.

The basis for OntoDL's syntax relies on five main structures: the name of the ontology, the list of concepts, the list of individuals, the list of relation types and the triples that actually declare relations. Listing 1 shows the basic declaration for each structure. A few basic rules:

- The language follows basic principles of ignoring whitespaces and line-breaks, as well as the use of curly brackets as group delimiters;
- The order of the declarations matters;
- Concepts, relationships and triples are mandatory;
- Comments are line based, beginning with the percent sign (`%`) and ending with a line-break;
- There are pre-defined relation types for specialization (`isa`), composition (`pof`), and instantiation (`iof`);
- Triples are directional, and may be formed using any combination of concepts and individuals.

[8]Available at: https://webontodl.epl.di.uminho.pt

[9]DOT is a graphics format written in plain text that is used to define visual elements in a diagrammatic form. It is part of the Graphviz project, available at https://graphviz.org

Listing 1 Basic syntax for OntoDL.

```
% Identifiers must follow the C standard.
Ontology OntologyName

% The order of declarations matters.
concepts { Concept1, Concept2, ... }

individuals { Individual1, Individual2, ... }

% Whitespace and line-breaks are ignored.
relationships {
    RelationType1,
    RelationType2,
    ...
}

triples {
    % Triples may relate both concepts and
    individuals, in any combination.
    Concept1     =RelationType1=> Concept2;
    Individual4  =RelationType2=> Individual5;
    % Specialization.
    ConceptChild  =isa=> ConceptFather;
    % Composition.
    ConceptPart   =pof=> ConceptWhole;
    % Instantiation.
    IndividualA   =iof=> ConceptA;
}
% The period indicates the end of the ontology.
.
```

There are more rules for defining properties, axioms and other elements, that have not been used in this paper. With the foundation firmly established on OntoDL, the first step to create a valid ontology that would allow for reasoning on the surveyed Programming Cocktails was to model its main concepts.

B. Open Conceptual Model

Before delving into the actual ontology and its concepts, it is important to establish a graphical notation that will be used and referenced in figures that represent them. Fig. 1 shows the basic elements and how they are graphically styled, more specifically for OntoDL's pre-defined relations. Other relationships use a simple arrow. This notation is emblematic because it allows for quick identification of element types (concepts or individuals), and pre-defined relation types that carry specific semantics (instantiation, specialization and composition).

The main concepts of the ontology were incrementally created. The first concepts meant to establish a foundation, based on the fact that each *Cocktail* is directly associated to a *System* that either is or has been under *Development*. Listing 2 and Fig. 2 present them.

The next step in the development of the conceptual model was the addition of the Ingredients and their types. The results of the survey reaffirmed the initial proposal for their types (Language, Library, Framework and Tool), but also highlighted a basic problem: some tools that were listed by the companies did not participate directly in the development process. In some cases, such as the main Operating System that programmers chose or a note taking application, it was evident

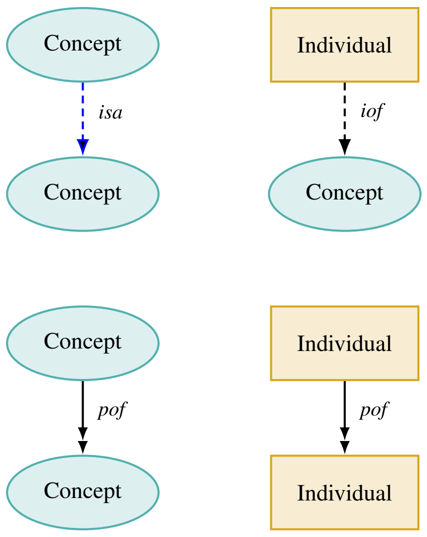

Fig. 1. Graphical notation for the ontology.

Listing 2 Initial concepts for the ontology.

```
Ontology Cocktails

concepts { System, Development, Cocktail }

relationships { uses, requires }

triples {
    % Foundation.
    System      =requires=> Development;
    Development =uses=>     Cocktail;
}
.
```

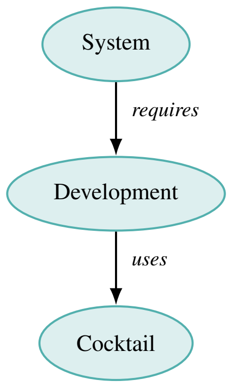

Fig. 2. Initial concepts for the ontology.

that their inclusion as Ingredients would be a stretch that could backfire in later developments. Other elements were not so easy to distinguish, such as Database Management Systems, and Queue Managers.

