# Hypermodelling for Drag and Drop Concern Queries

Tim Frey

Institut für Technische und Betriebliche Informationssysteme
Otto-von-Guericke University
Magdeburg, Germany
tim.frey@tim-frey.com

**Abstract:** Imagine a developer, who wants to alter the service layer of an application. Even though the principle of separation of concerns is widespread not all elements belonging to the service layer are clearly separated. Thus, a programmer faces the challenge to manually collect all classes that belong to the service layer. In this paper we present the Hypermodelling approach that can be used to query the necessary code fragments that belong to the service layer. We utilize methods and ideas from Data Warehousing to enable this functionality. Slicing and dicing the code in this new way overcomes the common limitation of having only one viewpoint on a program. The service layer can now be uncovered just via a query.

## 1 Introduction

Programmers spend most of their time reading and navigating source code [1]. Often developers alter an application layer. Usually the developer tediously gathers the artifacts that belong to the desired layer. This is a typical instance of the concern retrieval problem that programmers face commonly. Locating concerns in software is indicated to be a challenge for software developers [2,3,4]. Probably the main reason why locating concerns in source code is so difficult is because of their intertwining and the difficulty to apply separation of concerns (SOC) [5]. This principle addresses the goal to encode functionality into modules that have one primary responsibility. Normally, not all concerns are packed into separate modules and therefore a module encodes multiple functionalities at the same time [6].

Modern development environments allow building working sets, consisting of one or more fragments corresponding to the various interests for a developer [7]. Such working sets need to be built manually by a developer and can be used to filter the navigation just for the elements belonging to the working set. Typical for working sets is to split the application into their distinct layers and organize those as different working sets. This concurrent view of elements belonging to a working set is beneficial for developers by allowing them to navigate between the various fragments. But, still, the challenge of code investigation to build the working sets exists. For instance, when new classes are added to a layer, they need to be associated with the corresponding set.

To support programmers in investigating source code beyond working sets several tools are available for concern analysis or query operations [1,8,9]. The query tools offer complex and advanced query languages. This leads to the fact that composing a query is quite complicated [8]. It is indicated that a few query tools seem not to be enhancing productivity. This could result from the complexity of the tools [8].

In order to overcome the above mentioned limitations of manually creating working sets and complex queries, we present the Hypermodelling approach for the development environment. Hypermodelling enables developers to design concern queries without the need to learn a complex query language like it is used in other tools. Developers can use known views and elements via drag and drop in their queries. The same view wherein the working sets are shown is used to present the result of a query. This way, we avoid dealing with a custom query result representation view, like other tools use it. Developers can now do queries for the various layers of an application without the need to organize them into working sets.

Our contribution consists of two points: First, we show a tool that enables developers to drag and drop a query from well-known views and concerns. The result is also presented in a well-known view to enable a good usability for developers. And second, we address the complex query problem by utilizing the Hypermodelling approach for the development environment. This allows developers to design queries for code slices and their combinations.

In prior work, we described the Hypermodelling approach [10,11]. In this paper, we transfer the idea to the area of Integrated Development Environments (IDE). We present the Hypermodelling approach on an abstract level and use it to build a prototype. Also, a video is available, showing the tool in action.[1]

The paper is organized as follows. First, we present the Hypermodelling approach. Then, we describe a prototype and its architecture. Afterwards, we present a preliminary evaluation by showing a few sample use cases. Finally, related work is described and compared to the prototype, conclusions are done and future work paths are explained.

## 2 Hypermodelling

In this section we recap the most important facts of Hypermodelling. A more detailed description can be found in our earlier work [10,11]. Hypermodelling is founded on the idea to combine mechanisms to separate concerns and Data Warehouse (DW) like queries. Several mechanisms exist to separate concerns at the source code level. In this paper we focus on Metadata annotations [12] and normal object-oriented programming features. First, we describe an example of source code how concerns are separated. We use annotations as example for other programming mechanisms, because they are straightforward to understand. Then, we show the similarities to DW data structures and present the Hypermodelling approach.

