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 a starting point 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 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 developer's 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 anything 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 concerns. Furthermore, 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. 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, 65]. 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 respects research about categorization and SOC. Hence, we enrich program comprehension through the direct association with

PLATEAU'11, October 24, 2011, Portland, Oregon, USA.

[1] We assume a reading familiarity with the idea of SOC [21]. 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.

an area of psychology. We enable future research about the detailed role of categorization in program comprehension.

First, we give 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 arguments against it. Related work shows similar approaches and differences to our work. Finally, we draw 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 to use knowledge of a category. New objects are associated with a category and the knowledge of 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 make predictions of 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 do not exist as physical objects [47]. For instance, democracy represents a conceptual ideal of government. This way, categories classify abstract functionality for a democratic system directly. Likewise, it is possible that categories of physical objects are used to deduce functionality about category members. Thus, no simple distinction between categories and functionality of those can be made. However, categorization is a dynamic process which updates the knowledge of categories, their members and their relations constantly throughout new experiences. Thereby, even minimal prior knowledge has the effect of greatly speeding up learning of category [56]. Likely, there are basic categories, representing important and often used elements, enabling faster processing and association [50].

One feature of categorization is to make quick predictions [46]. These can be derived from limited available information, but can lead to avoiding understanding of 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 beware of a snake. As the examples show, predictions are used for acting adequately or for adapting 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. Thematic categories group objects that are associated to each other or have a complementary relationship like a dog leash or a clothesline [41]. We humans use taxonomic, script and thematic categories to equally categorizing and understanding objects [40]. This means that every kind of category is used for categorizing objects while none of them is favored. It is also possible to combine different categories and to generate new categories [62, 63]. Also categories can be structured hierarchically. Subcategories have features of their super-ordinate in a certain probability [37].

Categories can be formed spontaneously to fulfill a certain task. These are 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 different / various categories at the same time. This is called cross-classification. The context an object is seen in influences which category it is associated with [39]. Therefore, the objects categories can be different kinds of category (e.g. the formerly mentioned taxonomic and thematic) [41, 42]. This is important, because research indicates that the speed of association with a category differs with regard to the different kinds of category [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 possess equal quality in relation to the respective category [33]. According to results of various experiments it seems that categories cannot be clearly defined, and categories are not exactly distinguishable [22, 34]. Therefore new theories have been developed.

The prototype theory claims that categories are represented by a bundle of characteristics, which are typical for a certain category, but not in any case inevitable or sufficient [35]. A category “bird” might have characteristics like “flying” or “building a nest”. Even if not all birds show this features, they still belong to the category “bird”. For instance, a penguin is a bird that cannot fly. The prototype of a category merges typical characteristics of a category, but no exemplar has to match them completely by showing all characteristics. New objects are classified by comparing them 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 about them, 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 contrast to the prototype theory, single exemplars are engrained together with the category denotation. Each new exemplar represents a category of its own. By recognizing a new exemplar a learner is reminded, more or less, of formerly seen exemplars. The learner assumes that an object might have the same features as the exemplar compared to which it has the most similarities. Thus comparisons of similarities are made with the exemplar in question 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 in the same category, because it reminds of a squirrel or a chipmunk. Research certifies some prognoses that were made on the basis 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 with the area of programming are explained. Thereby, similarities between SOC and categorization emerge. Finally, we show a model containing all the facts.

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, methods, annotations and similar means. Other concepts are realized by compositions of various programming constructs. Design patterns are examples of such compositions [61]. Developers associate the elements of realizing concepts with the corresponding design pattern. The functionality of these elements has discrete, flexible characteristics whereas not every implementation is completely equal to another one. Thus, the category of a design pattern is more like an abstract concept than a concrete mechanism. Categories manifesting in code are similar to physical object categories and abstract concepts like democracy. Hence, there are concrete mechanisms in programming to realize categories and composition mechanisms to realize elements of conceptual categories.

