Characterization and Identification of Programming Languages

Júlio Alves
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal

Alvaro Costa Neto
Federal Institute of Education, Science and Technology of São Paulo, Barretos, Brazil

Maria João Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics, Polythechnic Insitute of Bragança, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre/LASI, University of Minho, Braga, Portugal

Abstract

This paper presents and discusses a research work whose main goal is to identify which characteristics influence the recognition and identification, by a programmer, of a programming language, specifically analysing a program source code and its linguistic style. In other words, the study that is described aims at answering the following questions: which grammatical elements – including lexical, syntactic, and semantic details – contribute the most for the characterization of a language? How many structural elements of a language may be modified without losing its identity? The long term objective of such research is to acquire new insights on the factors that can lead language engineers to design new programming languages that reduce the cognitive load of both learners and programmers. To elaborate on that subject, the paper starts with a brief explanation of programming languages fundamentals. Then, a list of the main syntactic characteristics of a set of programming languages, chosen for the study, is presented. Those characteristics outcome from the analysis we carried on at first phase of our project. To go deeper on the investigation we decided to collect and analyze the opinion of other programmers. So, the design of a survey to address that task is discussed. The answers obtained from the application of the questionnaire are analysed to present an overall picture of programming languages characteristics and their relative influence to their identification from the programmers' perspective.

2012 ACM Subject Classification Software and its engineering → Language types; Software and its engineering → Formal language definitions

Keywords and phrases Programming Languages, Programming Language Characterization, Programming Language Design, Programming Language Identification

Digital Object Identifier 10.4230/OASIcs.SLATE.2023.13

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Computers have evolved to be capable of recognizing and translating sentences, written according to formal rules, to machine code, which enables them to execute the tasks they are being asked to do. These sets of sentences are called *programming languages*. Every programming language has its own syntactic and semantic rules that make them unique [8], and that must be strictly followed in order to construct valid programs.

Each programming language has been developed with certain goals in mind, the so called programming paradigm, with some being being applied to specific fields of application, such as artificial intelligence or web development, while many others have been denominated as

general purpose languages. It can be said that behind the design of a programming language lies a philosophy of problem solving, usually suited for its intended area of application [9]. It is also evident that all languages have different characteristics, that can become obstacles not only to the development process, but also in teaching, sometimes creating such a mixture of diversified technologies applied to a system's construction that one could denominate as a *technological cocktail*. Due to each language's nature, it may be harder for programmers and students to understand how to properly apply the language rules to meet their needs [13]. The study of these differences and the identification of languages' main characteristics are not common practices in typical academia courses, making it a very important subject to be tackled.

It is then possible to raise some interesting questions that deserve further research: what indicates that two programming languages are different if what distinguishes them is not clear? What characterizes and identifies a language? Is it a syntax driven feature, such as the opening and closing symbols of code blocks? Could it be how a variable is declared? If a programming language is incrementally modified, at what point does it lose its identity?

This paper starts with a short section regarding fundamentals of programming languages, where it is explained how a programming language is specified and the role of its formal grammar. Section 3 presents the most relevant characteristics of a set of programming languages that have been elected for this study: `C`, `C++`, `C#`, `Java`, `Python` and `Haskell`. Section 4 discusses the design process of a survey, through which programmers were inquired about how they identified a programming language through several source code snippets. Section 5 presents the analysis of results collected using the referred survey. Finally, the paper concludes emphasizing what was discovered from the study so far conduced and proposes the potential improvements to be done through future research.

2 Fundamentals in Programming Languages

Programming languages are defined by sets of rules. These rules define both its *structure* – syntactic rules – and *meaning* – semantic rules. A *context-free grammar* is usually defined to generate a parser, that checks if a source code abides to these syntactic rules. The sentences derived from these rules are used to create programs that perform specific tasks, solve problems, and interact with other systems. A considerable number of programming languages has been developed, each one portraying its own syntax and features that make them more suitable to solve problems in a certain domain [2, 5].

As previously stated, the main way to define the syntax of a programming language is by using a grammar. A formal grammar is composed of a set of rules that must be used to derive correct, or valid, sentences of that language. In order to specify a grammar, four elements are needed:

- **Finite set of tokens** - The elementary symbols of the language defined by the grammar, usually called the Alphabet or Vocabulary;
- **Finite set of non-terminal symbols** - Represent the language concepts that give raise to, or derive, the sentences and sub-sentences;
- **Finite set of grammar rules** - Substitution rules in which a non-terminal symbol (on the left) may be substituted by a sequence of terminals and non-terminals defined on the right side of the derivation operator;
- **Start non-terminal symbol** - The starting point for the derivation process.

