Harnessing the Concept of an Array in Swift Programming Language. Abstract Concepts vs Natural Semantic Metalanguage

Bartłomiej Biegajło

Siedlce University of Natural Sciences and Humanities

Abstract

The contemporary digital world we all inhabit can have a twofold distinction; the majority of people participate in it in a passive manner; however, there is a growing number of IT enthusiasts who are keen to assume a more active role in the digitalized world society. The latter group of people want to take an active role not only in utilizing the digital inventions but, more importantly, in creating the digital world. This cannot be accomplished without developing a set of basic programming skills. To learn how to program applications requires perseverance and extended practice. This is especially problematic with regards to beginners who often feel overwhelmed by the number of keywords, programming patterns and various types of 'good practices' that they are advised to follow. Many of them quit prematurely as they suffer from the spreading preconception implying that programming is too exclusive. The intention of the following study is to test whether Natural Semantic Metalanguage could possibly help in more immediate accommodation of the newly acquired programming knowledge. Specifically, it is focused on developing an NSM explication of one of the core programming concepts, called an array, and possibly estimating the future potential behind such research.

Keywords: Natural Semantic Metalanguage, programming concepts, explications, an array, Swift programming language

Abstrakt

Cyfrowy świat w którym dzisiaj zamieszkuje każdy z nas można podzielić na dwie części: większość z nas zamieszkuje go w sposób pasywny, rośnie jednak liczba osób, które w stosunku do świata informatyki przyjmują rolę aktywnych uczestników. Ta druga grupa nie jest zainteresowana wyłącznie użytkowaniem wynalazków cyfrowych, ale chce brać udział we współtworzeniu cyfrowego świata. Odnalezienie się w tej roli nie może obejść się bez rozwinięcia umiejętności programowania. Nauka programowania wymaga wytrwałości i długotrwałej praktyki. Z uwagi na znaczną ilość specyficznych słów kluczy, potrzebę zrozumienia podstawowych praktyk designerskich, początkujący adepci programowania doświadczają poczucia przemożnego przytłoczenia ilością wstępnych informacji, które muszą przyswoić na początku swojej programistycznej przygody. Wielu z nich rezygnuje z dalszej edukacji sądząc, że programowanie jest dziedziną dostępną tylko dla wybranych osób. Niniejszy artykuł próbuje odpowiedzieć na pytanie, czy Naturalny metajęzyk semantyczny mógłby pomóc w przyswojeniu podstawowych wiadomości z dziedziny programowania. Niniejsza analiza skupia się na próbie zbudowania definicji pojęcia tablicy zmiennej (array) z wykorzystaniem dorobku Naturalnego metajęzyka semantycznego oraz oszacowania potencjału badań opartych na powyższych założeniach.

Słowa kluczowe: Naturalny metajęzyk semantyczny, koncepcje programistyczne, eksplikacje, array (tablica zmiennych), Swift (język programowania)

1. Why Learn to Code?

Why learn to code is the question posed in the opening chapter of one of Tariq Rashid's newest books, *Creative Coding for Kids* (2019). The author, being a recognized researcher in the realm of deep learning and artificial intelligence, makes a substantiated claim that "coding is considered by many to be as essential as reading and writing – Reading,

Writing, Coding" (Rashid 2019, 6). The growing importance, influence and presence of the digital universe in the contemporary world can be hardly disputed. Apart from seasoned programming professionals, who are in constant pursuit to push the technological boundaries to the territories never seen before and for whom learning new technologies and new coding skills is part and parcel of their everyday activity, there is also growing awareness among educationalists for the need to develop professional coding courses dedicated to educating the newcomers to the field. There seems to exist a shared and grounded agreement in the education sector that early exposure to the nuts and bolts of technology is a prerequisite for a successful entering into the modern market:

> Many education curricula have been updated to ensure that children are digitally literate, equipped to participate in a digital economy, able to develop their own technology ideas, and be better informed consumers and citizens. (Rashid 2019, 6)

The recent proliferation of various kinds of coding camps (both online and onsite), coding meet-ups, IT conferences as well as wide availability of a plethora of online programming courses clearly testify to the fact that learning how to code is not merely a temporary fad but has become a skill which is recognized and urgently sought-after.

