Automatic Recognition of Class Blueprint Patterns

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Marc-Philippe Horvath

Leiter der Arbeit:
Dr.Michele Lanza
Prof.Dr.Oscar Nierstrasz
Prof.Dr.Horst Bunke
Institut für Informatik und angewandte Mathematik

Abstract

In reverse engineering, *class blueprint* patterns are an efficient way to determine the purpose and abilities of a class. Finding those patterns is not trivial because the graphical representation of a large software system is too complex to be grasped by a software reengineer or a group of reengineers to find all the similarities and patterns in it.
This thesis presents a technique to discover known and unknown class patterns automatically in a software system. Our approach is based on the theory of graph pattern recognition, mainly graph edit distance and maximal common subgraph (MCS) algorithms. Using MCS and hierarchical clustering we automatically detect known and unknown patterns.

Acknowledgments

The possibility of doing one of the first joint ventures between the Software Composition Group and the "Forschungsgruppe Künstliche Intelligenz" was a great honour and a great opportunity as well. I had the chance to do something no one has done before, and it was a dream-come-true for me because I always wanted to be a pioneer in a field. Of course being a pioneer always brings the risk of the unknown. Alas the usability of my work is reduced due to the NP-complexity of the exact maximal common subgraph algorithms.

I want to thank all the people in those two groups for supporting me, especially Prof.Dr. Oscar Nierstrasz and Prof.Dr. Horst Bunke for thinking of me as a possible candidate for this joint venture. Furthermore I want to thank Dr. Michele Lanza. He was very helpful and supporting and always had time for me. I owe him even more because my diploma project would not exist if he had not created the class blueprints. Thanks again. Last but not least I would like to thank Orla Greevy and Gabriela Arévalo for giving me much helpful advice.

Marc-Philippe Horvath,
July 2004

Contents

Chapter 1

Introduction

In the world of software engineering, reverse engineering is becoming an increasingly important task. Nowadays we know of hundreds of big software systems, which have to be adapted to new hardware, or have to be changed to implement new features. Reverse engineering is not an easy task and tends to consume a lot of time, thus a need of automatization of these tasks is clear. A simple idea is the comparison of two or more classes considering their similarities. This idea fails as soon as one remembers that the similarity of text does not mean that the two classes perform the same tasks, and have the same functions. We need a way to describe the inner workings of a class, before we can recognize similarities. Such an abstraction exists [LANZ 03]. This abstraction is basically a graphical representation of the inner workings of a class, which makes graph pattern recognition applicable. If we have a working graph pattern recognition method, we can use it to find the most similar classes in a system, and in the end we get a clustering of them based on their similarities. This allows us to pinpoint the main questions:

- Is it possible to write a software tool to recognize class patterns?
- Given the fact that exact graph pattern recognition algorithms are NP-complete, will our tool have the performance to deal with large software systems?
- Are there different ways to solve the problem of class blueprint pattern recognition?
- Are we able to detect already known class blueprint patterns?
- Has a developer used a new design pattern, or a class blueprint pattern without documenting it?
- Are we able to detect unknown class blueprint patterns?

The goal of this thesis is to provide answers for these questions by using a lightweight approach based on software visualization, enriched with software metrics information, coupled with the theories and algorithms of the artificial intelligence field of graph pattern recognition. We name these visualizations as *class blueprints*, and the discovered patterns as *class blueprint patterns*. Furthermore, we use the term *class profile* of a class to describe a list of values which determine how high the probability is for a class blueprint to have a given class blueprint pattern in it.

1.1 Pattern Recognition And Reverse Engineering

We are not the first ones who try to combine pattern recognition and reverse engineering. As an example we point to Niere's Work [NIER 02]. In his work he tried to find design patterns using subgraph matching in Java. Subgraph matching is a field of graph pattern recognition. Other researchers tried to find design patterns or software patterns using other approaches.

Recently Formal Concept Analysis has been applied. Two recent examples are Tone [TONE 99] and Arévalo and Buchli [ARÉV 04].
Lanza [LANZ 03] introduced a new view on software systems. We can look now into classes and we can find *class blueprint* patterns. Since the classes are represented as graphs in this view, it is more than natural to apply the techniques of graph pattern recognition to find patterns in a set of class blueprints. The idea is to use graph pattern recognition to automatically detect known patterns and unknown patterns. This will reduce the workload for a reverse engineer, because he has only to check the generated data, namely a set of profiles and a dendogram, a graphical representation of a clustering of the distances between the class blueprints. With this generated data a reverse engineer will find errors, bad designed classes and other problems faster.
Unfortunately graph pattern recognition has some limits too. The most restrictive property of exact algorithms, those that return always the best match for two graphs, is NP-completeness. This makes the computation of a match time consuming. For huge software systems it might take more time than a team of software reengineers would have, which degrades the usability of our approach. We have identified other problems with a graph pattern recognition approach. We have to adapt every algorithm of graph pattern recognition to get usable results. The algorithms have been designed to work with a specific set of test graphs, to support their claims of being exact. These graphs are often much simpler and feature less data than an abstraction of a class blueprint. The adaptation of graph pattern matching algorithms to make them compatible with class blueprints generates various changes in these algorithms. The adaptations are in some cases so different from the original algorithm, that we consider it a new algorithm. Furthermore it is necessary to introduce a new measure because the predefined class blueprint patterns are split into patterns which consist only of nodes, patterns which consist only of edges, and patterns which have both. This new measure incorporates the number of nodes and edges to get a distance between two graphs, making the algorithms suitable to find all possible class blueprint patterns.

1.2 Contributions

This thesis presents a methodology that describes how to use the theory of graph pattern matching to detect class blueprint patterns in a software system. We outline the limitations and benefits of our approach. The contributions in detail are:

- A new measure was needed for the pattern matching algorithms. It is shown in Chapter 4 in Section 4.2.1. The proof that this is indeed a metric can be found in the Appendix A.
- We propose an algorithm to quickly scan a software system for clusters. It is discussed in detail in Section 4.4.1. Our tool ClassProfiler incorporates this algorithm.
- We also propose an exact algorithm, which is shown in Section 4.4.2. It is the algorithm we used for our case studies, because it is fast and exact.
- To get a quick overview of a class, we present the idea of a *class profile*.The *class profile* enables us to see whether a class does have a certain *class blueprint* pattern without having to check the *class blueprint* itself.
- We are able to automatically find *class blueprint* patterns in software systems. As reference we provide some examples in Chapter 5 and the *class profile* lists in the Appendix D and E
- We can find new *class blueprint* patterns in software systems using clustering techniques. As proof we refer to Section to the examples of case studies in Chapter 5 and in the Appendix B and C.
- We present a new pattern which our tool found in all case studies. The *AllStateClean* pattern. An example is the Figure 2.9.

1.3 Structure Of This Document

- In chapter 2 we introduce the field of reverse engineering and we explain the class blueprints. We list all blueprint patterns we try to recognize.
- Chapter 3 is an introduction to the field of graph pattern recognition. We talk about graph edit distance and maximal common subgraph recognition. We also show the complexity problem of this field in calculating the worst case complexity for our chosen algorithms. Furthermore we show other approaches to solve the problem.
- In chapter 4 we explain the adaptations we were required to make in the pattern recognition techniques to find the patterns in blueprints. We also discuss our new algorithms and the metrics we defined to get the graph pattern matching to work with a better precision than the original one.
- In chapter 5 we discuss the results of our case studies. There we show that our techniques of clustering and subgraph matching work well with smaller software systems, and that checking for *profiles* works well even for large software systems.
- Chapter 6 presents the conclusion. Here we discuss the results, the solved problems and identify the unsolved issues. As an outlook, we outline some possible fields for future work.
- In the appendix we list the clusterings and profiles for our case studies Smallwiki and Moose.

Chapter 2

Object-Oriented Reverse Engineering

2.1 Introduction

Reverse engineering existing software systems has become an important task in the world of software engineering. It is defined by Chikosfky and Cross as “the process of analyzing a subject system to identify the system’s components and their relationships, and to create representations of the system in another form or at a higher level of abstraction” [CHIK 90]. Reverse engineering is required for the maintenance, reengineering, and evolution of software systems, because a modification of one part of a system can have a negative impact on other parts of the system. In the worst case the software system can even break. Since this possibility exists, we have to reverse engineer the software system first, such that we get a model of the software that emphasizes the trouble spots and highlights the possible problems, before the software system can be modified or reengineered.

Sommerville [SOMM 00] and Davis [DAVI 95] estimate that the cost of software maintenance accounts for 50% to 75% of the overall cost of a software system. It would seem advisable to rewrite software systems as soon as they fail to satisfy the requirements. However, certain software systems are too valuable to be replaced or to be rewritten, because their sheer size and complexity makes a reengineering attempt too expensive for the owning company concering time and money. In the case of such legacy software systems it is advisable to first reverse engineer and then maintain, reengineer, and evolve such systems. By adapting them to new requirements [CASA 98, RUGA 98] the lifetime of these systems can be extended and increase the return of investment of their owners. Indeed, the longer a software system can be used, the better it pays off for the company that owns it.

We focus on the reverse engineering of object-oriented legacy systems, mainly because most current software systems are written in languages implementing this paradigm, and because it is not only its age that turns a software system into a legacy system, but the developement rate [DEME 02]. Moreover, early adopters of object-oriented technology are discovering that the benefits they expected to achieve by switching to objects have been very difficult to realize [DEME 02] and find themselves with present and future legacy systems implemented with object-oriented technology. Moreover, reverse engineering object-oriented software systems comes with additional challenges [WILD 92] compared to non-object-oriented systems, such as polymorphism, late-binding, incremental class definitions, and many other problems.

There are many approaches to reverse engineer software systems, such as:

- reading the existing documentation and source code. This is difficult when the documentation is obsolete, incorrect or not present at all. Reading the source code is a widely used practice, but does not scale up, as reading millions of lines of code would take weeks or months without necessarily increasing the understanding of the system by the reader. Moreover, at the beginning of a reverse engineering process one does not seek detailed information, but rather wants to have a general view of the system.

- running the software and/or generate and analyze execution traces. The use of dynamic information, e.g., information gathered during the execution of a piece of software, has also been used in the context of reverse engineering [RICH 99], but has drawbacks in terms of scalability (traces of a few seconds can become big) and interpretation (thousands of message invocations can hide the important information one is looking for).

- interviewing the users and developers. This can give important insights into a software system, but is problematic because of the subjective viewpoints of the interviewed people and because it is hard to formalize and reuse these insights.

- using various tools (visualizers, slicers, query engines, etc.) to generate high-level views of the source code. Tool support is provided by the research community in various ways, and visualization tools like Rigi [MÜ 86] and ShrimpViews [STOR 95] are widely used.

- analyzing the version history. Still a young research field, understanding the evolution of a piece of software is done using techniques like graph rewriting, visualization, concept analysis, clustering, and data mining. The insights gained are useful to understand the past of a piece of software and to possibly predict its future.

- assessing a software system and its quality by using software metrics. Software metrics tools are used to assess the quality and quantity of source code by computing various metrics which can be used to detect outliers and other parts of interest, for example cohesive classes, coupled subsystems, etc.

Several of these approaches succeed in solving various problems, but come with advantages and disadvantages due to the challenges they all face: They must scale up, since legacy software systems tend to be very large, and they must be flexible and applicable in different contexts, as there is no such thing as a standard reverse engineering context. Each legacy system comes with its own problems and flaws. Furthermore, they must be simple and straight-forward to use, because in the fast-paced software industry there is little time to reverse engineer software systems.
We now present a different way to look at a software system. Like a physician uses x-rays to look into the body of a patient, we use a class blueprint to look into a class. It is noteworthy that we also have the some problem of a x-ray: The is no information about dynamic processes which happen.

2.2 The Class Blueprint

This section introduces the concept of the *class blueprint*, a visual way of supporting the understanding of classes. A class blueprint is a semantically augmented visualization of the internal structure of a class, which displays an enriched call-graph with a semantics-based layout. It is augmented in various aspects that are explained in the subsequent sections:

- A class blueprint is structured according to *layers* that group the methods and attributes.

- The nodes representing the methods and attributes contained in a class are colored according to semantic information, i.e., whether the methods are abstract, overriding other methods, returning constant values, etc.

- The nodes vary in size depending on source code metrics information.

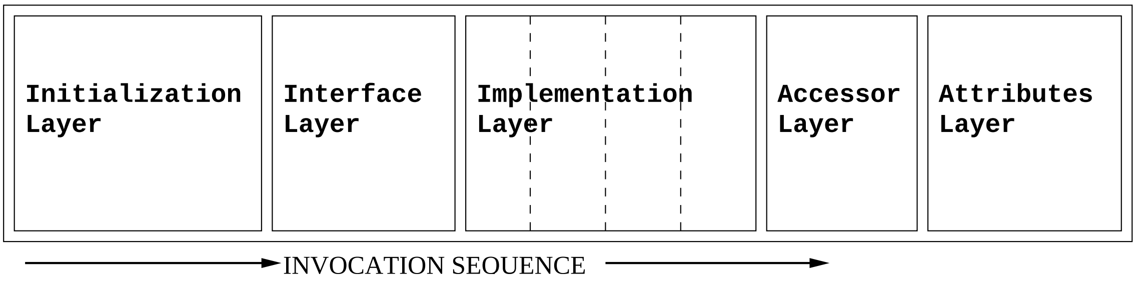

Figure 2.1: A class blueprint decomposes a class into layers.

2.2.1 The Layered Structure of a Class Blueprint

A class blueprint decomposes a class into layers and assigns the attributes and methods of the class to each layer based on the heuristics described below. In Figure 2.1 we see an empty template of a class blueprint.

The layers support a call-graph notion in the sense that a method node on the left connected with another node on the right is either invoking or accessing the node on the right that represents a method or an attribute. From left to right we identify the following layers: *initialization layer, external interface layer, internal implementation layer, accessor layer, and attribute layer*. The first three layers and the methods contained therein are placed from left to right according to the method invocation sequence, i.e., if method *m1* invokes method *m2*, *m2* is placed to the right of *m1* and connected with an edge.

For each layer we present the conditions that methods must fulfill in order to belong to a certain layer. Note that the conditions listed below follow a lightweight approach and are not to be considered as complete. However, we have seen that they are sufficient for our purposes.

A class blueprint contains the following layers:

1. **Initialization Layer.** The methods contained in this first layer are responsible for creating an object and initializing the values of the attributes of the object. A method belongs to this layer if one of the following conditions holds:

   - The method name contains the substring “initialize” or “init”.
   - The method is a constructor.
   - In the case of Smalltalk, where methods can be clustered in method protocols, if the methods are placed within protocols whose name contains the substring “initialize”.

   In this layer there should also be the static initializers for Java, however we do not take them into account, as they are not covered by our metamodel [DEME 01].

2. **External Interface Layer.** The methods contained in this layer represent the interface of a class to the outside world. A method belongs to this layer if one the following conditions holds:

   - It is invoked by methods of the initialization layer.
   - In languages like Java and C++ which support modifiers (*e.g., public, protected, private*) it is declared as *public* or *protected*.
   - It is not invoked by other methods within the same class, *e.g.,* it is a method invoked from *outside* of the class by methods of collaborator classes or subclasses. Should the method be invoked both inside and outside the class, it is placed within the implementation layer.

We do not include accessor methods to this layer, but to a dedicated layer as we show later on. We consider the methods of this layer to be the *entry points* to the functionality provided by the class.

3. **Internal Implementation Layer.** The methods contained in this layer represent the core of a class and are not supposed to be visible to the outside world. A method belongs to this layer if one of the following conditions holds:
   - In languages like Java and C++ if it is declared as *private*.
   - The method is invoked by at least one method defined in the same class.
4. **Accessor Layer.** This layer is composed of accessor methods, *i.e.,* methods whose *sole* task is to get and set the values of attributes.
5. **Attribute Layer.** The attribute layer contains all attributes of the class. The attributes are connected to the methods in the other layers by means of *access relationships* that connect the methods with the attributes they access.

2.2.2 Representing Methods and Attributes

We represent methods and attributes using colored boxes (nodes) of various size and position them within the layers presented previously. We map metrics information to the size of the method and attribute nodes, and map semantic information to their colors.

Mapping Metrics Information on Size

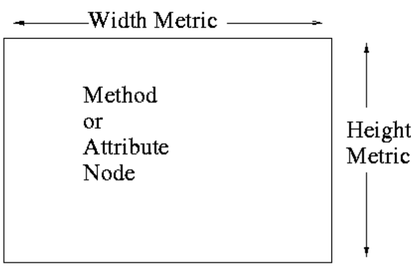

Figure 2.2: A graphical representation of methods and attributes using metrics: the metrics are mapped to the width and the height of a node.

The width and height of the nodes reflect metric measurements of the represented entities, as illustrated in Figure 2.2.

The class blueprint view visualizes method nodes and attributes nodes.

- *Method nodes.* In the context of a class blueprint, the metrics used for the nodes representing the methods are lines of code for the height and the number of invocations for the width.
- *Attribute nodes.* The metrics used for the boxes representing the attributes are the number of direct accesses from methods within the class for the width and the number of direct accesses from outside of the class for the height. The choice of these measures allows one to identify how attributes are accessed.

In Figure 2.3 we see how we distinguish a caller from a callee: the caller has outgoing edges at the bottom, while the callee has in-going edges at the top. Furthermore, the blueprint layout algorithm places the callee to the right of a caller.

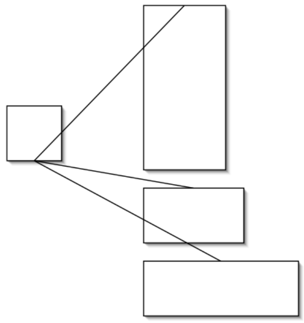

Figure 2.3: The caller has outgoing edges at the bottom, while the callee has in-going edges at the top.

Mapping Semantic Information on Color

The call-graph is augmented not only by the size of its nodes but also by their color. In a class blueprint the colors of nodes and edges represent semantic information extracted from the source code analysis. The colors play therefore an important role in conveying added information, as Bertin [BERT 74] and Tufte [TUFT 90] have extensively discussed. Table 2.1 presents the semantic information we add to a class blueprint and the associated colors.

Description	**Color**
Attribute	blue node
Abstract method	cyan node
Extending method. A method which performs a *super* invocation.	orange node
Overriding method. A method redefinition *without* hidden method invocation.	brown node
Delegating method. A method which delegates the invocation, *i.e.,* forwards the method call to another object.	yellow node
Constant method. A method which returns a *constant* value.	grey node
Interface and Implementation layer method.	white node
Accessor layer method. Getter.	red node
Accessor layer method. Setter.	orange node
Invocation of a method.	blue edge
Invocation of an accessor. Semantically it is the same as a direct access.	blue edge
Access to an attribute.	cyan edge

Table 2.1: In a class blueprint semantic information is mapped on the colors of the nodes and edges.

Certain semantic information such as whether a method is delegating to another object is computed by analyzing the method abstract syntax tree (AST) and by identifying certain patterns. For example we qualify as delegating, a method invoking exactly the *same* method on an attribute (pattern 2) or a method invocation (pattern 1). In addition to those patterns we consider also the case when the method is returning a value using ˆ in Smalltalk (pattern 3 and 4). Note that such an analysis is language dependent but does not pose any problem in practice.

Pattern 1: delegating to invocation result.

```
methodX
   self yyy methodX
```

Pattern 2: delegating to an attribute.

```
methodX
   instVarY methodX
```

Pattern 3: delegating to invocation result with return.

```
methodX
   ^ self yyy methodX
```

Pattern 4: delegating to an attribute with return.

```
methodX
   ^  instVarY methodX
```

The fact that a method is abstract is also extracted from the analysis of the method AST as in Smalltalk the only way to specify that a method is abstract is to invoke the method `subclassResponsibility` (see Pattern 5). For Java and C++, specific language constructs make the analysis simpler.

Pattern 5: Abstract method.

```
methodX
   self subclassResponsibility
```

Note that the color associations shown in Table 2.1 are not mutually exclusive. Therefore, a node could have more than one color assigned to it. In such a case the color determined by the source code analysis takes precedence over the color given by the layer a certain node belongs to, as this information conveys usually more semantics.

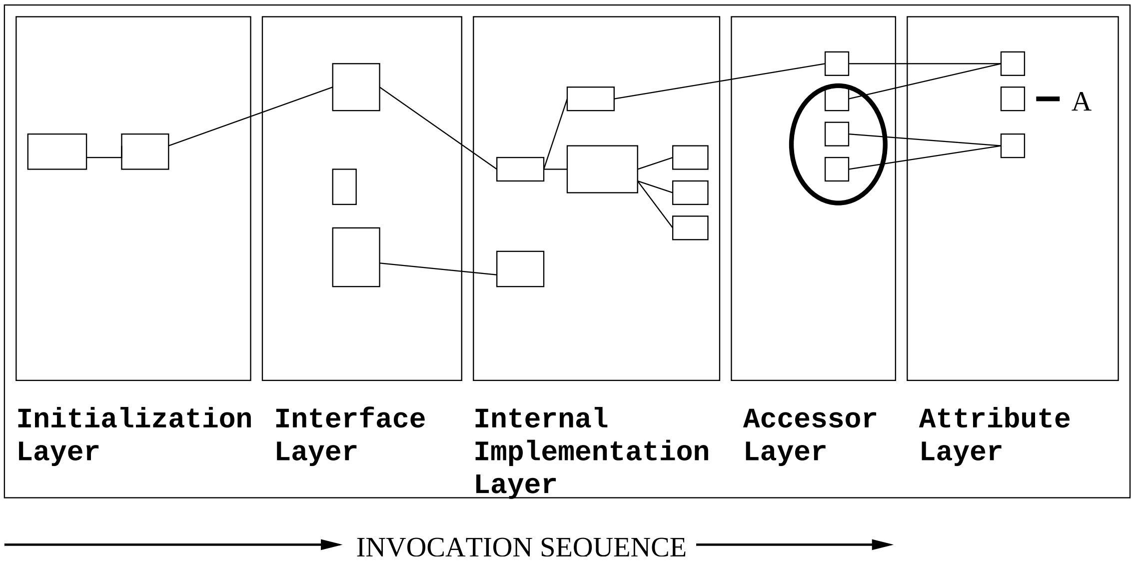

Figure 2.4: The methods and attributes are positioned according to the layer they have been assigned to.

2.2.3 The Layout Algorithm of a Class Blueprint

The algorithm used to layout the nodes in a class blueprint first assigns the nodes to their layers and then sequentially lays out the layers. Within each of the first three layers, nodes are placed using a horizontal tree layout algorithm: if method *m1* invokes method *m2*, *m2* is placed to the right of *m1* and both are connected by an edge which represents the invocation relationship. In case a method *m1* accesses an attribute *a1*, the edge connecting *m1* and *a1* represents an access relationship, as is denoted by the color of the edge. In the last two layers the nodes are placed using a vertical line layout, i.e., the nodes are placed vertically below each other. Although the layout algorithm can be considered lightweight, it shows acceptable results in terms of visual quality. The complex structure of a method invocation graph allows for cycles because of recursive calls, therefore the tree layout algorithm used as part of the overall blueprint layout is cycle-resistant.

In Figure 2.4 we see a template blueprint. We see that there are 2 initialization methods and 3 interface methods. We also see that some of its accessors (the ones in the ellipse) are not invoked and therefore unused and that one of the attributes (A) is not accessed by the methods of this class. In the next section we will define a vocabulary of blueprint patterns.