Despite this standard way of designing programming languages, the substitution rules and the concepts expressed by each non-terminal symbol provide distinct characteristics to each language, establishing a sense of identity. This sense may also be realised through the use and application of each language, which allows for their characterization beyond formal definitions.

3 Characterization of Programming Languages

As it is known by programmers, every programming language is different. These differences may be on their purpose, the paradigms they realize, features (such as portability or efficiency) or, and most notably, their lexicon and syntax. Programming languages may be divided in two groups that are defined by their programming paradigms: *declarative* and *imperative*. The former includes all the languages that are based on a set of declarations or specifications of several program elements, while the latter is based on a sequence of instructions, or orders. As can be observed in the following sections, out of all the selected languages the only language under study that belongs to the declarative paradigm is `Haskell`. In order to better understand these differences from the programmers' points of view and construct a well founded survey, it was important to collect the main features that define each languages identity, gathering the syntactic and semantic characteristics that are either uniquely or commonly recognizable to each of them. In this way, it became easier to determine which questions to ask and how to ask them, as will be discussed in Section 4.

The main reasons for choosing these languages were:

- General similarity (`C` *versus* `C++`, `Java` *versus* `C#`);
- General dissimilarity (`C` *versus* `Python` and `Haskell`);
- Variation of paradigms;
- Ascendancy (`C` *versus* `C++`, `C#`, and `Java`);
- Possible familiarity of the respondents.

It is important to note that these reasons were meant to compare similarities and dissimilarities, in order to raise or lower the linguistic contrast of the snippets in the questionnaire.

3.1 C

Being one of the most used and recognizable programming languages in history, `C` was an important entry point for later comparison in the survey. Many other languages, such as `C++`, derived their syntactic and semantic rules from those previously found in `C`. Some of the defining characteristics of `C` are [14]:

- `C` is a Procedural Programming language;
- `C` is strong and statically typed;
- A mandatory `main` function that defines the entry point for execution;
- Curly brackets are used to enclose the contents of a code block;
- Semi-colons indicate the end of a statement;
- Variables and constants must be explicitly declared using the `type identifier` pattern;
- Functions are the structural elements of the source code;
- Function signatures must have a return type, followed by its name and a pair of parenthesis grouping all arguments;
- Functions may return nothing (`void`);

- The fixed-length array is the only data collection built into the language and its declaration follows the basic variable declaration pattern. Lengths and dimensions are defined using square brackets after the identifier;
- Pointers are memory addresses and are used to indirectly access memory addresses;
- A pointer is declared like any other variable, with an added asterisk prefix before the identifier;
- An ampersand is used to fetch the address of a variable;
- Characteristic and recognizable functions are present in the standard library, such as `printf` and `scanf`.

3.2 C++

`C++` was designed as a *super set language* of `C`, inheriting a lot of its syntactic and semantic characteristics. The concept of classes is introduced in `C++`, which brings `C`'s structure and form to a new paradigm. Some of its main characteristics are [15]:

- Just like `C`, `C++` is also strong and statically typed;
- `C++` imported all the basic syntactic and semantic elements from `C`. Therefore, characteristics such as the entry point for execution, code block notation, declaration syntax and so on were either unaltered or simply augmented;
- The concept of classes was the most structural aspect that `C++` implemented on top of `C`, defining an Object Oriented Programing (OOP) language;
- Classes are also structural elements of the source code;
- Classes are declared using the `class` keyword, in a similar fashion to composite structures (`struct`) in `C`;
- Data members are declared inside the class definition using the standard variable declaration pattern;
- Methods may be declared either inside the class definition or outside, using a specific notation (`class::method`);
- The standard library added several data structures, such as dynamic vectors, linked lists, and queues, while also provided several new redundant methods and operators that implemented the same functionality as `C`'s standard library – `«` instead of `printf`, `»` instead of `scanf`, and so on;
- Some other additions to `C`'s mechanisms are also present, such as operator overloading, `new` and `delete` commands, templates, virtual functions *etc.*;
- In `C++`, the notion of *namespaces* is introduced. A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.