Marina Umaschi Bers and Mitchel Resnick, the latter being the architect of a highly successful and extremely well-received visual programming language explicitly devised for children called Scratch, affirm that "coding is often seen as difficult or exclusive, but we see it as a new type of literacy – a skill that should be accessible for everyone" (2016, 2). Both claims cause no major controversy. Niklaus Wirth, a Swiss pioneering computer scientist who designed a number of programming languages, including the famous Pascal language, suggests that "the key to successful programming is finding the 'right' structure of data and program" (1980, 1). This is the opening line of Wirth's discussion of the programming concept of a module which is further defined along the following lines: "the module concept is oriented towards the need of separating large programs into partitions with relatively 'thin' connections" (1980, 9). Striving for lucidity and best possible precision, Firth's explanation is already vague enough to put off potential programming enthusiasts from entering the already steep learning curve that programming inevitably brings. As these types of definitions are not exclusive to the programming world only, it seems true that they do contribute to the spread of the idea that coding is indeed difficult and exclusive. At the same time, Rashid optimistically asserts that "coding is also fun and creative. Many people do it just for pleasure" (2019, 6) which

correctly implies that the inherent creativeness lying behind the very skill combined with a playful nature of the activity should indeed 'be accessible for everyone'.

Learning to code in the 21st century has become based more on the visual content where a dialogue between programming concepts taking place 'under the hood' is largely invisible to the user. The already mentioned example of Scratch is one such example. Recently, Apple company has also been attentive to the need of educating beginner programmers via a user-friendly interface. The world-famous Californian corporation launched what its designers call Playgrounds – an app containing a selection of cartoon characters which are expected to accomplish a number of different tasks through specific instructions supplied by a learner. Undeniably, both Scratch and Playgrounds offer an extremely attractive user experience especially to the younger students; however, these approaches will ultimately prove to be a half step on a journey to become a fully-fledged programmer able to compete on the market.

Rashid observes that "coding and algorithmic thinking are important life skills in the increasingly digital world we live in" (2019, 6) and in order to develop comprehensive expertise in the field, it is crucial to understand the mechanisms governing the behaviour of the events happening on a device's screen. The extended and regular practice seems to be the key in switching to the career of a professional programmer; however, all things being equal, it has to have a starting point. Definitions of basic programming concepts further explained with the support of real-life examples, are conventionally a place to begin developing the skills in question. To avoid the tag of 'exclusiveness', explaining programming concepts should seek clarity and conciseness and anyone who has got one's feet wet with learning how to code will corroborate the view that programming is hardly a straight road to success. Bestselling manuals for popular programming languages are packed with jargon words which most beginners find overwhelmingly too complex to grasp. It is frequently a straight road to giving up any further attempts at exploring the intriguing world of digital experience. This paper aims to propose a different stepping-stone to a programming experience, which is conventionally still seated in language-based definitions, unsupported by a visual content characteristic to Scratch or Playground, yet framed in a language that is jargon-free and culture-independent. This paper is, therefore, an attempt to use Natural Semantic Metalanguage to harness the meaning of what is called an array and is an example of a Collection Type in Apple's Swift programming language.

2. Methodology

The methodology behind Natural Semantic Metalanguage has been discussed in a number of publications for the past four decades. Its leading proponents are two linguists, Anna Wierzbicka and Cliff Goddard. However, NSM (short for Natural Semantic Metalanguage), having received growing attention, is now also being studied/referred to, or was studied/referred to by scholars who are not/were not immediately engaged in linguistic research (e.g. Christian 2018; D'Andrade 2001; Fabrega 2012; Harré 2012; Hryniewicz 2012; Peeters and Marini 2018; Shweder 2004 among others). Wierzbicka observes that "some of the scholars [...] refer explicitly to the NSM program and its practitioners, while others don't" (2014, 197) and she recognizes and accepts the fact that "needless to say, they don't all agree on every point with either myself and my colleagues, or with each other" (2014, 197). Notwithstanding these reservations, the prospect of NSM's core tenets being keenly adopted in thinking about cultures and languages across various academic curricula is a unique phenomenon.