[1] http://eclipse.hypermodelling.com

We present sample source code in Listing 1. There, a *Customer* class and a Data-Access-Object (*CustomerDAO*) are shown. The *CustomerDAO* class extends a helper class (*DaoSupport*), for table access. This is commonly done when frameworks are used. Both classes make heavy use of annotations. We show exemplary concern associations through the boxes. The elements are associated with various concerns at the same time. For instance, the *Customer* class is associated with the entity concern that belongs to the persistency part of Java. At the same time the *Customer* class is marked as deprecated. Hence, every programmer associates it with the concern that it should not be used any more by other code. Another example is the *createCustomer* method where compile errors are suppressed. The *createCustomer* method is consuming the deprecated *Customer* and, thus, a deprecated element consumer.

```
1: @Entity                                          Entity concern association
2: @Deprecated                  Deprecated concern association
3: class Customer{
4:         @Deprecated                              Deprecated concern association
5:         Customer(){
6:         …}
7: …}
8: class CustomerDAO extends DaoSupport             DAO concern association
9:
10:        @SuppressWarnings("deprecation")  SupressWarnings  concern association
11:        Customer createCustomer(){        Deprecated element consume
12:                return new Customer();            concern association
13:        }     ...}
```

**Listing 1. Example for annotated source code and associations with concerns**

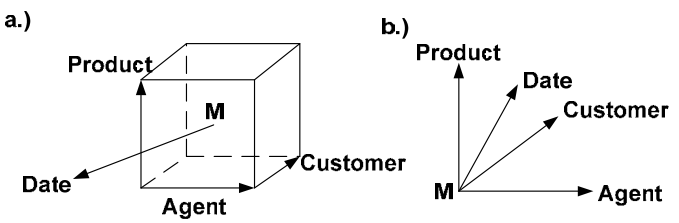


**Figure 1: Multi-dimensional data within a Data Warehouse**

DWs and associated techniques are used to analyze multi-dimensional data. Normally, an Online Analytical Processor (OLAP) is used to query multi-dimensional data that is stored within a DW [10,11]. We show an OLAP data structure in Figure 1a. There, the measure revenue (M) is associated with different dimensions at the same time. Revenues can be viewed by different associations with dimensions. Such associations are a product, date, a customer and an agent. Each one of those dimensions is describing an aspect of a product sale. The agent that made the deal, the bought product, the customer

that acquired it and the date of the deal. Out of these associations queries can determine the revenues for the diverse dimensions. This way, the revenues for a specific product or the revenues for a specific customer can be determined. Likewise, it is possible to discriminate through the dimensions. A sample for such discrimination can be the computation for revenues at specific date in combination with a specific product.

Figure 1b shows the cube abstracted to a pure vector model. This can be compared to the concern associations of a code fragment in Listing 1. Thus, in Hypermodelling a source code fragment is associated with multiple concerns at the same time like a measure is associated with dimensions in a DW. Hence, the main idea is to leverage principles and technologies from the area of DW to advance source code analysis.

**Table 1. Concerns of Listing 1**

| Element / Dimensions | extends DaoSupport | @Entity | @Deprecated | @SupressWarnings |
|---|---|---|---|---|
| Customer | | X | X | |
| Customer.Customer() | | | X | |
| CustomerDAO | X | | | |
| CustomerDAO. createCustomer() | | | | X |

We present a more detailed concern association example in Table 1. The table shows an alternative representation of Listing 1 and its associations. The rows are representing source code fragments and columns representing concerns. The columns show that the concerns are source code fragments themselves. Hence, the table shows the relations in the source code. The "X" indicates that a fragment is associated to a concern. For example the constructor of the class *Customer* is marked *@Deprecated*. Likewise the rest of the table-listing associations can be done.

We define: All the code associated to a specific concern is forming the corresponding concern slice. A concern is, like described before, similar to a dimension in a DW. Our approach is that similar queries to the ones used within a DW can be executed on source code. Such queries can define slices of various concerns.

Furthermore, combinations of slices can be used to discriminate results. This is needed because a slice is containing all the corresponding code fragments that belong to that concern. But, code fragments are often encoding multiple concerns at the same time. Thus, code fragments belong to multiple slices at the same time. This can also be seen in Table 1. For instance, Customer belongs to the slice Entity and also to the deprecated

slice. Therefore, queries can address combinations of slices. A sample query for Table 1, resulting in the Customer class can be: All fragments that belong to @Entity “and” to @Deprecated at the same time.