At that moment, a decision had to be made: would these tools be included as Ingredients? Which category would they belong to? At first, the intention was to include all of them as Tools, but as more Cocktails were obtained, it became clear that this choice could easily distort the category's meaning. In the end, in order to avoid this negative effect, the concept of a *Resource* was created to implement this solution. A Resource represents an external system (or service) that is used at runtime by the application, but that does not participate directly during the development process. For obvious cases, such as Operating Systems, the separation between a Tool and a Resource was clear. Alas, that was not always the case. Some Resources required some kind of implementation in the development phases, such as a communication library, or a configuration language. In these cases, the runtime of the system (or service) was considered a Resource, while any mandatory Application Programming Interface (API), library, framework or language that was used to interact with it was considered an Ingredient (of the correct type). As an example, in order to communicate with MySQL, an external library or framework (such as *libmysqlclient*) is usually included as part of the project. Given that the runtime of the MySQL server provides support for the *execution* of the application, it would be considered a *Resource*. On the other hand, *libmysqlclient* would be considered a *Library*, as it participates directly in the development phase to program the interaction to the server. In order to highlight this difference in purpose (supporting the application execution *versus* the development process), the concept of the Resource was moved from its initial relationship (a specialization of Ingredient) to a supporting role to the System itself. Listing 3 and Fig. 3 show the inclusion of these concepts to the ontology. It can be seen that while the concepts of Language, Library, Framework and Tool are specializations of the more general concept of an Ingredient, Resource is directly connected to the System concept.

The last two additions to the conceptual model consisted in a series of relationships that highlighted the central role of Languages in the Cocktail, and the definition of Tasks. Languages are usually the central element in the development of almost any kind of System. In fact, it is very unusual that a Language choice will depend on other types of Ingredients, such as Frameworks, or Libraries. The reverse, although, is commonplace: the choice of a Language usually dictates which other Ingredients will be part of the Cocktail.

In order to illustrate and define Language's central role, three relationships were added, each connecting one of the other Ingredient types to it (see the *extends*, *encloses* and *supports* relationships that terminate in *Language* in Listing 4 and Fig. 4).

The definition of Tasks and how they were modelled was the last step in the construction of the conceptual model for the ontology. As with any specification for concepts that rely

Listing 3 The inclusion of Ingredients and Resources to the ontology.

```
Ontology Cocktails

concepts {
    System,
    Development,
    Cocktail,
    Resource,
    Ingredient,
    Language,
    Library,
    Framework,
    Tool
}

relationships { uses, requires, supports }

triples {
    % Foundation.
    System      =requires=> Development;
    Development =uses=>     Cocktail;
    % Runtime resources (OS, DBMS, etc.)
    Resource    =supports=> System;
    % Ingredients and their types.
    Ingredient  =pof=> Cocktail;
    Language    =isa=> Ingredient;
    Library     =isa=> Ingredient;
    Framework   =isa=> Ingredient;
    Tool        =isa=> Ingredient;
}
.
```

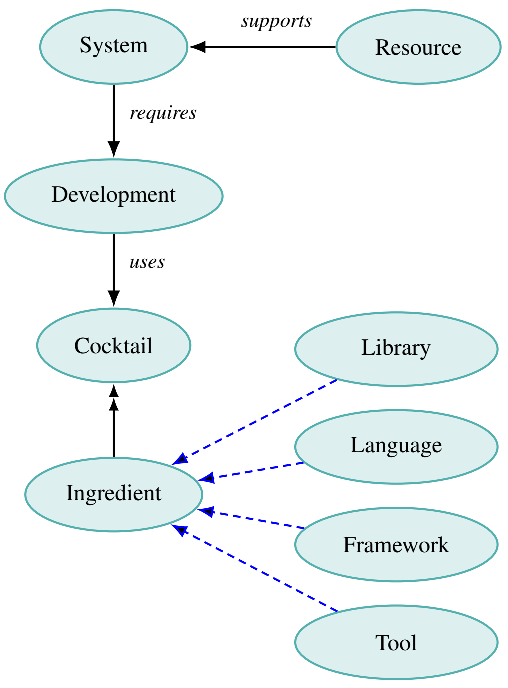

Fig. 3. The inclusion of Ingredients and Resources to the ontology.

