# On the Role of Human Thought

## Towards A Categorial Concern Comprehension

Tim Frey, Veit Köppen
Otto-von-Guericke University
Magdeburg, Germany
tim.frey@tim-frey.com
veit.koeppen@iti.cs.uni-magdeburg.de

Marius Gelhausen, Hardo Sorgatz
TU Darmstadt
Darmstadt, Germany
marius.gelhausen@gmx.de
sorgatz@psychologie.tu-darmstadt.de

**ABSTRACT**
Separation of concerns comes from the inspiration how scientists study and understand problems. However, do humans always think scientifically? It is time to consider the human factor in software engineering. We show first indications that our minds do not structure information in an absolutely separated and sharp way. This is done by comparing concerns with research from psychology about categories. The ramification is: Research about categories needs to be considered for separation of concerns. Our goal is to show the emerging need for further investigations.

***Categories and Subject Descriptors*** D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.3.3 [**Programming Languages**]: Language Constructs and Features

***General Terms*** Human Factors

***Keywords*** Separation of Concerns, Categorization

## 1. INTRODUCTION

In 1982 Dijkstra formulated the principle of separation of concerns (SOC) [8]. The principle addresses the goal that modules should have a primary and only responsibility. This enables reducing complexity and the impact of changes. Ever since, software developers and researchers carry this principle with great exultation like a mantra in front of them. Though still, concerns in object-oriented programming languages, like Java, are intertwined and often not absolutely separated [9]. Developers face the problem of detecting concerns in a program [20]. Concerns can be used to encode them in their own modules and be composed together by the means of the programming language. Thereby, programming languages offer various means to separate and compose concerns [4].

Hence, a program comprehension model that respects the way of SOC is needed. Current models only consider the fact how code is understood, but not the way how concerns are encoded [6, 14, 22]. We want to improve the SOC concept and therefore consider the role of human thought in programming. The role of humans is considered explicitly, because the SOC principle is comes from the inspiration how scientists study problems. Scientists split problems into smaller ones and solve them [8]. Similar research in psychology about categorization uncovers that categories are fuzzy and not clearly separated [15, 18]. We assume that such categories equal concerns. If concerns and categories equal, SOC would not to be natural in every case for humans. We consider the assumption of their equality through a first comparison of SOC and categorization. Hence, our contribution is to improve program comprehension theories through a new aspect: We show first indications that categories in psychology and concerns in programming share similar characteristics.

The paper is organized as follows. First the background about the distinct areas is briefly presented. Then, we describe categorization theories and similar constructs in programs. A discussion presents possible arguments against our comparison. Afterwards, related research is described and conclusions are presented.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*FREECO-Onward! 2011,* October 23, 2011, Portland, Oregon, USA.
Copyright © 2011 ACM 978-1-4503-1025-3/11/10…$10.00.

## 2. BACKGROUND

Following, a brief introduction about the area of SOC and categorization in psychology is given.

### 2.1 Separation of concerns

One of the most important principles in software engineering is separation of concerns (SOC) [8]. Programming language designers have developed numerous mechanisms for SOC at the source code level. In this paper, we focus mainly on Metadata Annotations [3], classes, methods and Aspects [9].

Annotations are elements that can be used as meta-information at classes and their members, like fields or methods. Normally, this information is used to enrich annotated elements with certain capabilities. For example, such annotations are used to indicate persistency of various classes. Aspects are a programming concept that is used to enable coding of crosscutting concerns in their own modules. Crosscutting concerns represent functionality of a program that is not clearly belonging to one module and would be scattered over various modules if implemented in a normal object-oriented style. The Pointcut – Advice mechanism (PA) of aspect languages enables grouping of functionalities together. Pointcuts allow defining places where Advices, containing code functionality, are applied within a program. Similarly, other approaches are made to enhance SOC. The approach of Hyperslices/HyperJ is another application of

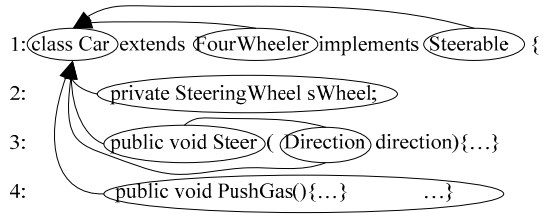
what is commonly called multi-dimensional SOC (MDSOC) [21].