## 3 Implementation

The Hypermodelling tool is implemented by a custom query processing engine. This was done to show the application of Hypermodelling approach within the IDE before implementing DW techniques within the IDE. We inspect source code files in real time within a query. This has the advantage that new concerns in altered source code files are detected when a new query is executed. Nevertheless, the real time detection and the custom query engine have several limitations. Within a DW queries and analysis for slowly changing dimensions are possible. These dimensions are hard to indentify in source code. Our tool shows the first application of multi-dimensional queries within the IDE to investigate such scenarios at a later point. Additionally, the query speed slows down for larger code bases, because every file is inspected in real time. However, DW technology offers plenty of functionally to process queries faster. For instance, DW technology is capable to do pre calculations and aggregations or query analyzers to speed up queries [22]. Our earlier work [11] shows an example using actual DW technology to perform queries on code. This can be used as a way to speed up the tool with DW technology in the future. In this paper, we focus on how the Hypermodelling approach can be applied within the IDE to ease the work of a developer.

Next, the reasons to choose the different views for query composition and result presenting are described. Finally, we explain how a query is processed.

### 3.1 Views

As described before; programmers spend most of their time in exploring source code and it is indicated that structured code investigation is effective [13]. In the case of Eclipse the most used tool is the package explorer [14], which shows a hierarchical structure of projects, packages, classes and methods. Hence, we assume that it is the tool most used for source code exploration. However, composing a query in current query tools is difficult [15]. Therefore, we put some effort into developing an innovative way of presentation for our implementation. The goal was to respect the way programmers use Eclipse and allow structured investigations of code.

We support well known views as drag sources to allow composing a query. This avoids learning. Therefore, a developer can compose a query by dragging elements from known views into the query view. Hence, the concerns defined in the source code, libraries and in the various tools can be added to a query. As addition and as an alternative to this drag source, we realized a view where well known concerns are listed. We encoded a few of those for demonstration issues. The intention for this view is to enable a systematic query creation. Additionally, we can offer combinations of concerns as concern units in this view. We call this view the pre-defined concerns, because standard concerns can be

offered this way.

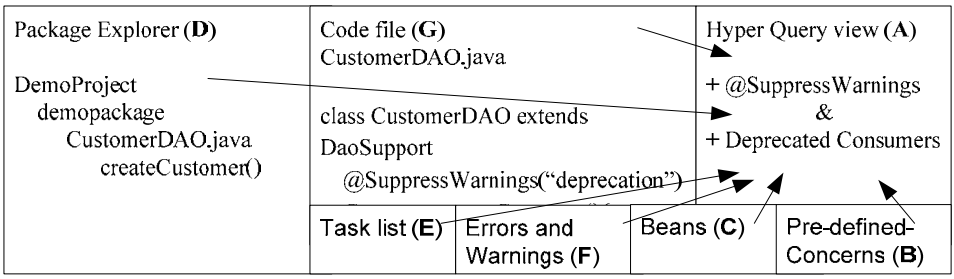


**Figure 2: The Hypermodelling Tool**

We use the package explorer (D) to present the results of a query to a developer. We made this choice to avoid that programmers need to learn a new view. We can justify this because it is one of the main tools that developers use for code exploration. Hence, it seems a good fit since queries are also used for exploration. For us, information reduction is a central element to inspect query results effectively. Thus, the presentation in the package explorer (D) is based on a filter that just shows the elements belonging to the result. Figure 2 presents distinct common used views in Eclipse as a schematic overview. The arrangement of the package explorer (D) on the left, the code file (G) in the middle and other views (E,F,C,B) at the bottom and right (A) is quite common. The important fact is that the Hypermodelling view can be placed anywhere. The various views serve as drag sources (B,C,D,E,F,G) to drop concerns into a Hypermodelling query (A). The arrows visualizing that concerns can be dragged and dropped from the various views into a query. For a better comprehension, we uploaded a screenshot of the Eclipse that shows the different views.[2]

