Evaluating Emerging Programming Paradigms: An Artifact-Oriented Approach

H. James Nelson (jnelson@ukans.edu)
The University of Kansas, Division of Accounting and Information Systems
David E. Monarchi (david.monarchi@colorado.edu)
University of Colorado, Graduate School of Business
Kay M. Nelson (knelson@ukans.edu)
The University of Kansas, Division of Accounting and Information Systems

Abstract

A number of programming paradigms have risen and fallen in popularity since the invention of the computer. Many of these have been accompanied by claims that they are more natural ways to program. Little evidence has been offered to justify these claims. One possible way of providing this evidence is to compare programming paradigms to a standard of “naturalness.” In this paper, we compare the structured and object-oriented programming paradigms to such a standard: the “cognitive paradigm” that supplies the general-purpose cognitive tools for one’s daily life. To do so, we propose a methodology that draws upon techniques used in the fields of anthropology, archaeology, and comparative linguistics to compare paradigms indirectly through their artifacts. The results of the comparison indicate that the object-oriented programming paradigm can be considered to be more natural than the structured programming paradigm. Further, it indicates that the method of comparing and exploring programming “cultures” though their artifacts opens a number of interesting possibilities.

1. Introduction

We have all heard the claims from software vendors, consulting houses, and industry pundits about the latest and greatest technology aimed at solving the “software crisis.” These claims are very enticing. Information systems managers are critically concerned with getting programmers up to speed quickly on new technologies, getting more productivity out of fewer people, and improving the quality of their software [18]. One of the more recent technologies, the object oriented (OO) programming paradigm promises to do all this and more. OO analysis and design is supposed to be a more natural way to produce software, and this translates to faster learning, faster development, and higher quality. However, OO’s claims of “natural” are largely unsupported.

To be natural, the tool must fit the person using it, whether it is a physical tool which fits the hand or a cognitive tool that fits the mind. Assessing the “fit” of a programming paradigm to a mind is a much more difficult process than testing the fit of a hammer to a hand. A paradigm is essentially a way of thinking. It supports deeper cognitive processes by supplying the conceptual and physical tools for solving various kinds of problems [11, 12]. The fit that must be tested is between the paradigm and the mind; a difficult prospect since neither can be directly observed.

We propose an indirect method of examining emerging programming paradigms. Much can be learned from various societies and cultures by examining the evidence and artifacts that they leave behind. This paper draws upon techniques used in the fields of anthropology, archaeology, and comparative linguistics, and others to compare paradigms indirectly through their artifacts. We begin by discussing the some of the problems of examining programming paradigms. We then move on to a description of the methodology and proceed to use it to compare the structured and object-oriented programming paradigms to the cognitive paradigm in the area of language. We conclude with the results of the comparison, and some implications about the process we used and the results we found.

2. Comparisons

There are several methods available for examining programming paradigms. One method is to put the paradigm into practice and observe the long-term effects on productivity. A better-fitting paradigm should result in higher productivity. This method relies on the day-to-day real world testing of tools and techniques by industry [1]. But there are problems when using information from industry. A new paradigm generally takes quite some time to be accepted. Many large programming projects need to be completed and the results analyzed before there is any conclusive evidence of whether or not the technology works. Anecdotes, war stories, and publications take a long time to circulate throughout
industry. Many of the best examples are proprietary and will never be disseminated.

Another method is to perform experiments and observe how well experts or novices learn and use the new paradigm [24]. Although experiments can be useful, their often limited scope can cause problems. Considerable time and knowledge may be necessary for someone to gain proficiency with the new tools and the new mindset. Experiments may only test how well someone learns a specific tool.

A third method is to study the paradigm itself rather than any particular tool associated with it. In doing so, we can compare the paradigm in question with a standard of “naturalness.” If a programming paradigm is found to be fairly close to this standard in a number of important areas, then the tools and techniques it supplies could be said to be relatively natural to use. The standard we will use will be the “cognitive paradigm” that supplies the general purpose tools for one’s daily life. This brings us back to the original problem: how to compare things that cannot be directly observed.