### 2.2 Categorization

Categorization is an essential part of cognition, and is also fundamental for the process of comprehension. Categories play a central role in perception, learning, communication, and thinking [12]. By applying categorization, objects are grouped together into classes, based on similarities. These classes are called categories or concepts [15, 18]. Categories can be concrete objects like specific animals, but also immaterial entities like democracy [16].

The membership of elements, as part of a category, enables the possibility to use prior knowledge, about a category, for new objects, that belong to the same category. For instance, inferences on an unknown object can be done this way. Thereby, knowledge about other category members is used to derive features of a foreign object [27]. One of the most important features of categorization is prediction [26]. These may be necessary due limited available information, but also to avoid having to understand an object completely. It is too time consuming to comprehend every object totally. In this way it is possible to drive a car just because all cars are similar [2].

Categories are normally formed on basis of features of an object. In case of a car this can be: four wheels, a steering wheel, a gas pedal, a break, a car body, and so on. The categorization theories cover the way how a category is created in the mind based on features. We map this feature definition to vertices of a graph of a programming language. For instance, Listing 1 shows the class Car. The ellipses mark the code elements corresponding to the graph's vertices. The super class and interface, fields, methods, and their assignments represent features that are associated with this class. Even more, an object instance of this class would be a vertex. Field values of an instance would also be vertices. Programming languages allow to apply SOC and to group vertices together to another vertex. For instance, Direction can be a category itself but also a feature of another one. Thus, we need to think about the difference between a category and the members belonging to it.




**Listing 1: Sample source code**

Like described, an element can be a category itself or an element of a category. For instance, a pen is an object and a category label at the same time. We refer to a pen as a specific object and also as label for the category. In a shop, we search for the pen category. We buy a specific pen and refer to an element of the category. On the bill the category is listed, but it refers to an object. Circles that may not ever end can be discussed to discriminate between categories and elements of them. Generally, the thought defines the category, not the object itself. The knowledge that there are many pens, sharing similar features, creates the category. In the following, we present theories that are explaining the category creation thought and the association of elements with the category.

## 3. CATEGORIZATION OF CONCERNS

In this section, we present categorization theories and give examples of similar programming constructs. Afterwards, we present research about category types and cross-categorization.

### 3.1 Three different categorization theories

For classifying objects, three main theories exist how objects are categorized. The distinct theories were created, because some examples are not explainable through one main theory. First, the classic view tried to explain how a human categorizes. Then, the prototype theory was built and, finally, the exemplary view was created. Nowadays, hybrid theories claim that reality is consisting of a combination of theories [1, 13, 25]. We cannot clearly say which theory or which combination represents the whole truth. We can state that experiments support one or another theory. Hence, we present these different theories and compare their essence with example mechanisms to apply SOC

#### *3.1.1 The Classical View*

The classical view claims that categories are discrete entities characterized by a set of features which are shared by all of their members. Features are characteristics of a category member. Such characteristics can be physical entities but also more abstract concepts like activities. These features are assumed to establish conditions, which are necessary and sufficient, to capture the meaning of a category. All members of a category possess equal quality to the respective category [7]. In short: all members of a category need to have the same features.

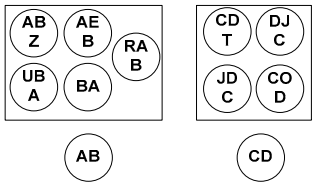



**Figure 1. Classic category view**

We show an example for the classic category view in Figure 1. Two categories, AB and CD, are shown in different squares. The circled elements are members of the corresponding categories and the letters within the circles represent different features of an element. All the different members share the same features (AB or CD), defining the categories. As described in Section 2.2, such features are the vertices of the program graph.

Typical examples for such categories in source code are classes. Objects created from a class share exactly the same features, just with different values. Thus, a class can be

considered as a classic category example. Another example is inheritance of various classes from a common super class or interface. In such a way all methods and fields are inherited and various extenders share the same feature set.

*3.1.2 The Prototype Theory*

The prototype theory claims that categories are represented by a bundle of features that are typical for a certain category, but not inevitable or sufficient [24]. A category "bird" has features, like "flying" or "building a nest". Even if not all birds got these, like for instance a penguin, they still belong to the bird category. The prototype of a category merges typical features of a category. No exemplar has to match the prototype completely. New objects are classified, out of an affinity with the prototype. In short: A category is defined by a kind of family resemblance where the mother and the father have no shared features, but the children have features of mother and father. All together, they form the category of a family and no member owns all the features of the abstract family prototype.