### 3.2 Tool Usage

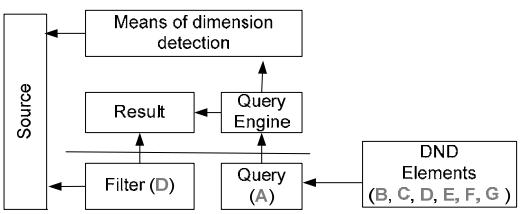


Figure 3 shows how the tool works. The letters (A-G) correspond to the ones Figure 2. Concerns can be selected via dragging and dropping from the various views (B,C,D,E,F,G), into the query view (A).

**Figure 3: Tool Operation**

In B we offer well-known concerns, like various annotations or compiler outputs, to a developer. Furthermore, we offer popular concerns like inheritance. A programmer can just add the extenders and implementers of classes or interfaces as concern slices to a

[2]http://eclipse.hypermodelling.com

query. A generic template for inheritance and also commonly used super classes are offered in this view. Furthermore, we support other well-known concern kinds in this view. For instance, fragments that contain unsafe type casts can be added to a query. Even a template for a string in a class or method name can be added to a query. Generally, the idea of the pre-defined concerns view (B) is to enable a fast access to often used concerns.

The other views (C,D,E,F,G) show normal views of the IDE from which concerns can directly be dragged and dropped to a query. The tool supports various code elements as concerns on a generic base. This means that, e.g. new annotations just can be dragged and dropped out of a code fragment into a query. This way, we enable the addition of annotations that are defined in programming libraries and in the code itself to a query. This is possible, because annotations in Java are defined through a language mechanism. This has the advantage to present a subsidiary possibility to add concerns to a query by using familiar views. This way, it is possible to populate concern data of an actual investigated program into queries.

View C shows a bean view of the IDE where classes that are defined as beans are shown. E shows Mylyn Tasks with which different fragments may be associated. D shows the normal package explorer. From that, Java Elements or groups of Java elements can be added to the query. This makes it very easy to include or exclude elements in the view from a query. F shows the error and warnings view as drag source. From G, the editor, annotations can be added to a query.

The query view (A) shows the Concern slices that are included("+") and excluded("-") in the query. For example, from Listing 1, the @Entity slice would consist of the class Customer. The @Deprecated slice contains createCustomer(). Another result for the query of "+" DeprecatedConsumer is createCustomer(). So, if "-" SuppressWarnings is added an empty result would be the outcome. For this reason it is possible to tie slices with an "AND" together which we visualize with the "&" in the figure. The query +DeprecatedConsumer & + SuppressWarnings determines all consumers of deprecated marked elements that show no compiler warnings. This way, developers can reveal deprecated element consuming code fragments which do not show compiler warnings. This is quite useful, to determine not updated spots in a program that do not excel at compile time or in the errors and warnings view.

At execution the query engine uses different means to determine fragments in the source code, belonging to the claimed dimensions. When all fragments that correspond to the query are found, they are stored in a result. This result is used to apply a filter to the package explorer (D). Thereby elements are excluded that are not in the result set.

A specialty of the tool is the support for drill downs. A package can be dragged and dropped into a query and be the source for a drill-down. All the consumers of the package and its types are belonging to the result slice. The types of the package (classes, enums, interfaces) can be expanded and (de)selected from the query. Likewise, the types themselves that are in a package can be extended to methods which can also be (de)selected. This way, systematic top-down investigation is possible.

## 4. Evaluative Use Cases

Here we present example applications for the tool to evaluate possible use cases.

### 4.1 Resolving the service layer

A possible application scenario of our tool is to uncover code that belongs to a certain application layer. A developer can compose a query to uncover elements that are belonging to the service layer. Therefore, he composes code slices that belong to this layer together in a query. When he executes the query he can see results like working sets in the package explorer. The slices in the query can be for example: (1) Classes that are annotated by typically in the service layer appearing annotations, like @Service; (2) Classes that have parent classes or interfaces that belong to the service layer, like controller classes; (3) Classes that contain strings that are typical for service layer, like "service" or "controller". This way a query would reveal all source code that belongs to the different concern slices. Additionally, concern slices that belong to a layer can be offered as a group in the pre-defined concerns view. This way, a developer could just drag and drop a layer query from the pre-defined concerns view without having the need to assemble the query himself. Generally, the query for a layer avoids building working sets manually.