3. Methodology

A paradigm provides the framework for the organization of knowledge which “can guide a scientist into the unknown, telling him where to look and what he may expect to find” [11]. In fact, a particular paradigm so colors a person’s perceptions that the fields of metaphysics and ontology have evolved to explore the nature of reality. A paradigm not only affects the way a problem is perceived, but also the way a problem is solved. It supplies the tools such as laws, theories, applications, and instrumentation needed to solve the problem [12].

Figure 1 shows a possible model for paradigmatic problem solving. The mind is supported by the general-purpose cognitive paradigm. A problem in some domain is recognized and processed by the paradigm most appropriate for that domain. Figure 2 shows an example of two different paradigms applied to the problem of getting food. Two people from different societies see this problem in different ways. The first person is from a society of hunter-gatherers. This person sees the problem of getting food and applies the hunter-gatherer societal tools (spear and horse) to solve it. The second person sees the same problem and applies the agrarian societal tools (plough, oxen) to get food.

The paradigms discussed so far cannot be directly observed or compared, but they can be indirectly compared through artifacts uniquely associated with them. Using artifacts to make inferences about objects or phenomena is nothing new; it is basic to the scientific method. A person’s attitude can be inferred by a score on some profile; a company’s success can be inferred by its profits. These inferences, however, only just begin to get at the nature of an object or phenomenon. They don’t answer questions such as “How is the diffusion of technology in company A different from company B?” or “How is programming paradigm X different from programming paradigm Y?”

Answering these questions requires much deeper inferences. The question is not so much “What is the artifact?”, but rather “What does this artifact reveal about
the nature of the thing that produced it?” This requires some care. Artifacts must be chosen which can be uniquely identified with the phenomena being studied. Once found, they can be used to describe and compare the underlying processes that produced them.

Choosing these artifacts, making inferences, and making these comparisons has a long tradition in many of the sciences that perform research in human culture and prehistory. Archaeologists use this technique when making inferences about ancient societies. Certain artifacts such as pointed sticks, spears, knives, and scrapers are associated with hunter-gatherer societies. Other artifacts such as tools for digging plants, trapping wild fowl, as well as tools for preparing and cooking this kind of food are associated with agrarian societies [27] (see Figure 3).

Choosing these artifacts, making inferences, and making these comparisons has a long tradition in many of the sciences that perform research in human culture and prehistory. Archaeologists use this technique when making inferences about ancient societies. Certain artifacts such as pointed sticks, spears, knives, and scrapers are associated with hunter-gatherer societies. Other artifacts such as tools for digging plants, trapping wild fowl, as well as tools for preparing and cooking this kind of food are associated with agrarian societies [27] (see Figure 3).

![Figure 3. Two societies and their representative tools.](image)

Although the same basic technology is used for creating all of these tools, the society dictates which set of tools is created and used. Most prehistorians depend on this to identify ancient civilizations and their various cultural traditions. Their research would not be possible without the assumption that a culture is associated with a kind of tool, and that a specific kind of tool is representative of a particular culture.

Physical tools and their use can give an indication of the broad paradigm a society is operating under, but cannot necessarily identify the cognitive processes of the user. Artifacts such as language and problem-solving approaches give researchers insights into these cognitive processes. Researchers in the fields of sociolinguistics and social psychology use language to categorize and compare members of various social, ethnic, educational, and other groups [29]. The field of intercultural communication uses personal interactions to determine the values, social structure, interaction rules, and politics among various peoples. Anthropology uses a similar methodology to compare one society to another. Comparative anthropologists identify and highlight specific items within a culture assuming that social facts are constructed of these items [8].

It should be noted that not all artifacts are appropriate for making deep inferences about a society or paradigm. Care must be taken to ensure that the artifact chosen is more generally associated with one paradigm than another. This does not mean that a tool cannot be used by two different paradigms. A large bent stick with a sharpened side may be used for hunting but it is better employed as a plough. Likewise, bows and arrows make poor digging sticks but are good for hunting. In programming, data flow diagrams, entity relationship models, and functional decomposition techniques have all been adapted to the object paradigm [17]. For example, while data flow diagrams can be used in any paradigm, they were developed for, and are generally associated with, the structured programming paradigm.