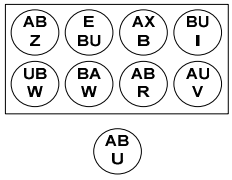



**Figure 2. Prototype theory**

The theory is visualized in Figure 2. The prototype is the ABU element where different features occur together in the various elements. We show that not all elements share the same features and there is element existing that realizes the complete prototype. The inclusion of characteristic features illustrates, why some elements are perceived as typical instances of a category, in comparison to others. Characteristic features of a category are abstracted during learning and merged as a representation of a prototype representing the category. Thus, all typical features are associated with the prototype. Hence, the prototype is a representation of an amount of objects that share similar features [23]. Therefore, the category definition is in this case not sharp any more.

In code, this theory can be realized through annotations. For instance, absolutely different classes can be marked with various persistency annotations in Java. Typically, developers can choose between various annotations that are indicating persistent features. Developers can map classes and their fields directly to tables and columns. There is no necessity that all classes need to be marked this way. Thus, the annotations used can differ for various classes. Even though, developers associate all classes with such annotations with the persistency category. The classic view is not sufficient to explain the phenomenon, because annotations can differ for every class. We consider the prototype theory as explanation. A prototype class serves as mental representation for the developer. This prototype contains all persistency annotations. Developers compare this prototype with other classes to decide if they are belonging to the persistency category.

*3.1.3 The Exemplary View*

The exemplary view assumes that in contrary to the prototype theory, single exemplars are engrained together with the category denotation. Each new exemplar represents a category on its own. By recognizing a new exemplar a learner compares it to already known categories. The learner assumes that an object might have similar features as the exemplar compared to which it has the most similarities. Thus, similarity comparisons are made with the exemplar itself and not with an abstract prototype. This way, a certain animal might be categorized as a rodent, because it reminds of a mouse, whereas another animal is categorized to the same category, because it reminds of a squirrel or a chipmunk [5].

We visualize the exemplary view in Figure 3. Objects are depicted by their features and arrows connect same features of exemplars. Every exemplar is a category on its own and just the features remind of another object. The AB exemplars remind stronger of each other through two shared features. If any exemplar is recognized, the exemplars that share the most features are taken as reference to derive further features or functionality of the object.

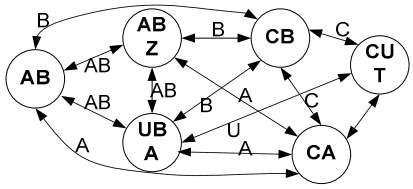



**Figure 3. Exemplary view**

The PA mechanism of Aspect-oriented programming allows in many cases to avoid modules with multiple concern associations. Also, the Hyperslice approach tries to avoid such kind of overloaded modules. Generally, the Hyperslices follow the idea to encode every concern in its own module. Modules, encoding only one concern at a time, are similar to the vantage point to regard every exemplar on its own. Every resemblance to other modules is just defined because of shared features. Hence, we see Hyperslices as similar to the exemplary view. The PA mechanism enables to group program elements together and apply advice code at them. The grouped elements share common features like network accessibility or security. Thus, we classify the PA mechanism as hybrid similar in between the exemplary view and prototype theory.

*3.1.4 Mapping Theory and SOC*

We see the means to apply SOC and categorization as similar, because of previous described comparison. In Table 1, we show mechanisms to apply SOC and the categorization theories that share similarities. The different values in the rows present the mechanisms of the before described examples and their mapping to the columns.

**Table 1. Similarities of theories and means to apply SOC**

| Classic | Prototype | Exemplary |
|---|---|---|
| Classes-objects | Pointcut - Advice | |
| Inheritance | Annotations | Hyperslices |

### 3.2 Types of Categories

Besides the three theories, different types of categories are used to group objects together [17, 19]. Following, we explain these types of categories and describe a cross-classification.

#### *3.2.1 Taxonomic Categories*

Taxonomic categories represent hierarchies of increasingly abstract categories. An example for a taxonomic structure is terrier-mammal-animal. In case of programming, this kind of categorization is solved by specialization through inheritance.[1] Thus, we see inheritance as a representative for taxonomic categories. Another taxonomic structure is represented through package hierarchies. We see this as similar to the taxonomic category kind, because the elements within a package normally have a somehow related meaning and packages are arranged in taxonomy. Thus, we see the hierarchic structure of the packages as taxonomic category, but the membership of elements in the distinct package itself has to be described in a different way.