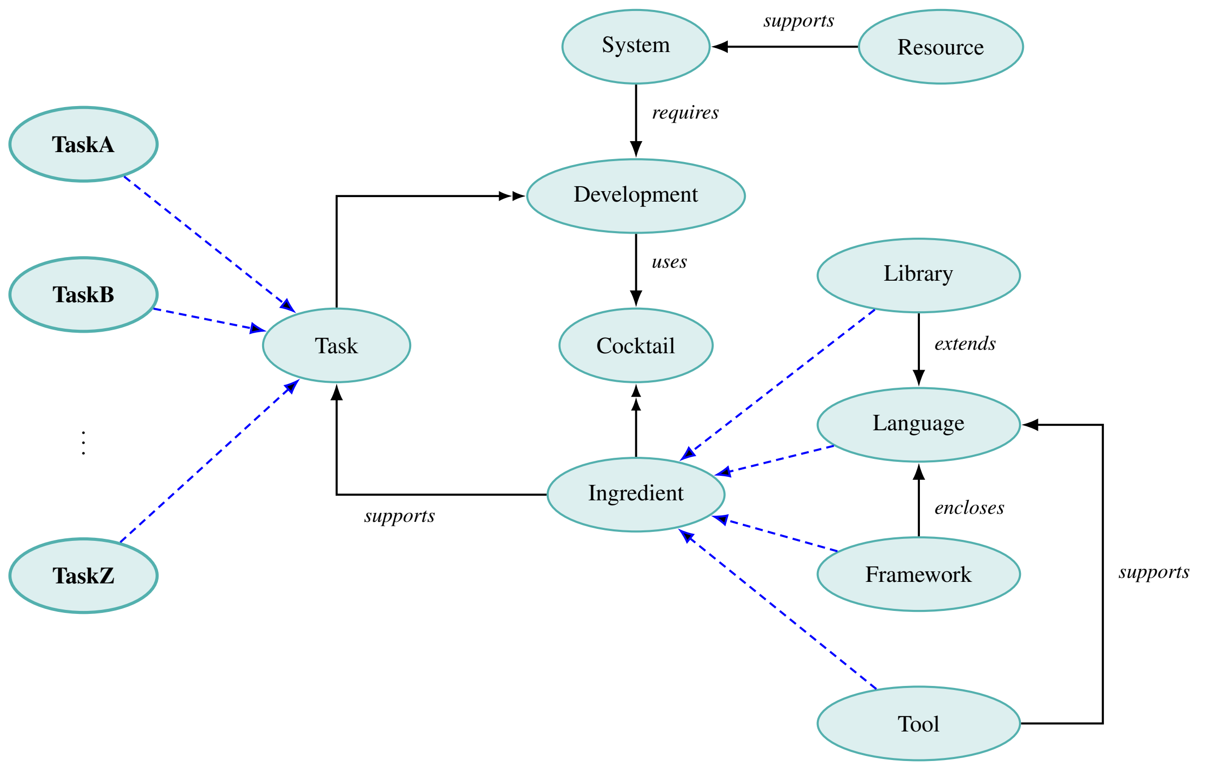

Fig. 4. The conceptual model for the ontology.

heavily on a particular context, there is no optimal solution. A few strategies were considered:

- A general concept is created in order to future-proof the definition, such as *ProgrammingTask*, or equivalent. It does not conceive any particular information about real Tasks that are effectively conducted in the development context. By doing so, the model will not require further updates, while remaining valid for any future Programming Cocktails. The drawback relies on the lack of effective representation, given its generalist nature;
- Given a survey of Programming Cocktails, a set of pre-defined Tasks is established in order to achieve more practical representation than one general concept. These Tasks are fixed and any future use of the ontology will require "fitting" of the actual development context into the set of pre-defined Tasks. This option has the evident risk of rapidly becoming outdated, specially in dynamic domains, such as Web Development. It may also require distortions on the division of Tasks in order to fit the pre-defined concepts, potentially losing its representativeness;
- The conceptual model becomes open and adaptable to specific development contexts. This was the chosen strategy for the ontology. Given that the main intention is to construct knowledge on Programming Cocktails and that each project, team, company, or organization has specific demands and requirements, keeping the ontology adaptable was a better solution to the definition of Tasks.

The final solution for the ontology and how it must deal with Tasks relies on adapting the Task concept (via specialization) to include in the model a logical division of Programming Tasks, fit for the context in case.

As an example, a company with different projects might establish different conceptual models for their ontologies. Suppose that the first project is a simple Web Application, with just a few Ingredients. In this case, the team behind it might simplify the conceptual model and only specialize general Tasks, such as *FrontEndProgramming* and *BackEndProgramming*. In another project, with multiple Ingredients and a much larger problem to solve, the team might find it appropriate to specialize *Task* into a more granular level, such as *LandingPageStructuring*, *ClientInterfaceStyling*, *DatabaseCommunication*, so on, and so forth. This is what the term *open* in *open conceptual model* means.

This strategy future-proofs the conceptual model by making it adaptable, while providing both flexibility and a solid foundation for Programming Cocktails analysis.

C. Cocktail Identity Cards

The conceptual model of an ontology is crucial to define its structure, how the modelled concepts are related, and what level of detail is expected for the overall organization of knowledge. Nonetheless, the concepts, besides being fundamental, are usually materialized into *individuals*, on their occurrences in the context being modelled.

Listing 4 The conceptual model for the ontology.

```
Ontology Cocktails

concepts {
    System,
    Development,
    Cocktail,
    Resource,
    Ingredient,
    Language,
    Library,
    Framework,
    Tool,
    Task,
    TaskA, TaskB, ..., TaskZ
}

relationships {
    uses,
    requires,
    supports,
    extends,
    encloses
}

triples {
    % Foundation.
    System      =requires=> Development;
    Development =uses=>     Cocktail;
    % Runtime resources (OS, DBMS, etc.)
    Resource    =supports=> System;
    % Ingredients and their types.
    Ingredient  =pof=> Cocktail;
    Language    =isa=> Ingredient;
    Library     =isa=> Ingredient;
    Framework   =isa=> Ingredient;
    Tool        =isa=> Ingredient;

    % Language's central role.
    Library     =extends=>  Language;
    Framework   =encloses=> Language;
    Tool        =supports=> Language;
    % General Task concept.
    Task        =pof=>      Development;
    Ingredient  =supports=> Task;
    % Context-specific Tasks. These tasks depend on
    the development context and its structure.
    TaskA       =isa=> Task;
    TaskB       =isa=> Task;
    ...
    TaskZ       =isa=> Task;
}
.
```

The conceptual model of the ontology was applied to the Cocktails in order to test its validity and aid in structuring the information from the survey. Initially, all individuals and concepts were pictured, which resulted in a convoluted image and many overlapping connections. In order to establish a clearer picture of each Cocktail, the focus shifted to showing the individuals, their relations and, when necessary, some concepts to avoid misidentification of the individuals. The concepts that were kept in the diagram were:

- *Resource* to explicitly show which supporting systems and services each application used;
- *Language*, *Library*, *Framework*, and *Tool*, to categorise each *Ingredient*;
- *Task* specializations to identify the parts of the application that each *Ingredient* tackles.

