Categorization of Concerns

A Categorical Program Comprehension Model

Tim Frey
Otto-von-Guericke University
Magdeburg, Germany
tim.frey@tim-frey.com

Marius Gelhausen
TU Darmstadt
Darmstadt, Germany
marius.gelhausen@gmx.de

Gunter Saake
Otto-von-Guericke University
Magdeburg, Germany
gunter.saake@ovgu.de

Abstract

Program comprehension models lack associations with the paradigm of separation of concerns. We present a holistic program comprehension model based on categorization studies of psychology. A comparison of research about categorization and separation of concerns is used to develop the model. The cognition in this model is influenced by the context wherein a programmer investigates the code. The comprehension process starts with some ad-hoc concerns that are about to be refined by following an investigation strategy and a vertical process study. Through this study, the concerns refinement may imply an update on the knowledge and the adoption of a new behavior for the investigation strategy. Our model can serve as starting point for further investigations how developers recognize concerns.

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 Program comprehension, separation of concerns, categorization

1. Introduction

Companies spend a large amount of money for software maintenance [28]. Since source code is more read than written, a crucial element in maintenance is the comprehension of source code [25]. Thus, research about maintenance tasks and the corresponding inspection of source code by a developer is done. It seems that some elements are more important for specific maintenance tasks [58]. Likewise, first indications show that effective programmers inspect source code systematically to uncover elements, belonging to a task [51].

In order to explain the source code investigation process, different program comprehension models were created. These models [2, 6, 7, 8, 9, 10] describe the creation process of a developers mental program model. Some of them [8, 7, 3, 27] state that previous knowledge of a programmer is used to gain intelligence about a program. Some models propose that developers use so called beacons [2, 6, 4, 1, 3] to detect familiar structures in code. Beacons are well known concepts (e.g. method calls) to a developer that are used to derive indications about the code. Hence, the role of such conceptual knowledge in program comprehension is of interest to researchers [12, 26]. Thus, how conceptual knowledge of a programmer manifests in code is a central element for program comprehension [13, 14].

In order to realize mental concepts in code, developers follow the paradigm of separation of concerns (SOC). SOC recommends encoding one concern in one module and weaving those together [21]. Commonly, the term concern is overloaded, in respect to programming structures, to be any matter of interest of a software system [24]. Thus, we restrict a concern to a logical classification of a source code fragment. Programming languages support, by varying terms, the application of SOC [29].[1] In reality, often modules are responsible for multiple concern or concerns are scattered over various modules [17]. Consequently, programmers face the challenge to comprehend the concerns and their encoding.

Surprisingly, none of the former referenced program comprehension models addresses the fact that programmers need to comprehend concerns and their separation. In favor, to fill this gap, we did research about areas in psychology that seem similar to the idea of SOC. In prior work, we identified the area of categorization in cognitive science as interesting for program comprehension [60]. Now, we show the following research about categorization and present a program comprehension model based on it.

Our contribution is a holistic program comprehension model based on categorization theory. This model

[1] We assume a reading familiarity with the idea of SOC [21] and the corresponding coding techniques. For the comprehension of this paper a basic knowledge about Annotations [18], Aspects [19, 31], design patterns including architectural layers [23, 40, 61] and the basic idea of multidimensional SOC [15, 16, 17] is expected.

respects research about categorization and SOC. Hence, we enrich program comprehension through the direct association with an area of psychology. We enable future research about the detailed role of categorization in program comprehension.

First we do a brief introduction about categorization. Afterwards, we present similarities of the SOC paradigm in contrast to categories and establish step by step our comprehension model. We argue for the validity of our model by presenting threats against it. Related work shows similar approaches and differences to our work. Finally, we do conclusions and propose future work.

2. Categorization

Categorization is a part of cognition and fundamental for the process of comprehension. Categories play a central role in perception, learning, communication and thinking. Categorization is used to group objects together into classes, based on similarities. These classes are called categories or concepts [36, 38]. In this chapter, we give a brief introduction about categorization. We present the different theories how elements are categorized.

2.1 A brief introduction

Category membership enables the usage of knowledge, about a category. New objects are associated with a category and the knowledge about the category is used to assume properties of the new object [36, 38, 45]. To argue, in favor of the existence of categorization, it has to be considered that without it, every object or incidence would appear inimitable. Hence, there would be no way to do predictions about new unknown objects. Thus, categories can be seen as a product of interactions, containing perceived resemblance relations in the environment, prior knowledge and context of utilization.

Categories can be formed for physical objects like specific animals, but also for entities that don't exist as physical objects [47]. For instance, democracy represents a conceptual ideal of government. This way, categories classify abstract functionality for a democratically system directly. Likewise, it is possible that categories of physical objects are used to infer about the functionality of category members. Thus, no simple distinction between categories and functionality of those can be done. However, categorization is a dynamic process, which updates the knowledge about categories, their members and their relations constantly throughout new experiences. Thereby, even minimal prior knowledge has the effect to greatly speed up category learning [56]. Likely, there are basic categories, representing important and often used elements, enabling faster processing and association [50].

One feature of categorization is to do quick predictions [46]. This can be done out of limited available information, but can lead to avoid understanding an object completely. It would be too time consuming to comprehend every object totally. This way, it is possible to drive a car just because all cars are similar or to be careful towards a snake. Like the examples show, predictions are used to act adequately or to adapt behavior [32].