3.3 C#

Just as `C++` was designed to be an evolution of `C`, `C#` was meant to pursue this path further concerning `C++`, while also competing in the same space of `Java`. `C#` simplifies many of `C++`'s features and solves some of its problems (such as pointers and memory allocation issues) while maintaining its power and paradigm. Some of `C#` characteristics are [1]:

- Like its predecessors, `C#` is also strong and statically typed;
- The need of a `Main` method, instead of a function;
- As a successor of `C++`, it maintains the same syntatic features to delimit code blocks and to end statements, both carried over from `C`;
- When declaring a variable, one can explicitly state what the variable type will be or can use the keyword `var`, and the variable type will be inferred by the compiler;

- As C# is an Object Oriented Programming language, classes are the structural elements;
- C# offers a variety of data structures, such as arrays, lists, stacks, queues, sets and dictionaries;
- Directives indicate that a specific namespace will be used. It is composed by the keyword `using`, followed by the desired namespace identifier;
- References types supplant pointers and remove the asterisk notation.

3.4 Java

Java is an Object Oriented Programming language that, despite not being based on C and C++, certainly was influenced by them. Some of Java characteristics are [6]:

- Java is a strong and statically typed language;
- As Java was inspired by C and C++, many syntactic and semantic elements have been imported into it, such as the mandatory entry point (`main` method), curly brackets to delimit code blocks, semi-colons to end statements and the standard variable declaration pattern;
- All variables must be declared with a type;
- There are specific declaration modifiers, such as `final`;
- Just like C#, classes are the structural element;
- Java also offers a significant variety of data structures, such as arrays, lists, stacks, queues, sets and maps;
- Types can be divided into two semantic categories: primitive types, such as `boolean` and numeric types; and reference types such as classes, interfaces and arrays. Values in reference types relate to objects;
- The special type `null` has no name and can be assigned or cast to any reference type. Its reference is the only possible value of an expression of type `null`;
- Conversions, such as identity conversion, widening and narrowing primitive conversion, unchecked conversion and capture conversion;
- In order to use other packages, one must use the keyword `import` followed by the name of the package, similar to C++ and C# namespaces.

3.5 Python

Python is an Interpreted, Scripting and Procedural language but also supports other programming paradigms such as Object Oriented and Functional. Some of Python characteristics are [10]:

- Unlike the previously discussed languages, Python is dynamically typed;
- A `main` function or method is not required;
- In Python, instead of using braces to delimit code blocks and scope, these are defined by indentation;
- Semi-colons are not required to mark the end of a statement, a new line is enough;
- Variables are declared when a value is assigned to them, without needing to declare their type;
- In order to define a function, its name must be preceded by the keyword `def`. After the function name, a pair of parenthesis must be used containing and optional a set of arguments;
- Despite being an OOP language, Python's structural elements are functions;
- There are different data structures like lists, tuples, sets, dictionaries, strings and range available in the standard library, some of which have special syntax associated with their use.

3.6 Haskell

Differently from all other languages previously discussed, `Haskell` is a purely Functional Programming language meaning, instead of telling the computer what to do, the programmer tells the computer what something is. `Haskell` is also lazily executed, so the interpreter will compute values only when they are actually needed, unless told otherwise. Some of `Haskell` characteristics are [11]:

- `Haskell` is a Declarative language since is based on the specification of a set of functions and expressions;
- `Haskell` is a strong and statically typed language;
- Code blocks are delimited by indentation, just like `Python`;
- A new line is used to define the end of a statement;
- In order to declare a variable in `Haskell`, the programmer must use either the keyword `let` or `var`, followed by the variable name. Variables are immutable by default, however the keyword `var` allows the variable to be mutable;
- Due to `Haskell`'s inference mechanism, it is not needed to specify variables types in declarations;
- In order to declare a function type, the programmer declares its name followed by the `::` operator – which can be read as "type of" – and all its parameters. The parameters are separated by the arrow operator (`->`), with the last parameter being the return type;
- Functions are the main structural element of the language;
- The data structures offered by `Haskell` are: lists, tuples, sets, maps, strings and ranges;
- Instead of requiring multiple lines of code to write nested `if...then...else` conditions, `Haskell` implements *guards*. Guards are indicated by a pipe character (`|`) followed by a boolean expression and what will be evaluated in case the expression is true. Should it be false, the next guard is evaluated, and so on;
- `where` - `Haskell` doesn't allow to store variables for a future use. However, with the keyword `where`, users can declare variables to be used inside a function with its biggest limitation being the fact that `where` scope is limited to the function it was declared;
- `Haskell` also has a `switch` like statement, with the use of the keyword `case` in the format `case <expression> of <pattern> -> <result>`;
- Recursion plays a big role on the regular use of the language.