In all of these cases, a concept is only added to the diagram if a relation to or from it is also present. As an example, if there are no libraries in the Cocktail, the *Library* concept will not be shown.

The instantiation of an application from the survey is presented in Fig. 5[10]. It is a Question & Answer (Q&A) Web Application used for internal communication (and documentation) in the company. It requires three different supporting systems for its execution[11]: Elasticsearch [44], MongoDB [45], and RabbitMQ [46]. The three basic *Ingredients* for almost any Web Application are present (HTML, CSS, and JavaScript), as are two well known *Frameworks* (Node.js and React.js).

The three *Task* concepts (*WebSiteStyling*, *WebSiteMarkup*, and *WebSiteScripting*) have been determined based on the main areas of the development context. They do not represent specific tasks, as these have not been provided by the company. Their role is to exemplify how *Ingredients* relate to their supported *Tasks*.

As a first proof of concept, the instantiation in Fig. 5 is able to visually represent each development component and how they relate to the application. It does so in a compact form, with enough elements to quickly provide interesting insights into the Cocktail:

- Dependency on external services and systems is directly represented by the number of *Resources* that support the *System*;
- Dependency on *Ingredients* is represented by the number of the equivalent instances;
- Possible redundancies (too many *Ingredients* of the same type supporting the same *Task*) are quickly identified;
- *Tasks* that are too reliant on many *Ingredients*—a possible weakness point—can be directly identified by their number of *support* relations.

The instantiation provides enough information about the application and its development (specially its Cocktail) that it is effectively an *Identity Card* (CIC). It has been successfully applied to the other Cocktails obtained from the survey, providing CICs to all of them. Fig. 6 shows two more CICs for comparison.

The Identity Card shown in Fig. 6a represents a mobile educational game. In this case, the tasks have been chosen in a more granular manner, in order to better represent specific parts of development context. Fig. 6b also represents a mobile application, but not a game. It is a Covid-pass related front-end application. Differently from Fig. 6a, which applied a multi-platform engine (Unity [47]) to create and deploy the game to both mobile application stores (Apple's App Store and Google's Play Store), Fig. 6b shows how one application

[10]The names of the applications have been changed to a generic *App#* format for privacy concerns. Nonetheless, they have all been gathered in the survey and represent real software.

[11]Instantiated from the *Resource* concept.

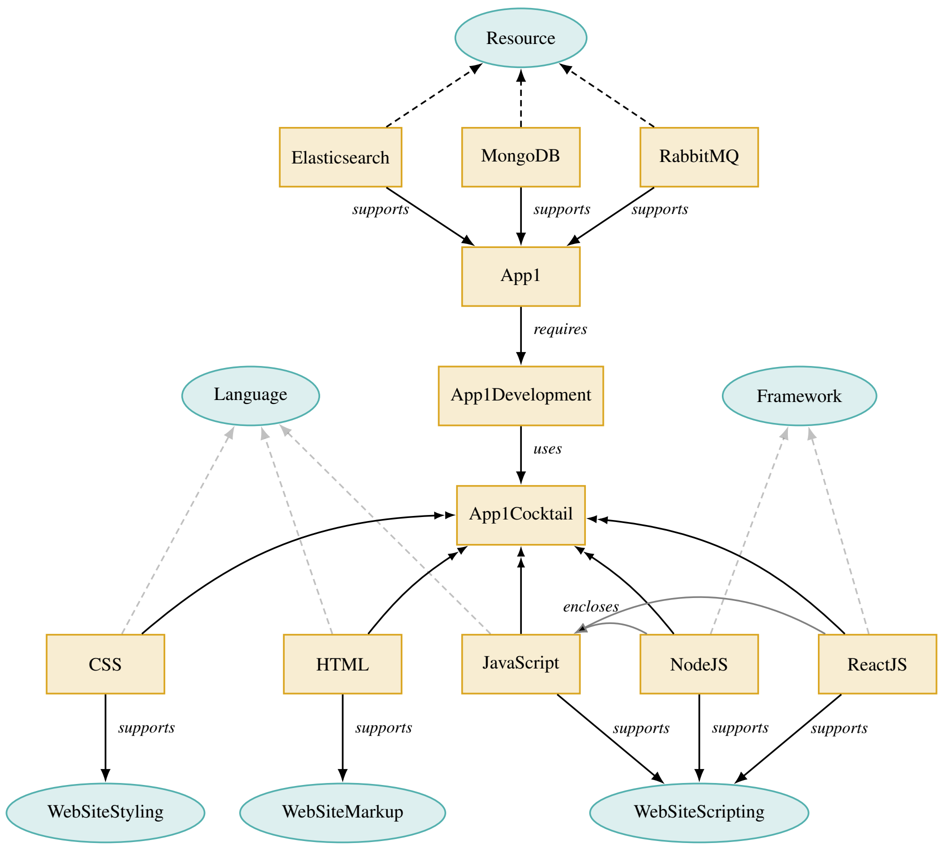

Fig. 5. An example of a Cocktail Identity Card modelled for a Q&A Web Application.