In order to use another tool, developers need to learn the complex query language of that tool, e.g., JQuery [9], first. The Hypermodelling idea of code fragments that belong to multiple concerns is simpler. Furthermore, all the elements that belong to the service layer have to be defined by hand in a query in other tools. The Hypermodelling tool in contrast offers pre-defined concerns and offers the possibility to drag and drop concerns into a query. Lastly, the other query tools present the result of their queries within new views, where our tool enables to work with the package explorer.

### 4.2 Unsecured Methods

Another challenge for developers is to uncover unsecured publicly accessible methods to identify potential security leaks. A developer can compose a query for those with the Hypermodelling tool. For instance can a developer query for classes that inherit a framework class that makes them publicly available. Additionally, he can exclude the secured classes in the query. Excluding secured classes is possible, because applications are normally secured via Aspects or annotations. This structural information can be added as excluded slice to the query. Hence, all publicly accessible methods that are not secured can be uncovered easily. With other tools a complex query building process and result representation in standard views is not possible.

### 4.4 Deprecated Update

When software evolves, often elements are deprecated. A typical task for developers is to upgrade code not to use deprecated elements any more. Currently, the warnings and

error view lists all elements that consume deprecated elements. A programmer has to click through the whole list to get to the desired elements that need to be upgraded. With the Hypermodelling tool, a developer can add the slice of deprecated consumers to a query. This way, all the methods and classes consuming deprecated elements are revealed. Now, the developer investigates the first class and recognizes that it is extending a deprecated class. The documentation is checked to determine how the code should be updated. The developer applies the new knowledge how to update and changes that code. In that very moment the knowledge how to update this kind of case is at hand. Hence, the developer has the desire to update all elements of the same kind to avoid checking the documentation again. It would take quite a while to find the elements with the same problem in the errors and warnings view. Hence, he uses the Hypermodelling tool and adds just the slice of extenders of the deprecated classes to the query. The result of the query shows all elements using this deprecated class instantly. He can update all classes without having the need to use the errors and warnings view.

Other query tools are designed to query for code. Hypermodelling is built to query for plenty of code and other associated facts. Thus, concerns like deprecated consumers that are from the error and warnings view can only be used within the Hypermodelling tool. Moreover, this example shows that elements from code can be added to a query. Also, any kind of code slices can be combined with the error and warnings concern. Other tools are not that flexible.

## 5 Related Work

ConcernMapper is a tool for associating source code fragments with concerns manually [1]. The Concerns can be used to filter the package explorer and only show elements belonging to the selected concerns. Hypermodelling on the contrary has persuasive query capabilities that unleash queries with combinations of concerns far beyond the simple filtering. It can be interesting to extend the Hypermodelling tool with the capability to add concerns of ConcernMapper to a query.

Flat^3 [16] offers the possibility to search for code fragments via text strings and presents the search results in a result view. Also, concerns can be discovered via the execution of a program. Flat^3 offers rudimentary visualizations. Compared to Hypermodelling the multi-dimensional concern structure is ignored and just one concern kind is supported. Again, the detected concerns could be used to extend the Hypermodelling tool and be used within queries.

Mylyn is a tool that can be used to track down elements that are relevant for a task [15]. Hypermodelling is in contrast to Mylyn not a tool to track down concerns automatically. But tasks of Mylyn can be used within the queries. Mylyn itself lacks the possibility to calculate elements that belong to one task and not to another. Through the support of Hypermodelling this limitation can be overcome. Now it is possible to compute elements that belong to one and (not) another task or to combine queries for tasks with other concerns.

JQuery is a source code query tool with a predicate logic language [9]. The main issue is

the complexity and the custom language to do the queries [8]. Additionally, no result filter for the known views is offered. With Hypermodelling, developers have a tool that is integrating into their well known views. There is no necessity to learn a new complex query language and new views.