Additionally, there are different kinds of categories. Taxonomic categories represent hierarchies of increasingly abstract categories like terrier-mammal-animal. Script categories are used to group elements that play the same role together. For instance, in the case of breakfast: Eggs and bread belong to the category breakfast foods and are exchangeable. Both can be eaten. Finally, thematic categories that group objects that are associated or have a complementary relationship like a dog leash or a clothesline [41]. We humans use taxonomic, script and thematic categories equally to categorize and understand objects [40]. This means that every category kind is used to categorize objects and none is favored. It is also possible to combine different categories and their objects to generate new categories [62, 63]. Also categories can be structured hierarchically. Subcategories have features of their super ordinate for a certain probability. The "inheritance" of features is therefore not absolute [37].

Categories can be formed spontaneously to fulfill a certain task. This kind is called ad-hoc categories [43]. For example, things that can be sold in a garage sale can be defined in a spontaneous category. Also, an object can be associated with multiple categories at the same time, what is called cross classification. The context wherein an object is viewed influences which category is associated with it [39]. Thereby, the objects categories can be the different category kinds (e.g. the former named taxonomic and thematic) [41, 42]. This is important, because research indicates that the association speed with a category differs for the different category kinds [44].

2.2 The different theories

The classical view claims that categories are discrete entities characterized by a set of properties which are shared by all of their members. These properties are assumed to establish the conditions, which are both necessary and sufficient, to capture the meaning of a category. All members of a class posses equal quality to the respective category [33]. Through results of various experiments it seems that categories have diffuse boundaries, and categories are not discrete [22, 34]. Therefore new theories have been built.

The prototype theory claims that categories are represented by a bundle of characteristics, which are typical for a certain category, but not inevitable or sufficient [35]. A category "bird" might have characteristics, like "flying" or "building a nest". Even if not all birds exhibit this features, they still belong to the bird category. For instance, a penguin is a bird that can't fly. The prototype of a category merges typical characteristics of a category, but no exemplar has to match completely, with all characteristics. New objects are classified, out of an affinity composition with the prototype of an already existing category. 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 [22].

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 exemplary a learner is reminded, more or less, to prior seen exemplars. The learner assumes, an object might have the same features like 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. Research certifies some prognoses that were made out of this theory [59]. Nowadays hybrid theories are available that try to combine the different views [20, 30, 64].

3. Categorization of Concerns

To bring categorization and SOC together, we present different supporting points. Several comparisons of the area of categorization to the area of programming are shown. Thereby, similarities between SOC and categorization are emerging. Finally, we show a model with all the facts included together.

3.1 Characteristics of categories

Categories are essential for the comprehension of specific objects and discrete conceptual entities [47]. In source code both of them are occurring, too. There are concrete mechanisms to indicate concern associations of elements. Such mechanisms are packages, classes, inheritance, annotations methods and similar means. Other concepts are realized through compositions of various programming constructs. Design patterns are an example for such compositions [61]. Developers associate the realizing elements with the corresponding design pattern. The functionality of these elements has discrete, flexible characteristics where not every implementation is completely equal. Thus, the category of a design pattern is more like an abstract concept then a concrete mechanism. Categories manifesting in code are similar to physical object categories and abstract concepts like democracy. Hence, there are concrete mechanisms to realize categories and compositions to realize elements of conceptual categories.

The comprehension of fragments seems to be dependent on the experience [51] and thus probably of the knowledge of a developer. That is also the case in categorization. The learning of categories and also comprehension speed differs, depending on prior knowledge [56, 50]. We see supporting arguments in research about program comprehension. There, some source code lines are more important than others for the comprehension of a developer [3]. We see these lines as a representation of categories or features of categories. To categorize the fragment, features (those lines) are recognized and used for category association. Another possibility is that the line itself represents an element of a category that is recognized. We claim that concerns are acquired and accessed by studying programs. This is similar to how categories are learned. Therefore, we assume that the importance of a concern and the frequency how often it is studied, influences if it is recognized as a basic concern. This distinction is done, to respect the existence of basic categories. Like for categories, the comprehension of basic concerns is done quicker as for normal ones. We consider framework classes as an example for often used elements. Therefore, we assume that framework classes are an instance of basic concerns.

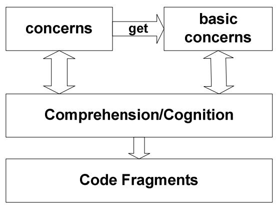

Figure 1. Learning basic concerns

We present our graphic representation of the concern learning process in Figure 1. Developers comprehend code and use basic and normal concerns. We visualize the access of the code with the arrow from comprehension down to the code fragments. A developer accesses a concern every time it is studied. Thus, the usage frequency of the category gets up. This means: Just the study of source code can lead to classify a concern as important and it can be faster comprehended when it is occurring again. We visualize the continuous usage of concerns for comprehension as well as their update as double sided arrows. Additionally, the importance of a concern and its usage frequency refines a first recognized normal concern into a basic concern. When a code study begins there is no knowledge about the source code. But as the knowledge grows, certain concerns can get important ones. We present this transfer as arrow from concerns to basic concerns.

3.2 Usage of prior knowledge for prediction

Developers predict functionality of source code through prior knowledge. This is obvious, because often reoccurring elements are used in a program. Also, different program comprehension theories support this. It is assumed that a developer creates hypotheses about the way of operation of a program out of prior knowledge [2, 8, 7, 3, 27]. That usage of prior knowledge for comprehension seems also to be the case in categorization [45]. We believe that building hypothesis about a program is equal to predictions out of category membership [46]. Hence, the prediction about the functionality comes out of categories. A prediction leads to adequate behavior. Such a behavior can be a change of the investigation strategy. In categorization adapting behavior is an important appliance [32]. Thus, it seems equal to the code inspection process.