4 Survey Design

A crucial part of this research was the development and implementation of a survey[1] to gather the different perspectives of a programmer regarding their known languages identification. While not directly applied, the general concepts behind Value-Focused Thinking [7] founded the rationale throughout the construction of the survey, as was the case with previous studies [4].

Since the survey contemplated multiple programming languages and the target audience had various degrees of knowledge of these languages, it became critical to weigh one's answers. For this reason, the survey started with a section containing only one question on the respondent's familiarity with the six chosen programming languages (`C`, `C++`, `C#`, `Java`, `Python` and `Haskell`). Respondents could choose his or her level of knowledge in each language from five different options, ranging from complete ignorance to profound knowledge.

[1] Available at `https://forms.gle/6kBuHhYD5FPHK8Cp9`.

From this section forward, all questions regarded different code snippets written in the programming languages under study. These code snippets were created aiming for conciseness, while also contrasting or emphasizing specific linguistic traits. As previously stated in section 3, both similarities and dissimilarities were implemented when two snippets were compared. This was made to evaluate not only how minor details influence the differentiation of similar languages, but also to identify which linguistic elements have stronger identification roles, raising similarity between very different languages.

The second section aimed to evaluate if the respondent could identify the language of a snippet of code, justifying his or her answer. There were six pairs of questions ("What is the language present in the following snippet?" and "Justify your rationale for the previous answer."), one for each language. The purpose of this section was to understand the reasoning behind the identification of a programming language and to detect if multiple people use the same thought process, specifically if the snippet contains a distinctive feature that is commonly reported as important for the identification. Using a similar strategy as applied to the previous section, for each language the respondent could choose one of four possible answers: doesn't know, it sure isn't, maybe and absolutely is. Listing 1 presents the snippet used for the question regarding the C language identification.

Listing 1 C language snippet used in the second section of the survey.

```
int main()
{
    if( n <= 1 || (n = atoi(argv[1])) <= 0 ) n = 8;
    int hist[n];
    solve(n, 0, hist);
}
```

For the third section, the identification of programming languages was evaluated through comparisons. In this section, the respondent was presented with two code snippets representing solutions to the same problem, each written in a different language, and asked to identify which programming languages were used in each snippet. Listings 2 and 3 exemplify the comparisons that were made in each of the third section's questions. The similarities in general form were intentional to pinpoint characteristics that respondents justified as crucial to the identification of each language – or his or her inability to do so.

Listing 2 Java snippet used in the third section of the survey. This snippet was presented in the same question as listing 3 for comparison.

```
class FileIOTest {
    public static void main(String[] args) throws Exception {
        var lines = Files.readAllLines(Paths.get("input.txt"));
        Files.write(Paths.get("output.txt"),lines);
    }
}
```

Listing 3 C# snippet used for comparison with the one presented in listing 2, in the third section of the survey.

```
class FileIOTest
{
    public static void Main(string[] args)
    {
        var lines = File.ReadLines.("input.txt");
        File.WriteAllLines("output.txt",lines);
    }
}
```

In the fourth and last section, a new approach was applied. This new approach consisted in progressively modifying a given a code snippet in one programming language until it became a substantially different. The respondent was then asked at which point of these progressive changes he or she believed that the snippet of code no longer resembled the original language, that is, at which point the language lost its identity. Figure 1 shows one of the original snippets, followed by their progressively altered versions. The changes were small and related to one of the language's main characteristics at each version. This section aimed to establish what is the *breaking-point* to a language's identification, meaning which characteristics are so entrenched into its definition that once it is changed, programmers can no longer associate code with it.

```
int main(int argc, char *argv[])
{
    char t[255]="alphaBETA";
    str_toupper(t);
    printf("uppercase: %s\n",t);
    str_tolower(t);
    printf("lowercase: %s\n",t);
    return 0;
}
```

```
int main(int argc, string argv)
{
    string t="alphaBETA";
    str_toupper(t);
    printf("uppercase: %s\n",t);
    str_tolower(t);
    printf("lowercase: %s\n",t);
    return 0;
}
```

```
int main(int argc, string argv)
{
    string t="alphaBETA";
    str_toupper(t);
    write("uppercase: ",t);
    str_tolower(t);
    write("lowercase: ",t);
    return 0;
}
```

```
main(argc, argv)
{
    t="alphaBETA";
    str_toupper(t);
    printf("uppercase: ",t);
    str_tolower(t);
    printf("lowercase: ",t);
    return 0;
}
```