may require more than one development context, since it used mutually exclusive technologies for each platform (Swift for Apple's ecosystem, and Kotlin for Google's). In both cases, the CICs quickly present the previously listed properties for their Programming Cocktails.

The Identity Cards are valuable for quick identification of several Cocktail properties, as previously shown, but also form a foundation for the further, deeper analysis. Risks, costs, or any other form of valuation that would be layered on top of their relations could become valid augmentations to the CICs.

D. Structured Knowledge on Cocktails

The instantiations are valuable in their own merit, by organising the relations between the technologies that application development depends on. Nonetheless, its construction, by itself, relayed valuable information about the survey.

The definition of the category columns in the spreadsheet (mentioned in Subsection III-B) is a direct and practical result in this case. The initial version of these columns had several problems, from the lack of domain definitions, to redundancy in values. Since the categorization of the Ingredients will be paramount in future studies, columns *B* to *D* are of great importance. After the definition of the ontology and its application to the several Cocktails that have been gathered, the final version of the category columns was finally obtained.

The first category column (*Type*, column B) was a direct implementation of the *Ingredient* specializations (*Language*, *Library*, *Framework* and *Tool*). It is a direct definition of the ingredient's nature. The next column (*Language*, column C) represents what language is used for each other type of ingredient. It was derived directly from the relationships that the different types of ingredients establish to *Language* in the conceptual model (see the bottom-right relations in Fig. 4). Finally, column D (*Task*) represents the *Task* specializations, as previously explained. In the case that an ingredient is applied to more than one task, or used with more than one language, its line would be duplicated and changed to reflect these variations.

Another direct result from the construction of the ontology was the possibility to determine which languages are more auto-sufficient (have fewer libraries and frameworks connected to them) or more dependant of complements.

A third point-of-view for knowledge construction based on the ontology relates to project management. A few possibilities include:

- By superimposing the Identity Cards, teams and companies can quickly identify which ingredients they are more dependent on, or have more experience with;
- The definition of the *Task* specializations render an opportunity to identify common threads between projects, in