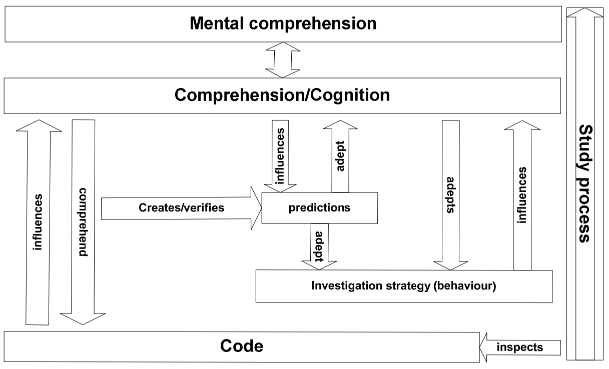

Figure 2. Prediction and behavior

We developed visualization for the process of predictions and behavior. We present it in Figure 2. In respect to the program comprehension scenario, the behavior is called investigation strategy. On the left side, the investigation process is vertically visualized. It is arranged in a box with an arrow to express the continuous process. All shown actions happen within this process. The arrow directions visualize the way of interactions between the elements: Code is studied and comprehended through the influence of known concerns. The comprehended code influences the cognition and thus the comprehension of further fragments. For example, new concerns can be learned by studying and therefore the cognition is changed. Comprehending code leads also to predictions about the features of the code under the influence of known facts. By building such a prognosis the current state of comprehension influences the prediction. Because of predictions about studied source fragments, the investigation strategy can be adapted. The strategy also influences the cognition of a studied code fragment and is adapted by the current state of comprehension. It is a mutual influence: In the study process this can happen multiple times and when the comprehension adepts the investigation strategy the perception of new studied elements will be changed. Anyway, sometimes wrong or right predictions are made. In such an ongoing comprehension process, predictions can be verified wrong or right. It is important to note that the strategy has not the demand to be verifying predictions continuously. A direct verification is probably rather happening, when uncertainness about the predictions is at stake. Likely, in a normal study process, the predictions are verified by the assumption they are true and further investigation support of falsify these.

The verification of these predictions is then used to update the knowledge about concerns. Such an update can be the proof or falsification of the predictions that are associated with a concern. Since these concerns influence the cognition, the strategy can change again. For example can the recognition of a code fragment lead to predictions about its functionality. Further investigations can then lead to support or falsify the previous made assumptions. Because of this, the knowledge about the concern has to be updated. This leads to new predictions that can lead to a strategy change in investigation.

3.3 Different means of separating concerns

Different programming paradigms and mechanisms exist to apply SOC [21, 17, 19, 15, 16, 18, 31, 61]. In the following we will exemplary mention a few of them on an arbitrary level. This way, we show a relation between categorization theories and means to separate concerns.

A mean to apply SOC are classes in object-oriented languages. The inheritance of fields and methods is absolute for subclasses. We see this as corresponding to the classic category view. Also, we see the same for classes and their instances. Instances of classes have all the same features.

Annotations [18] enable developers to mark absolutely different classes with them. For example, classes can be marked through persistency Annotations. All marked persistent classes are belonging to the persistency layer. This layer can also be named persistency category. The classes belonging to this layer cannot be defined as precisely as the classic view demands: The annotated classes are totally distinct and share varying feature sets. Obviously, no class needs to have all of the persistency annotations. Thus, we recognize that the classic view is sufficient to represent this kind of categorization. Hence, we propose the prototype theory as explanation for this phenomenon. A prototype of a layer groups all the different features, associated with the layer, together. This way, an abstract prototype groups all persistency Annotations together. Classes are getting annotated with the Annotations that occur in the prototype. Each class just has to have a few features of the prototype. Not all features need to be shared with other classes of the category.

However, SOC proposes to encode each concern in a separated module. If this would be possible the result would be modules with no shared features. So, we see the necessity to conclude that the prototype theory is not sufficient to explain SOC. Advanced mechanisms to apply SOC support this assumption: Aspects [19] group elements together and allow modifying their behavior. All the elements have in common is the application of the Aspect. Therefore, aspect-oriented programming is not comparable with the prototype theory. Similar to Aspects, MDSOC allows advanced modularization [17, 16]. All together, the advanced separation beyond the prototype seems quite similar to total distinct exemplars.

All over, we see source code and the means to separate concerns similar to the different theories how categories are arranged. Our main point is that means to separate concerns are similar to categorization theories. Elements in source code are recognized and grouped into categories.

We developed a comprehension process and visualization and present it in Figure 3. Concerns in the source code are shown (Code Fragments). The single sided arrows visualize that code is comprehended and studied. The letters represent the concerns and the circle around them represents a module. The different means to separate concerns and modules can be mapped to the corresponding cognitive representation. The mapping box shows different possibilities of mapping. This mapping between the mental representation and the realization in source is maintained continuously during the study process. Therefore, the arrow in the box indicates the continuous comprehension of a developer. The small arrow, to the mapping box, indicates the constant mapping refinement during the comprehension. In the mapping block the different circles and arrows show different variants of mapping. We symbolize this mapping as arrows from the mental space (grey) to code mapping (white). The grey circles in the mapping are concerns and the white code circles are modules. Not every concern is realized in a single module and sometimes modules represent multiple concerns. This is because of limitations to separate concerns in the source code, but also through mistakes of programmers.

