Building a programming language without the dependency on English

CMP6102 Individual Project: Literature Review

Pandelis Zembashis — S15101590
Supervisor: Emmmett Cooper

December 2018

1 Area Of Research

The fundamental aim of this project is to build an accessible programming language. To do so we will need to explore the base principals of programming language theory. We will look into some of the existing research in place for what has been defined as good programming practice. This shall be followed by a review of a subset of languages from various decades and compare how they have evolved from each other and improved on the ideas of previous generations of languages.

A second goal of the project is to make the language accessible to users who are not native english english speakers and have little to no programming experience. We will be taking a subjective look at the implementation of the languages and the symbols they use. We'll also take into account the country of origin of the main developers of the language to see if this has an effect on the symbols used. From this we may be able to deduce if there are indeed hints of linguistic culture within the design of languages which strengthens the case for a language which strives to be more generic in its choice of symbols.

2 If you See What I Mean

One of the earliest attempts at a programming language that attempted to make use of natural language to define more readable programs can be found in the design of ISWIM (If you See What I Mean). This language was first described in a 1966 edition the Association for Computing Machinery's journal titled "The Next 700 Programming Languages" (Landin, 1966).

In this paper the author describes the mistakes make by languages of the time, such as ALGOL60. They describe programming languages as a means to express things in terms of other things as well as existing as defining a basic set of given things that can be used. The author attempts to outline a language that better defines the given things defined by the language by utilising linguistic structure.

The author makes particular reference to mathematical communication in which he describes how naturally a *where* clause reads.

$$\begin{gathered} f(bA-2c)+f(2b-c) \\ \textbf{where } f(x)=x(x+a) \\ \textbf{and } b=u/(u+l) \\ \textbf{and } c=v/(v-t-1) \end{gathered}$$

In the expression above it is very natural to follow the grammar of the equation and deduce what is happening. The where clause is naturally extended by adding **and**, which we can follow and logically determine that this is an addition to the previous statement.

This is the type of grammatical reliance we must aim to avoid in the design of our language. We cannot assume the end user of the language will be familiar with these grammatical primitives.

The design of ISWIM attempts to take this into consideration with multiple levels of abstraction in the design of the language itself. The abstraction we are most interested in is what the author characterises as logical ISWIM. This abstraction is defined as:

"uncommitted to character sets and type faces, but committed as to the sequence of textual elements, and the grammatical rules for grouping them"

This is in theory would allow ISWIM to be translated to a different language and grammar completely as long as the definition of ISWIM could be adapted to the end language's grammar. As it's definition is also character set agnostic, it would naturally make sense that we could build a unique series of characters which we assign meaning and grammar to in order to represent the language.

3 Naturalistic Programming

In (Lopes et al., 2003) the authors outline a future beyond aspect-oriented programming (AOP), a method of improving system modularity by modularizing crosscutting concerns (Murphy and Schwanninger, 2006). The authors propose that high level programming do not follow natural human communication patterns or the way that we think. The next breakthrough will be by taking aspects of natural language to make descriptions concise, effective and understandable.

There is a wide breadth of naturalistic type languages in use by programmers which are outlined in the survey (Pulido-Prieto and Juarez-Martinez, 2017). This is a fantastic survey of a wide array of high level languages that follow grammatical and natural english rules to provide programmers with what is ideally an easier understanding of code that is being written.

One such language which is among the most used languages in the world with multiple dialects is Structured Query Language (SQL) (*Stack Overflow Developer Survey 2018* n.d.). SQL is very expressive and can be read like a regular English sentence. Without much knowledge of SQL at all you can make deductions as to what the program will do by following its natural grammar rules.

Take the following expression for example:

```
SELECT id, name, age, grade
FROM students
WHERE grade > 90
```

Reading the expression we can deduce that this is accessing data by selecting various properties *from* some sort of students record *where* the individuals grade is greater than 90. Very natural to follow and conforms to some of Landin's ideas mentioned previously (Landin, 1966).

The programming language Pegasus is an example of the most complete naturalistic programming languages that have been designed (Knöll and Mezini, 2006).

The authors of this language highlight 4 major unsolved gaps in developer's expectations and programming techniques.

1. Mental problem. An idea must be adapted to a programming language, decomposing orgrouping its elements.

2. Programming language problem. An algorithm must be translated into several languages that use new technologies and concepts, meaning that old languages still have a strong presence.

3. Natural language problem. In the modern era, while people from all around the worldwork in teams, they generally have a different native language and their own particular regionalisms.

4. Technical problem. In system design, the developer's contribution and time is in-vested by programmers thinking about how to best adapt ideas to produce an efficient implementation.

We should pay particular attention to point number 3. The natural language and culture problem. With teams spanning the globe and working in increasingly more diverse and integrated environments this problem becomes more prevalent. Even things as simple as color vs colour throw off developers every day when working with CSS, a markup that distinctly american.

Pegasus actually tackles this problem head on and is able to generate programs in English and German languages. Using the definition of Pegasus proposed by the authors there has also been an extension to the language Ara Pegasus (Mefteh and Knöll, 2012). This extension to the language allows for users of Arabic to program in a natural manner making it much more accessible for them to begin programming. Although the language itself is defined in a way where the ideas and implementations can be translated between languages it still means a single Pegasus program cannot actually easily be read by another language speaker unless it is translated.