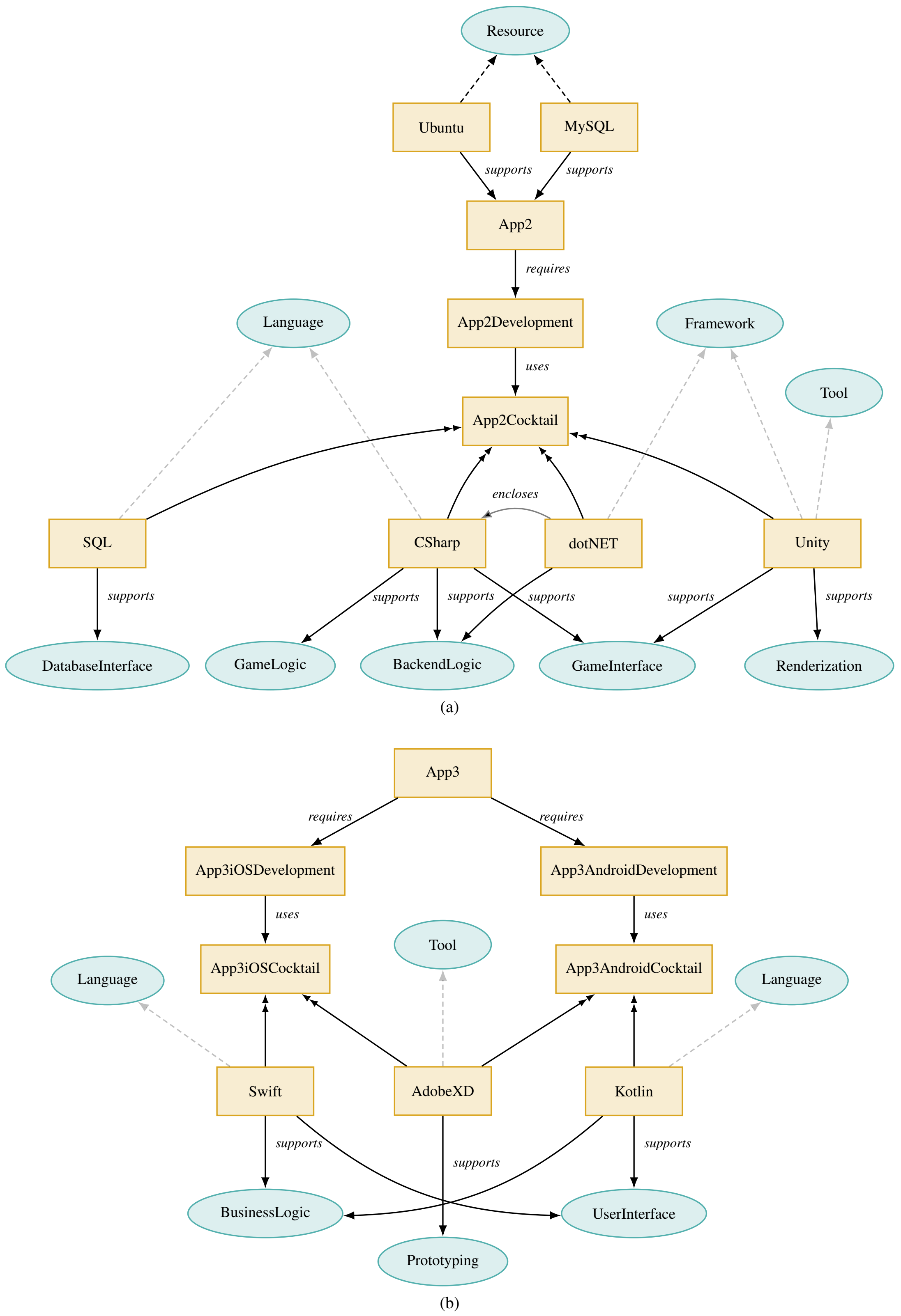

Fig. 6. Identity Cards for two different applications: a mobile educational game (a) and a Covid-pass management front-end (b).

order to standardize or evaluate how teams are structured, personnel is allocated, *etc.*

- The Identity Cards provide quick documentation about a project's technological evolution. Given that it can also be encoded in OntoDL, it can be easily registered in version control systems, such as Git [48].

As with any kind of structural representation, this ontology may be applied to analyse and support decisions on many facets of project developments, from simple documentation, to critical factors such as risk and dependency.

V. Conclusion

This paper presented an ontology based on an open conceptual model of Programming Cocktails. This ontology was initially idealised to aid in defining classifications for ingredients that were surveyed from several companies, and how they related to their development context. This result was successfully accomplished, and a new point-of-view also emerged from the instantiation of the Cocktails: the Identity Cards.

As can be observed from the previous sections, the act of constructing the ontology has already structured knowledge for the surveyed Cocktails, that will provide valuable information for future studies. Nonetheless, the ontology itself and the Identity Cards also presented more interesting opportunities. By creating a foundation on top of which further, more complex analysis could be applied, the Cocktail Identity Cards became a strong result from the ontology. It allows for quick visualization, identification and extraction of knowledge on Programming Cocktails.

Future projects will continue on this development by extending both the methodology for gathering and identifying Cocktails, but also by applying the Identity Cards on consequent studies. For attaining the former, a project for the automatic extraction of Programming Cocktails from public open source projects would be of great use, both for overall analysis, but also as a further proof of concept for the ontology. Also functioning as a direct application of the ontology, the Identity Cards will be augmented with cognitive analysis for a subsequent decision-support system for developers, project managers and teachers.

Appendix — Survey Message

The following is a redacted version of the email used to survey Programming Cocktails, as explained in Subsection III-A. Personal details were removed for privacy concerns.

* * *

In the context of our research project, we need to collect information about what we call *Programming Cocktails*, that is, clusters formed by programming languages (marking, formatting, communication…), libraries, frameworks and development environments that are used together to develop complex Applications (software systems).

For example, a very common Cocktail used to develop a current web application is:

- HTML and CSS for formatting;
- JavaScript for client-side programming;
- React as a framework for JS;
- VSCode for development.

As such, we have been contacting prominent companies in the area of Software Development to find out which Cocktails they have effectively used. As you have already shown interest in the academic context, given your participation in our Conferences/Seminars; or even proposing themes for master's theses, I come to ask: would it be possible for you to reply to this email sending me information about the Programming Cocktails that are used in your company? If this matter is not within your possibilities, we would be extremely grateful if you send us the contact of someone from your company who can help us with this information.

We make it clear that we do not intend to collect any type of confidential information. The answer is informal and should be as simple as shown in the examples below:

Application A, we used:

- Front-end: HTML, CSS, JavaScript, React;
- Back-end: Java, libssh;
- Communication: JSON;
- Database: MySQL;
- Other: Visual Studio, ActiveMQ, …

Applications B and *C*, we used:

- Full-stack: Node.js;
- Communication: gRPC;
- Database: MongoDB;
- Other: VSCode, …

Application D…

We don't even need the names of the applications, just the list of technologies used in their construction. However, the more details you can provide, such as the area or domain of the application, the more valuable your contribution will be to our research[...]