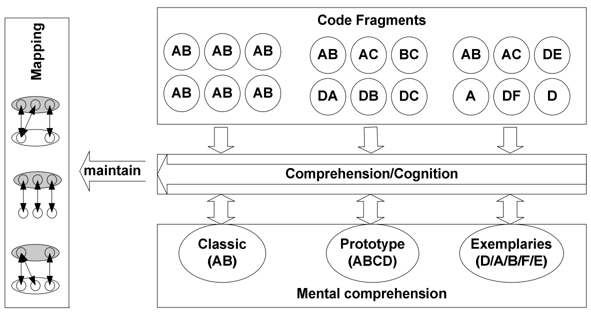

Figure 3. Concern theory mapping

The arrow across the comprehension box symbolizes the continuous comprehension process and shows that concerns will be recognized differently. At their first occurrence they will probably be recognized as a single exemplar. For instance, an class is seen for the first time. Since a developer does not know other occurrences of this exemplar just a single exemplary is created. As more and more code fragments appear, using the class, a developer is reminded to the other fragments. Then, an abstract prototype will be created if similarities between the different code fragments excel (prototype theory). If all the studied features are fixed, a category following the “classic view “ category is created. We show this use of prior seen exemplars through the double ended arrows. Additionally, we show exemplary code fragments, whereby the letters describe features of the fragments. Vertically at the bottom are the corresponding different categories arranged.

3.4 Multi-category-association

In software development, maintenance tasks appear, as for instance, fixing bugs. Dynamically multiple elements of the code are associated with such a task [58]. Likewise categories can be formed dynamically [43]. Hence, this indicates that program elements can be part of a spontaneous category. Similar is the multi category membership in categorization. Elements can be part of script, thematic and taxonomic categories at the same time. For instance, a class belongs to a certain package, but in the same moment it can also be a member of an inheritance hierarchy and can be affected by Annotations. Furthermore, a code fragment can be comprehended differently. The actual comprehension depends on the intention of a developer when he studies a fragment. This can be compared with polysemous words where the actual meaning is driven by the context [39]. Thus, likely, the association of a source code fragment with multiple concerns, is the same like cross-classification [41, 42] .

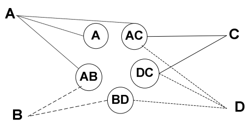

Figure 4. Cross concern association

We present our view of a multi concern association in Figure 4. There code fragments are associated with different concerns. The concerns are the letters and the code fragments are the circled letters. The fragments are belonging to different concerns at the same time. We visualize this through the concern letter associations.

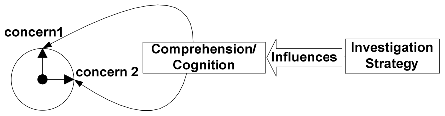

Figure 5. Cross concern comprehension

Figure 5 shows a single fragment that belongs to two concerns. The comprehension which concern is recognized is influenced by the context in what the fragment appears. For the study process is this context the investigation strategy. Thus, the investigation strategy influences how and which concerns are comprehended.

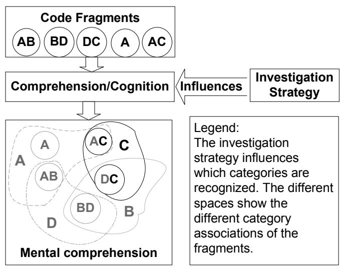
Code Fragments
AB BD DC A AC
Comprehension/Cognition
Influences
Investigation Strategy
A AC C
A AB DC
BD B
D
Mental comprehension
Legend:
The investigation strategy influences which categories are recognized. The different spaces show the different category associations of the fragments.

Figure 6. Cross concern association comprehension

In Figure 6, Figure 4 and 5 are combined and shown with multiple fragments. The discovering of varying concerns through different investigation strategies is visualized by the example of grey and black concerns. However, the framed spaces represent the concerns. As shown, a fragment is framed by multiple lines to indicate the multi- concern fragements. In short, a fragment is an intersection of various concerns.

3.5 Composition

The composition of code elements seems similar to the combination of categories to build new categories [62, 63]. Hence, we assume that new concerns can be created through the composition of other concerns, what happens quite often in source code; for instance, by creating data-access-objects (DAO). Such classes are clearly to be counted to the persistency layer. At the same time this classes are composed together with domain objects that normally represent business entities. Therefore, it is wrong to count a DAO only to the persistency layer, since they also belong to a distinct business domain in the application itself. Thus, a DAO is a composition of a business concern with the concern of persistency.

Developers often deal with abstract definitions and concrete implementations. We consider that a class can represent a concern. But when it is used somewhere (e.g. as instance object) then it is like a feature of another element. This reveals the question between a concern and its instance. The different categorization theories hold no answer about categories and their members. Hence, for the sake of simplification we don't distinct between concern or category instance.

In order to understand a composed concern a developer needs to know the underlying concerns. Another option to determine the way of operation of a composed fragment is by meta-information associated with the composed concern itself. Such meta-information can be realized through comments or a meaningful name of fragment.

Compositions are quite common. We assume that compositional concerns are only recognized as concerns by a developer, when their meaning is quite clear. Not every composition will be recognized as new concern. The usage frequency of a composed concern is an indicator if it is comprehended as concern itself. If a composition is never used, it will probably not be recognized as a concern on its own; likely it will be comprehended as pure composition, but not as a concern.

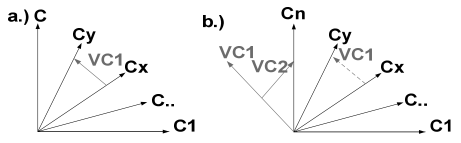

Figure 7. Concern composition

Figure 7 shows a composition of concerns based on concerns. The concerns are named C and the composed concerns are named VC (virtual concern). VC1 shows a composition of two normal concerns. The a. part of the graphic shows the construction of the first composed concern (VC1 – composed of Cx and Cy). The b. part shows the construction of another composed concern on top of a composed concern (VC2). To visualize, VC1 is equal to any other concern, a shifted VC1 is also shown in Figure 7b. We assume that VC1 will be recognized as concern, because it is used by VC2. The developer needs to know it to comprehend VC2. It can be that VC2 will not be recognized as a concern itself because it is not used anywhere else.