2.3 A Vocabulary based on Patterns in the Class Blueprints

While the approach is already an excellent vehicle to support the understanding of classes, it also provides the basis to develop a visual vocabulary that enables programmers to communicate recurrent situations they encounter. Indeed, recurrent situations in the code produce similar blueprint patterns in terms of node colors and flow structure. These (blueprint) patterns stem from the experiences we obtained while applying our approach on several industrial case studies. We subdivide the discussion of the patterns in two separate sections depending on the context in which a blueprint is presented:

1. **Single class perspective**, where we look at a single blueprint without considering surrounding sub- or superclasses (Section 2.4).

2. **Inheritance perspective**, where we extend the context to the inheritance hierarchy where the class resides ([LANZ 03]).

We use the term *pure* class blueprint when it is composed of only one and exclusively one pattern. Furthermore we have to mention that we focus on the class perspective. The inheritance perspective is discussed in the Thesis of Lanza [LANZ 03].

2.4 Single Class Blueprint Patterns

In this part we present the patterns that are included in blueprints without considering surrounding sub- and superclasses. Note that one class blueprint may include several patterns. As case study we use the software system Moose, a language independent reengineering environment, which has been developed at the University of Berne. The blueprint patterns are ordered first by their kind, whether ithey are class size or method size patterns, and then alphabetically.

2.4.1 *Method Clumps*

Graphically this pattern is composed of one large or huge node surrounded by some tiny nodes. It contains clusters of methods each with one very large method that is calling many of small methods. The large methods are not structured following a functional decomposition, but have a monolithic structure (one big chunk of code). *Method Clumps* are large nodes which represent methods having more than 100 lines of code. To give an idea of the disproportionality between those methods and the small ones, note that the average number of lines of a Smalltalk method is 7 [KLIM 96]. We did not find a 100% *Method Clumps* pattern in the classes of Moose. There were a few in the metaclass parts, but none in the classes themselves. Although we found an almost *Method Clumps* in MSECDIFSaver Figure 2.7.

2.4.2 *Size*

There are four sizes for class blueprints:

- *Single* : if the blueprint consists of only one node.
- *Micro* : if the blueprint consists of only a few nodes.
- *Giant* : if the blueprint consists of a lot of nodes.
- *Normal* : if the blueprint is bigger than micro and smaller than giant.

2.4.3 *Adders*

Graphically this pattern represents class blueprints that are mainly white (adding), orange (extending), or brown (overriding). These patterns present the way classes add, extend, or override inherited behavior. The weight of these patterns, for example, the number of methods in one of these three colors compared with the total number of methods is an indication on the way the class fits within its inheritance hierarchy. If the class blueprint is not bigger than micro, it is not considered an Adder.

AdderExtender

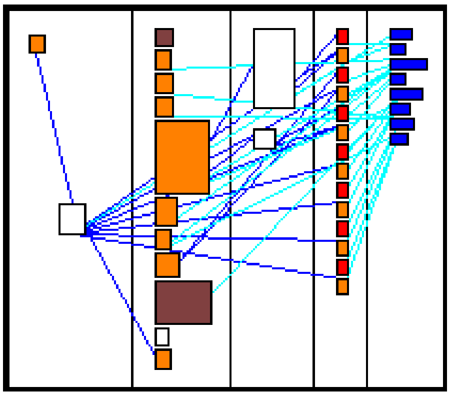

Figure 2.5: The class blueprint of VisualWorksParseTreeMetricCalculator, which is an *AdderExtender* .

AdderExtender (Figure 2.5) is a rather rare class pattern, mainly because classes which extend other classes are often small.

AdderNormal

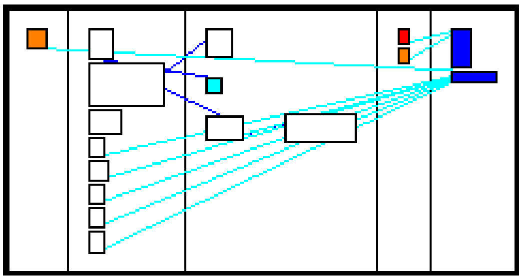

Figure 2.6: The class blueprint of FAMIXAbstractImporter, which is an *AdderNormal* .

Often there are many AdderNormal (Figure 2.6) patterns in a software system. *AdderNormal* can be anything from an algorithm implementing class to a data storage class with some additional functions.

AdderOverrider

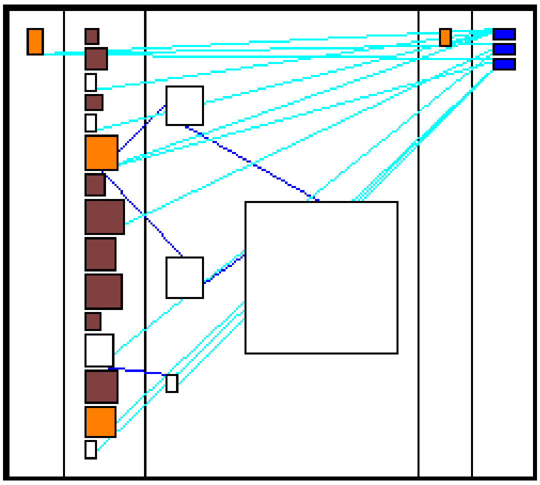

Figure 2.7: The class blueprint of MSECDIFSaver, which is the closest to an *AdderOverrider* which can be found in Moose.

AdderOverrider (Figure 2.7) is a uncommon pattern like the AdderExtender. Blueprints of classes which override other classes have often micro size, which of course reduces the number of *AdderOverrider* even in well designed object-oriented software systems.

2.4.4 *AllState*

Graphically this pattern presents groups of method nodes that have edges arriving to *all* the blue attribute nodes. It describes the fact that a group of methods accesses *all* the attributes of a class. When the class presents a *SingleEntry* it often presents also the *AllState* blueprint. The inverse is not true. Figure 2.8 shows an example where we see that all the attributes in the class are accessed by the three methods annotated as *A*. It comes in three versions: Initialization layer *AllState* , public layer *AllState* and private layer *AllState*. We use the abbreviations *AllState*1, *AllState*2 and *AllState*3 in a class profile.

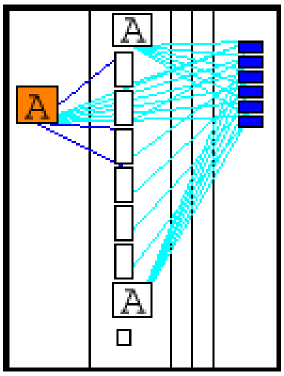

Figure 2.8: MGPreferences is an example with two *AllState*2 in the public layer and one *AllState*1 in initialization layer.

2.4.5 *AllStateClean*

This pattern presents groups of method nodes that have edges arriving to *all* the getter or to *all* setter nodes. It is basically, in terms of semantics, the same as the *AllState* with the exception that all attributes are accessed using the getter and setter methods. Like the *AllState* it comes in also in three versions: *AllStateClean*1, *AllStateClean*2 and *AllStateClean*3. This pattern has been found automatically while we clustered the case studies. Figure 2.9 shows a setter *AllStateClean* .

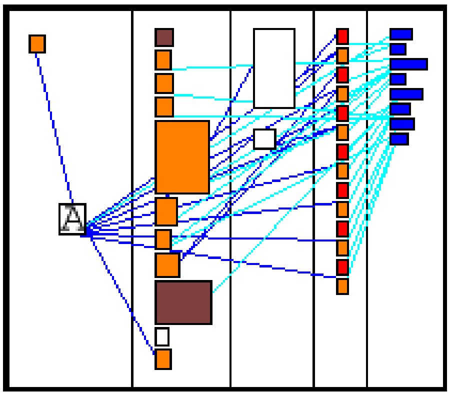

Figure 2.9: VisualWorksParseTreeMetricCalculator features a setter *AllStateClean* .

2.4.6 *ConstantDefiner*

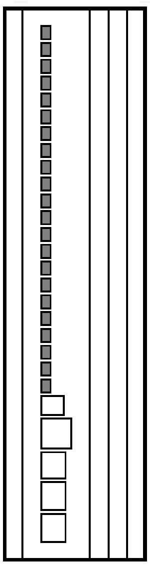

Figure 2.10: The class blueprint of the class FAMIXModel: it contains a distinct *ConstantDefiner* pattern.

Graphically this pattern is composed of grey nodes often residing in the interface layer. It describes a class which defines methods that return constant values such as integers, booleans, or strings. Pure ConstantDefiner blueprints are rare as a class is seldom limited to define constants. The class blueprint in Figure 2.10 is a ConstantDefiner.

2.4.7 *DataStorage*

Graphically this pattern presents mainly two layers: one red and orange coloured of accessors and one blue of attributes. Sometimes it also has one extra method to initialize the attributes. The *DataStorage* pattern describes a class which mainly defines accessors to attributes. Such a class usually does not implement any complex behavior, but merely stores and retrieves data for other classes. The implementation layer is often empty. An example of a *DataStorage* is shown in Figure 2.11. Looking for duplicated logic in the clients of such classes is usually a good way to reduce duplicated code and to enforce the law of Demeter [LIEB 89], [DEME 02].

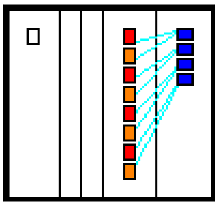

Figure 2.11: The class blueprint of the class MofAssociationEnd: it contains a pure *DataStorage* pattern.

2.4.8 *Delegate*

Graphically this pattern is composed of yellow nodes often found in the interface layer. *Delegate* (Figure 2.12) describes a class which defines delegating methods, *i.e.*, it forwards invocations to attributes or to accessor invocations. A *Delegate* pattern can be an indication for design patterns such as *Facade* or *Wrapper* [GAMM 95].

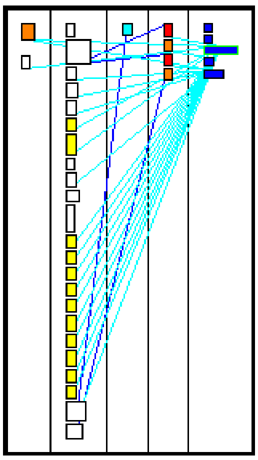

Figure 2.12: The class blueprint of the class ImporterFacade: it mainly consists of a *Delegate* pattern.

2.4.9 *Funnel*

Graphically this pattern is composed of an inverse (right-to-left) tree of nodes whose root is on the right, forming a funnel. *Funnel* describes a group of methods that all converge towards a final functionality. It often occurs when a complex data structure is used that can be accessed by various interfaces. Identifying the final functionality is often the key to understand how data abstraction is used in the class. In addition, in Smalltalk metaclasses providing multiple examples or initialization possibilities exhibit this behavior. Figure 2.13 represent a *Funnel* pattern, a lot of invocation edges are connected to the abstract(cyan) node.

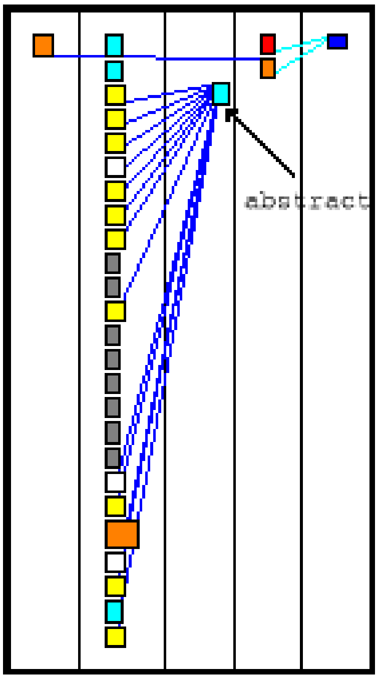

Figure 2.13: A *Funnel* pattern in the class blueprint of the class MSEAbstractGroup.

2.4.10 *Interface*

Graphically this pattern represents one predominant interface layer. It occurs when a class acts as an interface, which is frequent for abstract superclasses. It also occurs when the class acts as a pool of constants. In the Smalltalk programming language there is no construct for defining constant values, therefore class methods are often used to return constant values. Such classes can also contain a *ConstantDefiner* pattern as shown by the top class blueprint in Figure 2.14.

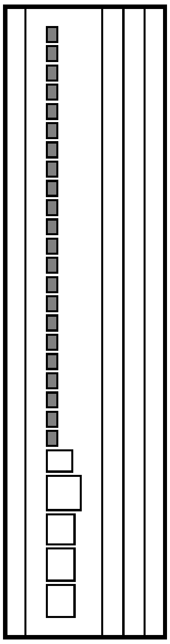

Figure 2.14: A *Interface* pattern in the class blueprint of the class FAMIXModelRoot.

2.4.11 *MicroSpecialExtender*

If a class has the size pattern micro , and it consists mainly of extending methods, we call it a *MicroSpecialExtender*. In a well designed object-oriented software system, the MicroSpecials are quite common. An example is Figure 2.15.

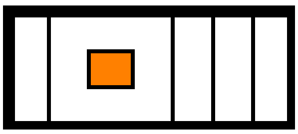

Figure 2.15: The single sized MSEApplicationModel features a *MicroSpecialExtender* pattern, making it the perfect example for the other classes of its kind.

2.4.12 *MicroSpecialOverrider*

We call an overriding class, which has micro size, an *MicroSpecialOverrider*. It is, like the *MicroSpecialExtender* a common sight in well designed Object-Oriented Software Systems.

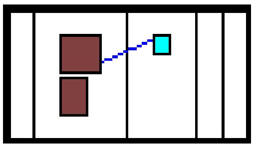

Figure 2.16: The micro sized DSAbsoluteOperator has a *MicroSpecialOverrider* pattern.

2.4.13 *SharingEntries*

Graphically the attribute nodes are accessed uniformly by groups of method nodes. This pattern represents the fact that multiple methods access the same state. Therefore it reveals a certain cohesion of the class regarding its state management. This pattern comes in two flavours. It depends whether this group of methods accesses the attribute using an accessor or directly. We refer to the first as a SharingEntryAccessor and to the later as a SharingEntry. The abbrevations for those patterns are SharingEntry4 or SharingEntry5. An example of such a pattern is emphasized in Figure 2.12 where nearly all methods access the third attribute from the top. Figure 2.17 presents *SharingEntries*4 pattern as almost all nodes in the public layer access the attribute over the accessor.

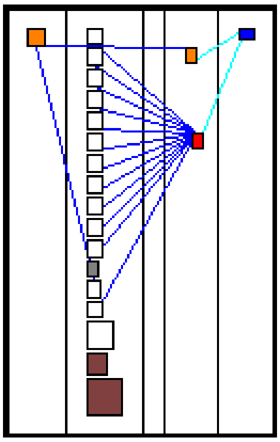

Figure 2.17: A *SharingEntries*4 pattern in the class blueprint of the class *FAMIXNamespace*.

2.4.14 *SingleEntry*

Graphically this pattern is composed of a minimal interface layer, often limited to one node, but which is connected to all the nodes of the larger implementation layers. *SingleEntry* describes a class which has very few or only one method in the external interface layer acting as entry point to the functionality of the class. It then has a large implementation layer with several levels of calls. Such classes are designed to deliver only little yet complex functionality. Classes which implement a specific algorithm (for example, parsers) show this pattern. Figure 2.18 shows one *SingleEntry* pattern in one class blueprint.

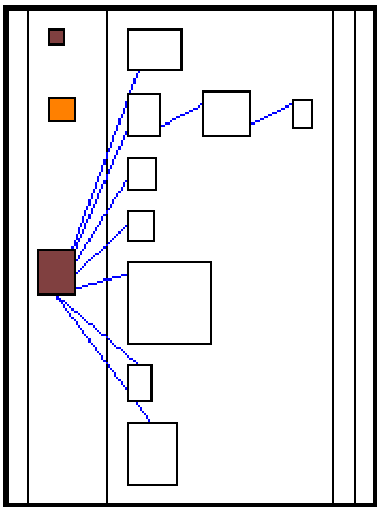

Figure 2.18: The class blueprint of the class DSMetricOperator with one *SingleEntry* pattern.

2.4.15 *StructuredFlow*

Graphically this pattern presents a cluster of methods structured in a deep and often narrow invocation tree. This pattern reveals that the developer has decomposed an implementation into methods that invoke each other and possibly reuse some parts. It supports the reading of the methods. A typical example is the decomposition of a complex algorithm into pieces. The class blueprint in Figure 2.19 shows a nice *StructuredFlow* pattern.

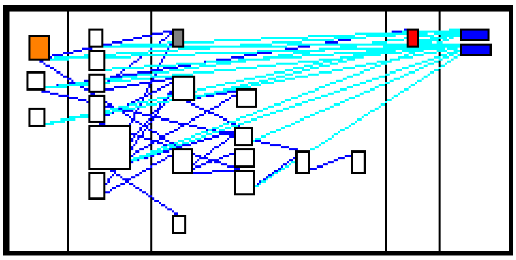

Figure 2.19: The *StructuredFlow* pattern of the class ModelManager.

2.4.16 *ThreeLayers*

Graphically this pattern is composed of three to four colored bands with few nodes: one or two bands for the interface layer, one red for the accessor, and one blue for attributes. This pattern describes classes that have few methods, some accessors, and some attributes. Usually these classes are small and implement primitive behavior and access to data. The class in Figure 2.20 has a *ThreeLayers* pattern. It is one of the most common patterns found in software systems.

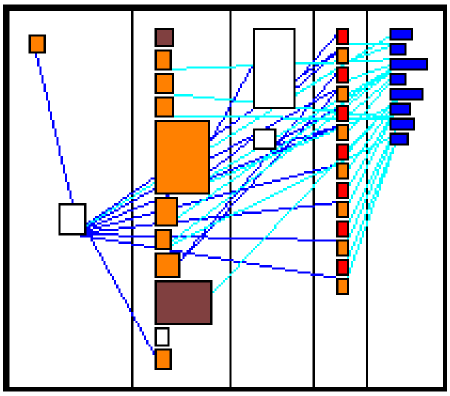

Figure 2.20: The class *VisualWorksParseTreeMetricCalculator* also features a *ThreeLayers* pattern.

2.4.17 *Wide Interface*

Graphically this pattern is composed of a large interface layer relative to the rest of the class. A *Wide Interface* blueprint is one that offers many entry points to its functionality proportionally to its implementation layer (see Figure 2.21).

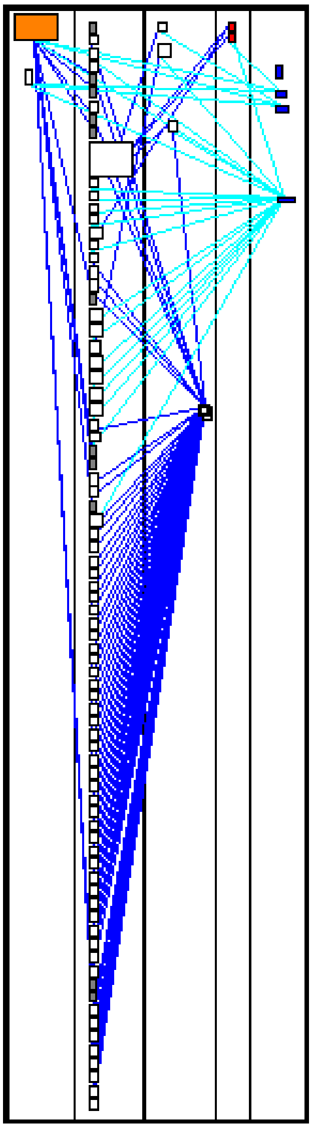

Figure 2.21: The class MSEMetricManager has a *Wide Interface* pattern.

2.5 Tool Support: CodeCrawler and Moose

To obtain the class blueprint visualizations we use CodeCrawler [LANZ] as visualization tool and Moose [DUCA 00] as metamodel and provider of the metrics and semantic information. CodeCrawler supports the synergy between opportunist reading of the code and the visualization of classes in the following ways:

- **Interactivity.** The blueprint visualizations do not merely represent source code, as in the case of static visualizations (*e.g.*, static pictures which cannot be manipulated), but they support direction manipulation. When the proposed layout does not suit the reengineer wishes, he can select, move, or delete connected, recursively connected, or disconnected nodes.

- **Code Proximity.** At any moment the reengineer can access the code by clicking on any node and see the corresponding definition at the level of a method, at the level of the class, and using code browsers presenting superclasses and subclasses. Moreover he has the possibility to see permanently a floating window showing the code of the node over which the mouse pointer is passing.

2.6 Related Work

Among the various approaches to support reverse engineering that have been proposed in the literature, graphical representations of software have long been accepted as comprehension aids [PRIC 93] [Sta 98].
Many tools make use of static information to visualize software, like Rigi [TILL 94], Hy+ [CONS 93] [MEND 95], SeeSoft [EICK 92], Dali [KAZM 99], ShrimpViews [STOR 95], TANGO [STAS 90], as well as commercial tools like Imagix [1] to name but a few of the more prominent examples. However, most publications and tools that address the problem of large-scale static software visualization treat classes as the smallest unit in their visualizations. There are some tools, for instance the FIELD programming environment [REIS 90] or Hy+ [CONS 93] [MEND 95] which have visualized the internals of classes, but usually they limited themselves to showing method names, attributes, etc. and use simple graphs without added semantical information.
Substantial research has also been conducted on runtime information visualization. Various tools and approaches make use of dynamic (trace-based) information such as Program Explorer [LANG 95], Jinsight and its ancestors [PAUW 93] [PAUW 99], Graphtrace [KLEY 88] or [RICH 99]. Various approaches have been discussed like in [JERD 97] where interactions in program executions are being visualized, to name but a few.
Nassi and Shneiderman proposed flowcharts to represent in a more dense manner the code of procedures [NASS 73]. Warnier/Orr-diagrams allow us to describe the organization of data and procedures [HIGG 87]. Both approaches only deal with procedural code and control-flow. Cross *et al.* defined and validated the effectiveness of Control Structure Diagrams (CSD) [CROS 98] [HEND 02], which is a graphical representation that depicts the control-structure and module-level organization of a program. Even if CSD has been adapted from Ada to Java, it still does not take into account the fact that a class exists within an hierarchy and in presence of late-binding.
We provide a visualization of the internal structure of classes in terms of their implementations and in the context of their inheritance relationships with other classes. In this sense our approach proposes a new dimension in the understanding of object-oriented systems.

2.6.1 Advantages

The visualisation of software system using class blueprints has some advantages:

[1] see http://www.imagix.com

- We can analyse the behaviour and function of a class without having to read the whole source code. The time needed to reverse engineer software systems is dramatically reduced.
- The class blueprint reduces classes to their main aspects. The resulting graph helps the reengineer to formulate hypotheses and to gain quickly insights over the inner working of a class, thus the class blueprints allow the reenigneer to select the relevant methods to inspect deeper and may either validate or invalidate the hypotheses formulated.
- After looking at a few dozen class blueprints, a reengineer can identify the programming style of the developer of the software system and find reoccuring patterns in the classes.
- The patterns found in the class blueprints can be used to define a vocabulary for a software system. Using those patterns in discussion between the rengineers during a reverse engineering task, like using design patterns while building a software system, leads to less explaination and a quicker understanding of the problems the other reengineers found.

2.6.2 Drawbacks

- The approach presented here relies heavily on an efficient layout algorithm in terms of space and readability. Especially in the case of very large classes, *i.e.*, having hundreds of methods, it may happen that the only real statement we can make is that the class is large (the *Giant* blueprint). However, patterns often occur in such classes providing important pieces of information.
- The blueprint of a class can give the viewer a “taste” of the class at one glance. However, it does not show the actual functionality the class provides. The approach proposed here is thus complementary to other approaches used to understand classes.
- The algorithms on which the class blueprints are based on do not always lead to the right marking of a method node. A better analysis of the source code will lead to better blueprints and thus to better understanding of the software systems.
- The approach presented here does not make use of dynamic information. This means we are ignoring runtime information about which methods get actually invoked in a class. This is especially relevant in the context of polymorphism and switches within the code. In this sense the class blueprint can be seen as a visualization of every possible combination of method invocations and attribute accesses.