Figure 1 Four progressively altered snippets of C code used to establish what is the breaking point for the identification of a programming language.

5 Survey Results Analysis

To start the results analysis, it is important to begin with the first question, as it shows how well the respondents knew the six chosen languages. `Java`,`C` and `Python` were the most well known languages, with 95%, 88% and 93%, respectively, of the respondents stating to be, at least, capable of using the languages. These results are indicative that questions regarding `Java`,`C` and `Python` can give a higher understanding on what identifies these languages. On the other hand, `Haskell`, `C++` and `C#` respectively had only 47%, 45% and 31% of respondents stating to be, at least, capable of using the languages.

5.1 Second Section: Language Identification

Regarding `C`, respondents successfully identified it, stating the main reasons to its identification were:

- Variable declaration syntax;
- Use of pointers;
- Syntax features like semicolons, brackets to delimit code blocks and functions signature;

Like `C`, `Java` was also correctly identified, with the main reasons being:

- Methods access modifiers;
- Methods signatures;
- Variables types;
- The usage of `System.out.println`;

On the third snippet shown, the language present was `C#`, however, as the respondents weren't very familiarized with this language, there was a consistent lack of confidence to distinguish whether the language was `Java` or `C#`. Out of the total, nine people said they were sure it was `Java`, and nine said they were sure it was `C#`. On the other hand, twenty four respondents said it could be `Java`, and twenty one said it could be `C#`. There were also fourteen respondents who believed it could be `C++`. It was unanimous, however, that it was neither `Python` nor `C`. The audience revealed a big difficulty in finding characteristics which allowed them to distinguish between `Java` and `C#`. The respondents who could successfully identify it as `C#`, expressed that what made them sure was the use of PascalCase naming convention on identifiers. Since it represents a writing style other than a language characteristic, it is safe to say that the identification wasn't a consequence of the language's formal definition, but its conventions of use instead.

The fourth snippet consisted of a piece of `C++` code. It was very obvious for the audience that this snippet wasn't `C`, `Java` or `Python`. There was some uncertainty if it could be `C#` as the respondents could identify this language as being a part of the `C` language's branch of derived languages. Nonetheless, there was a major consistency as identifying the language as being `C++` due to:

- The usage of the `std` library;
- `Unsigned long long` type;
- The method or member function call notation.

On the fifth snippet shown, the language present was `Python`. It was very evident for the audience that this was `Python` as no other language was given a positive answer and `Python` was picked as the right language by almost one hundred percent of the users. What made them answer `Python` was:

- Block syntax and indentation;
- The use of `def` when defining a function;
- Type inference;

For the last snippet on this section, respondents were finally presented with a `Haskell` code snippet. Once again, the respondents were certain it couldn't be any other language other than `Haskell`. The reasons given for this were:

- Functional programming style;
- Point free syntax, a style of writing Haskell code that avoids explicit mention of the arguments of a function;
- Function signatures;
- The complete overall difference to the other languages;

5.2 Third Section: Language Comparison

5.2.1 `Java` vs `C#`

As `Java` and `C#` were two languages which it was expected some difficulties to arise, two comparisons were made. On the first comparison, the respondents were able to successfully identify the languages correctly, stating – as they did on the previous section – the difference

in naming convention as the biggest contributor for the identification. Respondents also pointed the way exceptions are thrown in `Java` as a differentiating factor. On the second comparison however, there was a big uncertainty on the `C#` snippet, with respondents being mainly divided between `C#`, `C++` and `Java`. There were also some answers that were open to the possibility of it being `C` or even another language. This indecision is due to respondents not being able to associate the types `ulong` and `uint` to a language. `Java`'s identification was very straightforward to the respondents, stating that the use of the keyword `final` on an argument as the reason the snippet was written in `Java`.