Anyway, we see this as simplification of the real world where some compositions (VC1) get used more often and others do not (VC2). The usage of VC1 in VC2 in Figure 7 is just one example for multiple usages within other concerns. Developers recognize such often used composition concerns as VC1 because of their reoccurrence in code. This is supported in the prototype theory. There, categories are learned through the occurrence of elements that are similar. Developers recognize the occurrences of a composition as category members. Thus, every occurrence of the used composition is associated with the composition itself and the composition evolves to its own category.

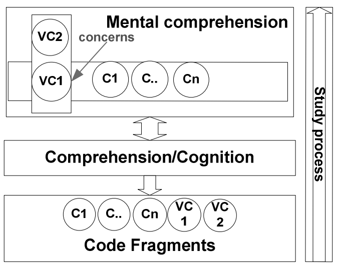

Figure 8. Concern composition comprehension

We visualize the integration of the compositional concerns in Figure 8. The code fragments contain compositions that represent compositional and normal concerns. A continuous study process is shown. We show the mental representation of a developer wherein the composition is expressed through the vertical block. All normal concerns (C1…Cn) are arranged horizontally. VC1 is a member in the horizontal and vertical. We do this to respect the previous discussed fact, that a composition can be recognized as a normal concern, too. Anyways, it has to be clarified that there could be multiple levels of composition and this crossing has to be seen as example. Developers build up the knowledge about the compositions through the study process. They can recognize compositions first as a pure "horizontal" concern. A further study process can then refine it to recognize it as composition. For example when VC2 is comprehended first, probably VC1 would get recognized as a normal concern and part of VC2. When it is recognized later that VC1 is also a composition, then the knowledge about it would be updated. In short, Figure 8 expresses the process of composition comprehension.

3.6 Putting it all together

We present the abstract model of the study process in Figure 9. The cycle expresses the continuous process of program comprehension. The arrow visualizes the mind of the developer where the knowledge is build up and the comprehension grows. The direction of the arrow also indicates the permanent knowledge level. We do this to respect that concerns can first be current concerns of interest, so called ad-hoc concerns, named after the ad-hoc categories. Such a-hoc concerns can evolve into permanent ones depending on their importance and occurrence in the study process. Finally, concerns can be even more generic basic concerns. Such basic concerns have a high usage frequency or a high importance, like described in section 3.1.

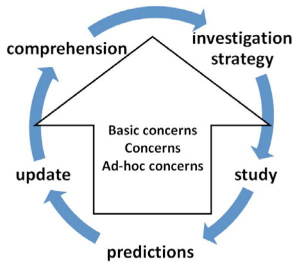

Figure 9. The comprehension process

Figure 10 is assembling all facts, from section 3.1. - 3.5, together. The two axes, study process and comprehension/ cognition, show the main actions that happen in program comprehension. The boxed arrows indicate the continuous comprehension process. All the elements that are in the range of the arrows are affected by the comprehension and the study process. The study process crosses the cognition/comprehension to indicate the consistency of the comprehension all over a study. Vice-versa, the comprehension process crossing over the boundaries of the study process shows the holistic approach of comprehension, which is not limited for a specific study process. Moreover, even multiple study processes can happen.

The mental comprehension contains different representations of the concerns in the mind. Like shown, they are updated during the study process and have an influence to the comprehension/cognition. The actual cognition adepts the knowledge and the knowledge can influence the cognition. Also, mappings between the mental representation and the concrete program elements exist and get maintained during the comprehension processes. The concerns are represented by the various kinds of category views/theories (section 3.3). The concerns beneath are separated into the different kinds of basic, normal and ad-hoc concern. The ad-hoc concerns are used for a specific task like described in section 3.4. The basic ones are important and often used concerns as described in section 3.1. The comprehension speed and usage frequency is increasing from ad-hoc concerns to basic concerns.

The bars crossing the concern kinds are the different other facts that were discussed allover in section 3. The grey vertical boxes represent the different concern types (exemplary, basic normal). The concern box is colored grey to indicate it is a simplification for the different theories that are also colored grey. The Means of separation block indicates that a concern can also be