2.7 Conclusion

As in object-oriented programming, classes are the primary abstractions based on which applications are built, we focus on supporting the reengineer to understand the internal structure of classes and how class behavior is developed in the context of the inheritance hierarchy in which it is defined. Our approach is based on the synergy between the class blueprint visualization and opportunist code reading [LITT 96] as the visualization helps building hypothesis, raising questions that a selective code reading verifies. As such it supports an understanding at multiple levels of abstractions [VON 96].
In this chapter we have presented the class blueprint, a polymetric view targeted at the understanding of classes and class hierarchies. The class blueprint visualizes the internal structure of classes, *e.g.*, an augmented call-graph enriched with metrics and semantic information. The class blueprint permits to identify patterns that help to understand the structure of classes. We have identified and described several of these patterns. Furthermore we have used the class blueprints on a case study and have discussed and verified our findings.
We return again to the questions we asked ourselves in the introduction (Chapter 1) :

- Is it possible to write a software tool to recognize class patterns?
- Given the fact that exact graph pattern recognition algorithms are NP-complete, will our tool be fast enough to deal with large software systems?
- Are there different ways to solve that problem?
- Are we able to detect already known class blueprint patterns?
- Has a developer used a new design pattern, or a class blueprint pattern without docmenting it?
- Are we able to detect unknown class blueprint patterns?

What we are presenting in this thesis is how we combined the theory of graph pattern matching in Chapter 4 with the class blueprints presented in this chapter. A class blueprint is nothing more than a graph and this made the graph pattern matching algorithms applicable to the problem of finding new patterns, or automatically find the ones given in this chapter.

Chapter 3

Graph Pattern Recognition

3.1 Introduction

In this chapter we introduce the basic graph pattern recognition theory. First we define the main elements of the field: what a graph is, later what a line-graph is and then we look at the isomorphism between graphs and between subgraphs. With all the necessary definitions provided, we look more closely at two of the better known maximal common subgraph algorithms. Afterwards we take a glance at the theory of graph edit distance, and point out the advantages it has compared to the maximal common subgraph algorithms. At the end of this chapter, we provide a short description of other possible approaches to solve the problem, which we considered during our research, but they were either not precise enough or the implementation was too time consuming. In the end of the chapter, we briefly explain what techniques we used to cluster the results.

3.2 Definitions

The first thing we have to define are the graphs. We use the following definition throughout the entire text.

Definition 1:
A Graph is a 4-tuple $G = (V, E, \alpha, \beta)$, where
V is the finite set of vertices
$E \subseteq V \times V$ is the set of edges
$\alpha : V \to L_V$ is a function assigning labels to the vertices
$\beta : E \to L_E$ is a function assigning labels to the edges

A definition of a graph is not enough to explain how graph pattern recognition works. Usually we do not only want to know if two graphs are equal, but also if one graph is a subgraph of the other. To achieve this, we have to define what a subgraph is:

Definition 2:
Given a Graph $G = (V, E, \alpha, \beta)$ and a Graph $S = (V_s, E_s, \alpha_s, \beta_s)$, then S is a Subgraph of G if and only if the following properties hold:

1. $V_s \subseteq V$
2. $E_s = E \bigcap (V_s \times V_s)$
3. α_s and β_s are the restrictions of α and β to V_s and E_s, respectively, i.e.,

$$\alpha_s(v) = \begin{cases} \alpha(v) & \text{if } v \in V_s \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\beta_s(v) = \begin{cases} \beta(e) & \text{if } e \in E_s \\ \text{undefined} & \text{otherwise} \end{cases}$$

We also need a proper way to define how similar two graphs are. One way to get a similarity distance is to define the difference between two graphs:

Definition 3:
Given a graph $G = (V, E, \alpha, \beta)$ and a subgraph $S = (V_s, E_s, \alpha_s, \beta_s)$ of G, the difference of G and S is the subgraph of G which is defined by the set of vertices $V - V_s$. We denote the difference of G and S by $G - S$.

Another useful definition is the one for a union of two graphs.

Definition 4:
Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$, where $V_1 \bigcap V_2 = \emptyset$, and a set of edges $E' \subseteq (V_1 \times V_2) \bigcup (V_2 \times V_1)$ with a labeling function $\beta' : E' \rightarrow L_E$, the union of G_1 and G_2 with respect to E' is the graph $G = (V, E, \alpha, \beta)$ such that:

1. $V = V_1 \bigcup V_2$
2. $E = E_1 \bigcup E_2 \bigcup E'$
3. $\alpha(v) = \begin{cases} \alpha_1(v) & \text{if } v \in V_1 \\ \alpha_2(v) & \text{if } v \in V_2 \end{cases}$
4. $\beta(e) = \begin{cases} \beta_1(v) & \text{if } e \in E_1 \\ \beta_2(v) & \text{if } e \in E_2 \\ \beta(v) & \text{if } e \in E' \end{cases}$

The most important definition: How do we know if two graphs are the same? Like often in a field near to mathematics, we simply define that two graphs are equal, if there exists a bijective function between the two graphs.

Definition 5:
A bijective function $f : V \rightarrow V'$ is a graph isomorphism from a graph $G = (V, E, \alpha, \beta)$ to a graph $G' = (V', E', \alpha', \beta')$ if:
1. $\alpha(v) = \alpha'(f(v))$ for all $v \in V$.
2. For any edge $e = (v_1, v_2) \in E$ there exists an edge $e' = (f(v_1), (v_2)) \in E'$ such that $\beta(e) = \beta(e')$, and for any $e' = (v_1', v_2') \in E'$ there exists an edge $e = (f^-1(v_1'), f^-1(v_2')) \in E$ such that $\beta(e') = \beta(e)$.

Now we know how to check if two graphs are the same. To check if a graph is included in another graph, we have to define a way to get an isomorphism between the smaller graph and a subgraph of the larger graph. One can see that an isomorphism between a smaller graph and a subgraph of a larger graph is simply an injective function:

Definition 6:
An injective function $f : V \rightarrow V'$ is a subgraph isomorphism from G to G' if there exists a subgraph $S \subseteq G'$ such that f is a graph isomorphism from G to S.

We can check now if a graph is included in another graph. What we need is a definition to check if a subgraph of a graph is included in another graph. Such a graph is simply called a *common* *subgraph*. Now we do not know if the found subgraph isomorphism is the largest of all possible subgraph matching. Maybe there is another injective function, which hits more vertices than the one we first found? We need a definition which tells us exactly if the found graph is the largest of all possible subgraph isomorphism. Both ideas unified in a definition looks like the following:

Definition 7:
Let $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs. A common subgraph of G_1 and G_2, $CS(G_1, G_2)$, is a graph $G = (V, E, \alpha, \beta)$ such that there exist subgraph isomorphisms from G to G_1 and from G to G_2. We call G a largest common subgraph of G_1 and G_2, $MCS(G_1, G_2)$, if there exists no other common subgraph of G_1 and G_2 which has more vertices than G.

3.3 Maximal Common Subgraph

Before we look at the algorithms and the needed definitions in detail, we have to see that there are four ways to define a maximal common subgraph. First we have to see the difference between a maximal induced common subgraph and a maximal common edge subgraph. A vertex subgraph is a set S of vertices of a graph G and those edges of G with both endpoints in S. A graph G_{12} is a common induced subgraph if and only if G_{12} is isomorphic to induced graphs G_1 and G_2. A maximal common induced subgraph (MCIS) consists of a graph G_{12} with the largest number of vertices meeting the aforementioned property. Related to the MCIS is the maximal common edge subgraph(MCES). An MCES is a subgraph consisting of the largest number of edges common to both G_1 and G_2. Normally the term MCS refers two both MCES and MCIS.

The MCS between two graphs can be classified further by distinguishing between the connected and disconnected case. A connected MCS is a MCS whereby each vertex is connected to every other vertex by at least one **path**. Note that we say path connected and not edge connected. This means simply that the MCS consists of only one subgraph. A disconnected MCS on the other hand, can consist of multiple subgraph components. Figure 3.1 illustrates the difference between a connected and unconnected MCS.

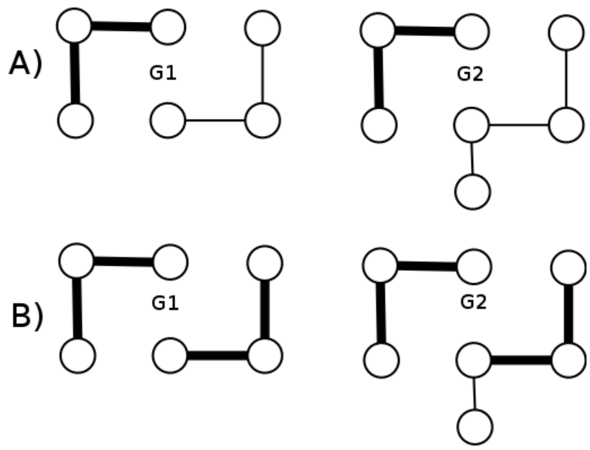

Figure 3.1: A) is a example for a connected MCS and B) is a example for a disconnected MCS

We describe the most useful way to define a MCS for the problem of finding similarities in class blueprints in the next chapter (Chapter 4).
Since most of the published literature considers the MCIS, an algorithmic transform for translating a MCIS to a MCES is desirable. One such technique is the work of Withney[WHIT 32], who proved that an edge isomorphism between two graphs G_1 and G_2 , induces a vertex isomorphism

provided that a ΔY-exchange does not occur. A ΔY-Exchange may happen if a graph is transformed into a linegraph. An example of a ΔY exchange is show in Figure 3.2.

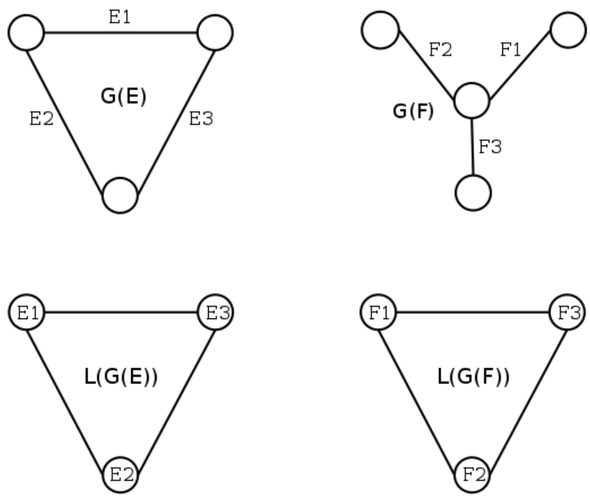

Figure 3.2: On the left side (graph G(E) and L(G(E))) no ΔY-exchange occurs. On the right side (graph G(F) and L(G(F)) a ΔY-exchange occurs.

A way to transform a MCIS algorithm to a MCES algorithm, is to transform the graph G in a line graph of G. The following definition clarifies the idea.

Definition 8:
A line graph $L(G)$ is a graph whose vertex set consists of the edge set of$G$; therefore, if (v_i, v_j) is an edge in G, it is also a vertex in $L(G)$.
A pair of vertices in $L(G)$ are adjacent if and only if the two corresponding edges in G are incident on each other.

Using a line graph to transform a MCIS to a MCES allows us to use the same algorithm for a different kind of problem. In some case we are more interested in a MCES than just an MCIS, or vice versa.

We now go ahead with the thematic of maximal common subgraphs and present two of the better known MCS algorithms. The first one we discuss is the algorithm of McGregor[McGr 82] and it is capable of detecting a disconnected MCES. The paper is noteworthy since it appears to be the first that draws a distinction between MCIS and MCES. The algorithm itself belongs to the class of backtracking algorithms. The McGregor algorithm tries to reduce the number of backtrack instances necessary by inspecting the set of possible solutions remaining at some point in the depth-first search and determining whether it is necessary to extend the current solution. The set of possible solutions is evaluated by enforcing a connectivity relation with the currently detected solution.

Algorithm 1 (McGregor):

```
function McGregor(s, n1, n2)
  begin
    if(nextpair(n1,n2)) then
      begin
        if(isFeasiblePair(n1,n2)) then
          AddPair(n1,n2);
          CloneState(s,s');
        while(s' not leaf in the search tree)
          begin
            McGregor(s',n1,n2);
            Backtrack(s');
          end
      delete(s');
      end
  end
```

The algorithm example is derived from the one described by McGregor, but it is more general and is discussed in more detail in [BUNK 02]. It is suitably described by a state space representation. Each state s represents a common subgraph of the two graphs under construction. This common subgraph is part of the MCS that will be eventually formed. In each state a pair of vertices not yet considered, the first belonging to the first graph and the second to the second graph, is selected (whenever it exists) through the function `NextPair(n1,n2)`. The selected pair of vertices is considered legal if the function `IsFeasiblePair(n1,n2)` returns true. If the pair is legal, then the function AddPair(n_1,n_2) extends the current partial solution by the pair (n_1, n_2). Now if the current state s is not a leaf of the search tree, it copies itself through the function CloneState(n,n'), and the new state is immediately checked. After the new state has been analyzed, a backtrack function is invoked which restores the common subgraph to the previous state. Now the algorithm chooses a different state. Whenever a branch is chosen, it will be followed as deeply as possible in the search tree until a leaf is reached. It is noteworthy that every branch of the search tree has to be followed, because, except for trivial examples, it is not possible to predict if a better solution exists in a branch not yet explored. One should mention too, that whenever a state s is not useful anymore, it will be removed from the memory with the function delete(s).

Now we calculate the worst case complexity for the McGregor algorithm. Let N_1 and N_2 be the number of vertices of the first and second graph and let $N_1 \leq N_2$. In the worst case, if the two graphs are completely connected with the same label on each vertex and the same label on each edge, the number of states s examined by the algorithm is:

$$S = N_2! \cdot (\frac{1}{(N_2 - N_1)!} + \ldots + \frac{1}{(N_2 - 1)!}) \tag{3.1}$$

which for the case of $N_1 = N_2$ reduces to the following approximation:

$$S \cong e \cdot N!$$

It is noteworthy that only $O(N_1)$ space is necessary for the algorithm.

Another way to detect a MCS is using a clique based approach. The reduction of the search for the MCS to the problem of finding a MC (maximal clique) in a graph is well known and often applied. The first step of all clique based algorithms, is to build the modular product of the to be compared graphs. The MCS can then be obtained by detecting the MC of the corresponding modular product.

Definition 9:
The modular product of two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ is defined on the vertex set $V_1 \times V_2$ with two verticed (u_i, v_i) and (u_j, v_j) being adjacent whenever
$(u_i, u_j) \in E_1$ and $(v_i, v_j) \in E_2$ or
$(u_i, u_j) \notin E_1$ and $(v_i, v_j) \notin E_2$

We now look in detail at a classical maximal clique based algorithm, also discussed in [BUNK 02]:

Algorithm 2 (Durand-Pasari):

```
function DurandPasari(vertex_list)
  begin
    level=length(vertex_list);
    nullcount=numberOfNulls(vertex_list);
    if(nullcount>=bestnullcount) then
      return;
    else if(level==maxlevel) then
      save(vertex_list);
      bestnullcount=nullcount;
    else
      P=collection of vertices(n1,n2) such that n1==level;
      P=P ⋃ null_vertex;
      P do foreach v in P
        if (v is legal) then
          DurandPasari(vertex_list + v)
  end
```

The Durand-Pasari algorithm generates a list of vertices using a depth-first search on a search tree, by systematically selecting one vertex at a time for successive levels, until it is not possible to add further vertices. When a vertex is considered, the forward search part of the algorithm first checks to see if this vertex is a legal vertex. It is considered legal if it is connected to every other vertex already in the list. At each level, the choice of the vertices is limited to the ones which correspond to pairs (n_1, n_2) whereas $n_1 = level$, such that it is ensured the search space is actually a tree. If there is no legal vertex left, the algorithm inserts an always legal null vertex. The algorithm next checks whether the current number of nulls in the list is smaller or equal to the number of nulls in the best solution so far.

If N_1 and N_2 are the number of vertices of the starting graphs, with $N_1 \leq N_2$, the algorithm execution will require a maximum of N_1 levels. Since at each level the space requirement is constant, the total space requirement of the algorithm is $O(N_1)$. To this space requirement the space needed to represent the modular product must be added. In the worst case the modular product is a complete graph of $N_1 \cdot N_2$ vertices, and the algorithm has to explore $(N_2 + 1)$ vertices at level 1, N_2 at level 2, and up to $(N_2 - N_1 + 2)$ at level N_1. If we multiply these numbers we obtain the worst case number of states.

$$S = (N_2 + 1)(N_2) \cdot \ldots \cdot (N_2 - N_1 + 2) = \frac{(N_2 + 1)!}{(N_2 - N_1 + 1)!} \tag{3.2}$$

which for $N_1 = N_2$ reduces to $O(N_1 \cdot N_1!)$.

The problem the MCS algorithms have is their inability to correct the errors which can occur. We provide an example in the next chapter.

3.4 Graph Edit Distance

The graph edit distance is derived from the well known string edit distance. To be precise, a string is nothing more than a specialised graph, with at most one incoming and one outgoing edge per vertex. It seems clear, that since string edit distance is a specialised form of graph edit distance, we have to add some more edit operations to the set of operations for string edit distance to make it applicable on graphs. The advantage of graph edit distance is that it will correct the errors. For example if we copy a graph and change the marking of one vertex, the maximal common subgraph is one vertex smaller than the original graph. If we do the same with graph edit distance, the difference between the two almost equal graphs is just the cost for the exchanged marking. We have to define first the edit operations before we can go on with the details of the theory.

Definition 10:
We define the edit operations as follows:

- Delete a vertex: $v \to \epsilon; v \in V$
- Insert a vertex: $v \leftarrow \epsilon; v \in V$
- Substitute a vertex marking: $a \to b; a, b \in L_V$
- Delete an edge $e = (u, v) : e \to \epsilon; e \in E$
- Insert an edge $e = (u, v) : e \leftarrow \epsilon; e \in E$
- Substitute an edge marking: $c \to d; c, d \in L_E$

With the operations defined, we need to define costs for each operation.The costs are used to define how probable an operation is. There are two ways of finding out how expensive an operation should be: One can either let the computer learn the costs, which will consume a lot of computing power or one has to start with your own costs based on heuristics and adapt them after each run through the test set until the results are acceptable.

Definition 11:
We define costs with the cost function $c(\ldots)$. The general costs are:

- $c(v \to \epsilon)$
- $c(v \leftarrow \epsilon)$
- $c(a \to b)$
- $c(e \to \epsilon)$
- $c(e \leftarrow \epsilon)$
- $c(c \to d)$

The cost of a order of edit operations $S = s_1, s_2, \ldots, s_n$ is defined as:
$c(S) = \sum c(si)$.

Note that in general, the cost of a vertex deletion or insertion is independent of the vertex. It is a direct function of the marking. The same applies to edges.

The more markings a vertex has, the more costs one has to define. With the costs and the operations defined, we are now able to define the graph edit distance:

Definition 12:
The graph edit distance $d(G_1, G_2)$ between two graphs G_1 and G_2 is defined as follows:
$d(G_1, G_2) = \min\{ c(S) | S$ is a sequence of edit operations for the translation of G_1 to $G_2 \}$

Lemma 1:
The graph edit distance $d(G_1, G_2)$ is a metric if and only if the following hold:

1. $c(s) \geq 0$ for all edit operations
2. $c(s) = 0$ if and only if s is an identical vertex or edge substitution
3. $c(s) = c(s^-1)$ for all edit operations s, where s^-1 is the inverse edit operation of s.
4. $c(s) \leq c(s') + c(s'')$ if $s = s' \circ s''$, where $s = s' \circ s''$ is the concatenation of s' and s''

If it is defined that $(s) = \begin{cases} 0 & \text{if s is a vertex deletion} \\ \infty & \text{for all other not identical edit operations} \end{cases}$
then holds $d(G_1, G_2) = 0 \leftrightarrow G_1 \subseteq G_2$

Now the computation of a subgraph isomorphism can be seen as a special case of the computation of the edit distance. It follows that there cannot exist an algorithm with polynomial complexity for the computation of $d(G_1, G_2)$.

For the real computation of the graph edit distance, one can use a classical tree search algorithm. There are three different search strategies for a tree search: breadth-first, depth-first and A$*$. Breadth-first finds the optimal match, but has an exponential space-complexity and time-complexity. Depth-first has only a space-complexity of $O(bd)$, but the time-complexity is still exponential and due to its design, the Depth-first strategy may miss the optimal solution in large search trees, even if it is located in close vicinity to the starting point. The A$*$ strategy is a generalisation of the breadth-first strategy with an heuristic function ordering the current candidates and therefore it has the same worst case time- and space-complexity like breadth-first.
Now we look at the complexity of the graph edit distance in detail. Note that we only show the worst case of the computation. The worst case would be if all vertices are identical and connected to all others with edges. Since now all vertices are indistinguishable, we have to compare each vertex with all the other vertices and consider the deletion/ insertion of a vertex. For a worst case we compare two identical graphs with $n_1$vertices, n_2 vertices respectively $n_1 + 1$ cases in the first of n_2 steps and already n_1 times n_1 plus $n_1 + 1$ in the second step. If we follow this to the end we can conclude that the worst case time-complexity of the graph edit distance is

$$O(n) = {n_1}^{n_2} \tag{3.3}$$

This is the price we have to pay for having an error correcting algorithm. Note that in practical implementations many optimisations are applied. For example we can take the lowest distance between graphs as an upper limit. As soon as the current distance is bigger than the shortest distance so far, we do not have to follow the branch to the end. This already reduces the search tree drastically and is easy to implement. Other optimisation approaches would be doing some sort of precalculation, using an approximate graph edit distance for hints of the best matches or implementing a rule of structure conserving.

3.5 Other Approaches

In this section, we outline other approaches to solving the problem of graph recognition which we identify but did not take into consideration in this work. We describe briefly each idea we have studied.

We divide all algorithms in two different classes: the exact ones and the approximate ones. In this document we have already discussed two kinds of exact algorithms, namely the McGregor, an exact backtracking algorithm, and the Durand-Pasari, an exact clique-based algorithm. There is another kind of exact algorithms not yet discussed. These are called dynamic programming, and based on the technique with the same name. There was only one example of this kind of algorithm in the literature: the one from Akutsu. It is relatively easy to implement, and the most interesting aspect is: We can compute a MCS in polynomial time. Unfortunately this solution [AKUT 93] can only be applied to a special case of graphs. The algorithm works only on graphs which are "almost trees of bounded degree". Such a tree is a graph G in which the equation $|E(B)| \leq |V(B)| + K$ holds for every biconnected component B of G, where K is a constant. A biconnected component can be defined as a maximal edge induced subgraph in a connected graph such that the subgraph cannot be disconnected by eliminating a vertex. Since we planned to apply graph pattern recognition on very dense graphs, which were neither almost trees nor trees, the algorithm was not applicable, but nevertheless worth mentioning since dynamic programming represents a rather new solution attempt for this kind of problems.

Having outlined briefly these exact algorithms, we enter the field of dozens of approximates. We can also divide these in different categories: Genetic-algorithm based, combinatorial optimisation or *ad hoc*. The approximate algorithms try to solve NP-complete problems within reasonable time. One limitation of these algorithms is, just as the name indicates, that there is no guarantee that they will find a MCS close in size and composition to the real MCS. However since our graphs are rather big in terms of graph pattern matching, we checked approximate class of algorithms for its usability to solve the graph recognition problem.

We have to admit that there exist dozens of approximate algorithms, they are often made specifically for a given problem, making their general usefullness for other application fields rather limited. This means if we try to apply one of these kinds of approximate algorithm on our problem, we have to write our own algorithms to get useful results.