This is an aspect we aim to tackle with our programming language, building on principals of natural language but defining out own grammatical rules along the way which are assigned specific symbols for meaning.

4 Learning barriers to non native English speakers

When teaching logical principals of mathematics there are also many barriers that non native speakers face when taught about mathematics in English (Mallet and G., 2010). If students are not familiar with the language they are being taught in natural sounding ideas like the *where* clause which we continually point to as being so natural sounding, can be completely foreign and alien. Mallet highlights how without additional teaching support, students which do not adapt to the spoken form of the language the subject is being taught in, the students fall behind.

This is worsened by digital teaching aids which are also not in the students native language which further puts the students at a disadvantage.

Preliminary results from (Bretag, Horrocks, and Smith, 2002) suggest that students from non-English speaking backgrounds consistently achieve lower than average grades in information systems courses in South Australia.

5 Programming Assistance Tools

One tool which exists to aid in assisting programming techniques is RoboMind (*RoboMind* n.d.). This tool is of particular note because it has been translated to 26 languages. RoboMind utilises regular coding practices and the entire process of writing a program is done by writing the characters into its editor as you would program traditionally.

There are other programming assistance tools which abstract the writing of programs. Such environments use a block based approach such as Scratch (Resnick, Kafai, and Maeda, 2005). There are studies however which point to such block environments not being as effective as students typing commands in regular plaintext in their own language. In south african secondary schools (Koorsse, Cilliers, and Calitz, 2015) it was observed that users of RoboMind were more confident in their understanding of programming fundamentals.

6 Conclusions

There is allot of active research with the aims of making programming languages more and more abstract. The natural progression here is to make them follow natural language as that is how we commonly communicate ideas. As noted by those researching Naturalistic programming languages this limits our userbase to those who have prior knowledge of those grammatical rules.

Users who are non native speakers of teaching subject matter are at a distinct disadvantage when the basis for teaching is within the language, like with mathematics and many programming languages.

It would be beneficial to many new learners to have an environment in which they can adapt to new programming concepts without first needing to grapple with language specific grammar like the case is with SQL for example.

References

Bretag, Tracey, Sam Horrocks, and Jeff Smith (2002). “Developing classroom practices to support NESB students in information systems courses: Some preliminary findings”. PhD thesis. Shannon Research Press.

Knöll, Roman and Mira Mezini (2006). “Pegasus: First Steps Toward a Naturalistic Programming Language”. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications.* OOPSLA ’06. Portland, Oregon, USA: ACM, pp. 542–559. ISBN: 1-59593-491-X. DOI: `10.1145/1176617.1176628`. URL: `http://doi.acm.org.ezproxy.bcu.ac.uk/10.1145/1176617.1176628`.

Koorsse, Melisa, Charmain Cilliers, and André Calitz (2015). “Programming assistance tools to support the learning of IT programming in South African secondary schools”. In: *Computers & Education* 82, pp. 162–178.

Landin, P. J. (Mar. 1966). “The Next 700 Programming Languages”. In: *Commun. ACM* 9.3, pp. 157–166. ISSN: 0001-0782. DOI: `10.1145/365230.365257`. URL: `http://doi.acm.org.ezproxy.bcu.ac.uk/10.1145/365230.365257`.

Lopes, Cristina Videira et al. (2003). “Beyond AOP: Toward Naturalistic Programming”. In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications.* OOPSLA ’03. Anaheim, CA, USA: ACM, pp. 198–207. ISBN: 1-58113-751-6. DOI: `10.1145/949344.949400`. URL: `http://doi.acm.org.ezproxy.bcu.ac.uk/10.1145/949344.949400`.

Mallet and Dann G. (Nov. 2010). *Walking a Mile in Their Shoes: Non-Native English Speakers’ Difficulties in English Language Mathematics Classrooms.* URL: `https://eric.ed.gov/?id=EJ940646`.

Mefteh, Mariem and Roman Knöll (2012). “Ara_Pegasus: A new framework for programming using the Arabic natural language”. In:

Murphy, Gail and Christa Schwanninger (Jan. 2006). “Aspect-Oriented Programming”. English. In: *IEEE Software* 23.1. Copyright - Copyright IEEE Computer Society Jan/Feb 2006; Document feature - references; photographs; illustrations; Last updated - 2013-05-01; CODEN - IESOEG, pp. 20–23. URL: `https://search-proquest-com.ezproxy.bcu.ac.uk/docview/215842786?accountid=10749`.

Pulido-Prieto, Oscar and Ulises Juarez-Martinez (Sept. 2017). “A Survey of Naturalistic Programming Technologies”. In: *ACM Comput. Surv.* 50.5, 70:1–70:35. ISSN: 0360-0300. DOI: `10.1145/3109481`. URL: `http://doi.acm.org.ezproxy.bcu.ac.uk/10.1145/3109481`.

Resnick, Mitchel, Yasmin Kafai, and John Maeda (2005). “A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities”. In:

RoboMind (n.d.). URL: `http://robomind.net/`.

Stack Overflow Developer Survey 2018 (n.d.). URL: `https://insights.stackoverflow.com/survey/2018/`.