The comprehension of fragments seems to be dependent on the experience [51], and thus probably on the knowledge of a developer. That is also the case with categorization. The learning of categories and also speed of comprehension differs, depending on prior knowledge [56, 50]. We find supporting arguments in research about program comprehension: Some source code lines are obviously more important than others [3]. We see these lines as a representation of categories or features of categories. To categorize the fragment, features (i.e. 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 the decision whether it is recognized as a basic concern. This distinction is made to acknowledge the existence of basic categories. As with categories, the comprehension of basic concerns is gained quicker than of normal ones. We consider framework classes as an example for often used elements. Therefore, we assume that framework classes are an example of basic concerns.

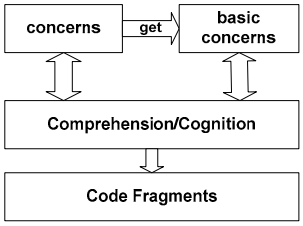

Figure 1. Learning basic concerns

We present 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 increases. This means: Just studying the source code can lead to classifying a concern as an important one and it can be understood faster when it is occurring again. We visualize the continuous usage of concerns for comprehension as well as their update by double-sided arrows. Additionally, the importance of a concern and its frequency of usage is refining a firstly recognized normal concern into a basic concern. No knowledge of the source code exists, when a code study begins. But as the knowledge grows, certain concerns can get important ones. We present this transfer by an arrow from ‘concerns’ to ‘basic concerns’.

3.2 Usage of prior knowledge for prediction

Developers predict functionality of source code with the help of prior knowledge. This is obvious, because often recurring elements are used in a program. It is assumed that a developer creates hypotheses about the functionality of a program by using 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 a hypothesis about a program is equal to predictions of category membership [46]. Hence, the prediction about the functionality comes from 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.

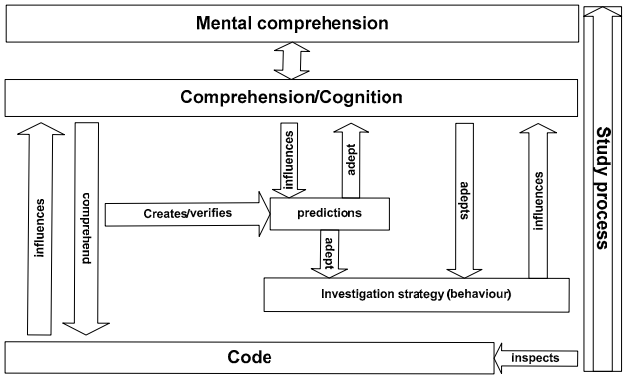

Figure 2. Prediction and behavior

We developed visualization for the process of predictions and behavior. We present it in Figure 2. With 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 actions shown 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 code and thereby the cognition is changed. Comprehending code leads also to predictions of 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 newly studied elements will be changed. Anyway, sometimes wrong or correct predictions are made. In such an ongoing process of comprehension, predictions can be verified as being wrong or right. It is important to note that this strategy does not claim to be verifying predictions continuously. A direct verification is probably happening sooner, when uncertainness about the predictions is at stake. Likely, in a normal study process, the predictions are, in a first step, verified by the assumption that they are true, but further investigation might support the falsification of them.

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 be changed again. For example, the recognition of a code fragment can lead to predictions about its functionality. Further investigations can then lead to supporting or falsifying the previously made assumptions. Because of this, the knowledge of the concern has to be updated. This leads to new predictions that can lead to a change of strategy 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 describe a few examples at random. This way, we show a relation between categorization theories and means to distinguish concerns.