Genetic algorithms are one class of algorithms frequently used for maximizing or minimizing a specified objective function. They work basically on the theory of Darwin, namely that only the fittest will survive, which also holds in the graph pattern recognition, if we think of fitting the real MCS. We first generate a population of candidates and a fitness function, then we let nature have its way. In the terminology of computer science, that means we apply randomly crossover and mutation operators on the population. Crossover can be described as a randomly mixing of two candidates, which generates a sibling that might fit better, and mutation is simply changing randomly a few properties of the original candidate. Note that we do not yet delete the parents, or the original candidate. Afterwards we apply a fitting function on the new population and remove the unfittest. The threshold is dependent on the application. If we do this a few times in a row, we get a good approximation of the real MCS. An interesting way to adapt genetic algorithms to the field of pattern matching can be found in the paper of Wang [WANG 97].

Combinatorial optimisation algorithms are basically just problem adapted algorithms, which rule out the impossible matchings. As an example we briefly discuss the 2 DOM algorithm of Nobuo Funabiki and Junji Kitamichi, a 2 stage combinatorial optimization algorithm.The first stage is a greedy heuristic method in order to produce a feasible initial solution with an acceptable quality in short time. The second stage consists of a random discrete descent method which iteratively minimizes the number of unmatched edges by visiting a neighbour state from the current state while keeping the feasibility. According to their paper [FUNA 99], the algorithm is fast and provides a high solution quality. However as we were looking for an algorithm which provides better results, we did not use it.

Ad hoc algorithms are algorithms designed for a specific problem, which makes them inapplicable to other graph matching problems than the ones they were designed for. This is one of the reasons why we have so many approximate algorithms around, the list of the *ad hoc* algorithms is big, and often the inventor did not understand that he designed a new *ad hoc* algorithm for a graph matching problem.

3.6 Clustering

Equipped with the theory to compute the MCS and the graph edit distance, we are able to find out how similar two graphs are, which works well if we look for patterns in software systems. But we have to be able to find new patterns as well, and to find these, we have to compute the distance of each graph to each other graph, and then to cluster the results. Of course we can not apply the classical clustering algorithms for statistical pattern recognition, since they demand a vector with a constant length, and the graphs can vary in size. However we can apply hierarchical clustering. The idea of this kind of clustering is actually quite simple: Each graph will be put into a cluster of its own, such that if we have twelve graphs, we now have twelve clusters. Then we compute a distance matrix of the clusters, resulting in a twelve times twelve matrix. Having the distance matrix ready, we now merge the two clusters with the shortest distance, resulting in a eleven times eleven matrix. The merged cluster needs to update its distance values. For each distance value we have to take either each time the smallest of the two original, or the biggest, or the average value. We cluster a distance matrix using the minimal values, or the maximal values or the average values. This procedure has to be repeated until there is only one cluster left. This last cluster unfolds in a hierarchy which we can draw as a dendogram. In our dendograms we give each cluster a number which indicates how similar the graphs are. A low number means that the graphs are almost identical, whereas a high number means that the graphs in it do not have a lot in common, compared to the other clusters. In Figure 3.3 we provide an illustration of a small dendogram.

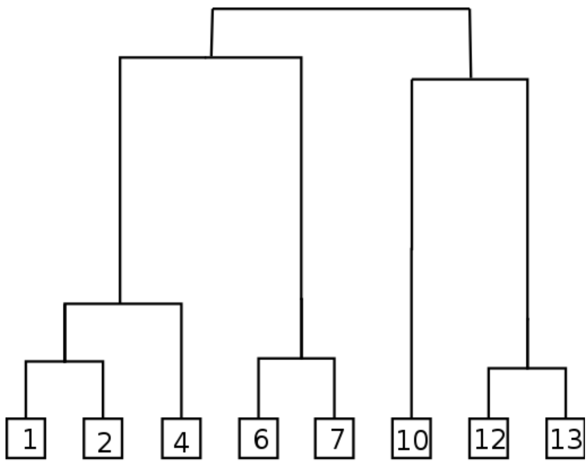

Figure 3.3: Example of dendogram: the shorter the distance between the numbers, the more similar they are

3.7 Conclusion

In this chapter we outlined the theory related to graph pattern recognition. We first defined graphs, then isomorphism between graphs or subgraphs. Later we discussed the inner workings of two maximal common subgraph algorithms and graph edit distance. The final part of the chapter discusses different approaches to solve graph pattern recognition and clustering. With all the necessary parts of graph pattern recognition defined and described, we are ready to go on to the next chapter and apply these theories to the class blueprints.

Chapter 4

Applying Pattern Recognition in Reverse Engineering

4.1 Introduction

In this chapter we discuss how we applied the theories of graph pattern recognition and clustering to the problem of recognizing class patterns using class blueprints. First we explain what problems arise while trying to find a suitable solution, and how we tried to solve them. Then we discuss in detail how we adapted some of the algorithms, and how they performed compared to each other. The first adapted algorithm is based on the Durand-Pasari but with emphasis on the edges, such that it will find the maximal set of independent edges. The definition of an independent edge is discussed in the section of the MaxIndEdge algorithm, as well as its advantages and drawbacks. Next we present an ad hoc algorithm for finding the MCS between two blueprints. It may be *ad hoc* but the idea is useful for other cases of finding an approximate MCS, if we require an algorithm with emphasis on finding specific nodes. It is based on the idea of graph edit distance and was actually our basis of our first approach. The final part of this chapter includes an evaluation of the four algorithms implemented.

4.2 Adapting Pattern Recognition to Class Blueprints

Our first step is to define a graph model which is not as memory consuming as the model in Moose. The Moose model was overloaded with unnecessary information and manipulating the internal data is not very easy. However, when we used this model directly, it did not even provide a solution for small cases. We had to reduce the information stored in the original model to the amount we really needed. The resulting graph model consists of a node with four markings, a numeral one named “lines of code”, and three symbolic ones, “type”, “class instance” and “belongs to layer”. The marking “lines of code” stores the number of lines of code of a method, or in the case of an attribute simply the number “1”. Such a node has a type as well. There are eight different types for methods, and one type for attributes. As an additional marking we have also included a boolean marking for whether this specific node has been found in the class part of the blueprint or in the instance part. The last marking consists of five values, one for each layer, and it specifies in which layer the given node has been found. For the edges there were no markings necessary. We first believed there would be a need for a marking to define whether the edge is an access edge or an invocation edge, but since this is already defined by the two nodes the edge connects, the marking for the edges has been considered redundant. We then applied the theory and algorithms of graph pattern matching on this graph model.

There are a few issues concerning the combination of class blueprints and pattern recognition.

One of the main issues is size dependence. A class may be huge or small, and still have a similar class profile, and uses, in a software system. Since the distance of two graphs is computed using the maximal number of nodes in the MCS, divided by size of the bigger graph, it is much harder to find size-independent class patterns with MCS algorithms. We have tried to solve this using a “discount” function in combination with graph edit distance based algorithms, such that reoccurring subpatterns are becoming cheaper each time they are found in a class, however the results were not satisfying. The other problem is that the orignal measure based on the number of nodes does not work with the kinds of class patterns we have to recognize. Many patterns consist of a few single nodes with a few edges, and having the same just without the edges should not lead to the same result. The solution was a measure combining the number of nodes and the number of edges.

4.2.1 A Measure Combining Number of Edges and Nodes

The first attempt to design a measure which includes the number of edges and nodes failed. Not because it broke any axioms for being a metric but the way it was designed,

$$1-(\frac{numberOfmcsNodes}{numberOfmaxNodes}+\frac{numberOfmcsEdges}{numberOfmaxEdges})*.05 \tag{4.1}$$

did not take into account whether there was only one edge or dozens. The next thing we tried was to treat nodes and edges as equals, such that a graph has distance zero to another graph if and only if their MCS has the same amount of nodes and edges as the original graph.

Definition 13:
The distance between two class graphs is:

$$d(Graph_1, Graph_2) = 1-\frac{mcsNodes+mcsEdges}{max(nodesGraph1, nodesGraph2)+max(edgesGraph1, edgesGraph2)} \tag{4.2}$$

We now show that this distance measure is indeed a metric:

Lemma 2:
$d(Graph_1, Graph_2)$ is a metric if and only if the following axioms hold:

1. $d(Graph_1, Graph_2) \geq 0$
2. $d(Graph_1, Graph_1) = 0$
3. if $d(Graph_1, Graph_2) = 0$ then $V(Graph_1) = V(Graph_2)$ and $E(Graph_1) = E(Graph_2)$
4. $d(Graph_1, Graph_2) = d(Graph_2, Graph_1)$
5. $d(Graph_1, Graph_2) \leq d(Graph_1, Graph_3) + d(Graph_3, Graph_2)$

For the proof see Appendix A. We used it for our algorithms and this metric gives us really useful results. We show some examples as illustration of how well it worked in Chapter 5.

4.3 Adapted Algorithms

In this section we describe our adaptations to the algorithms, and what adaptation we applied to get results which we could use in combination with our new metric and the hierarchical clustering. The changes to graph edit distance and Durand-Pasari are relatively small, such that their results are still the same.

4.3.1 Graph Edit Distance Algorithm

We implemented the graph edit distance in combination with a depth first tree search algorithm. As a lower bound we used the shortest distance found so far, and the order of the candidates is sorted by how similar they were to the node we look for. The similarity is defined by a node edit distance measure. Those two improvements made the algorithm faster. However, as we will show in 4.7, it wasn't fast enough to cope with blueprint graphs. Nevertheless we were able to get some good values for the costs using a few rounds of trial and error and some weeks computing, which we will present in Tables 4.1, 4.3 and 4.2.

To make graph edit distance work we have to define markings first. A marking is a set of values added to a node or an edge, such that we are able to distinguish the edges with different properties. The markings of our model consists of nine different types and a void type, if the type does not matter, two different class instance values and five layer locations, an additional layer 6, if the location of the method may be either be layer 2 or layer 3 and a layer 0, if the location of the method does not matter. These additional "do not care" markings were necessary to make the graph edit distance capable of detecting specific patterns, which sometimes are too general for being found with the normal markings.

1. type values: abstract, attribute, constant, delegater, normal, overrider, reader, supersender, void, writer
2. layer values: 1,2,3,4,5,6,0
3. class instance values: true, false

If we add up these values, we see that we can transform this markings into a single one with 140 different values.

Table 4.1 lists the type exchange cost we use for the graph edit distance algorithms. From the table it is clear that, in comparision with the exchange of delegater and other method types, the costs are rather cheap. The delegater is expensive because high costs lead to a better recognition of some of the given patterns. Such an exchange has to be expensive since a method delegate is rarely changed to something else, unlike supersender or overrider, a change from a delegating method to a normal one is a very fundamental change in the inner workings of a class.

from type	to type	cost	from type	to type	cost
abstract	void	0	constant	delegater	5
attribute	void	0	constant	normal	2
constant	void	0	constant	overrider	2
delegater	void	0	constant	reader	∞
normal	void	0	constant	supersender	2
overrider	void	0	constant	writer	∞
reader	void	0	delegater	delegater	0
supersender	void	0	delegater	normal	5
void	writer	0	delegater	overrider	5
abstract	abstract	0	delegater	reader	∞
abstract	attribute	∞	delegater	supersender	2.5
abstract	constant	2.5	delegater	writer	∞
abstract	delegater	5	normal	normal	0
abstract	normal	2.5	normal	overrider	2
abstract	overrider	1	normal	reader	∞
abstract	reader	∞	normal	supersender	2
abstract	supersender	2.5	normal	writer	∞
abstract	writer	∞	overrider	overrider	0
attribute	attribute	0	overrider	reader	∞
attribute	constant	∞	overrider	supersender	2
attribute	delegater	∞	overrider	writer	∞
attribute	normal	∞	reader	reader	0
attribute	overrider	∞	reader	supersender	∞
attribute	reader	∞	reader	writer	1
attribute	supersender	∞	supersender	supersender	0
attribute	writer	∞	supersender	writer	∞
constant	constant	0	writer	writer	0

Table 4.1: Costs for type exchanges

Type of Change	Cost
Insert Edge	1
Delete Edge	1
Change Class Instance	20

Table 4.2: Costs for lines of code changes, edge changes and class instance changes in one table

from Layer	to Layer	Cost	from Layer	to Layer	Cost
layer0	layer0	0	layer1	layer5	∞
layer0	layer1	10	layer1	layer6	30
layer0	layer2	10	layer2	layer2	0
layer0	layer3	10	layer2	layer3	5
layer0	layer4	10	layer2	layer4	∞
layer0	layer5	10	layer2	layer5	∞
layer0	layer6	10	layer2	layer6	0
layer1	layer1	0	layer3	layer3	0
layer1	layer2	30	layer3	layer4	∞
layer1	layer3	30	layer3	layer5	∞
layer1	layer4	∞	layer4	layer4	0
layer1	layer5	∞	layer4	layer5	∞
layer1	layer1	0	layer4	layer6	∞
layer1	layer2	30	layer5	layer5	0
layer1	layer3	30	layer5	layer6	∞
layer1	layer4	∞	layer6	layer6	0

Table 4.3: Costs for layer exchanges

The cost of a layer change is of course much higher than the costs of type changes. A change of position from the initialisation layer (layer1) to the implementation layer (layer3) is a big change in the design of a class, thus it is very expensive.

With the last costs stated, we have everything we need to let the graph edit distance algorithms run on the class systems we chose as a testbed for our algorithms.

4.3.2 Durand-Pasari Algorithm

The Durand-Pasari proved to be the fastest algorithm when we apply it to rather dense graphs. However, our goal is to have a maximal common subgraph where the number of edges and the number of nodes are maximal, not just the nodes. This leads to the following adaptation of the Durand - Pasari:

Algorithm 3 (Adapted Durand-Pasari):

```
function AdaptedDurandPasari(vertex_list)
   begin
     level=length(vertex_list);
     nullcount=numberOfNulls(vertex_list);
     if(nullcount>=bestnullcount) then
       return;
     else if(level==maxlevel) and (currentNumberOfEdges≤ maxNumberOfEdges) then
       save(vertex_list);
       bestnullcount=nullcount;     currentNumberOfEdges≤ maxNumberOfEdges;
     else
       P=collection of vertices(n1,n2) such that n1==level;
       P=P ⋃ null_vertex;
       P do foreach v in P
         if (v is legal) then
           AdaptedDurandPasari(vertex_list + v)
   end
```

This small adaptation improves the results, and in many cases returns as a result a MCS which meets our expectations exactly. In combination with a heuristic ordering, based again on a node edit distance, we improve the speed of the algorithm compared to the original one. Another attempt to let the Durand-Pasari run on the line graph of a blueprint fails, mainly because it generated far too many incident edges. As a result of taking the Durand-Pasari into this direction, we developed the MaxIndEdge algorithm, which is basically a heavily modified Durand-Pasari.

4.4 New Algorithms

In this section we present our new algorithms, which we intend to ease the problem of the time complexity of the graph pattern matching algorithms. Of course these algorithms are faster, however we paid a price for the speed we gained. Star2Star is an approximation of the graph edit distance which often gives the same results, but not consistently and not often enough to be considered suitable for finding the blueprint patterns defined in [LANZ 03]. It is quite useful to do a quick scan of a big software system to reduce the number of input classes for the more exact algorithms. MaxIndEdge is an algorithm designed to find the maximal edge subgraph of two subgraphs. Its creation was logical because in the case of blueprint patterns there are not often totally identical edges. This algorithm is much faster than the Durand-Pasari or the graph edit distance when applied on our problem domain. A complete evaluation was not possible because of the size of the interesting graphs and the time complexity of the exact algorithms. Although we never found an example where the results between the adpated Durand-Pasari and the MaxIndEdge were different, we cannot prove that this is always the case.

4.4.1 Star2Star Algorithm

This approximate algorithm is based on a very simple idea: A graph is nothing more than a collection of nodes, which are "decorated" with incoming and outgoing edges. We refer to a node with its incoming and outgoing edges as a "star". Now all we have to do is to find the best match for a star in one graph with a star in another graph, and thus we achieve a quick and fairly good approximation of graph edit distance.

Algorithm 4 (Star2Star):

```
function Star2Star(vertex_listA, vertex_listB)
   begin
     nodedistances=matrix[vertex_listA size][vertex_listB size];
     while(i<(vertex_list site))
       while(j<(vertex_list site))
       nodedistances[i][j]=nodeeditdistance(vertex_listA[i],vertex_listB[j]);
       end
     end
     cost=0.0
     candidateindex=0
     while((matrix[0] length) == 0)
       candidateindex=calculateCandidate(matrix)
       tempcost=getCostForIndex(candidateindex, vertex_ListA, vertex_ListB)
       insertcost=(NodeEditDistance(nil, (vertex_ListA[1]))
         +(NodeEditDistance(nil,(vertex_ListB[candidateindex first]))
       if(tempcost>insertcost)
         tempcost=insertcost
         removeFirstRow(matrix)
       else
         removeRowAndColumnFrom(matrix, 1, candidateindex)
         removeFrom(vertex_ListA,1)
         removeFrom(vertex_ListB,candidateindex)
       cost=cost+tempcost.
     end
     cost=cost+calculateInsertCost()
     return cost
   end
```

Algorithm 5 (calculateCandidate):

```
function calculateCandidate(matrix)
     tempcandidatelist=new List
     for(i=1,i<(matrix length),i++)
       tempcandidatelist add(calculateColumnValue(i))
     end
     Sort(tempcandidatelist, (a[1]<b[1]))
     return (tempcandidatelist[0])[0]
   end
```

Algorithm 6 (calculateColumnValue):

```
function calculateColumnValue(index)
    tempvalue=new List
    tempcolumn=getColumn(matrix, index)
    for(i=2,i<(tempcolumn length),i++)
      if((tempvalue length)==0)
        add(tempvalue, i)
        add(tempvalue, tempcolumn[i]
      else
        if(tempvalue[1]>tempcolumn[index]
          tempvalue=new List
          add(tempvalue, i)
          add(tempvalue, tempcolumn[i]
    end
    minvalue=new List
    if((tempvalue length)==0)
      add(minvalue,1)
      add(minvalue,tempcolumn[0]
    else
      add(minvalue,tempvalue[0])
      add(minvalue,(tempcolumn[0]-tempvalue[1]))
    return minvalue
  end
```

We can see that this algorithm does not use recursion. Although this may be good in terms of computing speed, since the worst case efficiency is $O(n) = n^3$, it is bad for the precision of the results. For getting the NP-complete problem of graph edit distance into the P-computability, we had to make some assumptions. This means that the algorithm usually finds the shortest edit distance between two graphs, but not always. In fact the algorithm works for clustering, but fails if we use it to find given class patterns in a blueprint. We cannot conclude that this is a problem of the algorithm, or it is due to a general incompatibilty problem. A test run with graph edit distance had to be aborted after two weeks and as a result we did not proceed further with our investigations.

4.4.2 MaxIndEdge Algorithm

This algorithm was born out of the fact that using a line graph with the Durand-Pasari failed. The reason for the failure is that the number of edges, in the case of line graphs the incident edges, to be considered are high and that there is no such thing as a easy way to determine whether adding the edge to the result set is legal or not. The algorithm we propose here is almost a Durand-Pasari, without edge checking, but with rules to determine whether an edge is legal or not. Furthermore, it removes the unconnected nodes, checks for the maximal common edge subgraph (MCES), and then adds the nodes again and applies a matching algorithm for the nodes not part of the MCES.

```
Algorithm 7 (MaxIndEdge):
function MaxIndEdge(vertex_list,edge_list)
  begin
    level=length(edge_list);
    nullcount=numberOfNulls(edge_list);
    if(nullcount=bestnullcount) and (size(vertex_list<maxsize) then
      return;
    if(nullcount>bestnullcount) then
      return;
    else if(level==maxlevel) then
      save(edge_list);
      save(vertex_list);
      maxsize=size(vertex_list);
      bestnullcount=nullcount;
    else
      P=collection of edges(n1,n2) such that n1==level;
      P=P ⋃ null_edge;
      P do foreach e in P
        if (e is legal(vertex_list)) then
          MaxIndEdge(vertex_list ⋃ verticesof(e), edges_list + e)
  end
```

We see that the algorithm is indeed very similar to the Durand-Pasari. It just features two lists instead of one, and the legality of edges and vertices has changed. An edge is considered legal if and only if the two vertices it connects are legal. A vertex is legal if after its addition to the vertex list, the list is still a bijective mapping. The time complexity for the worst case is the same as Durand Pasari, but because we change the focus from vertices to edges, and edges are in blueprints rarely exactly the same, it is a lot faster and as precise as the original algorithm. In fact we did not encounter any differences in any of the testruns conducted, but we cannot conclusively prove that the results are always be the same.

4.5 Defining Prototypes for Blueprint Patterns

One of the major challenges we face is how to define prototypes which represented the various blueprint patterns. In fact we cannot train or apply a MCS algorithm on a set of samples for a pattern, because we do not have enough samples for the specific patterns. Instead we define the prototypes using a trial and error method, which proves to be quite time consuming, but still faster than looking for examples in the open source software systems. The procedure is quite simple: for each class blueprint pattern defined in [Lanz 03], we identify the properties of a class it must have to feature a specific pattern. As a result we can use the graph pattern matching algorithms to find patterns in a class blueprint based on those prototypes defined by trial and error. Clearly the results are only as reliable as the prototypes and do not always lead to good results. The procedure of finding given patterns in a set of class blueprints works now quite simply: we calculate the MCS for each of the prototypes of the graph to be tested. For each pattern and its given prototype graph, we define a threshold, whose value depends on whether we have an invocation pattern or a node pattern. If the given pattern is definately in the class blueprint we obtain a value of 1.0, if it is not in the class blueprint, we obtain a value of 0.0. Since we have prototypes based mainly on nodes and not invocations, we have to adapt the threshold sometimes to get good results. Mainly the *Adder* pattern is often too restrictive and often misses even a typical example for an *Adder*. For a reference we list in 4.5 for each pattern the threshold when it can be considered to be in the given blueprint graph.

Pattern	Threshold
Size	Not Applicable
No. Method Clumps	1
AdderExtender	0.4
AdderNormal	0.6
AdderOverrider	0.4
AllState	1.0
AllStateClean	1.0
ConstantDefiner	0.8
DataStorage	1.0
Delegater	0.8
Funnel	1.0
Interface	0.8
MicroSpecialExtender	1.0
MicroSpecialOverrider	1.0
SharingEntries	1.0
SingleEntry	1.0
Singleton	1.0
StructuredFlow	1.0
ThreeLayers	1.0
WideInterface	0.8

A listing of the probability of finding a pattern in a class is what we refer to a class profile. It is almost like a fingerpint of the class, and similar class profiles suggests that the classes have similar functions and capabilities.

4.6 Choosing the Right Hierarchical Clustering Strategy

There are three ways of computing a hierarchical clustering: named as single link, average link and complete link. In single link strategy the inner distance of a future cluster is defined as the distance between the two closest points of a cluster, in complete link it is defined as the distance between the two farthest points and in average link it is the calculated average distance of the points. As we can choose three different strategies for the hierarchical clustering, we have to find out, which one yields the best results. Based on a small experiment we concluded that single link gave us a too “deep” clustering, resulting in clusters which contain many different blueprints patterns. The maximal common subgraph of those clusters is often composed of only one or two edges. If we use complete link as strategy, we receive a too “swallow” clustering. By “swallow” clustering we mean that the clusters contain only one specialised version of one pattern. This approach has been proven to be too size dependent, which is a factor we want to avoid. We concluded that average linke strategy is the right strategy to apply in our specfic case. Indeed, average link has been used as strategy since these tests and the clusterings give as result exactly what we expected initially.