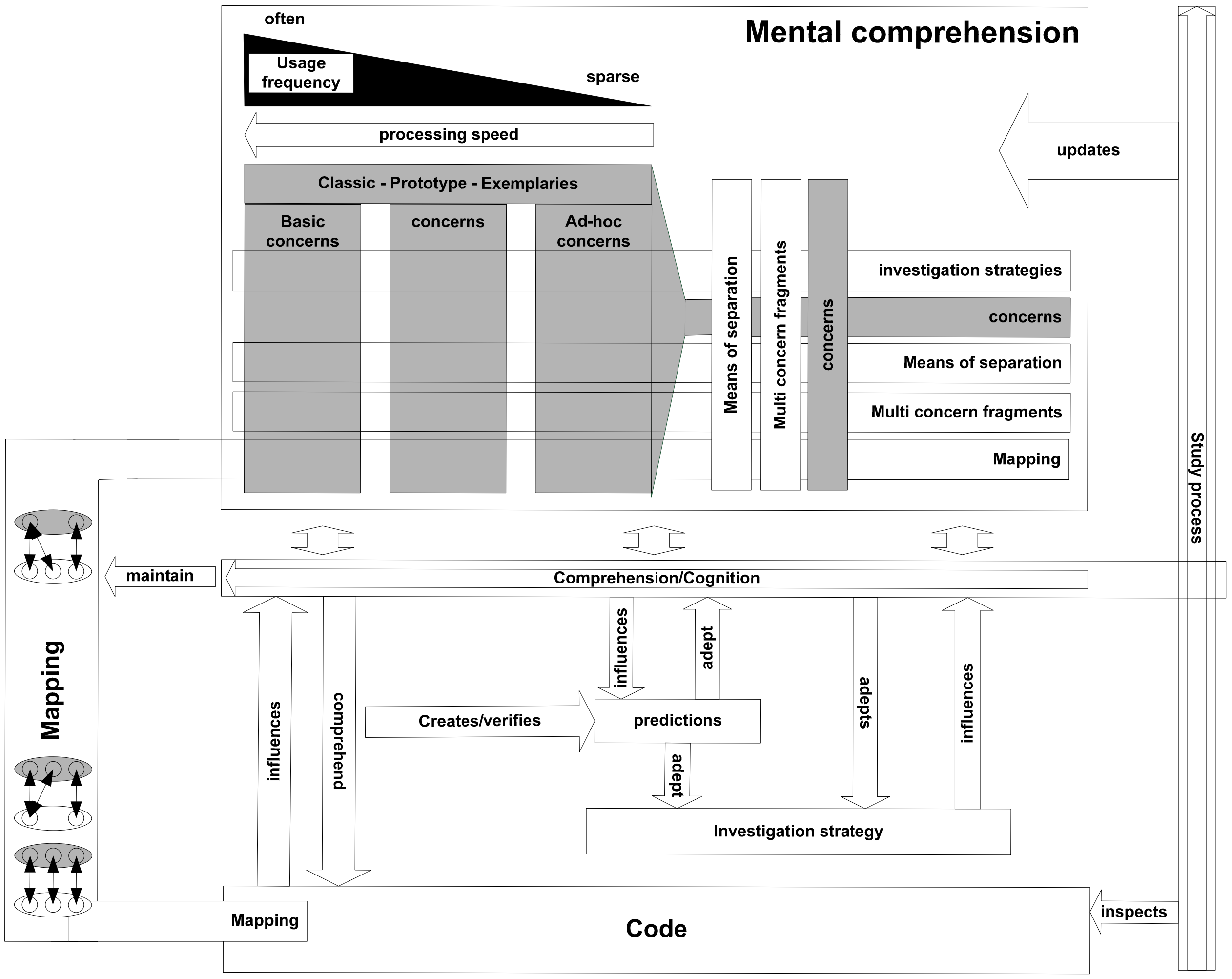

Figure 10. Holistic comprehension model

associated with the means how the separation manifests in the source code. The block named Multi concern fragments, indicates that a fragment can be mapped to different concerns at the same time and multiple concerns can be associated with the same fragment.

We show a block of investigation strategies. It was added to the mental knowledge, because we assume that the experience of a developer influences investigation strategies. Successful investigation strategies are somehow associated with concerns. Thus, there needs to be an association between investigation strategies and discovered concerns. Additionally, research [51] supports the assumption that investigation strategies vary for different skilled developers.

The vertical blocks with multi concern fragments and means of separation visualize compositions. However, the concern is enhanced with the other vertical elements that indicate the different associations.

Anyway, like in Figure 3 mappings of source code to the mental model are shown. The "Mapping" has been extended to visualize that they can also be associated with all the previous discussed elements and that they are in the mental comprehension as well as in the code. Through this combination of the mappings it is also possible to contain the means how concerns are separated and associate them with the mental knowledge. Like before, the small symbolization in the mapping bar indicates different mapping variants exist.

As discussed in section 3.2, different actions happen. The investigation strategy is adapted and predictions about the way of operation of studied source code fragments are verified or falsified and the mental knowledge is created and updated. Generally, all the arrow relations represent the same like described in the whole section.

4. Threats

Several threats have been identified that could corrupt the model. We show and address these facts to discriminate unsure pretenses against well known facts for validation with upcoming research.

The issue that effective programmers have different investigation strategies compared to others, that is stated in [51], needs further validation, since the study is not based on a large number of programmers. Therefore, the assumption that the investigation strategy can be associated with concerns could be wrong. We argue against this falsification, because of the fact that categories can change the behavior [32]. Since an investigation strategy is a kind of behavior, we argue for the accuracy of our conclusion, again.

Additionally, there is no complete model of the human mind available and there are only theories of how it works. This way, only the empiric supported research can be considered as fact. Therefore the different theories, exemplary, classic and prototype and also hybrid ones can be wrong. This could affect the comparison of the theories, with the means to separate concerns. We argue that these theories are based on empiric research and parts of them have been proofed. Our comparison was only on a basic level and showed already similarities. Furthermore, the comparison with the theories and means of separating concerns is only a small part of the model. Even with a nullity of these theories, only a small part of our model would be corrupted.

Also, the creation of categories just out of the pure composition of other categories could be used to argue against our model. We induce the fact that psychological research appears to go in the direction of cognitive concepts that are used to build inferences about the combination of categories [36]. We state that compositions are a concept. This way, our model holds for the specific case of composition and is not corrupted through a generalized conceptual combination of categories.

5. Related Work

Our model is in the research area of program comprehension [5,49]. In [2, 26] a model is presented, based on the idea of problem domain reconstruction through top-down hypotheses. Thereby, prior knowledge is used and the hypotheses verification is done through beacons. Programmers use these beacons to understand the way of operation of a code fragment [9]. Our comprehension model differs from the approach because it is founded on categories and concerns. We think that beacons are used to recognize the categories. The main benefit of our model is that it is starting to fill the gap between research in psychology and program comprehension. Furthermore, we consider the influence of the context in which source code is studied for comprehension. Our theory is connecting the manifestation of concerns in source code and the representation in a programmers mind.