References

[1] R. R. Fenichel, J. Weizenbaum, and J. C. Yochelson, "A program to teach programming," *Communications of the ACM*, vol. 13, pp. 141–146, 03 1970. doi: 10.1145/362052.362053. [Online]. Available: https://dl.acm.org/doi/10.1145/362052.362053

[2] M. J. V. Pereira and P. R. Henriques, "Visualization/animation of programs in alma: Obtaining different results," in *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments*, 2003. doi: 10.1109/HCC.2003.1260242 pp. 260–262. [Online]. Available: https://ieeexplore.ieee.org/document/1260242

[3] T. C. Freitas, A. Costa Neto, M. J. V. Pereira, and P. R. Henriques, "Nlp/ai based techniques for programming exercises generation," R. A. P. d. Queirós and M. P. T. Pinto, Eds., vol. 104, Open Access Series in Informatics (OASIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/OASIcs.SLATE.2022.14 pp. 1–15. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/16760

[4] S. A. Teixeira, "Automatic grading of programming exercises," Master's thesis, Minho University, Braga, Portugal, 2023, to be published.

[5] P. Vasconcelos, "Haskelite: A step-by-step interpreter for teaching functional programming," R. A. P. d. Queirós and M. P. T. Pinto, Eds., vol. 104, Open Access Series in Informatics (OASIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/OASIcs.SLATE.2022.14 pp. 1–15. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/16760

[6] S. A. Robertson and M. P. Lee, "The application of second natural language acquisition pedagogy to the teaching of programming languages: a research agenda," *ACM SIGCSE Bulletin*, vol. 27, no. 4, p. 9–12, 12 1995. doi: 10.1145/216511.216511. [Online]. Available: https://dl.acm.org/doi/10.1145/216511.216517

[7] M. V. P. Almeida, L. M. Alves, M. J. V. Pereira, and G. A. R. Barbosa, "Easycoding: Methodology to support programming learning," R. Queirós, F. Portela, M. Pinto, and A. Simões, Eds., vol. 81, Open Access Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 06 2020. doi: 10.4230/OASIcs.ICPEC.2020.1. ISBN 978-3-95977-153-5. ISSN 2190-6807 pp. 1–8. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12288

[8] J. L. Plass, B. D. Homer, and C. K. Kinzer, "Foundations of game-based learning," *Educational Psychologist*, vol. 50, no. 4, pp. 258–283, 2015. doi: 10.1080/00461520.2015.1122533

[9] A. Gomes and A. J. Mendes, "Learning to program: Difficulties and solutions," Proceedings of the 2007 International Conference on Engineering and Education (ICEE). International Network on Engineering Education and Research, 2007, pp. 283–287. [Online]. Available: http://icee2007.dei.uc.pt/proceedings/papers/411.pdf

[10] B. C. Wilson and S. Shrock, "Contributing to success in an introductory computer science course: a study of twelve factors," Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education. Association for Computing Machinery, 2001. doi: 10.1145/364447.364581. ISBN 1581133294 pp. 184–188. [Online]. Available: https://dl.acm.org/doi/10.1145/364447.364581

[11] P. C. Tavares, E. M. F. Gomes, and P. R. Henriques, "O impacto da animação e da avaliação automática na motivação para o ensino da programação," Ph.D. dissertation, 2017.

[12] A. Costa Neto, C. Araújo, M. J. V. Pereira, and P. R. Henriques, "Programmers' affinity to languages," P. R. Henriques, F. Portela, R. Queirós, and A. Simões, Eds., vol. 91, Open Access Series in Informatics (OASIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/OASIcs.ICPEC.2021.3. ISBN 978-3-95977-194-8. ISSN 2190-6807 pp. 1–7. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/14219

[13] J. Alves, A. Costa Neto, M. J. V. Pereira, and P. R. Henriques, "Characterization and identification of programming languages," A. Simões, M. M. Berón, and F. Portela, Eds., vol. 104, Open Access Series in Informatics (OASIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/OASIcs.SLATE.2022.14 pp. 1–15. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/16760

[14] C. Casalnuovo, E. T. Barr, S. K. Dash, P. Devanbu, and E. Morgan, "A theory of dual channel constraints," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '20. New York, NY, USA: Association for Computing Machinery, 2020. doi: 10.1145/3377816.3381720. ISBN 9781450371261 p. 25–28. [Online]. Available: https://doi.org/10.1145/3377816.3381720

[15] J. Figueiredo and F. J. García-Peñalvo, "Building skills in introductory programming," F. J. García-Peñalvo, Ed., Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality. New York: ACM, 10 2018. doi: 10.1145/3284179. ISBN 9781450365185 p. 46–50. [Online]. Available: https://dl.acm.org/doi/10.1145/3284179.3284190

[16] M. Fourment and M. R. Gillings, "A comparison of common programming languages used in bioinformatics," *BMC Bioinformatics*, vol. 82, no. 9, 02 2008. doi: 10.1186/1471-2105-9-82. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-9-82

[17] A. H. Odeh, "Analytical and comparison study of main web programming languages: Asp and php," *TEM Journal*, vol. 8, pp. 1517–1522, 11 2019. doi: 10.18421/TEM84-58. [Online]. Available: http://www.temjournal.com/content/84/TEMJournalNovember2019_1517_1522.pdf

[18] N. Walia and A. Kalia, "Programming languages for data mining: a review," *International Journal of Computer Trends and Technology*, vol. 68, pp. 38–41, 2020. doi: 10.14445/22312803/IJCTT-V68I1P109. [Online]. Available: https://ijcttjournal.org/archives/ijctt-v68i1p109