4.7 Comparing the four Implemented Algorithms

Now that we have identified these four algorithms, we are faced with the task of finding out which one is best suited to recognizing unknown blueprint patterns. We have to consider how much time an algorithm consumes, as well as how good the results are. An additional source of complexitiy is due to the fact that there is no way of directly comparing the results of a graph edit distance based algorithm and with those generated by MCS algorithms. The reason lies in the difference of the appearance of the results. With graph edit distance, we receive as a result a number between 0 and ∞. If we scale this result to the interval of 0.0 to 1.0 the values of a graph edit distance and a MCS algorithm are still not identical. So what we had to do is choose a pairing of two graphs and calculate the MCS and the graph distance by hand. It is frustrating if the programmer is faster than the program at finding the MCS or the edit costs, but in defence of the entire field of graph pattern matching, we have to say that, firstly a manual approach is subject to human error and secondly that humans have the ability to actually see the entire graph, whereas the automatic apporach based on our chosen algorithms only considers one vertex at a time, without any information about the neighbours of a vertex, or even the neighbours of the neigbours of a vertex. Of course this assumption that humans are faster holds only for small graphs, where you still can see the graph at with one glance, without taking it apart. The blueprints we have chosen for this test are HierarchicalClustering, ProfilChecker and GraphImporter Figure 4.1. Those are classes from the ClassProfiler, our tool, itself.

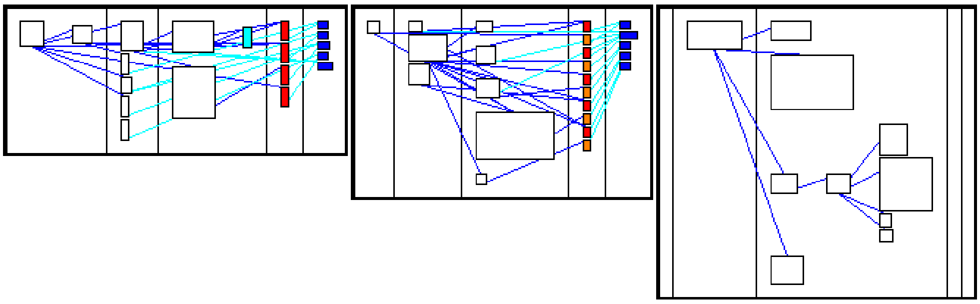

Figure 4.1: From left to right:HierarchicalClustering, ProfileChecker,GraphImporter.

If you check ProfileChecker and GraphImporter for common vertices and edges, you see that they have exactly six vertices and 4 edges in common, which gives us as a result a value of $1 - \frac{10}{58}$ or as a rounded decimal number 0.827586.

The Star2Star incorporates a heuristic to reduce the size dependence, and yields therefore a different clustering. The three other algorithms produce all the same clustering.
This small comparison is not fully reliable, it is at most a hint that we are on the right track with our chosen algorithm. Doing this with larger graphs, where the differences matter even more, is not possible due to the time complexity of the graph edit distance and Durand-Pasari and the complexity finding the edit distance and the MCS of bigger graphs for humans.

Algorithm	Computing Time in Seconds
Graph Edit Distance	2990
Durand Pasari Adapted	1779
MaxIndEdge	0.281
Star2Star	0.041

Table 4.4: Computing Time Of The Different Algorithms Compared

As for the speed, even with those small classes we already see huge differences between the algorithms in Table 4.4. Of course the results for the speed correspond with our expectations, with the Star2Star being the fastest, but also the most unreliable one, and MaxIndEdge coming in second with good precision. The adapted Durand-Pasari and the Graph Edit Distance offers a high level of precision, but the time needed for their computation is almost unbearable for blueprints, where we have to deal with classes conatining up to 100 methods or more. It is interesting to note that the distance matrices of Durand Pasari and MaxIndEdge are identical, as we show in Table 4.5.

0.0	0.816326	0.827586	0.0	0.816326	0.827586
0.816326	0.0	0.689655	0.816326	0.0	0.689655
0.827586	0.689655	0.0	0.827586	0.689655	0.0

Table 4.5: A comparison between the matrix generated for the Durand Pasari (left) and the MaxIndEdge (right).

4.8 Conclusion

In this chapter we outline our own metric for blueprints and we prove that it is indeed a metric. Additionally we show how we adapt the given algorithms to the problem of finding a maximal common subgraph in blueprints and finding the edit distance between graphs, including our costs for the exchange of markings of the vertices and edges. We describe our own, partly modified existing, partly invented, algorithms. Finally we present a comparison between the 3 graphs and explain why we chose the MaxIndEdge as algorithm best suited for the job.
Because the patterns we have to detect automatically consist sometimes of pure vertices or pure edges pattern, we have to invent a metric which can be applied on both situations, and even mixed ones, otherwise we would not be able to find new patterns using clustering techniques. The adaptation of the graph edit distance algorithms and Durand-Pasari are also necessary to allow the use of mixed patterns in our research.
The Star2Star algorithm may prove to be useful for other applications in which a quick screening for graph edit distances is needed, or as a pre stage for the normal graph edit distance, such that we can exclude certain branches of the search tree. MaxIndEdge can be used for other applications as well, where we have detailed edges. The focus on edges instead of vertices can in certain situations (for example comparing two dense graphs with all vertices being identical) be slower than the original algorithms, but such a situation is rare. Furthermore you can, by adding an edge for each edge in the opposite direction, make the MaxIndEdge algorithm even suitable for undirected graphs, at the expense of speed.

Chapter 5

Case Studies

5.1 Introduction

In this chapter we present the results we obtain by applying the `MaxIndEdge` algorithm from Chapter 4 on our case studies. Due to the volume of data we generate, we limit this description to a few examples for each case study in this chapter, the generated data can be found in the appendix. The next part of this chapter is a short explanation of how we conducted the case studies and we show some specifications of our test machine. We begin our discussion with the class profiles of SmallWiki version 1.303, and then a cluster example of the same software system. Afterwards we proceed to Moose version 2.84 and analyse it for clusters and class profile. We also analyse Jun, a framework for 3D graphics for VisualWorks, version 565 for class profiles, but since the class profile list generated a large amount of data, we did not include it in the appendix of the thesis. It is infeasible to calculate the clustering of Jun, because best case estimations show that we would need at least 200 days for a clustering of Jun, which is out of scope of this thesis. However, the data of the two smaller software systems is enough to fill dozens of tables and prove that our algorithm work.

5.2 Case studies in a nutshell

We use the term case study to refer to an example application, a student project or an open source system, to which we apply our analysis technique and generate results which we subsequently evaluate and interpret. In the appendix we provide the dendogram for each clustering and the class profile tables for Moose and Smallwiki. The numbers for each pattern in a class profile represent the probability for a class to have this specific pattern, where a 1.0 means that it features such a class profile, a 0.0 means that this pattern does not feature in the class. A case study is thus a collection of tables, with values of the probability for each pattern to be found in any of the classes and a dendogram. We also show the time it took to calculate the clustering and class profiles. All calculations are performed on a Intel Pentium 4 “Northwood” 2.4 Ghz, running at 2.52 Ghz with 1 Gbyte ram. The operating system is a linux distribution with kernel version 2.6.3, recompiled and adapted to the task to achieve maximal performance.

5.3 Case study 1 - SmallWiki

As an initial case study we choose a SmallWiki, a collaborative content management system written in Smalltalk, developed at the University of Berne. The advantage of choosing Smallwiki is that it is relatively small and its development was carried out by two developers. This results in a homogenous system with consistent use of coding conventions and programming styles. Since we cannot show you for each of the classes the class profile, we incorporated the class profiles into the clustering for this case study. We will outline the complete clustering in the appendix B. To illustrate the system complexity of SmallWiki, we have also included an overview of the software system Figure 5.1. Now let us have a look at some of the clusters found in SmallWiki version 1.303.

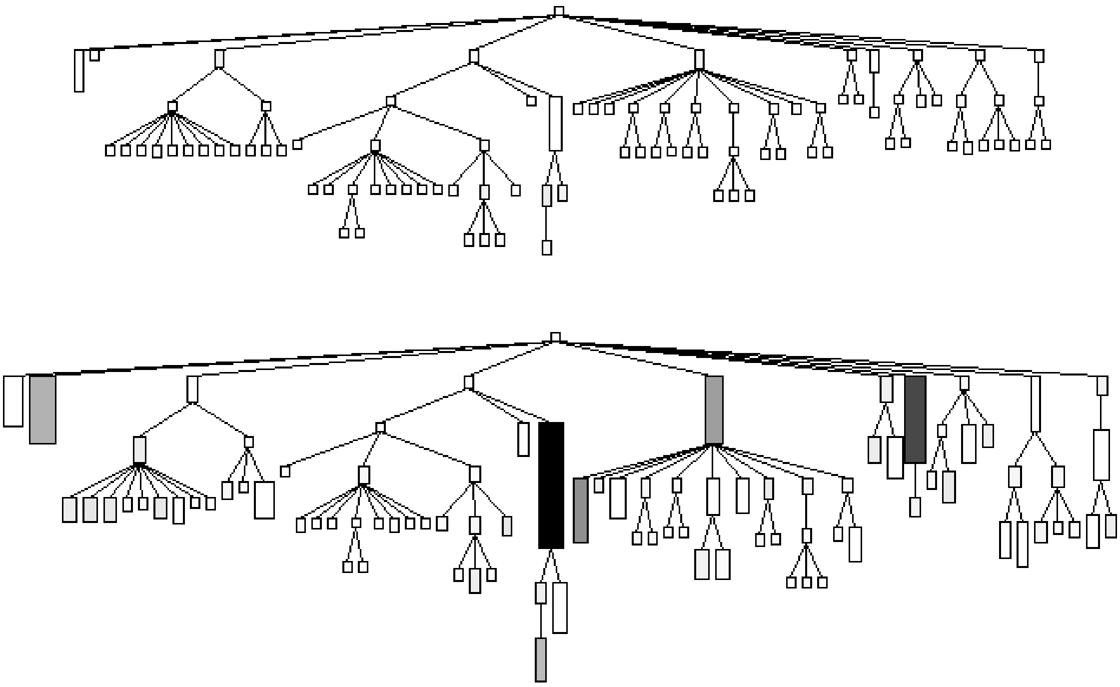

Figure 5.1: The main part of smallwiki: On top the metaclass hierarchy, below the class hierarchy.

SmallWiki 1.303 Cluster 40

In Figure 5.2 we illustrate cluster 40, which is a cluster of similar subclasses. They consists mainly of an extending method in the init layer, an overriding method in the public implementation layer and some getters and setters and attributes. The two single ConstantDefiner are the meta classes of two subclasses below.

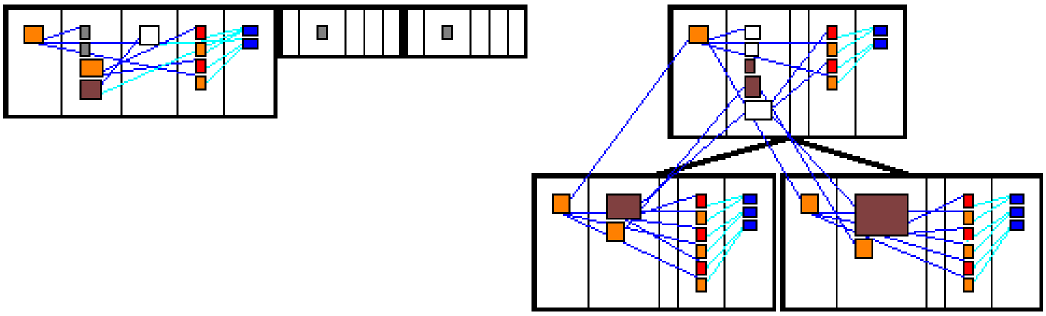

Figure 5.2: Cluster 40 in the Smallwiki 1.303 Clustering. From left to right: FifoCache, metaclass of TemplateBodyW3C, metaclass of TemplateBodySearch, TemplateBody. Below TemplateBody:TemplateBodyW3C and TemplateBodySearch

As an additional proof of similarity, a listing of the found class profiles for cluster 40 in Table 5.1.

Class name	(|#TemplateBody|)	(|#FifoCache|)	(|#TemplateBodyW3C|)	(|#TemplateBodySearch|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.111111	0.111111	0.111111
AdderNormal	0.111111	0.0	0.0	0.0
AdderOverrider	0.222222	0.111111	0.111111	0.111111
AllState1	0.0	0.0	0.0	0.0
AllState2	0.0	0.333333	0.0	0.0
AllState3	0.0	0.333333	0.0	0.0
AllStateClean1	0.666667	0.666667	1.0	1.0
AllStateClean2	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0
DataStorage	0.666667	0.666667	1.0	1.0
Delegater	0.0	0.0	0.0	0.0
Funnel	0.0	0.2	0.0	0.0
Interface	0.333333	0.222222	0.222222	0.222222
MicroSpecialExtender	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0
SharingEntries4	0.6	0.2	0.333333	0.333333
SharingEntries5	0.0	0.2	0.0	0.0
SingleEntry	0.0	0.333333	0.0	0.0
Singleton	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0
ThreeLayers	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0

Table 5.1: SmallWiki 1.303 class profiles for cluster 40

SmallWiki 1.303 Cluster 59

This is another cluster of subclasses. As we see in Figure 5.3 they are quite similar to cluster 40, the previously shown cluster in Figure 5.2, except that they are bigger. Smallwiki has a lot of subclasses and it is not surprising that many of the clusters consists are Extenders or Overriders. This cluster also underlines the problem of the size dependence. Actually they should belong into the same cluster as cluster 40.

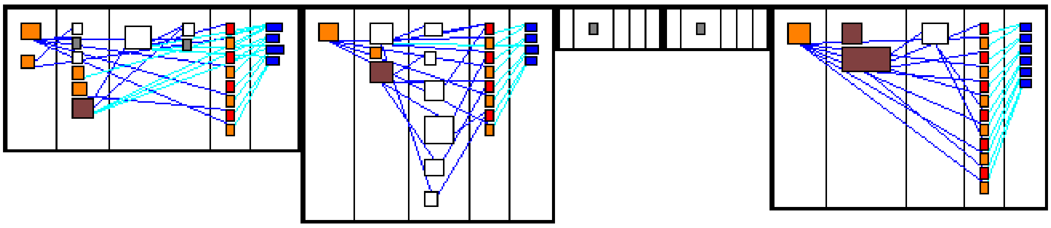

Figure 5.3: Cluster 59 in the Smallwiki 1.303 Clustering. From left to right: ExpiringCache, Search, metaclass of Search, metaclass of TemplateHeadMeta, TemplateHeadMeta. They all got a AllStateClean1 as a main pattern.

The found class profiles indicate that the classes got similarities. (Table 5.2).

Class name	(|#ExpiringCache|)	(|#Search|)	(|#TemplateHeadMeta|)
Size Pattern	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'
AdderExtender	0.117647	0.0588235	0.0
AdderNormal	0.0	0.0588235	0.0
AdderOverrider	0.0588235	0.0588235	0.117647
AllState1	0.0	0.0	0.0
AllState2	1.0	0.333333	0.0
AllState3	1.0	0.0	0.0
AllStateClean1	1.0	1.0	1.0
AllStateClean2	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0
DataStorage	0.8	0.8	1.0
Delegater	0.0	0.0	0.0
Funnel	0.25	0.125	0.25
Interface	0.176471	0.176471	0.117647
MicroSpecialExtender	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0
SharingEntries4	0.125	0.375	0.25
SharingEntries5	0.375	0.125	0.0
SingleEntry	0.333333	0.571429	0.5
Singleton	0.0	0.0	0.0
StructuredFlow	0.5	0.0	0.0
ThreeLayers	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0

Table 5.2: SmallWiki 1.303 class profiles for cluster 59

SmallWiki 1.303 Cluster 66

This cluster in Figure 5.4 consists of toplevel classes, means they only inherit from Object and no other class. The cluster number, 66 out of 93, which is rather low for such big classes, indicates that they must have a lot in common. If we look at the classes in detail, we see that they all feature the ThreeLayer pattern, and a similar public layer, almost an interface, and a lot of accessors and attributes.

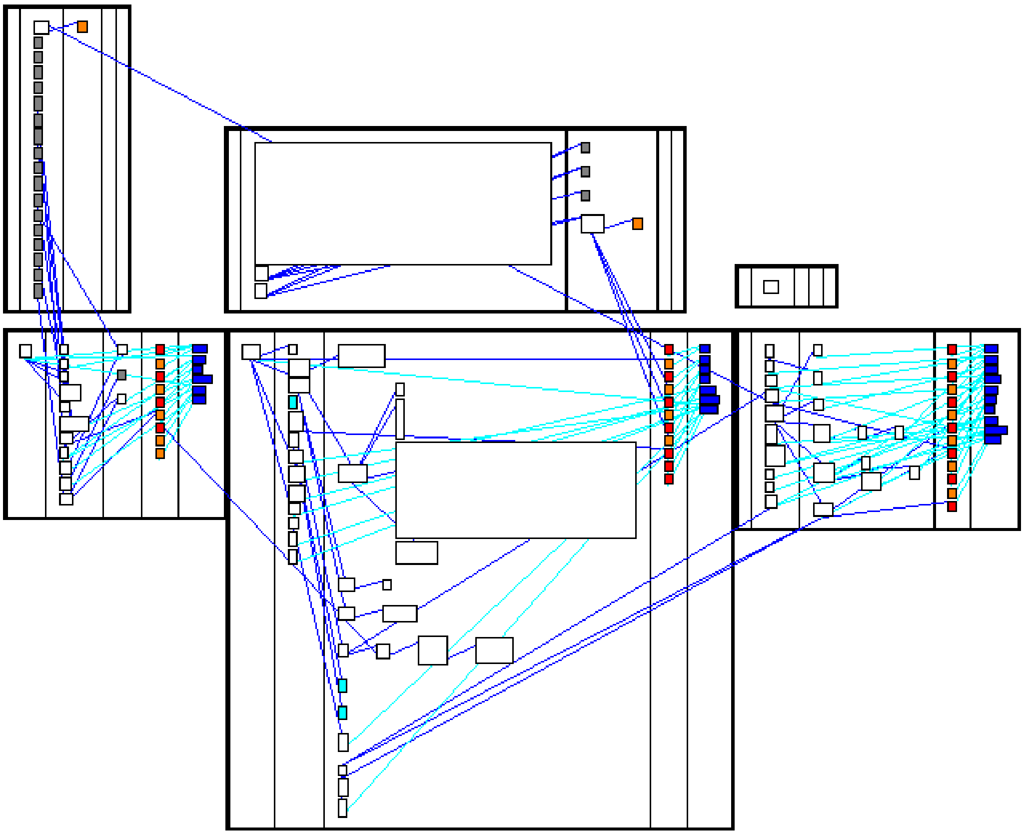

Figure 5.4: Cluster 66 in the Smallwiki 1.303 clustering. From left to right: metaclasses of : Response, Server, Request. Below the classes of: Response, Server and Request

We can see which similar patterns they feature if we look into the class profiles (Table 5.3) of these classes.

Class name	(|#Server|)	(|#Request|)	(|#ExpiringCache|)
Size Pattern	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.117647
AdderNormal	0.155556	0.30303	0.0
AdderOverrider	0.0	0.0	0.0588235
AllState1	0.333333	0.0	0.0
AllState2	0.333333	0.4	1.0
AllState3	0.333333	0.4	1.0
AllStateClean1	0.333333	0.0	1.0
AllStateClean2	0.0	0.333333	0.0
AllStateClean3	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0
DataStorage	0.366667	0.590909	0.8
Delegater	0.0	0.0	0.0
Funnel	0.125	0.125	0.25
Interface	0.177778	0.30303	0.176471
MicroSpecialExtender	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0
SharingEntries4	0.125	0.25	0.125
SharingEntries5	0.625	0.5	0.375
SingleEntry	0.0666667	0.454545	0.333333
Singleton	0.0	0.0	0.0
StructuredFlow	1.0	1.0	0.5
ThreeLayers	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0

Table 5.3: SmallWiki 1.303 class profiles for cluster 66

Applying our technique to SmallWiki does not yield many different results. This does not mean that the developers of this system produced bad code, it is more an indication for the opposite case. Because it is a small software system and has a lot of subclasses, many of the clusters look almost all the same.

Obtaining all this data is time consuming. The clustering took 1 day, 4 hours and 30 minutes, which is a long time for such a small software system. Due to our well chosen subgraphs for the class profile matching, we could keep the speed of profiling of the classes on a low level: It takes only 54 seconds. To make this case study complete, we have included all class profiles of all the classes in SmallWiki 1.303 (See Appendix D).

5.4 Case study 2 - Moose

The Moose reengineering system [Duca 00] is a base framework for our tool ClassProfiler and of CodeCrawler, a language independent reverse engineering tool which combines metrics and software visualization [Lanz]. Moose has been developed by many people and has been refactored and extended during its lifetime. It consists of 193 classes, making it almost twice as big as the SmallWiki software system. The computing time for the clustering was 3 days, 8 hours and 40 minutes, which is an increase of 283% compared to that of SmallWiki 1.303, whereas the size of the class system increased only by 209%. This is an indication that there are bigger classes in Moose than in SmallWiki. Profiling the 193 classes took 192 seconds, thus an average of 1 second per class. We assume that most of time is used for the 5 biggest classes in the system, if you keep in mind that the algorithm has an exponential time complexity. Compared to SmallWiki the profiling time rose by 355%.

We outline all the profiling results in the appendix E. Our approach in the case of Moose is different to the one with SmallWiki. Moose is developped by more people than SmallWiki, making it not so homogenous. Furthermore it has been refactored often and the programming styles have changed over time. These facts suggest that there are many different clusters to find in Moose.

We also handle the concept of showing the class profiles differently. This time around we use the examples of Chapter 2.3 to show that our ClassProfiler works as we expected it do to.

We have a lot of clusters to investigate, as you can see in appendix C. Normally, one would just browse the clustering using our tool ClassProfiler, and by selecting a cluster, we trigger CodeCrawler to generate a visualization of it. This is not possible using paper as a media. Nevertheless we present some examples of clusters in Moose version 2.84.

Moose 2.84 Cluster 95

This is one of the best examples of a DataStorage cluster (Figure 5.5) we have seen in Moose. It consists of The core cluster around the three MOF classes (cluster 35), with the other being added to that cluster one after another. MSEModelClassdescriptor and DetectionStrategy are added to the cluster at the end.

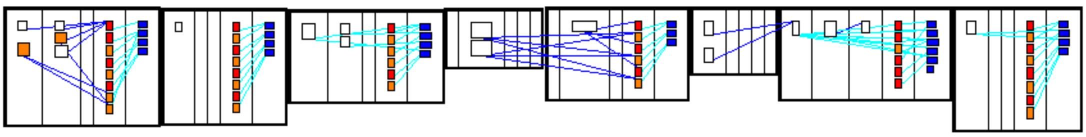

Figure 5.5: Cluster 95 in the Moose 2.84.Lanza.3 Clustering. From left to right: MSEModelClassDescriptor, metaclass of MGSingleMetricFlawDetector, MOFAssociationEnd, MOFAssociation, MGSingleMetricFlawDetector, metaclass of DetectionStrategy, DetectionStrategy and MOFStructuralFeature

Moose 2.84 Cluster 81

In this cluster we discover for the first time the *AllStateClean* pattern. It consists of a *DataStorage* where all the data is set using the accessors, which is a good example for a *AllStateClean* cluster. It is a proof that our method, combining clustering and our algorithm MaxIndEdge, can help to find new class blueprint patterns in a system.