A means 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 view of 'category'. 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 by 'persistency' annotations. All marked persistent classes belong 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 sets of features. Obviously, no class needs to contain all of the persistency annotations. Thus, we recognize that the classic view is sufficient to represent this kind of categorization. So, we suggest the prototype theory as an apt explanation for this phenomenon. A prototype of a layer assembles all the different features associated with the layer. This way, an abstract prototype gathers all persistency annotations. Classes are annotated with the annotations that occur in the prototype. Each class has to have only 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 separate module. If this was possible the result would be modules with no features shared with others. So, we recognize the necessity of concluding that the prototype theory is not sufficient to explain SOC. Advanced mechanisms of applying SOC support this assumption: Aspects [19] are applied to code elements and allow modifying their behavior. All that 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 in all, the advanced separation

mechanisms seem to be beyond the prototype. They seem more comparable to totally distinct exemplars.

All over, we see source code and the means of separating concerns as being similar to the different theories of how categories are arranged. Our main point is that means of separate concerns are similar to categorization theories; elements of source code are qualified and grouped into categories through them.

We developed a comprehension process and present its visualization 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 of separating 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 pointing to the mapping box, indicates the constant mapping refinement during the procedure of comprehension. In the mapping block the different circles and arrows show different variants of mapping. We symbolize this mapping by 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. The reason for this are limitations in separating concerns in the source code and mistakes of programmers.

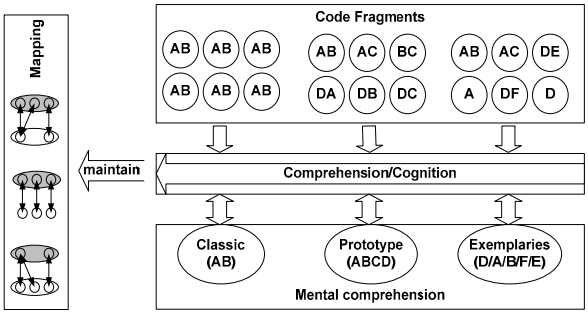

Figure 3. Concern theory mapping

The arrow across the comprehension box symbolizes the continuous process of comprehension and shows that concerns will be recognized differently. At their first occurrence they will probably be recognized as a single exemplar. For instance, a class is seen for the first time. Since a developer does not know other occurrences of this exemplar just a single model is created as mental representation. As more and more code fragments appear, using the class, a developer is reminded of the other fragments. This way, an abstract prototype is used for representing this mentally (prototype theory). The developer assumes that diverse reoccurring features can be used for discriminating the category. If all the features of the studied objects are the same, a category following the "classic view" can be used, instead of an abstract prototype. Thus, it depends on the features if the category of reoccurring elements is based on the prototype theory or the classic view. We show this use of formerly seen exemplars with the double ended arrows. Additionally, we show exemplary code fragments, where the letters describe features of the fragments. Vertically, at the bottom, the different corresponding categories are 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. 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, the association of a source code fragment with multiple concerns is the same as cross-classification [41, 42].

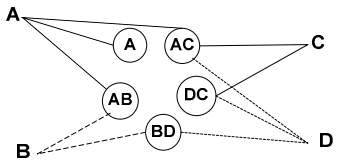

Figure 4. Cross concern association

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

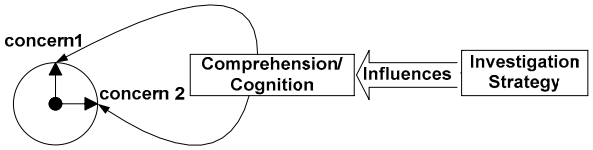

Figure 5. Cross concern comprehension

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

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. The framed spaces represent the concerns. As shown, a fragment is framed by multiple lines to indicate the multi-concern fragments. In short, a fragment is consisting out of various concerns.

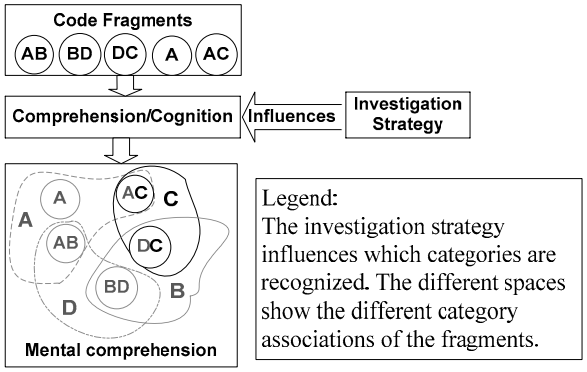

Figure 6. Cross concern association comprehension

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 in the persistency layer. At the same time these classes are composed of domain objects that normally represent business entities. Therefore, it is wrong to associate a DAO only with 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 domain concern and a concern of persistency.