In addition to the cultures already mentioned, there are also “cultures” of programmers. These cultures can be said to be following the many paradigms of programming such as structured programming, functional programming, and object-oriented programming. Within each of these cultures are the various camps of programming language advocates: C vs. COBOL, Smalltalk vs. C++, and so on. The languages in these camps are different but they still follow the mindset of their respective cultures.

Since language is perhaps the most distinctive part of being human, it will be the basis for our comparisons. We will compare the programming languages of the structured and object paradigms to the natural language of the cognitive paradigm. We are not interested in direct comparisons, but rather what the languages tell us about the nature of the underlying paradigm. The programming paradigm that most closely matches the cognitive paradigm should be more naturally and more easily supported by it and the mind (see Figure 4).

![Figure 4: Paradigms and interfaces to the mind.](image)
4. Language

Language is perhaps the most visible tool of the cognitive and programming paradigms, whether it is a natural language used for normal human interaction or a programming language for directing a computer’s operation and communicating with other people. Philosophers, psychologists, and linguists generally agree that language shapes our perception of reality, our processes of thinking and remembering, and our culture [4, 28]. Programming languages not only control computers, but they are also the primary means of notation and communication within a programming paradigm. “As concepts are discovered, explored, and become woven into the fabric of the science, they invariably find expression in programming languages” [30, p. 14]. Language, then, rather directly reflects the underlying paradigm.

Under the culturally-defined natural language, there is another layer of language which is universal across all cultures and people. This layer may be responsible for the general similarity of natural languages at many levels: the basic knowledge representation level [16], the general classification structures [21], the construction of the clause with subject / object / verb in varying arrangements [3], and the general structure of discourse [5]. Natural languages sit on top of, and are driven by, this universal linguistic layer.

Computer languages, on the other hand, were primarily developed for their ability to efficiently control a computer’s processing rather than as a vehicle for human communication. Although universal themes exist within each programming paradigm’s languages, there may be little in common between these themes and the universal aspects of natural language. It is this commonality, or lack thereof, that we will explore. We will now look at these language themes from three points of view: initial language acquisition, language production and comprehension, and discourse.

4.1. Language Acquisition

A child’s first words represents a significant developmental milestone. Where those first few words came from, and what happens next, has been the focus of considerable study. How a person learns a language, through mastery in some areas and mistakes in others, is an indication of the person’s natural abilities. The order and manner in which the language is taught is an indication of the most important aspects of the language from the instructor’s point of view.

Natural language is typically learned at an early age through interaction with people who know how to use the language well. Programming languages are learned later in life through books, instructors, and interaction with computers. This section will compare language instruction and acquisition of natural and programming languages. This should give an indication of how “natural” it is to learn a particular programming paradigm.

4.1.1. The Natural Language

Initially language acquisition in young children is very slow. They add words gradually, a few at a time, over an extended period [7]. During this time children are observing and connecting sound patterns, their future words, with the objects and events with which they occur most frequently. These word - object/event associations are not stored haphazardly, but rather by their semantic category: object, place, time, event, and so on, through some innate ability [2]. Even with their limited vocabulary, children are able to select a reasonable word from the proper category: “doggy” for small furry animals, “go” for motion.

Until children are around 16 months old, there are no relationships between any of the words; their language consists of single words or words and gestures. At around 50 words, however, children seem to form relationships between their words and lexical growth accelerates [7]. Words are added at a much faster rate and their first sentences appear. Syntax is added to the underlying semantics.

This stage of early speech reveals a great deal about the underlying framework of words and their relationships [2, 20]. Children are able to identify objects and their attributes such as size, shape, and color. Children are also able to identify an object’s position and motion relative to the observer as well as its causal roles. Semantic hierarchies appear with relationships among objects such as dog is a kind of animal. Objects are also associated with particular kinds of actions and behavior: water can splash, a rock cannot. These associations form the underlying semantic structure for language and thought.