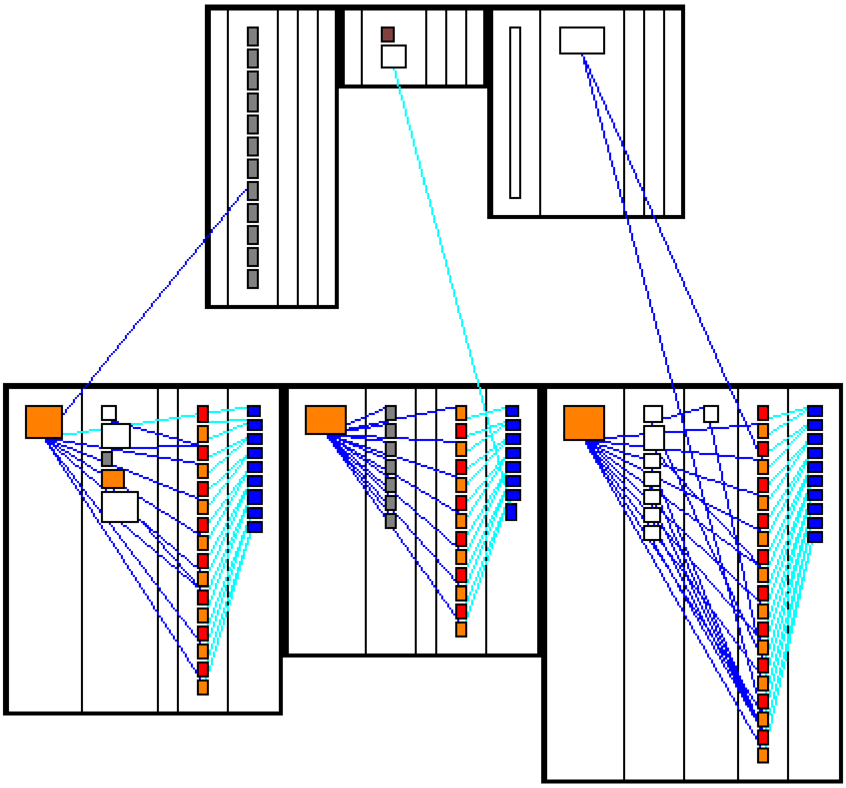

Figure 5.6: Cluster 81 in the Moose 2.84.Lanza.3 Clustering. The metaclass blueprints are above the classes in the same order.From left to right: MSEModelAttributeDescriptor, MGPreferences and MSEModelInformation

Moose 2.84 Cluster 112

This is a cluster(Figure 5.7) of interfaces with SharingEntries4. FAMIXPackage, FAMIXAbstractObject and FAMIXAbstractScopable are the core of the cluster and the two other classes were added later. This is a clear example of how size dependent the MCS algorithms are.

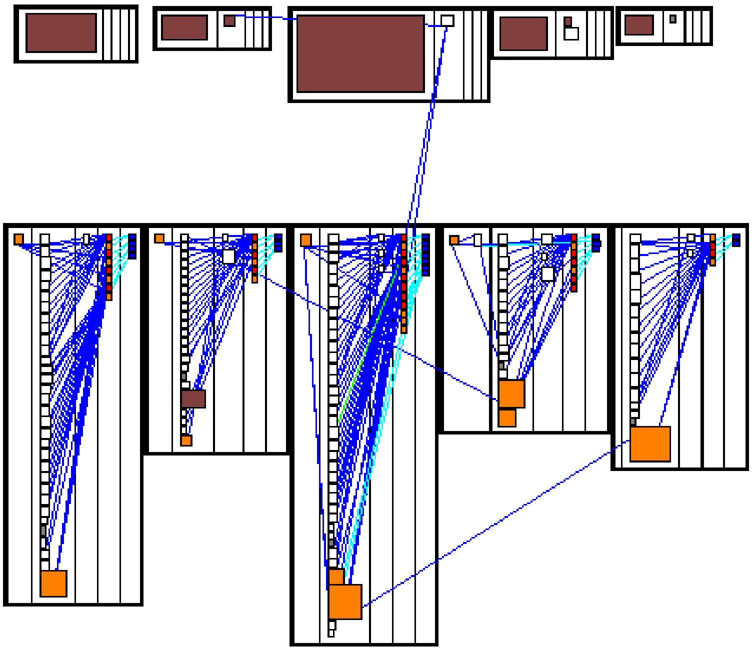

Figure 5.7: Cluster 112 in the Moose 2.84.Lanza.3 Clustering. From left to right (metaclass on top, class below):FAMIXAbstractStructuralEntity, FAMIXPackage, FAMIXInvocation, FAMIXAbstractScopable,FAMIXAbstractObject

Moose 2.84 Cluster 148

Cluster Figure 5.8 is an interesting example. Looking at the classes in detail, we see that they got an extended method , a constant method and the DataStorage like accessor and attribute array in common. Furthermore all the metaclasses got an overloaded method in the init layer and a normal node in the public layer. If we look closer at the cluster, we can even see the three subclusters, namely cluster 63 (MSEProperty and MSEMeasurement), cluster 71 (FAMIX-InheritanceDefinition and FAMIXAccess) and cluster 124 (cluster 71 and FAMIXMethod). This is a good example why it was wise to incorporate the metaclasses into the recognition. It proves that similarity of classes should not just be looked at the instance level, but better over the whole class.

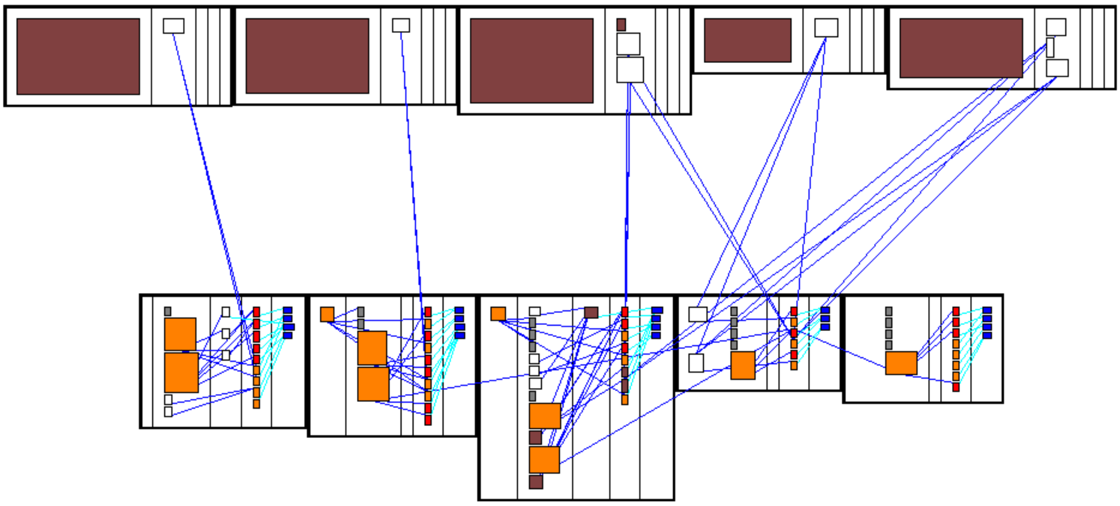

Figure 5.8: Cluster 148 in the Moose 2.84.Lanza.3 Clustering. From left to right (meta-class on top, class below):FAMIXAccess, FAMIXInheritanceDefinition, FAMIXMethod, MSEProperty,MSEMeasurement

Moose 2.84 Cluster 139

A cluster Figure 5.9 full with classes which have a tree like StructuredFlow pattern. These are all classes where structured flows occur, in this case, representing import, parser and layout algorithms.

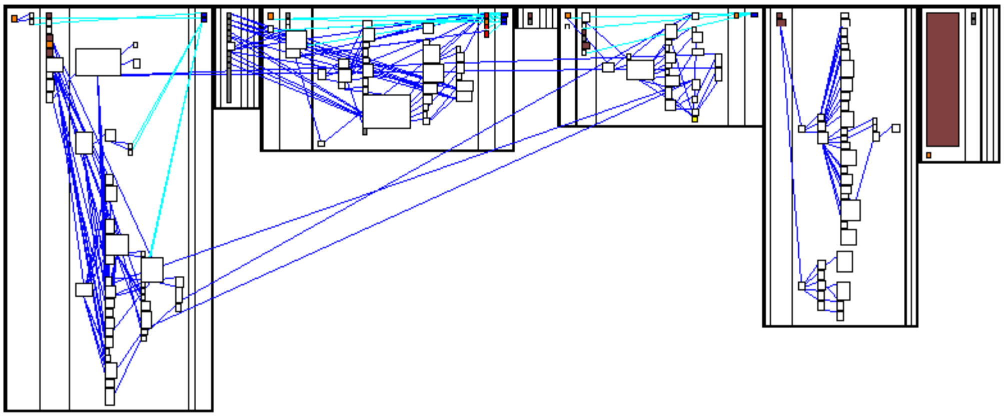

Figure 5.9: Cluster 139 in the Moose 2.84.Lanza.3 Clustering. From left to right:VisualWorksParseTreeEnumerator(no metaclass), MSEEntityView, VisualWorksImporter, LanguageIndependentMetricsOperator

Moose 2.84 Cluster 42

This cluster Figure 5.10 is well suited to show what the classes have in common. We see that the metaclasses are identical. The layer for attributes and accessors seems equal too. In the implementation layer we see that both classes got two constant methods and a minifunnel in common. We have marked every node which the classes do not have in common with a red shade instead of a black one.

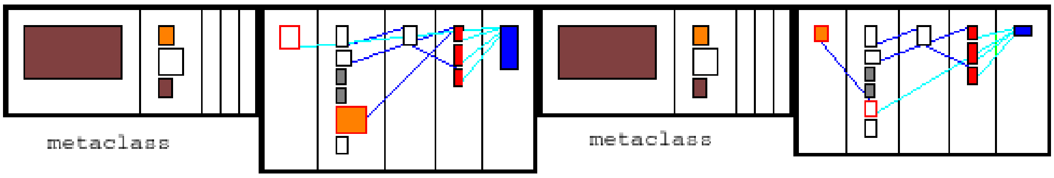

Figure 5.10: Cluster 42 in the Moose 2.84.Lanza.3 Clustering. From left to right:FAMIXAbstractNamedEntity and AbstractNamedEntity

Moose 2.84 Cluster 90

This is one of the stranger clusters (Figure 5.11). It seems clear that they should not have a lot in common, when we keep in mind the high cluster number and remember that the higher the cluster number, the later the classes are put into the clustering, and that both classes are relatively small. Nevertheless we can clearly identify the reason why the classes have been put together. They got an identical metaclass and six nodes in the class matched. Furthermore, they got four edges in common. The bigger class blueprint, DSExpression, has twelve edges and twelve nodes, which gives us a distance of $1 - \frac{6+4}{12+12} = \frac{14}{24}$ which is a fairly big distance as well. One could say that those are just two loners put together with someone they best match to.

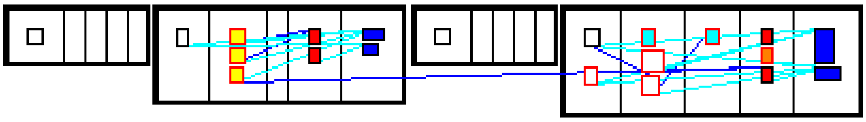

Figure 5.11: Cluster 42 in the Moose 2.84.Lanza.3 Clustering. From left to right:KeyValuePair and DSExpression

Examples of Section 2.3 revisited

We present here for each example we used in the Section 2.3 the corresponding class profile. The purpose of this chapter is to show that the profiles we found in Moose while looking through it ourselves, are also found by our algorithm. We are aware that it lengthens the thesis, but we think there should be a way to directly compare the class profiles to its corresponding class blueprint. Note that we do not show you the metaclasses, therefore there will be no example for a singleton pattern. Furthermore, it does not make much sense to show the *MicroSpecialExtender* and *MicroSpecialOverrider* because they cannot be anything else.

```
Class name              (|#VisualWorksParseTreeMetricCalculator|)
Size Pattern            'Normal'
No. Method Clumps       '0'
AdderExtender           0.296296
AdderNormal             0.037037
AdderOverrider          0.0740741
AllState1               0.25
AllState2               0.5
AllState3               0.25
AllStateClean1          1.0
AllStateClean2          0.333333
AllStateClean3          0.333333
ConstantDefiner         0.0
DataStorage             0.777778
Delegater               0.0
Funnel                  0.125
Interface               0.407407
MicroSpecialExtender    0.0
MicroSpecialOverrider   0.0
SharingEntries4         0.125
SharingEntries5         0.5
SingleEntry             1.0
Singleton               0.0
StructuredFlow          0.0
ThreeLayers             1.0
Wide Interface          0.0
```

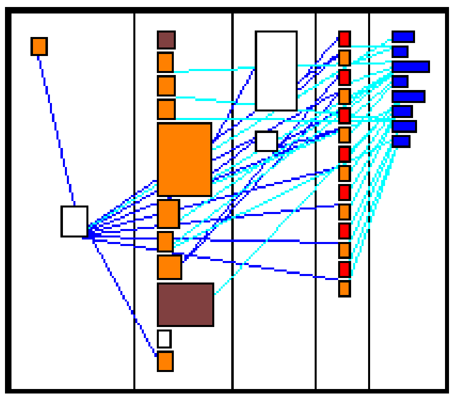

Figure 5.12: The class blueprint of VisualWorksParseTreeMetricCalculator, which is almost an *AdderExtender* and an *AllStateClean*, which its class profile confirms.

Class name	(|#FAMIXAbstractImporter|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0
AdderNormal	0.615385
AdderOverrider	0.0
AllState1	0.333333
AllState2	0.333333
AllState3	0.333333
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0
DataStorage	0.25
Delegater	0.0
Funnel	0.125
Interface	0.615385
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.75
SingleEntry	0.5
Singleton	0.0
StructuredFlow	0.5
ThreeLayers	1.0
Wide Interface	0.0

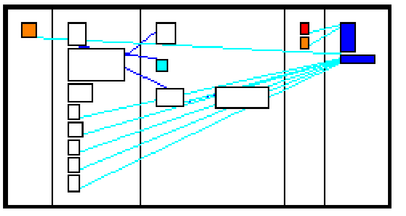

Figure 5.13: The class blueprint of FAMIXAbstractImporter, which is an *AdderNormal* and its class profile.

Class name	(|#MSECDIFSaver|)
Size Pattern	'Normal'
No. Method Clumps	'1'
AdderExtender	0.0952381
AdderNormal	0.190476
AdderOverrider	0.428571
AllState1	1.0
AllState2	0.666667
AllState3	0.333333
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0
DataStorage	0.0714286
Delegater	0.0
Funnel	0.25
Interface	0.714286
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.5
SingleEntry	0.5
Singleton	0.0
StructuredFlow	0.5
ThreeLayers	1.0
Wide Interface	0.0

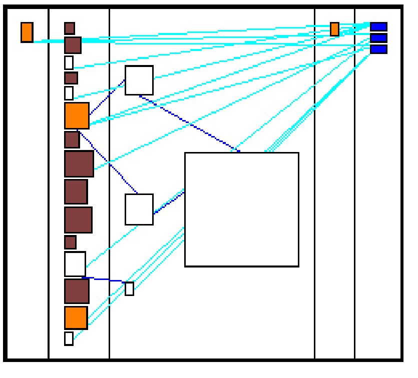

Figure 5.14: The class blueprint of msecdifsaver, which is the closest to an *AdderOverrider* which can be found in moose compared to its class profile.

Class name	(|#MGPreferencesUI|)
Size Pattern	'Normal'
No. Method Clumps	'1'
AdderExtender	0.0
AdderNormal	0.5
AdderOverrider	0.0
AllState1	1.0
AllState2	1.0
AllState3	0.333333
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0
DataStorage	0.0
Delegater	0.0
Funnel	0.0
Interface	0.5
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.375
SingleEntry	0.0
Singleton	0.0
StructuredFlow	0.0
ThreeLayers	0.666667
Wide Interface	0.0

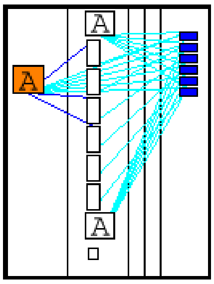

Figure 5.15: MGPreferences is an example with two allstates in the public layer and one allstate in init layer compared to its class profile

Class name	(|#FAMIXModelRoot|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0
AdderNormal	0.185185
AdderOverrider	0.0
AllState1	0.0
AllState2	0.0
AllState3	0.0
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	1.0
DataStorage	0.0
Delegater	0.0
Funnel	0.0
Interface	1.0
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.0
SingleEntry	0.0
Singleton	0.0
StructuredFlow	0.0
ThreeLayers	0.333333
Wide Interface	0.0

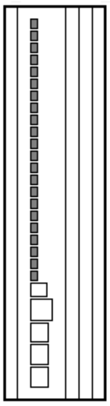

Figure 5.16: The class blueprint of the class FAMIXModelRoot: it contains a distinct *ConstantDefiner* pattern and is an *Interface*, which the class profile seems to confirm.

Class name	(|#MOFAssociationEnd|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0
AdderNormal	0.0
AdderOverrider	0.0
AllState1	0.0
AllState2	0.0
AllState3	0.0
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0
DataStorage	1.0
Delegater	0.0
Funnel	0.0
Interface	0.0
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.0
SingleEntry	0.0
Singleton	0.0
StructuredFlow	0.0
ThreeLayers	0.666667
Wide Interface	0.0

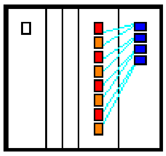

Figure 5.17: The class blueprint of the class MofAssociationEnd: it contains a pure *DataStorage* pattern, also according to its class profile.

Class name	(|#ImporterFacade|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0
AdderNormal	0.37037
AdderOverrider	0.0
AllState1	0.666667
AllState2	0.333333
AllState3	0.0
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.333333
ConstantDefiner	0.0
DataStorage	0.222222
Delegater	1.0
Funnel	0.125
Interface	0.814815
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.125
SharingEntries5	1.0
SingleEntry	0.333333
Singleton	0.0
StructuredFlow	0.0
ThreeLayers	1.0
Wide Interface	0.0

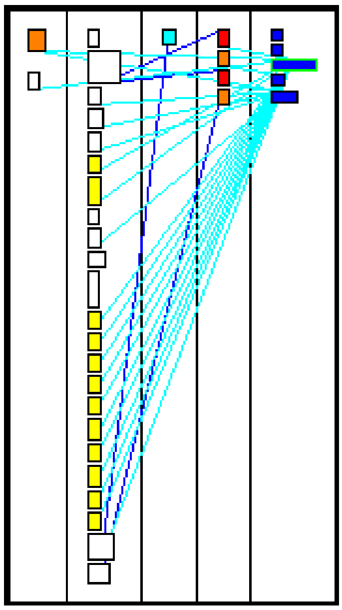

Figure 5.18: The class blueprint of the class ImporterFacade: it mainly consists of a *Delegate* and a *SharingEntries5* pattern. Its class profile says the same

Class name	(|#MSEAbstractGroup|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0434783
AdderNormal	0.130435
AdderOverrider	0.0
AllState1	0.0
AllState2	0.0
AllState3	0.0
AllStateClean1	0.333333
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.533333
DataStorage	0.142857
Delegater	0.833333
Funnel	1.0
Interface	1.0
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.0
SingleEntry	0.5
Singleton	0.0
StructuredFlow	0.0
ThreeLayers	1.0
Wide Interface	0.0

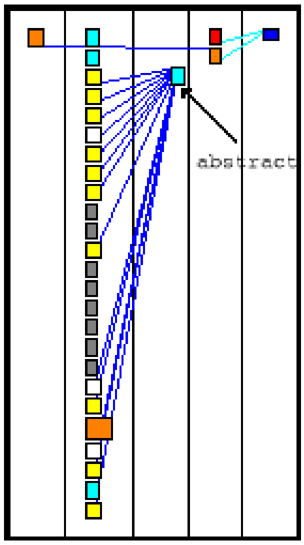

Figure 5.19: A *Funnel* pattern in the class blueprint of the class MSEAbstractGroup compared to its class profile. Which reveals us that it is also an *Interface* and a *Delegate*

Class name	(|#DSMetricOperator|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0833333
AdderNormal	0.0
AdderOverrider	0.166667
AllState1	0.0
AllState2	0.0
AllState3	0.0
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0
DataStorage	0.0
Delegater	0.0
Funnel	0.125
Interface	0.25
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.0
SingleEntry	1.0
Singleton	0.0
StructuredFlow	1.0
ThreeLayers	0.333333
Wide Interface	0.0

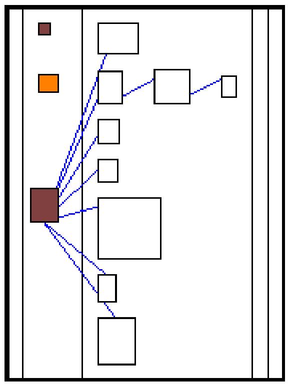

Figure 5.20: The class blueprint of the class DSMetricOperator with one *SingleEntry* pattern. Which the class profile confirms.

Class name	(|#ModelManager|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0
AdderNormal	0.333333
AdderOverrider	0.0
AllState1	0.666667
AllState2	0.666667
AllState3	0.333333
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0
DataStorage	0.0833333
Delegater	0.0
Funnel	0.5
Interface	0.333333
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.0
SharingEntries5	0.625
SingleEntry	1.0
Singleton	1.0
StructuredFlow	1.0
ThreeLayers	1.0
Wide Interface	0.0

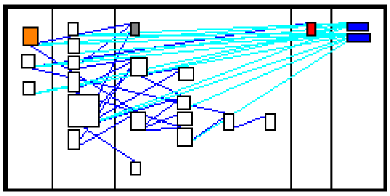

Figure 5.21: The *StructuredFlow* pattern of the class ModelManager and its class profile. Which tells us it is a *Singleton*, a *SingleEntry* and a *ThreeLayers* as well

Class name	(|#MSEMetricManager|)
Size Pattern	'Normal'
No. Method Clumps	'0'
AdderExtender	0.0
AdderNormal	0.971831
AdderOverrider	0.0
AllState1	1.0
AllState2	0.333333
AllState3	0.333333
AllStateClean1	0.0
AllStateClean2	0.0
AllStateClean3	0.0
ConstantDefiner	0.0212766
DataStorage	0.0434783
Delegater	0.0
Funnel	1.0
Interface	0.985915
MicroSpecialExtender	0.0
MicroSpecialOverrider	0.0
SharingEntries4	0.25
SharingEntries5	1.0
SingleEntry	0.75
Singleton	0.0
StructuredFlow	0.5
ThreeLayers	1.0
Wide Interface	0.985915

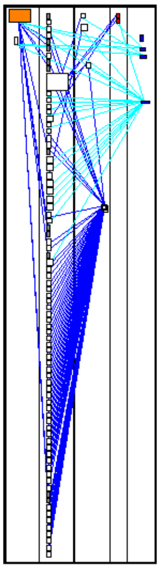

Figure 5.22: The class *MSEMetricManager* has a *Wide Interface* pattern and according to its class profile, an *AllState*1, a *Funnel*, an *AdderNormal* and a *SharingEntries*5 as well.

5.5 Case study 3 - Jun

Jun was our biggest case study. The version we analysed is number 565. It consists of 777 classes, which makes it one of the bigger software systems. With our newest algorithm and subgraph collection edition, we are able to get all the class profiles in just 5 hours. To be exact, it takes 17979 seconds, which is 93 times longer than it takes to calculate the class profiles for Moose. We now assume that the clustering of Jun takes about 60 times longer than the profiling, we end up with a computation time of about 200 days. which is definitely out of our scope. We first tried to cluster Jun 565, but after 4 weeks of computation we only had 206 classes clustered, which makes up for only 21115 of the 301476 distance calculations needed.
We thought that including the results of the clustering in the appendix was not helpful, because we did not see any need to include another 140 tables just to show that it works.