The broadest scope of the research agenda employed within the NSM framework can be summarized as follows:

> The notion of core or basic vocabulary depends on the idea that some meanings are simpler than others; and that the simpler meanings are necessary and useful in order to explain and to grasp more complex meanings. (Goddard and Wierzbicka 2007, 105)

The idea revolving around the notion of 'simpler meanings' addresses one of the most fundamental problems humanity has been facing since St. Jerome started to be called 'the father of translation studies' and is centred on flawed attempts to communicate with precision and clarity across all natural languages. Wierzbicka's own research is, in fact, an attempt to bridge this gap, as she herself admits: "what we need for real 'human understanding' is to find terms which would be both 'theirs' and 'ours'. We need to find *shared* terms, that is, universal human concepts" (1992, 27). This has to take us to the core theoretical assumptions endorsed by NSM: "as I have tried to demonstrate for a quarter of a century, the key to a rigorous and yet insightful talk about meaning lies in the notion of semantic primitives (or semantic primes)" (Wierzbicka 1996, 9), otherwise also labelled as semantic universals. Below is a list of 65 primes which, according to the theory, is an exhaustive and now closed number of concepts:

I, YOU, SOMEONE, SOMETHING~THING, PEOPLE, BODY	Substantives
KIND, PART~HAVE PARTS	Relational substantives
THIS, THE SAME, OTHER~ELSE	Determiners
ONE, TWO, SOME, ALL, MUCH~MANY, LITTLE~FEW	Quantifiers
GOOD, BAD	Evaluators
BIG, SMALL	Descriptors
KNOW, THINK, WANT, DON'T WANT, FEEL, SEE, HEAR	Mental predicates
SAY, WORDS, TRUE	Speech
DO, HAPPEN, MOVE	Actions, events, movement,
BE (SOMEWHERE), THERE IS	Location,
BE (SOMEONE/SOMETHING)	existence, specification
(IS) MINE	Possession
LIVE, DIE	Life and death
TIME~WHEN, NOW, BEFORE, AFTER, A LONG TIME, A SHORT TIME, FOR SOME TIME, MOMENT	Time
PLACE~WHERE, HERE, ABOVE, BELOW, FAR, NEAR, SIDE, INSIDE, TOUCH	Space
NOT, MAYBE, CAN, BECAUSE, IF	Logical concepts
VERY, MORE	Augmentor, intensifier
LIKE	Similarity

(Goddard 2018, 63)

For readability purposes, the primes have been grouped under specific headings which greatly eases navigation through the congested list above. Also, a brief explanation needs to be provided to account for what NSM arbitrarily assigns the name of allolexy: "the term *allolexy* refers to the fact that the same element of meaning may be expressed in a language in two or more different ways (Wierzbicka 1997, 28; original emphasis). In the list above, allolexes are highlighted with a tilde, and there are seven such instances: SOMETHING~THING, PART~HAVE PARTS, OTHER~ELSE, MUCH~MANY, LITTLE~FEW, TIME~WHEN and PLACE~WHERE. Terminologically, in this particular respect NSM borrows from studies in morphology and "by analogy with 'allomorphs' and 'allophones', such different exponents of the same primitive are called "allolexes" in NSM theory" (Wierzbicka 1997, 28). Every single prime indexed in the list above is thought to have an exact equivalent in any world language. In other words, all of

the primes are translatable into other languages without a loss in meaning which is why it is legitimate to claim that every single component of their meaning is reflected in any given language of the world.

This cursory overview of NSM is by no means comprehensive; however, it does introduce the core principles that govern its theoretical underpinnings. Additionally, it also implies how to effectively depart from theory in search of a more practical approach towards meaning which NSM evidently promotes. An in-depth examination of NSM carried out from the theoretical perspective, as well as the history of NSM can be found in a number of papers and books, especially those authored by the leading proponents of the theory (e.g. particularly Goddard 2010; Goddard 2018; Wierzbicka 2003 among many others).

The following section will try to compose a preliminary definition (NSM prefers to talk about explications rather than definitions) of an array – an example of one of the Collection Types used in Apple's Swift programming language. It has to be noted, however, that all major programming languages tend to apply the same, or closely related types of solutions to organize the storage of data. Storing data effectively and the extent to which a user is allowed to manipulate it is what differentiates the types of solutions supported by individual modern programming languages. What matters most is flexibility, clarity, and how fast a language can process vast amounts of data flowing through an app.