Developers often deal with abstract definitions and concrete implementations. We believe that a class can represent a concern. But when it is used in this way (e.g. as instance object) then it is like a feature of another element. This reveals problems of distinction between a concern itself and its instances. The different categorization theories offer 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 by comments or a meaningful name of the 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 a 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 most probably not be recognized as a concern of its own; likely it will be comprehended as pure composition, but not as a concern.

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. Maybe VC2 will not be recognized as a concern itself because it is not used anywhere else.

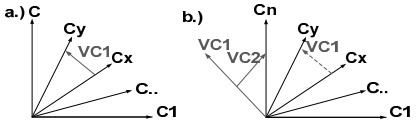

Figure 7. Concern composition

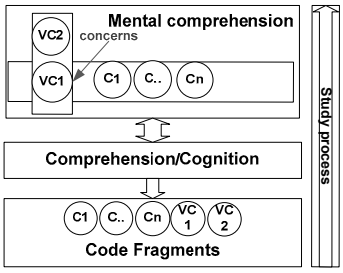

Figure 8. Concern composition comprehension

Anyway, we see this as a 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 like VC1 because of their reoccurrence in code. This is supported by the prototype theory. There, categories are learned via the occurrence of elements that are similar. Developers recognize the occurrences of a composition as category members. Thus, every occurrence of the composition in use is associated with the composition itself and the composition evolves into its own category.

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 by the vertical block. All normal concerns (C1…Cn) are arranged horizontally. VC1 is a member in the horizontal and the vertical block at the same time. We do this out of respect for the previously discussed fact, that a composition can be recognized as a normal

concern, as well. Anyway, it has to be made clear that there can be multiple levels of composition and this crossing has to be seen as an example of one of them. Developers build up the knowledge of the compositions throughout 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 of it would be updated.

3.6 Putting it all together

We present the concern study process in Figure 9. The cycle expresses the continuous process of comprehension of a program. The arrow visualizes how the knowledge of a developer is built up and comprehension grows. The direction of the arrow also indicates the permanent level of knowledge. We do this to acknowledge that concerns can be 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 basic concerns. Such basic concerns have a high frequency of usage or a high importance, as described in section 3.1.

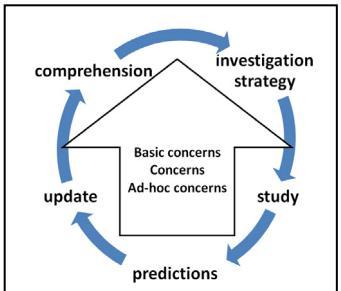

Figure 9. Study process

Figure 10 assembles all facts, from section 3.1. - 3.5. 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 elements in the range of the arrows are affected by the process of comprehension. The study process crosses cognition to indicate the consistency of the comprehension throughout a study. Vice-versa, the comprehension process crossing 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 occur.

The mental comprehension contains different representations of the concerns. They are updated during the study process and influence the cognition. Mappings