5.2.2 `C` vs `C++`

In this comparison, the respondents were on the same page, with virtually no problems to the correct identification. However, there were only two reasons on why they believed one language was `C` and the other `C++`: the functions used to output something (`printf` for `C` and `cout` for `C++`) and the use of `namespace` on `C++`.

5.2.3 `C` vs `Java`

Once again, there was no doubt within the respondents on what language was present in each snippet. Respondents stated that the output functions `printf` and `System.out.println` were the main reasons for their correct identification. There were also a few answers pointing that the way arrays are declared was also a contributing factor.

5.3 Fourth Section: Language Identification Breaking-point

For `C`, respondents were adamant that from the second snippet onward the language couldn't be `C`. The modification that triggered this opinion was the replacement of `char *` with a new type `string`, commonly used in other languages.

With `C++`, the answers were also unanimous on saying that, from the second snippet on, the language could not be `C++`. The modification on how libraries are imported, switching from `namespace` and `#include <libraryName>` to `with` was the defining change to arrive at the breaking-point.

On the third question, respondents agreed that the first snippet was `C#` and were also open to the possibility of it on the second snippet. It is possible to infer that the output function is not a very defining characteristic of the language, as that was the modification made from the first to the second snippet, changing `System.Console.WriteLine()` to `write()`. However, on the third snippet it became obvious to the respondents that the language wasn't `C#`. On this snippet, the way code blocks are structured was modified, replacing brackets with indentation, such as in `Python`.

Curiously, the exact same reaction was obtained with `Java`. On the second snippet, the code blocks syntax was modified from curly brackets to indentation, triggering the respondents to not identify the language as being `Java`.

With `Haskell` and `Python`, respondents were open to the possibility of being the original languages until the function declaration syntax were modified. In `Haskell`'s case, the modification was the replacement of the standard syntax `functionName :: argument -> argument -> result` with `functionName(argument, argument)`. `Python`'s modification was even more subtle, with the mere change of the keyword `def` to `function`.

The analysis of the results was obtained through the answers gathered from a class of fourth year software engineering students. A select group of people, that included university professors and software engineers, was also invited to answer, however, the survey has not

been yet made publicly available, as these results and the students' feedback will be used to make the necessary improvements for a future version.

The respondents were selected for their background in computer programming languages. The majority of the respondents were Master's Degree students present in a course of Languages Engineering at University of Minho (UMinho). In total, 44 people responded to the questionnaire, 39 of which were students. The other 5 people were professors of the Department of Informatics, also at UMinho.

6 Conclusion

Machine Learning [12, 16] has been extensively used on the subject of computer programming languages automatic identification. These studies applied different types of classifications such as Image Based Classification [3], Algorithmic Classification [9] and Source Code Classification [8, 17], however not many studies have been conducted to consider the programmers' points of view. What are the most relevant linguistic features that programmers observe, consciously or not, to correctly identify a language used in a source code? What and how much can a language's syntax and semantics change and still retain its identity? In order to answer these questions, the most straightforward solution is to actually ask people that deal with programming languages. The challenge then becomes not only what to inquire programmers about their perception of a language, how it is recognized and identified, but also *how to properly ask them*.

This paper presented an analysis of typical linguistic features, commonly found in six established programming languages (`C`, `C++`, `C#`, `Java`, `Python`, and `Haskell`) followed by the design and application of a survey that seeks to better understand the intrinsic relationship between programmers and their main tools of the trade: programming languages.

The survey was designed around three main approaches:

1. Direct identification of a programming language;
2. Comparison between similar languages;
3. Determination of the identity breaking-point.

The three approaches were implemented via corresponding sections in the survey. The questions for approaches 1 and 2 were constructed in pairs: a multiple-choice direct question, asking respondents to identify the language in the respective snippets of code; followed by an open-ended justification. The last section (approach 3) used only multiple-choice direct questions, gradually changing snippets of source code to evaluate at which point the original language (shown at the first snippet of each question) lost its identity.

The common threads of identification in the answers showed that some typically associated syntactic features, such as code block syntax, method signatures and variable declaration patterns are contributing factors to identify a programming language – which was expected. Other answers lead to some unpredicted results, pointing to the use of standard library elements (functions, classes, methods, etc.) and specially the naming convention that is typical – and sometimes, mandatory – in several programming languages. While not a characteristic of the formal definition of the language, it is nonetheless prevalent and linguistically related to it.