Choosing Swift's design of the concept of an array as a possible illustration of how NSM can fruitfully support a better understanding of core programming concepts, instead of a different programming language is conditioned by the fact that Swift's official documentation is extremely readable and therefore particularly easy to navigate. Hence, apart from being an obvious publicity stunt, Apple's own excitement at the range of Swift's capabilities might be truly contagious, especially to those who are new to programming:

> Swift is friendly to new programmers. It's an industrial-quality programming language that's as expressive and enjoyable as a scripting language. Writing Swift code in a playground lets you experiment with code and see the results immediately, without the overhead of building and running an app. (Apple 2019, 2)

Apple makes every effort to turn its newest invention into a widely accessible tool which is aimed to be both useful and friendly to all programming enthusiasts, regardless of their programming experience.

3. Explication of the concept of array

The key concept in commercial programming is creating usable apps that can respond to the user's actions. Any given app is essentially a collection of data that can be stored in various types of containers whose contents can be freely manipulated by a user. Modern programming languages, including Apple's Swift which, according to Apple, is "a new programming language for iOS, macOS, watchOS, and tvOS app development" (2019, 49), introduce a number of different types of such containers to help developers utilize the most basic, yet extremely powerful concepts that allow "to write software, whether it's for phones, desktops, servers, or anything else that runs code" (2019, 2). Apple proclaims that "Swift has been years in the making, and it continues to evolve with new features and capabilities" (2019, 3), and the following overview of Collection Types in Swift is carried out with reference to the latest version of Apple's new language.

An array is the first most basic example of a Collection Type offered by Swift. According to Apple, "arrays are ordered collection of values" (2019, 155). Additionally, a further extended description of the concept implies that "an *array* stores values of the same type in an ordered list" (2019, 156; original emphasis). These two preliminary observations might be transformed into the following opening NSM explication:

array of kind X:

(a) something
(b) there can be something inside it
(c) this something inside can be many things of kind X
(d) this something inside cannot be of any other kind than X
(e) all things inside can be in many places inside this something
(f) at one moment one thing inside is in one place inside this something, not in another place
(g) at one moment there can be one thing before this something inside, one thing after this something inside
(h) all things inside are like this

Component (a) is included for formal reasons – an array is not a human being; it is an inanimate object.

Component (b) points to the core feature of programming, i.e. its hardwired functionality that enables an app to store data, which programmers/developers traditionally refer to as values. It is values that travel between different 'containers', e.g. between two arrays that this component indicates. Additionally, component (b) is clear about the fact that an array does not necessarily always have to contain more than one value. Depending on the function an array is supposed to perform in a given context, its initial storage (a developer would be more precise and say: 'at the point of declaration') can be limited to just a single value, or no value whatsoever (an empty array) which can later be appended with more values with no upper limit as to the number of items that can be added. Thus, component (b) contains the crucial word 'can' ('there can be something inside it'), without being strict on the exact number of 'things' that an array can accept.

The two following components, component (c) and component (d), explicitly suggest the potential size (the word 'can' is meant to imply exactly this) of an array's storage capacity ('this something inside can be many things of kind X' and 'this something inside can not be of any other kind than X'). Additionally, both components contain a reference to the expected kind of values that a Swift's array can accept ('things of kind X'). Accompanied by the title of the explication ('array of kind X'), these pieces of information account for the preconditioned feature of an array, which is qualified to store only values of the same type. Swift provides "its own versions of all fundamental [...] types, including 'Int' for integers, 'Double' and 'Float' for floating-point values, 'Bool' for Boolean values, and 'String' for textual data" (2019, 49). In essence, an array cannot be seen to contain a mixture of dissimilar types of values, for example, a combination of integers and strings, as this would violate the original distinguishing principle with which an array is equipped.