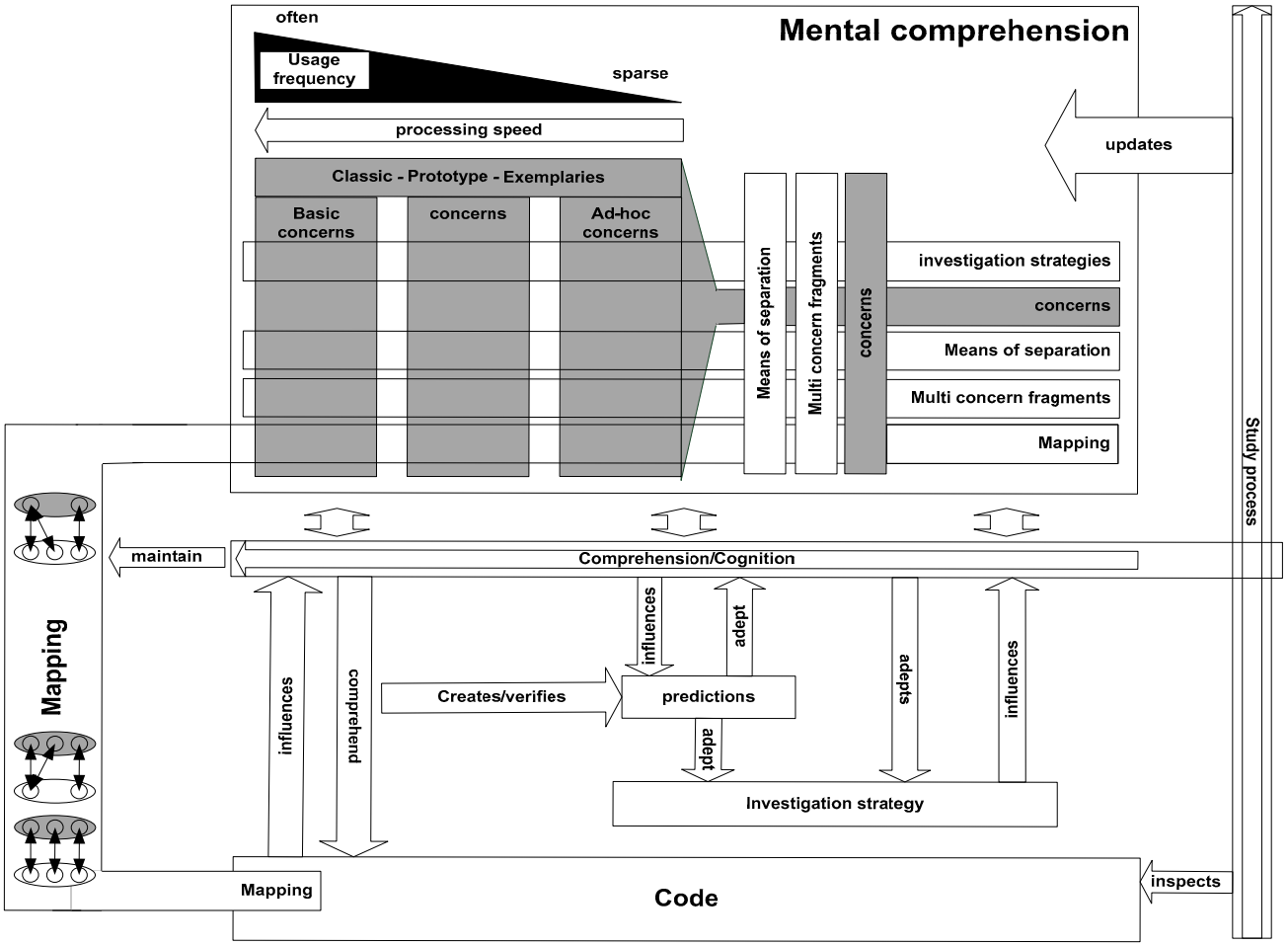

Figure 10. Holistic comprehension model

between the mental representation and the concrete program elements exist and are maintained during the comprehension processes. The concerns are represented by the various theories (section 3.3). Below, the concerns are separated into the different kinds of basic, normal and ad-hoc concerns. The ad-hoc concerns are used for a specific task as 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 kind of concerns symbolize the different other facts from section 3. The grey vertical boxes represent the different types of concern (exemplary, basic, normal). The concern box is colored in grey to indicate it is a simplification of the different theories that are also colored in grey. The block that is showing the Means of Separation indicates that a concern can also be associated with the way the separation as it is manifested in the source code. The block named Multi concern fragments, indicates that a fragment can be associated with multiple concerns at the same time. We show a block of investigation strategies. It was added to Mental comprehension, because we assume that the experience of a developer influences investigation strategies. Successful investigation strategies are somehow associated with concerns. Thus, there must be an association between investigation strategies and discovered concerns. Research [51] supports that investigation strategies vary depending on different skills of developers.