5.6 Conclusion

In this chapter we outline the results of applying out technique to our chosen case studies. The results show that the algorithms are able to find predefined patterns and clusters in the software system. Nevertheless we recognize the limitations of our approach from the time performance when applying it to larger systems like Jun. For smaller software systems, the process of finding patterns and clusters in software system is now automated. Finding class profiles can be done now fast enough even for bigger systems. It does not matter much for the computing speed how many classes a software system has. Due to the design of our algorithms, two huge class blueprints have a much bigger impact on the computing speed than 2000 small ones.
We show that we find the right patterns in a class blueprint and thus have a meaningful class profile for each class. We also show that we are able to detect clusters which make sense, but still are size dependent. There is no solution yet to make it independent of the size, without breaking the system and getting meaningless clusters. The new class blueprint patterns we find in these software systems are either combinations of patterns we already know or a further specialization. By the first few runs of the tool we found a new pattern, which has been identified as a cluster of test classes and thus they had their status of being classes removed, because basically a test class does not provide new functionality to a software system.

Chapter 6

Conclusion

6.1 Introduction

In this chapter we present the result of our work in a nutshell. Initially we answer the questions we have asked in the introduction, then we briefly summarize the work we have presented in this thesis. Subsequently we give a brief outlook into future fields in which we could do further research.

6.2 Answers of the Questions in the Introduction

In this section we answer the questions we asked ourselves at the beginning of this thesis 1.

- Is it possible to write a software tool to recognize class patterns?

Yes. It is possible, but you have to be aware of the limitations which might arise using graph pattern recognition. The negative part of the story is, we are still dependent on the NP-completeness of the problem. We still have to wait several hours or even days to get a result. The good part of the story is that the results are very good. The exact MCS algorithms always find the best matching and the quality of the matching mainly depends on the definition of our prototypes and markings for edges and nodes. The equation is simple: More time, more quality.

- Given the fact that exact graph pattern recognition algorithms are NP-complete, will our tool be fast enough to deal with big software systems?

No. It works only in special cases.
Graph pattern recognition algorithms work very well to find specific patterns, because most of the patterns are relatively small. You can find some examples of the computation time for software systems in Chapter 5. The problem of NP-completness arises if we try to cluster a software system with huge classes, we are able to cluster software systems. As a general rule one can say that as soon as we have two classes with more than 50 nodes, it is possible that we have to wait a long time for the tool to finish its calculations. The computation speed depends on how similar the two classes are and how many similar edges those classes have.

- Are there different ways to solve that problem?

There are a lot of other ideas how to solve the recognition of class patterns. We have focused on graph pattern matching. You can find an outlook of other approaches in Section 3.5.

- Are we able to detect already known class patterns?

Yes. It works because the pattern prototypes consist of only small graphs. The time complexity of the algorithm we used depends on the size of both classes, making it possible to check a huge class for a pattern quickly.

- Has a developer invented a design pattern, or a class pattern without docmenting it?

We do not know. We lack the data and the processor power to screen all available software systems. We did not find new patterns in the software systems we tested in Chapter 5, except for a further specialisation of *AllState*, *AllStateClean*. But we need more case studies to exclude the existence of new class patterns, because we found many combinations of known patterns, which is an indication that our techniques work for finding unknown class patterns.

- Are we able to detect unknown class patterns?

Yes we are. The *AllStateClean* pattern was found this way. And as a example we refer to cluster 81 of the Moose clustering5.4.

6.3 Summary

In this work we present an approach to recognize class blueprint patterns and to cluster a software system according to the similarities in the classes of this system. Those class blueprints and some of the already known patterns are discussed in detail in Chapter 2. We introduced the theory of graph pattern recognition, the maximal common subgraph algorithms and graph edit distance in Chapter 3 and explored the usability of graph edit distance and maximal common subgraph algorithms in Chapter 4, where in the end a maximal common subgraph algorithm provided us with the best mix of precision and speed. Furthermore we had to define a new metric to be able to use the algorithms with the given patterns. This situation arose due to the mixed nature of the already known patterns, such that we had to find a metric which combines the number of common nodes as well as the number of common edges of two graphs, or blueprints. The adaption of the algorithms, the proof of the metric and a quick comparision of the algorithms can be found in Chapter 4. In Chapter 5 we tested our chosen algorithm on three case studies, and discussed the results for two of them in detail. We used our metric to calculate a distance between the prototypes and the graphs to get a class profile, and we used the same metric together with hierarchical clustering to calculate a dendogram of the software systems. All class profiles and the two dendograms of the two smaller software systems are in the appendix. We show some examples of clusters and class profiles in Chapter 5.

6.4 Lessons Learned

As a result of our efforts and exploration we can clearly document pitfalls that can be avoided by researchers considering this path.

- In pattern matching your results are only as good as the prototypes you use, and of course the frameworks you use. Errors and faults in the underlying frameworks can cause a lot of problems. If you obtain weird results, the cause may not necessarily lie in our application, but may also be in the frameworks you use.
- Do not test the algorithms with test suites and small artificial graphs. The results will be really different if you test your algorithms with large artificial graphs or with real samples. It may happen that the new improved algorithm suddenly breaks if it is applied on larger graphs.
- There exist many different names for the same thing. For example, we call MCS maximal common subgraph, others call it maximum common subgraph, and again other scientists call it largest common subgraph. This makes finding the right papers harder than it should be.

- Understanding the correct meaning is crucial. The meaning of, for example, a big graph can be totally different for different fields. A big graph in graph edit distance is about 15 vertices and more, a big class blueprint is 70 nodes and more. It took a while until we got that there exists such a big difference in the ”semantics” of big graphs between the different branches of computer science.

6.5 Future Work

The fields in which we could conduct further research is huge, mainly because we have, on the one hand, combined two fields in which a lot of research is going on, and on the other hand, because we were one of the first to try to combine graph pattern recognition with class patterns.

- If we have the necessary processor power in a few years, we should test all available software systems for clusters at the same time, such that a cluster may consist out of classes from different systems. This is a more efficient way to find new clusters, and then the found new patterns do not come from only a single software developer group, making the found new clusters more meaningful.
- Apply the same techniques to class blueprint inheritance patterns. To achieve this one could use the class profiles found in a class blueprint as a marking for a class node. Then we only have to let the algorithms run over the new set of graphs. But this time we have a whole software system as one graph, which could prove to hard to compute.
- The best way to find new patterns, or class profiles, would be to throw all software systems into the same collection and let the algorithms find the clusters. This idea will be hard to compute as well.
- Use, or invent, better pattern recognition algorithms.
- Try to find or construct prototypes automatically.
- Adapt the tool ClassProfiler such that it becomes multiprocessor capable. This would actually work very well. It is easy to break down a search tree into disjoint subtrees.
- Use genetic algorithms instead of maximum common subgraph algorithms. According to Wang, Fan and Horng [WANG 97] it should speed up the computation.

List of Figures

List of Tables

Appendix A

Proof of Metric Properties of the Measure Combining Number of Edges and Nodes

1. Each graph has at least one node,thus the number of nodes or edges can not be negative, and since $max(nodesGraph1, nodesGraph2) + max(edgesGraph1, edgesGraph2)$ will be never smaller than one, it follows that point 1 holds.

2.

$$
\begin{aligned}
&d(Graph_1, Graph_1) \\
&\quad = 1 - \frac{nodesGraph1 + edgesGraph1}{max(nodesGraph1, nodesGraph1) + max(edgesGraph1, edgesGraph1)} \\
&\qquad = 1 - \frac{nodesGraph1 + edgesGraph1}{nodesGraph1 + edgesGraph1} = 1 - 1 = 0
\end{aligned}
$$

3. $d(Graph_1, Graph_2) = 0 \to (mcsNodes + mcsEdges) = (max(nodesGraph1, nodesGraph2) + max(edgesGraph1, edgesGraph2)) \to V(Graph_1) = V(Graph_2)$ and $E(Graph_1) = E(Graph_2)$. Because if the MCS has the same amount of edges and nodes as the bigger graph, it must be equal to the bigger graph, and if the smaller graph would be smaller, the MCS would be smaller than the bigger graph as well.

4.

$$
\begin{aligned}
&1 - \frac{mcsNodes + mcsEdges}{max(nodesGraph1, nodesGraph2) + max(edgesGraph1, edgesGraph2)} \\
&\quad \equiv 1 - \frac{mcsNodes + mcsEdges}{max(nodesGraph2, nodesGraph1) + max(edgesGraph2, edgesGraph1)}
\end{aligned}
$$

   The distance measure is by definition independent of the order.

5. Since the left side of the equation is by definition smaller than or equal to 1, we have only to show that the right side is bigger than or equal to 1.
   Since this proof will be rather long and we have to write a lot, we introduce first some abbreviations:
   E_n stands for number of edges of graph n
   N_n stands for number of nodes of graph n
   $maxE_{nm}$ stands for the maximum of number of edges of graph n and graph m
   $maxN_{nm}$ stands for the maximum of number of nodes of graph n and graph m

mE_{nm} stands for the of number of edges of the maximal common subgraph of graph n and graph m
mN_{nm} stands for the maximum of number of nodes of the maximal common subgraph of graph n and graph m

The right side is now equivalent to:

$$1 \leq \frac{mE_{13} + mN_{13}}{maxE_{13} + maxN_{13}} + \frac{mE_{23} + mN_{23}}{maxE_{23} + maxN_{23}}$$

which is again equivalent to:

$$\begin{aligned}&(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \\ &\quad \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13})\end{aligned} \tag{A.1}$$

Now, we can see that we have to possible cases, namely that the MCS of graph 1 and graph 3 and the MCS of graph 2 and graph 3 are either disjoint or not.

Case 1: The two MCS are disjoint.
Case 1.1:
$E_1 \geq E_2 \geq E_3$ and $N_1 \geq N_2 \geq N_3$
It follows that:

$$mE_{13} + mN_{13} + mE_{23} + mN_{23} \leq E_3 + N_3 \tag{A.2}$$

because the two MCS are disjoint. Using (4.4) in (4.3) we can show that:
$(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13})$
$\equiv (E_1 + N_1) \times (E_2 + N_2) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) + (mE_{13} + mN_{13}) \times (E_1 + N_1)$
$\rightarrow (E_1 + N_1) \times (E_2 + N_2) \geq (E_1 + N_1) \times (E_3 + N_3) \leq (E_1 + N_1) \times (mE_{13} + mN_{13} + mE_{23} + mN_{23}) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) + (mE_{13} + mN_{13}) \times (E_1 + N_1)$
Case 1.2:
$E_3 \geq E_2 \geq E_1$ and $N_3 \geq N_2 \geq N_1$
It follows that:

$$mE_{13} + mN_{13} + mE_{23} + mN_{23} \leq E_3 + N_3 \tag{A.3}$$

because the two MCS are disjoint.Using (4.5) in (4.3) we can show that:
$(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13})$
$\equiv (E_3 + N_3) \times (E_3 + N_3) \geq (mE_{13} + mN_{13}) \times (E_3 + N_3) + (mE_{13} + mN_{13}) \times (E_3 + N_3)$
$\equiv$ (4.5)
Case 1.3
$E_3 \geq E_2 \geq E_1$ and $N_1 \geq N_2 \geq N_3$
$(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13})$
$\equiv (E_3 + N_1) \times (E_3 + N_2) \geq (mE_{13} + mN_{13}) \times (E_3 + N_2) + (mE_{23} + mN_{23}) \times (E_3 + N_1)$
$\rightarrow (E_3 + N_1) \times (E_3 + N_2) \geq (E_3 + N_1) \times (E_3 + N_3) \geq (E_3 + N_1) \times (mE_{13} + mN_{13} + mE_{23} + mN_{23}) \geq (mE_{13} + mN_{13}) \times (E_3 + N_2) + (mE_{23} + mN_{23}) \times (E_3 + N_1)$ The other subcases of Case 1 can be shown in a similar way.

Case 2: The two MCS are not disjoint .
If the two MCS are not disjoint, then there must exist a nonempty graph u which is the MCS of the two MCS.

$$u = mcsEdges(mE_{13}, mE_{23}) + mcsNodes(mN_{13}, mN_{23}) > 0$$

it follows that:

$$mE_{13} + mN_{13} + mE_{23} + mN_{23} - u \leq E_3 + N_3 \tag{A.4}$$

Now the right side changed to:

$$1 - \frac{u}{maxE_{12} + maxN_{12}} \leq \frac{mE_{13} + mN_{13}}{maxE_{13} + maxN_{13}} + \frac{mE_{23} + mN_{23}}{maxE_{23} + maxN_{23}}$$

which is again equivalent to:

$$\begin{aligned}(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) \geq \\ (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) \\ + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13}) \times (maxE_{12} + maxN_{12}) \\ - u \times (maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23})\end{aligned} \tag{A.5}$$

Again, we show how to prove some of the subcases:
Case 2.1:
$E_1 \geq E_2 \geq E_3$ and $N_1 \geq N_2 \geq N_3$
Using (4.5) in (4.6) leads to:
$(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13}) \times (maxE_{12} + maxN_{12}) - u \times (maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23})$
$\equiv (E_1 + N_1) \times (E_2 + N_2) \times (E_1 + N_1) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) \times (E_1 + N_1) + (mE_{23} + mN_{23}) \times (E_1 + N_1) \times (E_1 + N_1) - u \times (E_1 + N_1) \times (E_2 + N_2)$
$\rightarrow (E_1 + N_1) \times (E_2 + N_2) \times (E_1 + N_1) \geq (E_1 + N_1) \times (E_3 + N_3) \times (E_1 + N_1) \geq (E_1 + N_1) \times (E_1 + N_1) \times (mE_{13} + mN_{13} + mE_{23} + mN_{23} - u) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) \times (E_1 + N_1) + (mE_{23} + mN_{23}) \times (E_1 + N_1) \times (E_1 + N_1) - u \times (E_1 + N_1) \times (E_2 + N_2)$
Case 2.2:
$E_3 \geq E_2 \geq E_1$ and $N_3 \geq N_2 \geq N_1$
Using (4.5) in (4.6) leads to:
$(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13}) \times (maxE_{12} + maxN_{12}) - u \times (maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23})$
$\equiv (E_3 + N_3) \times (E_3 + N_3) \times (E_2 + N_2) \geq (mE_{13} + mN_{13}) \times (E_3 + N_3) \times (E_2 + N_2) + (mE_{23} + mN_{23}) \times (E_3 + N_3) \times (E_2 + N_2) - u \times (E_3 + N_3) \times (E_3 + N_3)$
$\rightarrow (E_3 + N_3) \times (E_2 + N_2) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) + (mE_{23} + mN_{23}) \times (E_2 + N_2) - u \times (E_3 + N_3)$
$\rightarrow (E_3 + N_3) \times (E_2 + N_2) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) + (mE_{23} + mN_{23}) \times (E_2 + N_2) - u \times (E_2 + N_2) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) + (mE_{23} + mN_{23}) \times (E_2 + N_2) - u \times (E_3 + N_3)$
Case 2.3:
$E_3 \geq E_2 \geq E_1$ and $N_1 \geq N_2 \geq N_3$
This is the actual worst case you could ever encounter: $(maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) \geq (mE_{13} + mN_{13}) \times (maxE_{23} + maxN_{23}) \times (maxE_{12} + maxN_{12}) + (mE_{23} + mN_{23}) \times (maxE_{13} + maxN_{13}) \times (maxE_{12} + maxN_{12}) - u \times (maxE_{13} + maxN_{13}) \times (maxE_{23} + maxN_{23})$
$\equiv (E_3 + N_1) \times (E_3 + N_2) \times (E_2 + N_1) \geq (mE_{13} + mN_{13}) \times (E_3 + N_2) \times (E_2 + N_1) + (mE_{23} + mN_{23}) \times (E_3 + N_1) \times (E_2 + N_1) - u \times (E_3 + N_1) \times (E_3 + N_2)$
$\rightarrow (E_3 + N_1) \times (E_3 + N_2) \times (E_2 + N_1) \geq (E_3 + N_1) \times (E_3 + N_3) \times (E_2 + N_1) \geq (E_3 + N_1) \times (E_2 + N_1) \times (mE_{13} + mN_{13} + mE_{23} + mN_{23} - u) = (E_3 + N_1) \times (E_2 + N_1) \times (mE_{13} + mN_{13}) + (E_3 + N_1) \times (E_2 + N_1) \times (mE_{23} + mN_{23}) - (E_3 + N_1) \times (E_2 + N_1) \times u$ We now clearly see that:
$(E_3 + N_1) \times (E_2 + N_1) \times (mE_{13} + mN_{13}) \geq (mE_{13} + mN_{13}) \times (E_2 + N_2) + (mE_{23} + mN_{23})$
and
$(E_3 + N_1) \times (E_2 + N_1) \times (mE_{23} + mN_{23}) = (mE_{23} + mN_{23}) \times (E_3 + N_1) \times (E_2 + N_1)$

both satisfy the assumption. Altough we can not decide yet the relationship between: $(E_3+N_1)\times(E_2+N_1)\times u$ and $(E_3+N_1)\times(E_3+N_2)\times u$. Since we subtract this term, we would like to see something like:
$(E_3+N_1)\times(E_2+N_1)\times u \leq (E_3+N_1)\times(E_3+N_2)\times u$.
We can reduce this problem to three cases:
2.3.1. $(E_3+N_2)<(E_2+N_1)$
2.3.2. $(E_3+N_2)=(E_2+N_1)$
2.3.3. $(E_3+N_2)>(E_2+N_1)$
If we assume cases 1 and 2 we see clearly that we get exactly what we need. Case 3 leads us once again to an undecidable state. There we have again two cases which we have to prove:
2.3.3A. $(E_3+N_3)\leq(E_2+N_1)$
2.3.3B. $(E_3+N_3)>(E_2+N_1)$
Proof of Case 2.3.3A:
We replace (E_2+N_1) with (E_3+N_3) because it is now smaller than (E_3+N_2). $\rightarrow (E_3+N_1)\times(E_3+N_2)\times(E_2+N_1) \geq (E_3+N_1)\times(E_3+N_2)\times(E_3+N_3) \geq (E_3+N_1)\times(E_3+N_2)\times(mE_{13}+mN_{13}+mE_{23}+mN_{23}-u) = (E_3+N_1)\times(E_3+N_2)\times(mE_{13}+mN_{13})+(E_3+N_1)\times(E_2+N_1)\times(mE_{23}+mN_{23})-(E_3+N_1)\times(E_3+N_2)\times u \geq (mE_{13}+mN_{13})\times(E_3+N_2)\times(E_2+N_1)+(mE_{23}+mN_{23})\times(E_3+N_1)\times(E_2+N_1)-u\times(E_3+N_1)\times(E_3+N_2)$
Proof of Case 2.3.3B:
$\rightarrow (E_3+N_1)\times(E_3+N_2)\times(E_2+N_1) \geq (E_3+N_1)\times(E_3+N_3)\times(E_2+N_1) \geq (E_3+N_1)\times(E_2+N_1)\times(mE_{13}+mN_{13}+mE_{23}+mN_{23}-u) = (E_3+N_1)\times(E_2+N_1)\times(mE_{13}+mN_{13})+(E_3+N_1)\times(E_2+N_1)\times(mE_{23}+mN_{23})-(E_3+N_1)\times(E_2+N_1)\times u$
$(E_3+N_1)\times(E_2+N_1)\times(mE_{13}+mN_{13})+(E_3+N_1)\times(E_2+N_1)\times(mE_{23}+mN_{23})-(E_3+N_1)\times(E_2+N_1)\times uR(E_3+N_2)\times(E_2+N_1)\times(mE_{13}+mN_{13})+(E_3+N_1)\times(E_2+N_1)\times(mE_{23}+mN_{23})-(E_3+N_1)\times(E_3+N_2)\times u \rightarrow (E_3+N_1)\times(E_2+N_1)\times(mE_{13}+mN_{13})-(E_3+N_1)\times(E_2+N_1)\times uR(E_3+N_2)\times(E_2+N_1)\times(mE_{13}+mN_{13})-(E_3+N_1)\times(E_3+N_2)\times u$
Because the condition was $(E_3+N_3)>(E_2+N_1)$ it follows that:
$(E_3+N_1)\times(E_2+N_1)\times<u(E_3+N_1)\times(E_3+N_2)\times u$
together with: $(E_3+N_1)\times(E_2+N_1)\times(mE_{13}+mN_{13}) > (E_3+N_2)\times(E_2+N_1)\times(mE_{13}+mN_{13})$ it follows:
$(E_3+N_1)\times(E_2+N_1)\times(mE_{13}+mN_{13})-(E_3+N_1)\times(E_2+N_1)\times u > (E_3+N_2)\times(E_2+N_1)\times(mE_{13}+mN_{13})-(E_3+N_1)\times(E_3+N_2)\times u$
The other cases are solved in a similar way.