Samples of early speech in children indicates that sentence production is initially determined by the subject of the sentence. In the earliest speech, a single word had to carry the meaning of the communication. With the availability of syntax, that word becomes the subject of the sentence. The subject (noun) drives the semantics which selects the most appropriate action (verb) and object (another noun) to describe the event. Once selected, however, the verb drives the syntax. Its attributes determine the position and form of the subject and object.

4.1.2. The Object Paradigm

Learning a new programming paradigm is similar to learning one’s initial natural language. In a natural language a cultural paradigm is learned at the same time
as the language itself. This cultural paradigm is most often known as “social skills.” In a programming language similar social skills are also brought along as the language is learned. How the language is taught and the mistakes learners make also reveal much about the inner workings of the paradigm. For this comparison we will turn to teaching suggestions from McKim [14], LaLonde and Pugh [13], and Rosson and Carroll [23].

According to these authors, teaching the object paradigm should begin with the basic object concepts such as classes, objects, inheritance, and so on. The first few assignments introduce the student to the semantics of the object paradigm through explorations of the class hierarchy, and then some specific objects along with the object’s general behavior and relationships to other objects. Then, depending upon the experience level of the student, algorithms, data structures, and the actual code are introduced.

Rosson and Carroll have an explicit “semantics then syntax” approach. The student “spirals” through examples being given only the objects, relations, and messages between them before being given the actual source code. They do not “drill the students on the rules of Smalltalk” before going through their examples, and leave details of syntax to be acquired “through exposure to other examples” [23, p. 79].

4.1.3. The Structured Paradigm

Much more is known about teaching structured programming and the problems of novices compared to the object paradigm. Structured programming instruction tends to be primarily from the bottom up. Expert teachers, consultants, and programmers recommend starting with the primitives or features of the language, then moving to larger and larger sequences of code [25]. The sequence concludes with general design and problem-solving skills such as functional decomposition and stepwise refinement. No rationale for this sequence is given. It may be because that’s how they learned it, or because formal proofs of program correctness begin with the primitive elements.

Given this instructional sequence, Spohrer and Soloway’s [26] results are not surprising. They indicate that less than 10 percent of novice mistakes are directly attributable to misunderstandings about basic constructs. The rest of the mistakes are related to errors in understanding the structure of the larger problem. It appears that syntax is the primary focus of structured programming instruction. Semantics, the “big picture” of the program’s function and the domain of the application, comes later through experience.

4.1.4. Conclusions

Both the object and cognitive paradigm have the semantics of the language as the driving force with syntax following later. There is nothing in the structured paradigm which prohibits teaching the semantics of the language first. Research indicates that it may be a good idea. The structured instructional mindset, however, tends to favor the bottom-up approach. So from the viewpoint of language acquisition, the object paradigm rather than the structured paradigm is more similar to the cognitive paradigm.

4.2. Language Production and Comprehension

Language production and comprehension begins with a thought, or the “meaning” of what is to be said. This thought is combined with the semantics and syntax of the language, is molded by the restrictions of the society’s paradigm, and is constrained by the physical limitations of the brain and speech mechanisms to create the words, phrases, and clauses that make up a meaningful sentence. This section will look at language production and comprehension at a micro level, from the structures needed for generating a thought to the organization of words in a sentence in both natural and programming languages. This should reveal how “natural” it is to think and “speak” using a particular programming paradigm.

4.2.1. The Natural Language

Although there has been some controversy about the organization of lexical memory, most psychologists now agree that it is based on a hierarchical organization of nouns [15]. The branches of the hierarchy vary in length, but seldom go to more than ten levels. Miller and Fellbaum [15] suggest this may be due to information being stored in the structure redundantly in addition to being an inheritance structure. For example, a canary is a bird which is small, yellow, and flies. The canary inherits the attributes of birds such as a beak, wings, and feathers and adds the attributes small, yellow, and flies. If beak is inherited often enough, say by someone caring for a canary with a broken beak, that information may be stored redundantly with the other attributes of a canary.

Although verbs are perhaps the most important syntactic category of any language, since every complete sentence must contain at least one verb, the form of the verb lexical memory is much less clear than that for the noun. Where the noun structure is relatively deep, the verb structure seems to be rather shallow. The properties or attributes of verbs appear to be syntactic instructions determining the type and form of their arguments.