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

In Figure 3 mappings of source code in relation to the mental model are shown. The "Mapping" has been extended to visualize that they can also be associated with all the previously discussed elements and that they are to be seen 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 comprehension. Like before, the small symbolization in the mapping bar indicates that different mapping variants exist.

As discussed in section 3.2, different actions may take place. The investigation strategy is adapted. Predictions of functionality of source code fragments being studied are either verified or falsified and the mental knowledge is created and updated. Generally, all the arrows represent the same relations as described in the whole section.

4. Threats

Several threats have been identified that could corrupt the model. We show and name these facts to discriminate uncertain pretenses from well-known facts for validation by upcoming research.

The issue stated in [51], that effective programmers have different investigation strategies compared to others, 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 turn out 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.

Additionally, there is no complete model of the human mind available and there are only theories of how it works. This way, only empirically supported research can be considered as delivering undisputable facts. Therefore the different theories, exemplary, classic and prototype and also hybrid ones may be wrong. This could affect the comparison of these theories with the means of separating concerns. We argue that these theories are based on empiric research and parts of them have been verified. Our comparison was only on a very basic level and already showed similarities. Also, the comparison between the theories and means of separating concerns is only a small part of the model. Even if all of these theories are proved wrong, only a small part of our model would be corrupted.

The creation of categories just out of the mere composition of different categories can be used for arguing against our model. We induce the fact that psychological research appears to go in the direction of cognitive concepts that are used draw inferences about the combination of categories [36]. We state that compositions are a concept. This way, our model holds its own for the specific case of composition and is not corrupted by a generalized conceptual combination of categories.

5. Related Work

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

In [6], the comprehension is gained by the use 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 base of knowledge, 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 reached by applying the knowledge base to the program. Again this model does not differ very much from our approach. For instance, basic categories are comparable with the experience of a developer. Again, the main difference is that neither SOC nor research on categorization is mentioned.

Another model [10] describes program comprehension based on strategies. Again, neither the idea of categorization nor the impact of SOC in the comprehension process and in the mental model is mentioned.

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

In [48] the problem of semantic defects in a program is proposed to be solved by an ontology. Thereby, the problem is discussed how concepts in the mind of a programmer are mapped to program elements. This is similar to our approach, because we come up with explanations how a developer comprehends a program.

6. Conclusions and Future Work

We applied research of psychology to develop 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 is consisting out 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. Instead, our model links an approved research area in psychology with program comprehension. In contrast to previous comprehension 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 based on their 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 3.3 and 3.4) introduce the idea of similarity to different means of applying SOC. Former models did not take all these facts into consideration. Thus, future work needs to compare in detail program comprehension with categorization.

Our model enables further research on the role of categorization in program comprehension. With the help of our model, several areas can be identified where additional research is needed. For instance, we assume a difference in comprehension of normal and of basic concerns. Investigations need to verify this assumption. Such investigations can discover methods of detecting such basic concerns automatically. This can help developers to investigate a program. Also, the psychological experiments on categorization need to be studied in detail and compared with manifestations of concerns 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 for further investigation of the different areas. For example, categorization theories can be inspected to be leveraged for comparisons of programming languages.

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

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

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 M. Fath, F. Romer and the anonymous reviewers for their useful comments on earlier drafts.

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] T. 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

[65] T. Frey, M. Gelhausen, H. Sorgatz and V. Köppen. On the Role of Human Thought – Towards A Categorial Concern Comprehension. In Proceedings of the Workshop on Free Composition (FREECO-Onward!).ACM. 2011.