#### *3.2.2 Script Categories*

Script categories are used to group elements that play the same role together. Such elements are interchangeable within the role. For instance, eggs and bread belong to the category breakfast foods and are exchangeable. In programming, classes that are implementing interfaces, are interchangeable and offer the equal functionality. This is similar to script categories because all the classes that implement an interface are interchangeable.[2] Another case are object instances of a class. All of them offer the same functionality. Routines in code that work with one object also work with the other instances. All objects of a class are interchangeable. Thus, we see a mapping to script categories, too. Single Annotations can also partly be considered as a typical example for script categories. Classes can be marked persistent. The Classes can differ absolutely in their meaning and also in their functionality but they are interchangeable; any class can get a persistency annotation and is saved with the same routines within an application framework.

#### *3.2.3 Thematic Categories*

Thematic categories group objects together that are associated with each other or have a complementary relationship. Such a category can be a clothesline. Clothes and lines are distinct categories. The association forms the new category of a line to hang clothes on it. Nobody would talk about a line to hang clothes on it. Everybody understands the combined term of a clothesline as its own category. Such a grouping of elements is done through packages. All classes and interfaces in a package belong to a certain theme and are connected. Compositions are a classic way of combining. For instance a class uses multiple instances of other classes and combines these to derive a new functionality. Another often used way for composition is implementation of multiple interfaces, which concentrates the functionality of different concerns at single point of code. This code part is then probably recognized as its own category.

#### *3.2.4 Cross-categorization*

As indicated by composition of elements before; an object can be associated with multiple categories at the same time. This is called cross-categorization. An example can be an "Account" type in programming. It represents a domain concept and often a persistent entity in a database. Additionally, it can also represent an object, requiring secure access. However, programmers still refer to as the "Account" type. The actual meaning is thereby discriminated through the context. Research in psychology shows that cross-categorization is something absolutely natural [11]. Objects belong to different categories at the same time. The context influences the actual meaning and the inference of the way of operation of an object.

In Figure 4, we present a visualization of the recognition of an object, respecting the context in which the fragment appears. Thus, the different categories are recognized under influence of the current strategy. Additionally, categories of a fragment can be of a different category kind (taxonomic, script, or thematic), whereby the perception of a type is also influenced by the context.

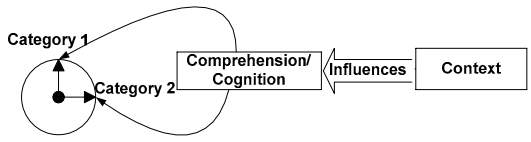



**Figure 4. Cross concern comprehension**

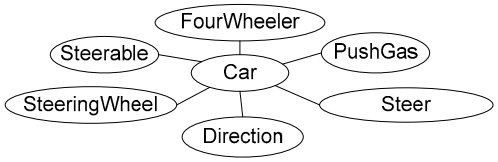



**Figure 5. Multi-category associations of Listing 1**

A more detailed example is given in Figure 5 by representing Listing 1 in a different graphical view. The Car class is belonging to its different vertices at the same time. In Listing 1 all the vertices are directly or indirectly connected with the Car. For instance, it consists of four wheels, is steerable and can drive in various directions. Thus, we

[1] Inheritance of multiple classes or interfaces is often possible in programming. Therefore, only class inheritance with one super class is a representation of this type of category.

[2] Classes can implement multiple interfaces. We assume that interfaces can represent categories. This way, a class can belong to multiple categories. Script categories do not explain the multi-category association. Later, cross-categorization is proposed as explanation.

visualize this simplified with direct connections of the vertices with the Car in Figure 5. We compare this with cross-categorization, where an object belongs to multiple categories at the same time. Here, these categories are just concerns. Thus, we see the association of a source code fragment with multiple concerns as the same like cross-categorization.

#### *3.2.5 Mapping Categories to SOC*

We present our described types of categories and described similar SOC mechanisms in Table 2. To respect the exemplary comparison, done before, several mechanisms are appearing multiple times. Columns indicate the category type, and rows represent their corresponding mechanisms. For instance, the package mechanism is used in two ways. Interfaces are appearing in script and in the thematic column, too. Advices and Pointcuts are added. The PA mechanism itself is not hierarchical and therefore not taxonomic. It can be used to affect totally distinct and interchangeable objects with concerns like security or logging. This is normally done through the application of Advices at various Joinpoints that are selected by Pointcuts. This seems similar to script categories. However, Advices add functionality or alter behavior of code, which is similar to a composition (thematic).