In [6], the comprehension is done by the usage of standard code conventions and patterns. Again we don't argue against this point. Patterns can be used to derive categories.

In [8], the comprehension is divided into a knowledge base, a mental model and an assimilation process. The knowledge base contains the experience of a programmer. The mental model links 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. Again this model does not differ completely from our approach. For instance, basic categories are comparable to the experience. Again, the main difference is that SOC and neither research about categorization is mentioned.

Another model [10] describes program comprehension based on strategies and again the idea of categorization and the impact of SOC in the comprehension process and in the mental model is not mentioned.

The research about the creation of a concern schema, COSMOS [52, 53, 54, 55] can be seen related, since the conceptual space is modeled. COSMOS proposes a schema with various concern kinds and rules how they can be associated with each other. This schema is multi-dimensional and can be used to define the concerns of a system. The goal was to develop a method to map these to programming constructs. In contrast to our approach no background from psychology was used.

Additionally, in [48] the problem of semantic defects is proposed to be solved through an ontology. Thereby, the problem of mapping of concepts in the mind of a programmer to program elements is discussed. This is similar to our approach, since we also try to come up with explanations how a developer comprehends a program. Probably, it can be interesting to compare semantic defects with the categorization approach.

6. Conclusions and Future Work

We presented the application of research in the area of categorization to develop a program comprehension model. In doing so, similarities between categories and concerns were shown. Different dimensions of concern comprehension were uncovered and visualized. Finally, a complete model, with all the different aspects was presented. In short, the presented comprehension model can be expressed as consisting of two different areas. First: The mental representation, where concerns are structured in categories and mapped to source code. Second: A continuous study process that influences the mental model and the comprehension. Vice versa, the current state of comprehension influences the actions of the study process.

However, we see the model not as alternative or as independent from other program comprehension theories. Related work showed already that our model does not stand orthogonal to previous research. Though, our model bridges an approved research area in psychology with program comprehension. In contrast to previous source code study models, we respect the idea that SOC plays a role in program comprehension and relate program comprehension to psychology. We introduce the idea of concerns and basic concerns, which are created on usage frequency and importance. The occurrence of multi-concern fragments is explained by cross categorization. Compositional concerns explain the creation of new concerns. Different category types and kinds (section 3 C and D) introduce the idea of similarity to different means to apply SOC. Former models did not respect this facts together. Thus, future work needs to compare the program comprehension with categorization in detail.

Our model enables further research about the role of categorization in program comprehension. Through the model several areas are identified where research needs to be done. For instance, we assume a difference in comprehension of normal and basic concerns. Investigations need to verify this assumption. Such investigations can uncover methods to detect such basic concerns automatically. This can help developers to investigate a program. Also, the psychological experiments about categorization need to be studied in detail and compared with manifestations in source code. This can lead to new concern revealing techniques. Generally, we see our model as a holistic view of the developer source code investigation interaction. It can be used as starting point to investigate the distinct areas further. For example categorization theories can be inspected to be leveraged to compare programming languages.

Finally, the model proposes a comprehension process. Modern development environments offer plenty of tracking functionalities where a developer clicks. We believe, future program investigation tools need to construct a category model of a developer through behavior tracking. Our presented model can serve as starting to develop such a category model.

All together, we see the emerging need to compare categorization in more detail with software development.

7. Acknowledgements

This work is (partly) funded by the German Ministry of Education and Research within the project ViERforES-II (grant no. 01IM10002B). Thanks go to Manuela Fath and the anonymous reviewers for their useful comments on earlier drafts.

8. References

[1] T.M. Shaft and I. Vessey. "The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension", Information Systems Research, pp. 286-299, 1995

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

[3] M. E. Crosby, J.Scholtz, S. Wiedenbeck, "The roles beacons play in comprehension for novice and expert programmers". In Proceedings of the 14th Workshop of the Psychology of Programming Interest Group. 2002

[4] C. Aschwanden and M. Crosby. "Code scanning patterns in program comprehension". In Proceedings of the 39th Hawaii International Conference on System Sciences, 2006

[5] M.-A. Storey, "Theories, Methods and Tools in Program Comprehension: Past, Present, and Future". 13th IWPC, pp.181-191, 2005

[6] E. Soloway, K. Ehrlich. "Empirical Studies of Programming Knowledge". IEEE Transactions on Software Engineering, 10(5), pp. 595-609, 1984

[7] B. Shneiderman, R. Mayer. "Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results". International Journal of Parallel Programming, 8(3), 1979

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

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

[10] D.C. Littman, J. Pinto, S. Letovsky, E. Soloway. "Mental Models and Software Maintenance". In Empirical Studies of Programmers: First Workshop, pp. 80-98, 1986.

[11] F. Detienne,"Software Design – Cognitive Aspects", Springer-Verlag London, 2002

[12] V. Rajlich, N. Wilde."The role of concepts in program comprehension." In International Workshop on Program Comprehension, 2002

[13] B. Cleary, C. Exton, "Assisting Concept Assignment Using Probabilistic Classification and Cognitive Mapping". In Proceedings of the 2nd International Workshop on Supporting Knowledge Collaboration in Software Development, 2006

[14] B. Cleary, C. "ExtonAssisting Concept Location in Software Comprehension", 19th Annual Workshop of the Psychology of Programming Interest Group, 2007

[15] P. Tarr, H. Ossher, W. Harrison, S. M. Sutton, Jr.. "N degrees of

separation: Multidimensional separation of concerns". In Proceedings of the 21st International Conference on Software Engineering, pp. 107–119. 1999.