A multi-dimensional programming approach (HyperJ) is described by Ossher and Tarr [6] which can be seen as related and also as inspiration for the multi-dimensional Hypermodelling. There, an enhancement of object-oriented languages is proposed to overcome the limitation of intertwined concerns. Hypermodelling has in contrast the advantage that it is not a programming paradigm and it can be used together with commonly applied programming languages so no extension is needed.

The Concern Modeling Environment (CME) [17] allows multiple concerns to be associated with a code fragment. CME offers a query interface to query the concern associations and program structures and show them in a result view. The main difference to Hypermodelling is that no inspiration from DW technology is taken. Additionally, the current result representation in the package explorer and the drag and drop capabilities make it simpler to compose queries in Hypermodelling and inspect their results.

The AspectJ Development Tools (AJDT) offer support for programming with Aspects [18]. In contrast to Hypermodelling, AJDT is based on the dominant viewpoint of Aspects. Our presented Hypermodelling tool supports various kinds of concerns like annotations and unsafe type casts. In our approach Aspects are just representing one concern. Our tool could be extended to support the concern kind of Aspects additionally. Furthermore, would it be interesting to adapt the Aspect visualizing view to show not only show Aspects and their files or packages together, but to use other concerns, like annotations or Beans, and see the Aspect distribution for those.

Semmle code[3] is a source analysis and investigation tool. It offers a query interface to investigate source code and its associated elements. The results of a query are presented in a result view. Hypermodelling allows queries to be composed more smoothly through the drag and drop functionalities without having the need to learn a query language. Thereby, a well known view can be used for result investigation. Finally, the explicit multi-dimensionality of source code, where a fragment belongs to multiple dimensions respected by Hypermodelling is targeting investigations to uncover concerns. Semmle is neither leveraged for those and neither nor does it target concern investigations.

Orthographic Software Modeling (OSM) [19] is a multi-dimensional viewpoint and navigation through the various software models which seems similar to the idea to use DW like methods. To support the multi-dimensional navigation the various characteristics of dimensions are selected and through their combination the desired model is shown. In contrast to Hypermodelling neither SOC nor queries are part of the technique. Generally, OSM targets models where Hypermodelling focuses mainly on source code and the associated elements.

[3] http://semmle.com/semmlecode/

## 6 Conclusions and future Work

Source code fragments belong to multiple concerns. The Hypermodelling approach utilizes this and allows DW like queries for concerns. We leveraged and transferred the Hypermodelling approach to the IDE. As a result, we presented a tool that enables developers to query code in an OLAP similar manner. The tool takes advantage of the fact that the package explorer is the commonly used view in Eclipse, by offering a filter for the result representation of a query. Various presented usage scenarios indicate already beneficial applications. Related work confirmed: Hypermodelling differs from other approaches and seems to solve some issues more easily than other approaches.

Altogether, we introduced the approach to use well-known information by a programmer, to enable multi-dimensional queries on source code. Now the structural information in the code and of frameworks can be used to reveal the code of a layer. The tool demonstrates that queries and the application of OLAP like queries in the IDE are possible. Therefore, studies should be performed, considering developer productivity with the tool. Currently, we are planning to perform a study with developers for the validation of our tool.

However, although our presented tool already supports plenty of concerns we have the desire for more. Software tests and their results can be an interesting candidate to be supported for queries. Generally, our method is only supporting clear and well defined concerns. Thus, it should to be investigated how fuzzy concerns can be supported within the tool. For further development of the tool, evaluations and discussions should be carried out, to decide which concerns should be supported in the next version. Clearly, the actual application of DW technology should be considered and used in future versions to have plenty of speed up mechanisms at hand.

The package explorer as the only view for result representation needs to be reconsidered. Definitely, the current application has the advantage of providing the possibility of summoning concerns from other views this way, but only one kind of hierarchy can be shown. Maybe, there should be other result views offered to enable further investigation capabilities with other hierarchies. Likewise, programmers often work with models, when they investigate a program. Therefore, further research is needed to investigate how models can be used to represent query results. Also, how a traceability of model elements and source code can be used within queries needs to be investigated.