**Table 2. Similarities to category kinds**

| Taxonomic | Script | Thematic |
|---|---|---|
| Single inheritance | Pointcut - Joinpoints | Advice |
| Package-hierarchy | Classes-objects | Package-members |
| | Annotations | Compositions |
| | Interface implementation | Multiple inheritance |

## 4. DISCUSSION

One could argue that intertwining of concerns does not mean that humans do not try to separate the concerns. Categorizing is what humans do to separate concerns. Categorization differs in its accuracy and in the numbers of categories associated with an object. Categorization is sometimes stricter and other times more intertwined. There exist cases where SOC is completely applicable and desired. The research about categorization shows that intertwining of categories happens through cross-categorization. It does not provide a distinction of cases where cross-categorization happens and where a single category association is done.

We argue against this point and mention that cross-categorization already provides examples where the context influences the chosen category. Hence, we do not argue that SOC does not work for us humans. There could clearly be examples provided where a clear separation is excellent. Nevertheless, for many cases it is difficult to come up with good mechanisms to realize the separation. In most cases, separation is not sharp. We cannot manage this in real life either. Many categorizations that we use for the same or similar objects support this. We think, it is natural for humans to try to separate concerns, but intertwining and the recognition of the right category of an intertwined object is a critical case that often appears. Hence, the more important question is: What are the right means to separate and to compose these modules? It is not important to discuss whether SOC is natural or not; it is more important to see it from the point of comprehension and which means of separation should be used in which case.

Probably one could come up with other examples where certain programming mechanisms are used to support certain categorization techniques. Such examples can differ from our comparison. We mention that our comparison shows well known cases and already similarities excelled. But, our comparison is a first observation. Therefore, such challenging cases can exist. Again, we see the necessity for more detailed discussions in a larger forum.

## 5. RELATED WORK

We already proposed to investigate categorization in the context of program comprehension in prior work [10]. In [6] programming is described as a process of the construction of mappings from a problem domain to source code. These mappings are done by using several intermediate knowledge domains that are all encoded in a code Comprehension is done by the reconstruction of those mappings. This is achieved by building top-down hypotheses. A hypothesis is verified by inspecting specific lines of code. Through the verification new hypotheses about advanced features of the program are built and the process of verification starts again. Thus reconstruction works iteratively until the program is comprehended. Our approach differs by setting the focus on the programmer's categories and their realization in the code.

In [14] the comprehension is divided into knowledge base, mental model and assimilation process. The knowledge base includes experiences of programmers. The mental model relates the representation in the mind and the implementation of the program. The assimilation of a program is done by applying the knowledge base to the program. The difference to our approach is that categorization is not mentioned as mechanism to understand programs.

In [22] the comprehension is done based on beacons representing key code controls and operations. Such beacons are well known elements by a programmer, used to derive predictions about functionality. We assume that beacons are categories or features of those, because they are used to derive knowledge about functionality. This is similar for categories. There, features are used to identify the category of an element and to make conclusions. Therefore, we assume a relation between concerns and beacons that needs to be investigated.

## 6. CONCLUSIONS AND FUTURE WORK

Some constructs in programming are similar to research of categorization in psychology. Examples showed similarities of means to apply SOC and categorization theories. Different categorization theories are similar to some mechanisms to separate concerns. Cross-categorization is comparable to intertwined code fragments representing various concerns simultaneously. Elements can be associated with multiple categories at the same time. Hence, it is natural that not all concerns in source code are separated, too.

This opens new questions on SOC: How many concerns should a fragment represent? How does the context of a program influence the recognition of its module's functionality? Can the context, in which a programmer studies code, be derived? Maybe, the context could be derived through code and tracking developer's behavior in the development environment? Can compositions be seen as context of a code element? Future research about composition needs to investigate results from psychological experiments deeper. It is important to lean the means of composition mechanisms against the ways of natural human comprehension. The right means of composition must be chosen in the right context.