[16] 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.

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

[18] J. A. Bloch, "A Metadata Facility for the Java Programming Language", 2004.

[19] G. Kiczales, E. Hilsdale, et. al. „An Overview of AspectJ". European Conference on Object-Oriented Programming, pp. 327-355. 2001

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

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

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

[23] M. Fowler, "Patterns of enterprise application architecture", Addison-Wesley, 2003

[24] S. M. Sutton Jr., P. Tarr. "Aspect-oriented design needs concern modeling", Workshop on Aspect-Oriented Design 1st International Conference on Aspect-Oriented Software Development, 2002.

[25] A. von Mayrhauser, A.M. Vans. "Program Comprehension During Software Maintenance and Evolution". Computer, 28(8), pp.44-55, 1995.

[26] R. Brooks. "Using a Behavioral Theory of Program Comprehension in Software Engineering". Proceedings of the 3rd international conference on Software engineering, pp. 196-201, 1978

[27] N. Pennington. "Stimulus structures and mental representations in expert comprehension of computer programs". Cognitive Psychology, 19, pp. 295–341, 1987

[28] B.P. Lientz, E.B. Swanson. "Software Maintenance Management". Addison-Wesley, 1980

[29] W. L. Hursch, C. V. Lopes. "Separation of Concerns". Technical report by the College of Computer Science, Northeastern University, Boston, 1995

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

[31] G. Kiczales, M. Mezini :"Separation of concerns with procedures, annotations, advice and pointcuts". In Proceedings of 19th European Conference on Object-Oriented Programming, 2005

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

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

[34] M. McCloskey, S. Glucksberg." Natural categories: Well-defined or fuzzy sets?", Memory & Cognition, 6,pp. 462-472. 1978

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

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

[37] Steven A. Sloman, Categorical Inference Is Not a Tree: The Myth of Inheritance Hierarchies, Cognitive Psychology 35, 1–33 (1998)

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

[39] 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

[40] R. Johnson, "Expert one-on-one J2EE design and development", Wrox 2003

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

[42] S. Nguyen. "Cross-classification and category representation in children's concepts". Developmental Psychology, 43, pp. 719-731, 2007

[43] L.W. Barsalou, "Ad hoc categories", Memory and cognition 11, pp. 211-227. 1983

[44] B. H. Ross, G. L. Murphy, "Food for Thought: Cross-Classification and Category Organization in a Complex Real-World Domain", Cognitive Psychology 38, pp. 495–553. 1999

[45] T. Yamauchi, A. B. Markman, "Inference Using Categories", Journal of Experimental Psychology: Learning, Memory, and Cognition, Vol. 26, No. 3, pp. 776-795, 2000

[46] M. F. Verde, G. L. Murphy, B.H. Ross, "Influence of multiple categories on the prediction of unknown properties", Memory & Cognition, 33 (3), pp. 479-487, 2005

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

[48] D. Ratiu, F. Deissenboeck, "How Programs Represent Reality (and how they don't)", Proceedings of the 13th Working Conference on Reverse Engineering, IEEE, 2006

[49] M. P. O'Brien, "Software Comprehension – A Review & Research Direction", Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report, 2003

[50] C. B. Mervis , M.A. Crisafi. "Order of acquisition of subordinate-, basic-, and superodinate-level categories". Child Development, 53, pp. 258-266. 1982

[51] M. P. Robillard, W. Coelho, G. C. Murphy, "How Effective Developers Investigate Source Code: An Exploratory Study", IEEE Transactions on Software Engineering vol. 30, No. 12, 2004

[52] S. M. Sutton, Jr, I. Rouvellou. ".Modelling of software concerns in Cosmos". Proceedings of the 1st international conference on Aspect-oriented software development. 2002.

[53] S. M. Sutton, Jr, I. Rouvellou. "Concern Space Modeling in Cosmos". OOPSLA Poster Session, 2001

[54] S. M. Sutton Jr. "Early stage concern modeling", Early Aspects Workshop, Held with AOSD, 2002

[55] W. Chung, W. Harrison, V. Kruskal et al.. "Working with Implicit Concerns in the Concern Manipulation Environment", AOSD '05 Workshop on Linking Aspect Technology and Evolution. 2005.

[56] A. S. Kaplan, G. L. Murphy, "Category Learning With Minimal Prior Knowledge", Journal of ExlXaimental Psychology: Leraning, Memory, and Cognition , vol. 26, No. 4, pp.829-846, 2000

[57] G. C. Murphy, M. Kersten, L. Findlater. "How Are Java Software Developers Using the Eclipse IDE?" IEEE Software, vol. 24, No. 4, 2006

[58] M. Kersten. "Focusing knowledge work with task context". PHD thesis, University of British Columbia, Canada. 2007

[59] 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

[60] Frey, M. Gelhausen. "Strawberries are nuts". CHASE '11 4th international workshop on Cooperative and human aspects of software engineering. page 49. ACM. 2011

[61] E.Gamma, R.Helm, R.E. Johnson, J. Vlissides. "Design Patterns. Elements of Reusable Object-Oriented Software", Addison-Wesley; 1st ed., 1994

[62] E.E. Smith, D.N. Osherson,L.J. Rips, M. Keane."Combining prototypes: A selective modification model".Cognitive Science,12, pp. 485–527. 1988

[63] G.L. Murphy. "Comprehending complex concepts". Cognitive Science, 12, pp. 529–562.1988

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