Before commenting on the four closing components, a critical remark has to be made with reference to why an array is designed to store a list of values whose order, as a rule, must be fixed at a given point in time. When an array becomes a collection of values, i.e. more values are appended to its initial storage range, those new values receive a new index number, i.e. the next available number/spot under which a new value, from now on, is going to be identified with. The size of an array conditions the number of indices and, as there is no upper limit to the number of values that a Swift array can store, there is no upper limit as to the number of indices at the programmer's disposal. An array and the

indices it automatically creates as they grow exponentially in tandem with the number of values inserted into such an array are therefore two inseparable entities. It is especially important to fully understand the system of logic behind indexing (Swift seems to be committed to label it as 'subscript syntax') in the context of accessing or modifying an array. Apple's instructions for accessing data from within an array clearly accommodate the idea of indexing:

> Retrieve a value from the array by using *subscript syntax*, passing the index of the value you want to retrieve within square brackets immediately after the name of the array. (Apple 2019, 162; original emphasis)

These remarks are crucial to understanding why values stored in Swift arrays are consistently designed to be an ordered list. These remarks also implicitly suggest that there is, in fact, a lot more to discuss with regards to arrays. However, to include these aspects into the present discussion would have to call for a much more extensive investigation which is outside of the scope of this analysis.

In light of the above, components (e) ('all things inside can be in many places inside this something') and (f) ('at one moment one thing inside is in one place inside this something, not in another place') imply that an array is designed to store its contents under a specific index, which is, by necessity (hence 'one thing inside is in one place' and not 'one thing can be in one place', which would point to varying indices), a fixed type of index ('not in another place'). Additionally, a given value occupies a distinct place on an array's map only at a given point in time ('at one moment'), which translates into the execution of a programme/application, i.e. when a code is run. Another execution round behaves differently, e.g. a new value is appended to the existing list of values inside an array or, alternatively, the value which is already part of an array is removed from the list, thus changing the arrangement of indices. It is essential to identify this capacity in order to show that an array is open to data manipulation at which component (e) clearly hints ('all things inside can be in many places inside this something').

Component (g) recounts the core feature of Swift's arrays and suggests that a unique order governs the location of values stored inside an array ('there can be one thing before this something inside, one thing after this something inside'). It is also worth noting that an array can be initially empty and only filled with values at later stages of a programme's execution. This is indicated by the use of the word 'can' (at one moment there can be [...]') which highlights this potential design.

Eventually, the closing component ('all things inside are like this') signifies that the behaviour of values expressed in components (e), (f) and (g) is typical to all values of a specific type ('of kind X') stored in an array.

4. Concluding remarks

The problem with explicating programming concepts (see Biegajło 2019) is that programming concepts themselves build their contextual meaning upon one another. Familiarizing oneself with one type of concept means, more often than not, having to study another one (or two and more, in fact) in order to comprehend what one is doing with code fully. The same would have to apply for developing NSM explications. Their role is to provide a succinct explanation of the core tenets behind a selection of programming concepts. These explanations, by necessity, would have to be unequivocal, accessible immediately and, which might sound surprising, would have to require a potential reader to have been already exposed to programming to at least a minimal extent. Explications of programming concepts could then take the form of a collection of dictionary entries, arranged in a responsive manner, possibly made into an app and therefore being accessible just one click away. The bottom line is to create a learning environment for those who intend to study programming and can receive an instant prompt as to how to handle a given programming concept.

The explication of an array proposed here certainly makes no claim to be conclusive and is open to further amendments. The urgency of the NSM explications to be instantaneously comprehensive upon first reading, which this overview is primarily concerned with, sets special constraints upon the potential manner in which these explications should continue to evolve. Strict reliance on Natural Semantic Metalanguage and its necessarily curbing theoretical underpinnings seem to work well on a theoretical plane, however, its application would have to be reconsidered from the viewpoint of the communicative value it offers. As long as the separate elements of the explication of the concept of array help one grasp the basic significance of an array, it is its readability that poses the greatest challenge, particularly if one intends to see it working in a learning environment. The answer to these reservations might come from NSM itself – the so-called Minimal English (or any other natural world language qualified with an adjective 'minimal') has recently been proposed by Goddard and Wierzbicka (Goddard and Wierzbicka 2018). Although it is outside of the realm of this study, it certainly merits further investigation.

Works cited

Apple. 2019. *The Swift Programming Language (Swift 5.1)*. Cupertino, California: Apple Inc.