[19] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993. doi: https://doi.org/10.1006/knac.1993.1008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1042814383710083

[20] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, vol. 25, no. 1, pp. 161–197, 1998. doi: https://doi.org/10.1016/S0169-023X(97)00056-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169023X97000566

[21] Apache Software Foundation, "Apache http server project." [Online]. Available: https://httpd.apache.org

[22] ——, "Apache activemq." [Online]. Available: https://activemq.apache.org

[23] Oracle, "Mysql." [Online]. Available: https://www.mysql.com

[24] Dormando, "Memcached." [Online]. Available: https://www.memcached.org

[25] B. W. Kernighan and D. M. Ritchie, *C Programming Language*, 2nd ed. Pearson, 03 1988.

[26] Python Foundation, "Welcome to python.org," 11 2019. [Online]. Available: https://www.python.org

[27] Mozilla Foundation, "Html: Hypertext markup language." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML

[28] ——, "Css: Cascading style sheets." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS

[29] D. D. Chamberlin, "Early history of sql," *IEEE Annals of the History of Computing*, vol. 34, pp. 78–82, 11 2012. doi: 10.1109/MAHC.2012.61. [Online]. Available: https://ieeexplore.ieee.org/document/6359709

[30] LibSSH, "Libssh." [Online]. Available: https://www.libssh.org

[31] R. Santamaria, "raylib." [Online]. Available: https://www.raylib.com

[32] Apple, "Swiftui," Apple Developer. [Online]. Available: https://developer.apple.com/xcode/swiftui/

[33] Meta Platforms, "React native." [Online]. Available: https://reactnative.dev

[34] The JUnit Team, "Junit." [Online]. Available: https://junit.org

[35] C. Araújo, P. R. Henriques, and J. J. Cerqueira, "Ontocne, characterizing learning resources for training computational thinking," in *2023 International Symposium on Computers in Education (SIIE)*, 2023. doi: 10.1109/SIIE59826.2023.10423710 pp. 1–6.

[36] S. Teixeira, R. V. Boas, F. Oliveira, C. Araújo, and P. R. Henriques, "Ontojogo: An ontology for game classification," in *2020 IEEE 8th International Conference on Serious Games and Applications for Health (SeGAH)*. Vancouver, BC, Canada: IEEE Xplore, 2020. doi: 10.1109/SeGAH49190.2020.9201876. ISBN 978-1-7281-9042-6. ISSN 2573-3060 pp. 1–8.

[37] C. Araújo, L. Lima, and P. R. Henriques, "An Ontology based approach to teach Computational Thinking," in *21st International Symposium on Computers in Education (SIIE)*, C. G. Marques, I. Pereira, and D. Pérez, Eds. IEEE Xplore, Nov 2019. doi: https://doi.org/10.1109/SIIE48397.2019.8970131. ISBN 978-1-7281-3182-5 pp. 1–6.

[38] D. R. Barbosa, "CnE-Ar: Teaching of Computational Thinking to Adults in Reconversion," Master's thesis, Minho University, Braga, Portugal, April 2021, MSc dissertation.

[39] M. de La Salete Teixeira, "Adequa, a platform for choosing Games suitable to Students' Profile," Master's thesis, Minho University, Braga, Portugal, March 2021, MSc dissertation.

[40] C. Araújo, P. R. Henriques, and J. J. Cerqueira, "Creating Learning Resources based on Programming concepts," in *Local Proceedings of the 15th International Conference on Informatics in Schools – ISSEP 2022*, A. Bollin and G. Futschek, Eds. Klagenfurt; Wien, Auatria: The Austrian Library Association, open-access net-library, Sep 2022. doi: 10.48415/2022/issep.2022 pp. 35–46.

[41] L. Martins, C. Araújo, and P. R. Henriques, "Digital Collection Creator, Visualizer and Explorer," in *8th Symposium on Languages, Applications and Technologies (SLATE 2019)*, ser. OpenAccess Series in Informatics (OASIcs), R. Rodrigues, J. Janoušek, L. Ferreira, L. Coheur, F. Batista, and H. G. Oliveira, Eds., vol. 74. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. ISBN 978-3-95977-114-6 p. 15:1–15:8. [Online]. Available: https://www.dagstuhl.de/dagpub/978-3-95977-114-6

[42] A. M. C. Dias, "ONTODL+, An ontology description language and its compiler," Master's thesis, Minho University, Braga, Portugal, Sep 2021, MSc dissertation.

[43] W3C, "Web ontology language (owl)." [Online]. Available: https://www.w3.org/OWL/

[44] Elastic, "Elastisearch." [Online]. Available: https://www.elastic.co/elasticsearch

[45] MongoDB, "Mongodb." [Online]. Available: https://www.mongodb.com

[46] Broadcom, "Rabbitmq." [Online]. Available: https://www.rabbitmq.com

[47] Unity Technologies, "Unity real-time development platform." [Online]. Available: https://unity.com

[48] Software Freedom Conservancy, "Git." [Online]. Available: https://git-scm.com