The comparison between languages pointed the inevitable mixed results between languages that are very similar and descendants from a common parent – `C#` and `Java` being the most prominent example, both derived from `C` and `C++`. The two main differentiating features, as pointed by the respondents, were naming convention and typical standard library elements, once more.

On the other hand, the results were not very conclusive on the breaking-point analysis, since there is an argument to be made for changing the way the questions were asked. Most respondents correctly identified the first snippet as being the original and unaltered language, and the second one forward as promptly loosing its identity. Since the changes were diminutive and triggered the breaking-point almost immediately, it is inconclusive as to which ones contributed the most to the lost of identity.

The feedback was one of the goals of this first application, as respondents pointed to two main difficulties they faced when answering the survey: they felt questions related to languages they didn't know should not be present and, as pointed in the previous paragraph, the breaking-point analysis must change in form, avoiding the immediate lost of identity. The latter showed us that a different approach to the breaking-point analysis must be tackled for a next version, aiming to gather enough information towards a more conclusive argument.

The results already showed interesting facets about the relationship between programmers and the languages they know and use. In a setting of multiple different technologies being applied to the construction of a modern computer system – a *technological cocktail* – this naturally occurring change of patterns and identities in the development process might be cumbersome. If well understood, this cognitive load may be lightened by diminishing the changes in language identity or even choosing technologies that use languages with similar identity features. Not only that but, at an initial stage of a programmer study, it can also aid a programmer to learn a programming language with more ease, given that it will allow him to choose a programming language more suited to his preferences.

This research is part of a larger group of studies that aim to identify and understand the human relations involved in computer programming. Future works include the review and refactoring of the survey, specially in the last section, followed by the next version to be openly applied for general public response. Finally, a guiding system will be implemented for the detection of a language's identity in snippets of source code, based on the programmers' points of view.

References

1 Sam A Abolrous. *Learn C-Sharp - Includes the C-Sharp 3.0 Features*. Wordware Pub, 2007.

2 Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques & tools*. Addison Wesley, 2007.

3 Francesca Del Bonifro, Maurizio Gabbrielli, Antonio Lategano, and Stefano Zacchiroli. Image-based many-language programming language identification. *PeerJ Computer Science*, 7, 2021.

4 Alvaro Costa Neto, Cristiana Araújo, Maria João Varanda Pereira, and Pedro Rangel Henriques. Value-Focused Investigation into Programming Languages Affinity. In Alberto Simões and João Carlos Silva, editors, *Third International Computer Programming Education Conference (ICPEC 2022)*, volume 102 of *Open Access Series in Informatics (OASIcs)*, pages 1:1–1:12, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2022.1.

5 Robert W. Floyd. The syntax of programming languages, 1964.

6 James Gosling, William N. Joy, and Guy L. Steele. The java language specification, 1996.

7 Ralph L. Keeney. Value-focused thinking: Identifying decision opportunities and creating alternatives. *European Journal of Operational Research*, 92(3):537–549, 1996. doi:10.1016/0377-2217(96)00004-5.

8 Jyotiska Nath Khasnabish, Mitali Sodhi, Jayati Deshmukh, and Gopalakrishnan Srinivasaraghavan. Detecting programming language from source code using bayesian learning techniques. In *MLDM*, 2014.

9 David Klein, Kyle Murray, and Simon Weber. Algorithmic programming language identification. *ArXiv*, abs/1106.4064, 2011.

10 Dave Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. Platypus Global Media, 2015.

11 Miran Lipovaca. *Learn You a Haskell for Great Good! A Beginner's Guide*. No Starch Press, 2011.

12 Tom Mitchell. Machine learning, 1997.

13 Alvaro Costa Neto, Cristiana Araújo, Maria João Varanda Pereira, and Pedro Rangel Henriques. Programmers' affinity to languages. In *ICPEC*, 2021.

14 Easy Programming. *C Programming Language The Ultimate Beginner's Guide*. CreateSpace Independent Publishing Platform, 2016.

15 Bjarne Stroustrup. *A History of C++: 1979–1991*, pages 699–769. Association for Computing Machinery, New York, NY, USA, 1996.

16 Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 3rd edition, 2010.

17 Shaul Zevin and Catherine Holzem. Machine learning based source code classification using syntax oriented features. *ArXiv*, abs/1703.07638, 2017.