Biegajło, Bartłomiej. 2019. "Explaining IT Programming Concepts Using NSM Explications: The Case of 'variable' and 'constant'." *Linguistics Beyond and Within (LingBaW)* 5(1): 7–16.

Christian, David. 2018. "Big History Meets Minimal English." In *Minimal English for a Global World. Improved Communication Using Fewer Words*, edited by Cliff Goddard, 201–224. Cham, Switzerland: Palgrave Macmillan.

D'Andrade, Roy. 2001. "A Cognitivist's View of the Units Debate in Cultural Anthropology." *Cross Cultural Research* 35(2): 242–257.

Fabrega, Horacio. 2012. "Ethnomedical Implications of Wierzbicka's Theory and Method. Emotion Review." *Emotion Review* 4(3): 318–319.

Goddard, Cliff. 2010. "The Natural Semantic Metalanguage Approach." In *The Oxford Handbook of Linguistic Analysis*, edited by Bernd Heine and Heiko Narrog, 459–484. Oxford: Oxford University Press.

Goddard, Cliff. 2018. "Minimal English: The Science Behind It." In *Minimal English for a Global World. Improved Communication Using Fewer Words*, edited by Cliff Goddard, 29–70. Cham, Switzerland: Palgrave Macmillan.

Goddard, Cliff, and Anna Wierzbicka. 2007. "Semantic Primes and Cultural Scripts in Language Learning and Intercultural Communication." In *Applied Cultural Linguistics*, edited by Farzad Sharifian and Gary B. Palmer, 105–124. Amsterdam/Philadelphia: John Benjamins Publishing Company.

Goddard, Cliff, and Anna Wierzbicka. 2018. "Minimal English and How It Can Add to Global English." In *Minimal English for a Global World. Improved Communication Using Fewer Words*, edited by Cliff Goddard, 5–27. Cham, Switzerland: Palgrave Macmillan.

Harré, Rom. 2012. "Methods of Research: Cultural/Discursive Psychology." In *Psychology for the Third Millennium: Integrating Cultural and Neuroscience Perspectives*, edited by Rom Harré and Fathali M. Moghaddam, 22–36. London: Sage.

Hryniewicz, Wacław. 2012. *God's Spirit in the World: Ecumenical and Cultural Essays*. Washington DC: Council for Research in Values and Philosophy.

Peeters, Bert, and Maria Giulia Marini. 2018. "Narrative Medicine Across Languages and Cultures: Using Minimal English for Increased Comparability of Patients'

Narratives." In *Minimal English for a Global World. Improved Communication Using Fewer Words*, edited by Cliff Goddard, 259–286. Cham, Switzerland: Palgrave Macmillan.

Rashid, Tariq. 2019. *Creative Coding for Kids*. Scotts Valley, California: Create Space Independent Publishing Platform.

Shweder, Richard A. 2004. "Deconstructing the Emotions for the Sake of Comparative Research." In *Feelings and Emotions: The Amsterdam Symposium*, edited by Antony S.R. Manstead, Nico Frijda, and Agneta Fischer, 81–97. Cambridge: Cambridge University Press.

Umaschi Bers, Marina, and Mitchel Resnick. 2016. *The Official ScratchJr Book*. San Francisco: No Starch Press.

Wierzbicka, Anna. 1992. *Semantics, Culture, and Cognition. Universal Human Concepts in Culture-Specific Configurations*. Oxford: Oxford University Press.

Wierzbicka, Anna. 1996. *Semantics. Primes and Universals*. Oxford: Oxford University Press.

Wierzbicka, Anna. 1997. *Understanding Cultures Through Their Key Words*. Oxford: Oxford University Press.

Wierzbicka, Anna. 2003. *Cross-Cultural Pragmatics: The Semantics of Human Interaction*. Berlin/New York: Mouton de Gruyter.

Wierzbicka, Anna. 2014. *Imprisoned in English. The Hazards of English as a Default Language*. Oxford: Oxford University Press.

Wirth, Niklaus. 1980. "The Module: A System Structuring Faculty in High-Level Programming Languages." In *Lecture Notes in Computer Science. Language Design and Programming Methodology*, edited by Jeffrey M. Tobias, 1–24. Berlin/Heidelberg/New York: Springer Verlag.