Furthermore, more application scenarios need to be considered to show what is possible with the Hypermodelling approach. For instance, we believe the application of the tool has benefits in the area of composite business applications (mashups) [20]. These wire various business domains together to derive a new functionality. The various business domains could be recognized as concerns and queries could be built, based on them.

Finally, it might be interesting to rank the importance of concerns with algorithms similar to CodeRank [21]. A usage scenario can be to suppose the most influential concerns, based on metrics, in the pre-defined concerns view.

## Acknowledgements

Thanks go to Veit Köppen, Gunter Saake and the anonymous reviewers for their useful comments on earlier drafts of this paper.

## References

[1] M. P. Robillard, F.Weigand-Warr. ConcernMapper: simple view-based separation of scattered concerns. In Proceedings of the '05 OOPSLA Workshop on Eclipse technology eXchange. ACM. 2005

[2] S. Letovsky, E. Soloway. Delocalized plans and program comprehension. IEEE Software, 3(3), pages 41–49. 1986

[3] T. Frey, M. Gelhausen, G. Saake. Categorization of Concerns – A Categorical Program Comprehension Model. In Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at the ACM Onward! and SPLASH Conferences. USA. 2011.

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

[5] E. W. Dijkstra. Selected Writings on Computing: A Personal Perspective. On the role of scientific thought. Springer.1982

[6] H. Ossher, P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In Proceedings of the Symposium on Software Architectures and Component Technology. Kluwer, pages 293-323. 2001

[7] A. J. Ko, B. Myers et al.. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. IEEE Transactions on Software Engineering, Volume 32, pages 971-987. 2006

[8] B. de Alwis, G.C. Murphy, M. P. Robillard. A Comparative Study of Three Program Exploration Tools. 15th IEEE International Conference on Program Comprehension ICPC 07. IEEE. pages 103-112. 2007

[9] K. D. Volder. Jquery: A generic code browser with a declarative configuration language. In P. V. Hentenryck, editor, PADL, volume 3819 of Lecture Notes in Computer Science. Springer, pages 88–102. 2006

[10] T. Frey. Vorschlag Hypermodelling: Data Warehousing für Quelltext. 23rd GI Workshop on Foundations of Databases. CEUR-WS, pages 55-60. 2011.

[11] T. Frey, V. Köppen, G. Saake. Hypermodelling - Introducing Multi-dimensional Concern Reverse Engineering**.** In 2nd International ACM/GI Workshop on Digital Engineering *(IWDE)*, Magdeburg, Germany, 2011.

[12] J. A. Bloch. Metadata Facility for the Java Programming Language. http://jcp.org/en/jsr/detail?id=175. 2004.

[13] M. P. Robillard, W. Coelho and G. C. Murphy How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Transactions on Software Engineering, Volume 30, pages 889-903, 2004

[14] G. C. Murphy, M. Kersten, L. Findlater. How Are Java Software Developers Using the Eclipse IDE? IEEE Software, pages 76-83. 2006

[15] M. Kersten. Focusing knowledge work with task context. PHD Thesis, University of British Columbia, Canada. 2007

[16] T. Savage, M. Revelle, D. Poshyvanyk. FLAT^3: Feature Location and Textual Tracing Tool, in Proc. of 32nd International Conference on Software Engineering (ICSE'10), ACM, pages 255-258. 2010

[17] 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 (LATE). 2005.

[18] A. Clement, A. Colyer, M. Kersten. Aspect-Oriented Programming with AJDT. Workshop on Analysis of Aspect-Oriented Software at ECOOP '03.Springer, pages 1-6. 2003

[19] C. Atkinson, D. Stoll. Orthographic modeling environment. In Fundamental Approaches to Software Engineering. Springer, pages 93-96. 2008

[20] J. Weilbach, M. Herger. SAP XApps and the Composite Application Framework. SAP Press, 1st edition, 2005

[21] B. Neate, W. Irwin, N.I. Churcher. CodeRank: A New Family of Software Metrics. In ASWEC: Australian Software Engineering Conference, IEEE, pages 369-378. 2006

[22] L. Langit, K. S. Goff, D. Mauri, et al.. Smart Business Intelligence Solutions with Microsoft SQL Server® 2008. O'Reilly. 2009