We restricted our presented research to a few mechanisms to separate concerns and compared them with categorization. This was done on varying abstraction levels. A more detailed and consistent comparison needs to be done. Our comparison of categorization and SOC needs to be verified through further investigations and improved through additional composition mechanisms. Quantitative research studies on manifestations of concerns in source code could determine and verify how categories are realized in code. The cited psychological studies have to be transferred and adapted to retrieve corresponding patterns in source code. A possible outcome of how categories manifest in code can be a new concern revealing technique.

## 7. ACKNOWLEDGEMENT

This work is partly funded by the German Ministry of Education and Research within the project ViERforES-II (grant no. 01IM10002B).

## 8. REFERENCES

[1] S. L. Armstrong, L. R. Gleitman, H. Gleitman. What some concepts might not be. Cognition, Vol. 13, No. 3. , pp. 263-308, 1983

[2] L. W. Barsalou, Ad hoc categories. Memory & Cognition, 11, pp. 211-217, 1983

[3] J. A. Bloch. Metadata Facility for the Java Programming Language. 2004.

[4] G. Booch. Object-oriented analysis and design with applications. 2nd ed. Addison-Wesley Professional. 1993

[5] L. R. Brooks, G. R. Norman, S. W. Allen. Role of specific similarity in a medical diagnostic task. Journal of Experimental Psychology: General, 120, pp. 278-287. 1991

[6] R. Brooks. Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies vol. 18, pp. 543-554, 1983

[7] J. S. Bruner, J. Goodnow, G. Austin, A study of thinking. Wiley, New York. 1956

[8] E. W. Dijkstra. A Personal Perspective. On the role of scientific thought. Selected Writings on Computing, Springer-Verlag. 1982

[9] R. E. Filman, T. Elrad, S. Clarke, M. Aksit. Aspect-Oriented Software Development Addison-Wesley Professional; 1st edition,. 2004.

[10] T. Frey, M. Gelhausen. Strawberries are nuts. CHASE '11 4th International Workshop on Cooperative and human aspects of software engineering. p. 49. ACM. 2011

[11] D. Klein, G. Murphy. Paper has been my ruin: conceptual relations of polysemous senses, Journal of Memory and Language, Vol. 47, No. 4, pp. 548-570, 2002

[12] G. Lakoff. Women, fire, and dangerous things: What categories reveal about the mind. University of Chicago Press. pp 614. 1987

[13] B. Landau. Will the real grandmother please stand up?, The psychological reality of dual meaning representations. Journal of Psycholinguistic Research, 11, 1982.

[14] S. Letovsky. Cognitive Processes in Program Comprehension. In Empirical Studies of Programmers, pp. 58-79. Intellect Books, 1986

[15] D. L. Medin, E. J. Heit. Categorization. In D. Rumelhart and B. Martin Handbook of cognition and perception. San Diego. Academic Press. 1999

[16] D. L. Medin, E. B. Lynch, K. O. Solomon. Are there kinds of concepts?. Annual Review Psychology, 51, pp.121 - 147. 2000

[17] G. L. Murphy. Causes of taxonomic sorting by adults: A test of the thematic- to taxonomic shift. Psychonomic Bulletin & Review, 8, pp. 834–83, 2001

[18] G. L. Murphy, The Big Book of Concepts, MIT Press, 2002

[19] S. P. Nguyen, G. L. Murphy. An apple is more than a fruit: Cross-classification in children's concepts. Child Development, 74, 2003

[20] B. Nora, G. Said, A. Fadila. A Comparative Classification of Aspect Mining Approaches. Journal of Computer Science 2 (4): pp. 322-325, Science Publications. 2006

[21] H. Ossher, P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2001.

[22] N. Pennington. Comprehension Strategies in Programming". In Empirical Studies of Programmers: Second Workshop, pp. 100-113, 1987

[23] E. Rosch, C. B. Mervis. Family resemblances: Studies in the internal structure of categories. Cognitive Psychology, 7, 573-605.1975

[24] E. E. Smith, D. L. Medin. Categories and concepts. Cambridge, MA.: Harvard University Press. 1981

[25] J. D. Smith, M. J. Murray, J. P. Minda. Straight talk about linear separability. Journal of Experimental Psychology: Learning, Memory, & Cognition, 23, pp. 659-680. 1997

[26] M. F. Verde, G. L. Murphy et al.. Influence of multiple categories on the prediction of unknown properties. Memory & Cognition, 33 (3). 2005

[27] T. Yamauchi, A. B. Markman. Inference Using Categories, Journal of Experimental Psychology. Vol. 26, No. 3, pp. 776-795, 2000