A sentence starts with a theme, an idea or thought, that a person wishes to communicate to another. The theme determines the subject and object for the phrase. The subject determines the proper verb to convey the action. For example, for approximately the same kind of motion, a balloon floats and a bird flies. At this point semantics
ends and syntax begins with the verb determining the form and position of the noun phrases within the sentence.

4.2.2. The Object Paradigm
The lexical structure of an object system is also based on the hierarchy. This hierarchy is composed of things representing concrete or abstract entities. Each entity has a number of attributes and/or a behavior which distinguishes it from other entities, and inherits attributes from higher in the hierarchy. Behavior is tightly bound to the entities; there is no separate structure for the “verbs” of the language.

The production of a “sentence” in the object paradigm begins with a model of the problem to be solved based on the conceptual entities of the problem [22]. The focus is on the objects, their interaction with other objects, and their behavior. Once the objects of the problem have been established, the task to be performed is considered. The objects determine the behavior, then the behavior determines the structure and flow of the “sentences” and program.

4.2.3. The Structured Paradigm
The lexical structure of a structured program is wider and bushier than it is deep. It is primarily based on the actions to be performed and the flow of data. For example, a generic square-root function may exist with specialized functions defined below it, say for integer, floating point, or complex square-roots. These may be grouped under general “mathematical functions,” and finally under “functions,” but rarely would the structure go very deep.

A “sentence” in the structured paradigm also begins with a model of the problem to be solved, but it is based on the flow and transformation of data rather than the data itself. The problem is decomposed by function, for example: input, process, and output functions. At the lowest levels the appropriate data transformation functions are selected from the lexicon. The flow of data determines the objects and the syntax in one step.

4.2.4. Conclusions
A sentence in the cognitive, object, or structured paradigms begins with a thought to be communicated to an outside entity. Within both the cognitive and object paradigms, semantic sentence production is based on the noun which drives the selection of the verb. The verb then determines the syntax. The structured paradigm semantics is based on the verb which also drives the syntax. It appears that the object paradigm is more similar to the cognitive paradigm in the semantic structures and initial sentence production. In the end, though, each paradigm uses the verb to determine the final form of the sentence.

4.3. Discourse
Language production and comprehension involves relatively small units -- clauses, phrases, and sentences. A discourse is a much larger structure composed of many sentences related to a single topic or theme. In a discourse, the speaker has much more information to send to the listener than can be contained in a single sentence. The speaker must take this information, arrange it in a logical sequence, break it into sentences, and finally form it into sounds. The listener reassembles the information in a complementary process.

A computer program, application, or system is very similar to discourse. It has a central theme, a logical structure, and component parts arranged in some sequence. This section will compare the production and general structure of a discourse to the production and general structure of a computer program. This should give an indication of the programming language’s
communicative abilities, and how “natural” it is to think in terms of that programming language.

4.3.1. The Natural Language

Kintsch et al. [10] asserts that natural language discourse takes on three levels of representation as it travels from the speaker’s mind to the external sound or text. The first level is the situation model (or mental model) where the macrostructure of the discourse is first organized. The second level is the propositional representation where the situation model is broken into discrete units, and the third level is the surface trace of the actual sentence structure. We are interested in the structure of the situation model, but primarily how it is transformed into propositions.

The structures of knowledge representation, the “world knowledge” of people, are too inflexible to react quickly enough to the constantly changing themes and situations of discourse [9]. Instead, a subset of this knowledge is “gathered up” by the context of the situation and interconnected as an associative net. This is the situation model of discourse. The nodes of the net are the propositions or concepts of the discourse. The connections represent the relationship between a particular node and another such as attribute, behavior, or argument relationships.

The discourse is grounded in a topic. “The overwhelming norm in verbally coded human discourse is to make perceptually salient, temporally stable entities -- coded grammatically as nouns” the grounding for the discourse [6, p. 9]. These entities are then strung together in a temporal-linear structure which finally emerges as the narrative. Thus in the short narrative “Bobby went to the store. He got some cookies.”, Bobby serves as the grounding, providing stable context for resolving he. The temporal-linear format serves as an organizer for the serial nature of human speech.

4.3.2. The Object Paradigm

The structure of any single program or entire system must also necessarily begin as a mental model of the problem to be solved. From that point to the production of the code, the paradigm determines the structure. In the object paradigm, the problem is mapped onto “computational entities,” which is “an abstraction bundling state and behavior,” in other words, an object [22].

These objects then serve as the grounding for the creation of the “object space” containing objects and their interconnections through their attributes and communication patterns. The thread of execution through these communication patterns adds the temporal dimension to the object space. This thread of execution is then what determines an application in the object paradigm.

4.3.3. The Structured Paradigm

In her studies of procedural programmers, Pennington [19] proposed that the situation model is based on real world objects and events. The proposition level, however, “may be dominated by procedural relations that largely reflect how programs in traditional languages are structured” [19, p. 102]. The proposition level, which is concerned with searches, merges, and computations, must be cross-referenced with the situation model before a program can be effectively understood or produced.

4.3.4. Conclusions

Although the structured program begins as real world objects and entities in a mental model, it is transformed into a procedural representation in the course of the discourse. The object program and the cognitive discourse also starts as a model structured around real world entities, but they maintain this structure as the grounding for the discourse. It appears that the object paradigm is closer to the cognitive paradigm even beyond the surface production of sentences into the deeper structures of discourse organization.

5. Summary

In 1980, when structured programming was the technique of choice, and object-oriented programming had not yet gained popularity, William Wulf wrote:

> There is an implicit Whorfian hypothesis that the nature of language shapes the ways in which we think about problems. Thus, although we may not be able to measure it directly, most experts believe that the user of one of the more modern, structured languages is better equipped to think about complex problems than the user of one of the older languages (e.g., Fortran) [30, p. 14].

How does OO programming compare to structured programming, at least within the domain of language? We summarize our findings in the table below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Natural Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquisition</td>
<td>Semantics then syntax</td>
</tr>
<tr>
<td>Production and</td>
<td>Semantics based on noun</td>
</tr>
<tr>
<td>Comprehension</td>
<td>Noun determines verb</td>
</tr>
<tr>
<td>Discourse</td>
<td>Verb determines syntax</td>
</tr>
<tr>
<td><strong>OO Programming</strong></td>
<td><strong>Noun based mental model</strong></td>
</tr>
<tr>
<td>Acquisition</td>
<td>Semantics then syntax</td>
</tr>
<tr>
<td>Production and</td>
<td>Semantics based on noun</td>
</tr>
</tbody>
</table>
6. Conclusions

We used a methodology amalgamated from parts borrowed from anthropology, comparative linguistics, sociology, and others to explore and compare three otherwise incomparable things: the structured, the object-oriented, and the cognitive paradigms. This methodology compared these paradigms through their most visible artifact, language. The results of this comparison indicate that object-oriented programming has more in common with natural cognitive structures and processes than does structured programming. This would tend to support the claim that objects are “more natural.”

This conclusion, however, is limited to the paradigm as a whole and does not address any particular implementation. For example, there are many languages which are associated with the object paradigm. The conclusion that the object paradigm is natural does not necessarily indicate that, for example, Smalltalk, C++, or Eiffel are natural languages. Nor does it imply that these languages are equally effective or natural. It also does not mean that they are necessarily easy to learn or easy to use since they still have to conform to the limitations of the host computer system. The technology of the host computer may not allow the programming paradigm to be completely reflected in the implementation language. Although the paradigm itself may be easy to learn, the limitations of a specific language may diminish this advantage. Once learned, however, these languages may allow mental models of a program to be more directly implemented than languages of the structured paradigm.

7. Future Research

The research method used in this paper opens a number of interesting possibilities. A great deal can be learned about a society by studying the artifacts and tools that society uses. The same methods what are used to study ancient and modern civilizations can be used for deeper study into programming paradigms. For example, what do programs say about the society of the programmers who created them? What do the physical tools of programming such as mice, icons, CASE tools, and other techniques such as flowcharts say about the cognitive structures of programmers? This method has the potential to answer these and other interesting questions.

8. References