$\square$

Appendix B

Smallwiki 1.303 Clustering

```
O--root
|--O--cluster 91
|  |--O--cluster 89
|  |  |--O--cluster 81
|  |  |  |--O--cluster 77
|  |  |  |  |--O--cluster 68
|  |  |  |  |  |--O--cluster 34
|  |  |  |  |  |  |--O--cluster 8
|  |  |  |  |  |  |  |--O--cluster 7
|  |  |  |  |  |  |  |  |--O--cluster 6
|  |  |  |  |  |  |  |  |  |--O--cluster 5
|  |  |  |  |  |  |  |  |  |  |--O--cluster 4
|  |  |  |  |  |  |  |  |  |  |  |--O--cluster 3
|  |  |  |  |  |  |  |  |  |  |  |  |--O--cluster 2
|  |  |  |  |  |  |  |  |  |  |  |  |  |--O--cluster 1
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |--HorizontalRule
|  |  |  |  |  |  |  |  |  |  |  |  |  |  \--UnorderedList
|  |  |  |  |  |  |  |  |  |  |  |  |  \--ListItem
|  |  |  |  |  |  |  |  |  |  |  |  \--MimeView
|  |  |  |  |  |  |  |  |  |  |  \--OrderedList
|  |  |  |  |  |  |  |  |  |  \--Preformatted
|  |  |  |  |  |  |  |  |  \--Document
|  |  |  |  |  |  |  |  \--TableCell
|  |  |  |  |  |  |  \--Paragraph
|  |  |  |  |  |  \--VisitorRecentChanges
|  |  |  |  |  \--O--cluster 54
|  |  |  |  |     |--O--cluster 51
|  |  |  |  |     |  |--O--cluster 36
|  |  |  |  |     |  |  |--O--cluster 19
|  |  |  |  |     |  |  |  |--O--cluster 12
|  |  |  |  |     |  |  |  |  |--O--cluster 11
|  |  |  |  |     |  |  |  |  |  |--O--cluster 10
|  |  |  |  |     |  |  |  |  |  |  |--O--cluster 9
|  |  |  |  |     |  |  |  |  |  |  |  |--PreviousStructure
|  |  |  |  |     |  |  |  |  |  |  |  \--TemplateHeadNavigation
|  |  |  |  |     |  |  |  |  |  |  \--PageView
|  |  |  |  |     |  |  |  |  |  \--NextStructure
|  |  |  |  |     |  |  |  |  \--ParentStructure
|  |  |  |  |     |  |  |  \--TemplateBodyTitle
|  |  |  |  |     |  |  \--TemplateBodyContents
|  |  |  |  |     |  \--O--cluster 25
|  |  |  |  |     |     |--ViewAction
|  |  |  |  |     |     \--AdminRole
```

Table B.1: Dendogramm of the clusters found in SmallWiki 1.303

```
|  |  |  |  |     \--O--cluster 22
|  |  |  |  |        |--Table
|  |  |  |  |        \--TableRow
|  |  |  |  \--O--cluster 63
|  |  |  |     |--O--cluster 43
|  |  |  |     |  |--O--cluster 18
|  |  |  |     |  |  |--TemplateBodyPath
|  |  |  |     |  |  \--O--cluster 13
|  |  |  |     |  |     |--ErrorUnauthorized
|  |  |  |     |  |     \--ErrorNotFound
|  |  |  |     |  \--O--cluster 21
|  |  |  |     |     |--O--cluster20
|  |  |  |     |     |  |--TemplateBodyReferences
|  |  |  |     |     |  \--Logout
|  |  |  |     |     \--RecentChanges
|  |  |  |     \--O--cluster 14
|  |  |  |        |--LinkExternal
|  |  |  |        \--LinkMailTo
|  |  |  \--RedirectAction
|  |  \--O--cluster 35
|  |     |--ResourceHistory
|  |     \--PageHistory
|  \--O--cluster 88
|     |--O--cluster 86
|     |  |--O--cluster 84
|     |  |  |--O--cluster 82
|     |  |  |  |--O--cluster 76
|     |  |  |  |  |--O--cluster 67
|     |  |  |  |  |  |--O--cluster 24
|     |  |  |  |  |  |  |--SessionAction
|     |  |  |  |  |  |  \--InvisibleAction
|     |  |  |  |  |  \--O--cluster 64
|     |  |  |  |  |     |--ImageStorage
|     |  |  |  |  |     \--O--cluster 32
|     |  |  |  |  |        |--O--cluster 30
|     |  |  |  |  |        |  |--VisitorSearch
|     |  |  |  |  |        |  \--Permission
|     |  |  |  |  |        \--VisitorRenderer
|     |  |  |  |  \--O--cluster 72
|     |  |  |  |     |--O--cluster 50
|     |  |  |  |     |  |--O--cluster 31
|     |  |  |  |     |  |  |--O--cluster 16
|     |  |  |  |     |  |  |  |--O--cluster 15
|     |  |  |  |     |  |  |  |  |--Header
|     |  |  |  |     |  |  |  |  \--Text
|     |  |  |  |     |  |  |  \--VisitorReferences
|     |  |  |  |     |  |  \--TemplateHeadTitle
|     |  |  |  |     |  \--O--cluster 38
|     |  |  |  |     |     |--Storage
|     |  |  |  |     |     \--Link
|     |  |  |  |     \--O--cluster 58
|     |  |  |  |        |--O--cluster 53
|     |  |  |  |        |  |--O--cluster 42
|     |  |  |  |        |  |  |--O--cluster 39
|     |  |  |  |        |  |  |  |--O--cluster 28
|     |  |  |  |        |  |  |  |  |--TemplateBodySession
|     |  |  |  |        |  |  |  |  \--VisitorCollector
|     |  |  |  |        |  |  |  \--TemplateBodyCustom

|     |  |  |  |        |  |  \--O--cluster 40 Illustrated on page 53

|     |  |  |  |        |  |     |--O--cluster 27
|     |  |  |  |        |  |     |  |--TemplateBody
|     |  |  |  |        |  |     |  \--FifoCache
|     |  |  |  |        |  |     \--O--cluster 17
|     |  |  |  |        |  |     |  |--TemplateBodyW3C
|     |  |  |  |        |  |     |  \--TemplateBodySearch
```

Table B.2: Dendogramm of the clusters found in SmallWiki 1.303

```
|     |  |  |  |        |  \--O--cluster 49
|     |  |  |  |        |     |--TemplateBodyActions
|     |  |  |  |        |     \--Page
|     |  |  |  |        \--LinkInternal
|     |  |  |  \--O--cluster 73
|     |  |  |     |--O--cluster 26
|     |  |  |     |  |--DocumentComposite
|     |  |  |     |  \--SwazooServer
|     |  |  |     \--O--cluster 65
|     |  |  |        |--Cache
|     |  |  |        \--O--cluster 33
|     |  |  |           |--PropertyManager
|     |  |  |           \--BasicRole
|     |  |  \--O--cluster 60
|     |  |     |--O--cluster 23
|     |  |     |  |--HistoryAction
|     |  |     |  \--ErrorAction
|     |  |     \--O--cluster 29
|     |  |        |--Code
|     |  |        \--WikiScanner
|     |  \--O--cluster 85
|     |     |--O--cluster 83
|     |     |  |--O--cluster 80
|     |     |  |  |--O--cluster 75
|     |     |  |  |  |--O--cluster 69
|     |     |  |  |  |  |--O--cluster 55
|     |     |  |  |  |  |  |--O--cluster 48
|     |     |  |  |  |  |  |  |--SwazooSite
|     |     |  |  |  |  |  |  \--SIXXStorage
|     |     |  |  |  |  |  \--O--cluster 46
|     |     |  |  |  |  |     |--FolderEdit
|     |     |  |  |  |  |     \--O--cluster 41
|     |     |  |  |  |  |        |--Login
|     |     |  |  |  |  |        \--EditAction
|     |     |  |  |  |  \--O--cluster 56
|     |     |  |  |  |     |--ResourceEdit
|     |     |  |  |  |     \--PageEdit
|     |     |  |  |  \--O--cluster 71
|     |     |  |  |     |--O--cluster 45
|     |     |  |  |     |  |--VisitorRendererWiki
|     |     |  |  |     |  \--VisitorRendererHtml
|     |     |  |  |     \--O--cluster 52
|     |     |  |  |     |--User
|     |     |  |  |     \--O--cluster 44
|     |     |  |  |        |--Folder
|     |     |  |  |        \--Resource

|     |     |  |  \--O--cluster 59 Illustrated on page 54

|     |     |  |     |--O--cluster 47
|     |     |  |     |  |--Search
|     |     |  |     |  \--ExpiringCache
|     |     |  |     \--TemplateHeadMeta
|     |     |  \--O--cluster 78
|     |     |     |--O--cluster 61
|     |     |     |  |--SnapshotStorage
|     |     |     |  \--TemplateEdit
|     |     |     \--O--cluster 74

|     |     |        |--O--cluster 66 Illustrated on page 55

|     |     |        |  |--O--cluster 62
|     |     |        |  |  |--Request
|     |     |        |  |  \--Response
|     |     |        |  \--Server
```

Table B.3: Dendogramm of the clusters found in SmallWiki 1.303

```
|     |     |        \--O--cluster 56
|     |     |           |--Action
|     |     |           \--WikiParser
|     |     \--O--cluster 79
|     |        |--Template
|     |        \--Visitor
|     \--O--cluster 70
|        |--HtmlWriteStream
|        \--Structure
\--O--cluster 90
   |--TemplateHead
   \--O--cluster 87
      |--Role
      \--O--cluster 37
         |--WikiItem
         \--SecurityInformation
```

Table B.4: Dendogramm of the clusters found in SmallWiki 1.303

Appendix C

Moose 2.84 Clustering

```
O--root
|--O--cluster 192
|  |--O--cluster 191
|  |  |--O--cluster 190
|  |  |  |--O--cluster 186
|  |  |  |  |--O--cluster 1
|  |  |  |  |  |-- SortExpressionVisitor
|  |  |  |  |  \-- SmallInteger
|  |  |  |  \--O--cluster181
|  |  |  |     |--O--cluster180
|  |  |  |     |  |--O--cluster170
|  |  |  |     |  |  |-- SelectExpressionVisitor
|  |  |  |     |  |  \--O--cluster 142
|  |  |  |     |  |     |--O--cluster 127
|  |  |  |     |  |     |  |--O--cluster 79
|  |  |  |     |  |     |  |  |--O--cluster 57
|  |  |  |     |  |     |  |  |  |--O--cluster 45
|  |  |  |     |  |     |  |  |  |  |--O--cluster 43
|  |  |  |     |  |     |  |  |  |  |  |--O--cluster 34
|  |  |  |     |  |     |  |  |  |  |  |  |--O--cluster 12
|  |  |  |     |  |     |  |  |  |  |  |  |  |--VisualWorksImporterParcelPanel
|  |  |  |     |  |     |  |  |  |  |  |  |  \--VisualWorksImporterNamespacePanel
|  |  |  |     |  |     |  |  |  |  |  |  \--OperatorManagerUI
|  |  |  |     |  |     |  |  |  |  |  \--VisualWorksImporterPackagePanel
|  |  |  |     |  |     |  |  |  |  \--VisualWorksImporterCategoryPanel
|  |  |  |     |  |     |  |  |  \--ModelViewerUI
|  |  |  |     |  |     |  |  \--O--cluster 32
|  |  |  |     |  |     |  |     |--PreferencesUI
|  |  |  |     |  |     |  |     \--MSEModelSaverUI
|  |  |  |     |  |     |  \--O--cluster 114
|  |  |  |     |  |     |     |--O--cluster 77
|  |  |  |     |  |     |     |  |--O--cluster 52
|  |  |  |     |  |     |     |  |  |--O--cluster 31
|  |  |  |     |  |     |     |  |  |  |--MGBypassedAccessorDetector
|  |  |  |     |  |     |     |  |  |  \--COBOLMetricOperator
|  |  |  |     |  |     |     |  |  \--MeasurementOperator
|  |  |  |     |  |     |     |  \--CppMetricOperator
|  |  |  |     |  |     |     \--MSEXMIDTDProducer
|  |  |  |     |  |     \--O--cluster100
|  |  |  |     |  |        |--O--cluster 46
|  |  |  |     |  |        |  |--MGCorrelationChart
|  |  |  |     |  |        |  \--MGStackedBarChartUI
```

Table C.1: Dendogramm of the clusters found in Moose 2.84

```
|  |  |  |     |  |        \--O--cluster 37
|  |  |  |     |  |           |--MGCorrelationMetricsChooserUI
|  |  |  |     |  |           \--MGSingleMetricDetectorUI
|  |  |  |     |  \--O--cluster 166
|  |  |  |     |     |--MSEConstants
|  |  |  |     |     \--FAMIXReporter
|  |  |  |     \--O--cluster 177
|  |  |  |        |--O--cluster 160
|  |  |  |        |  |--O--cluster 136
|  |  |  |        |  |  |--O--cluster 92
|  |  |  |        |  |  |  |--O--cluster 67
|  |  |  |        |  |  |  |  |--MOFNamespace
|  |  |  |        |  |  |  |  \--CompiledMethod
|  |  |  |        |  |  |  \--MSESchema
|  |  |  |        |  |  \--O--cluster 115
|  |  |  |        |  |     |  |--O--cluster 85
|  |  |  |        |  |     |  |  |--MOFReference
|  |  |  |        |  |     |  |  \--VisualLauncher
|  |  |  |        |  |     |  \--O--cluster 51
|  |  |  |        |  |     |     |--MOFImport
|  |  |  |        |  |     |     \--MOFConstraint
|  |  |  |        |  \--O--cluster 89
|  |  |  |        |     |--O--cluster 64
|  |  |  |        |     |  |--ItemService
|  |  |  |        |     |  \--O--cluster 18
|  |  |  |        |     |     |--SingleValueToXMLConvertor
|  |  |  |        |     |     \--MSESingleValueToCDIFConvertor
|  |  |  |        |     \--XMIReader
|  |  |  |        \--O--cluster 157
|  |  |  |           |--O--cluster 133
|  |  |  |           |  |--ModelViewerUISubcanvas
|  |  |  |           |  \--MGModel
|  |  |  |           \--O--cluster 134
|  |  |  |              |--FAMIXModelQueryFacade
|  |  |  |              \--MSEAbstractRoot
|  |  |  \--O--cluster 182
|  |  |     |--O--cluster 176
|  |  |     |  |--O--cluster 161
|  |  |     |  |  |--VisualWorksNamespaceImporter
|  |  |     |  |  \--ItemEditor
|  |  |     |  \--O--cluster 167
|  |  |     |     |--O--cluster 145
|  |  |     |     |  |--O--cluster 98
|  |  |     |     |  |  |--O--cluster 50
|  |  |     |     |  |  |  |--O--cluster 28
|  |  |     |     |  |  |  |  |--O--cluster 27
|  |  |     |     |  |  |  |  |  |--MGMethodCollectionUI
|  |  |     |     |  |  |  |  |  \--ItemChildren
|  |  |     |     |  |  |  |  \--O--cluster 9
|  |  |     |     |  |  |  |     |--O--cluster 8
|  |  |     |     |  |  |  |     |  |--MGSingleClassMetricFlawDetector
|  |  |     |     |  |  |  |     |  \--MGSingleMethodMetricFlawDetector
|  |  |     |     |  |  |  |     \--MGNOPOverrideDetector
|  |  |     |     |  |  |  \--O--cluster 7
|  |  |     |     |  |  |     |--DSAbsoluteOperator
|  |  |     |     |  |  |     \--DSRelativeOperator
|  |  |     |     |  |  \--O--cluster 15
|  |  |     |     |  |     |--CandidateInvocationsWithBaseOperator
|  |  |     |     |  |     \--SmalltalkAnnotatorOperator
|  |  |     |     |  \--O--cluster 105
|  |  |     |     |     |--O--cluster 68
|  |  |     |     |     |  |--O--cluster 11
|  |  |     |     |     |  |  |--MGClassCollectionZU
|  |  |     |     |     |  |  \--MGMisplacedMethodDetector
|  |  |     |     |     |  \--MSEPropertyOperator
|  |  |     |     |     \--SourceImporter
```

Table C.2: Dendogramm of the clusters found in Moose 2.84

```
|  |  |     |     \--O--cluster 154
|  |  |     |        |--O--cluster 29
|  |  |     |        |  |--DSAdapterOperator
|  |  |     |        |  \--DSACompositionOperator
|  |  |     |        \--DSOperand
|  |  |     \--O--cluster 130
|  |  |        |--O--cluster 120
|  |  |        |  |--O--cluster 26
|  |  |        |  |  |--O--cluster 25
|  |  |        |  |  |  |--O--cluster 24
|  |  |        |  |  |  |  |--O--cluster 23
|  |  |        |  |  |  |  |  |--O--cluster 6
|  |  |        |  |  |  |  |  |  |--O--cluster 5
|  |  |        |  |  |  |  |  |  |  |--O--cluster 4
|  |  |        |  |  |  |  |  |  |  |  |--O--cluster 3
|  |  |        |  |  |  |  |  |  |  |  |  |--O--cluster 2
|  |  |        |  |  |  |  |  |  |  |  |  |  |--DSAndOperator
|  |  |        |  |  |  |  |  |  |  |  |  |  \--DSBottomValuesOperator
|  |  |        |  |  |  |  |  |  |  |  |  \--DSHigherThanOperator
|  |  |        |  |  |  |  |  |  |  |  \--DSLowerThanOperator
|  |  |        |  |  |  |  |  |  |  \--DSTopValuesOperator
|  |  |        |  |  |  |  |  |  \--DSOrOperator
|  |  |        |  |  |  |  |  \--FAMIXSourceFile
|  |  |        |  |  |  |  \--ItemLabel
|  |  |        |  |  |  \--MSEModelMVAttributeDescriptor
|  |  |        |  |  \--O--cluster 10
|  |  |        |  |     |--MGAndCombinedFlawDetector
|  |  |        |  |     \--MGOrCombinedFlawDetector
|  |  |        |  \--FAMIXLocalVariable
|  |  |        \--FAMIXAbstractAssociation
|  |  \--O--cluster 188
|  |     |--O--cluster 187
|  |     |  |--O--cluster 184
|  |     |  |  |--O--cluster 183
|  |     |  |  |  |--O--cluster 179
|  |     |  |  |  |  |--O--cluster 174
|  |     |  |  |  |  |  |--O--cluster 163
|  |     |  |  |  |  |  |  |--O--cluster 141
|  |     |  |  |  |  |  |  |  |--O--cluster 121
|  |     |  |  |  |  |  |  |  |  |--O--cluster 113
|  |     |  |  |  |  |  |  |  |  |  |--O--cluster 78
|  |     |  |  |  |  |  |  |  |  |  |  |--O--cluster 53
|  |     |  |  |  |  |  |  |  |  |  |  |  |--O--cluster 21
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |--O--cluster 20
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |  |--O--cluster 14
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |  |  |--O--cluster 13
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |--MOFTypedElement
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  \--MOFConstant
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |  |  \--MOFAttribute
|  |     |  |  |  |  |  |  |  |  |  |  |  |  |  \--VisualWorksImporterAbstractPanel
|  |     |  |  |  |  |  |  |  |  |  |  |  |  \--FAMIXExpressionArgument
|  |     |  |  |  |  |  |  |  |  |  |  |  \--O--cluster 30
|  |     |  |  |  |  |  |  |  |  |  |  |     |--MSECDIFFilteringSaver
|  |     |  |  |  |  |  |  |  |  |  |  |     \--DSNamedExpression
|  |     |  |  |  |  |  |  |  |  |  |  \--O--cluster 36
|  |     |  |  |  |  |  |  |  |  |  |     |--DSFilteringOperator
|  |     |  |  |  |  |  |  |  |  |  |     \--O--cluster 17
|  |     |  |  |  |  |  |  |  |  |  |        |--MOFParameter
|  |     |  |  |  |  |  |  |  |  |  |        \--MOFTag
|  |     |  |  |  |  |  |  |  |  |  \--O--cluster 62
```

Table C.3: Dendogramm of the clusters found in Moose 2.84

```
|  |     |  |  |  |  |  |  |  |  |     |--FAMIXAccessArgument
|  |     |  |  |  |  |  |  |  |  |     \--O--cluster 55
|  |     |  |  |  |  |  |  |  |  |        |--FAMIXUnknownVariable
|  |     |  |  |  |  |  |  |  |  |        \--O--cluster 44
|  |     |  |  |  |  |  |  |  |  |           |--O--cluster 19
|  |     |  |  |  |  |  |  |  |  |           |  |--FAMIXGlobalVariable
|  |     |  |  |  |  |  |  |  |  |           |  \--FAMIXFormalParameter
|  |     |  |  |  |  |  |  |  |  |           \--FAMIXImplicitVariable
|  |     |  |  |  |  |  |  |  |  \--O--cluster 86
|  |     |  |  |  |  |  |  |  |     |--O--cluster 41
|  |     |  |  |  |  |  |  |  |     |  |--MOFDataType
|  |     |  |  |  |  |  |  |  |     |  \--O--cluster 22
|  |     |  |  |  |  |  |  |  |     |     |--MSECDIFFilteringImporter
|  |     |  |  |  |  |  |  |  |     |     \--MGCombinedFlawDetector
|  |     |  |  |  |  |  |  |  |     \--O--cluster 54
|  |     |  |  |  |  |  |  |  |        |--MGFlawDetector
|  |     |  |  |  |  |  |  |  |        \--MGComputedGroup
|  |     |  |  |  |  |  |  |  \--O--cluster 99
|  |     |  |  |  |  |  |  |     |--FAMIXInclude
|  |     |  |  |  |  |  |  |     \--FAMIXAbstractLocalEntity
|  |     |  |  |  |  |  |  \--O--cluster 159

|  |     |  |  |  |  |  |     |--O--cluster 90 Illustrated on page 62

|  |     |  |  |  |  |  |     |  |--KeyValuePair
|  |     |  |  |  |  |  |     |  \--DSExpression
|  |     |  |  |  |  |  |     \--O--cluster 150
|  |     |  |  |  |  |  |        |--O--cluster 138
|  |     |  |  |  |  |  |        |  |--O--cluster 129
|  |     |  |  |  |  |  |        |  |  |--O--cluster 118
|  |     |  |  |  |  |  |        |  |  |  |--O--cluster 103
|  |     |  |  |  |  |  |        |  |  |  |  |--O--cluster 87
|  |     |  |  |  |  |  |        |  |  |  |  |  |--O--cluster 40
|  |     |  |  |  |  |  |        |  |  |  |  |  |  |--MGChartDataGenerator
|  |     |  |  |  |  |  |        |  |  |  |  |  |  \--O--cluster 16
|  |     |  |  |  |  |  |        |  |  |  |  |  |     |--MGTableDataGenerator
|  |     |  |  |  |  |  |        |  |  |  |  |  |     \--MGQueryFacade
|  |     |  |  |  |  |  |        |  |  |  |  |  \--OperatorManager
|  |     |  |  |  |  |  |        |  |  |  |  \--FAMIXAbstractImporter
|  |     |  |  |  |  |  |        |  |  |  \--MSEAbstractTool
|  |     |  |  |  |  |  |        |  |  \--O--cluster 108
|  |     |  |  |  |  |  |        |  |     |--O--cluster 88
|  |     |  |  |  |  |  |        |  |     |  |--O--cluster 33
|  |     |  |  |  |  |  |        |  |     |  |  |--MGTableUI
|  |     |  |  |  |  |  |        |  |     |  |  \--MGItemMetricsUI
|  |     |  |  |  |  |  |        |  |     |  \--O--cluster 74
|  |     |  |  |  |  |  |        |  |     |     |--O--cluster 39
|  |     |  |  |  |  |  |        |  |     |     |  |--FAMIXAbstractArgument
|  |     |  |  |  |  |  |        |  |     |     |  \--Preferences
|  |     |  |  |  |  |  |        |  |     |     \--EntityCommentEditor
|  |     |  |  |  |  |  |        |  |     \--O--cluster 69
|  |     |  |  |  |  |  |        |  |        |--MGClassSelectorUi
|  |     |  |  |  |  |  |        |  |        \--ProgressUI

|  |     |  |  |  |  |  |        |  \--O--cluster 42 Illustrated on page 61

|  |     |  |  |  |  |  |        |     |--FAMIXAbstractNamedEntity
|  |     |  |  |  |  |  |        |     |--AbstractNamedEntity
```

Table C.4: Dendogramm of the clusters found in Moose 2.84

```
|  |     |  |  |  |  |  |        \--O--cluster 126
|  |     |  |  |  |  |  |           |--O--cluster 125
|  |     |  |  |  |  |  |           |  |--O--cluster 95 Illustrated on page 57
|  |     |  |  |  |  |  |           |  |  |--O--cluster 72
|  |     |  |  |  |  |  |           |  |  |  |--MGSingleMetricFlawDetector
|  |     |  |  |  |  |  |           |  |  |  \--O--cluster 48
|  |     |  |  |  |  |  |           |  |  |     |--O--cluster 35
|  |     |  |  |  |  |  |           |  |  |     |  |--MOFAssociationEnd
|  |     |  |  |  |  |  |           |  |  |     |  \--MOFStructuralFeature
|  |     |  |  |  |  |  |           |  |  |     \--MOFAssociation
|  |     |  |  |  |  |  |           |  |  \--MSEModelClassDescriptor
|  |     |  |  |  |  |  |           |  \--DetectionStrategy
|  |     |  |  |  |  |  |           \--O--cluster 97
|  |     |  |  |  |  |  |              |--O--cluster 75
|  |     |  |  |  |  |  |              |  |--MGValueSliderUI
|  |     |  |  |  |  |  |              |  \--MOFGeneralizableElement
|  |     |  |  |  |  |  |              \--O--cluster 82
|  |     |  |  |  |  |  |                 |--O--cluster 62
|  |     |  |  |  |  |  |                 |  |--MOFPackage
|  |     |  |  |  |  |  |                 |  \--MOFClass
|  |     |  |  |  |  |  |                 \--MOFModelElement
|  |     |  |  |  |  |  \--O--cluster 93
|  |     |  |  |  |  |     |--MSEEnumeratedGroup
|  |     |  |  |  |  |     \--FAMIXFunction
|  |     |  |  |  |  \--O--cluster 172
|  |     |  |  |  |     |--O--cluster 156
|  |     |  |  |  |     |  |--O--cluster 148 Illustrated on page 60
|  |     |  |  |  |     |  |  |--O--cluster 63
|  |     |  |  |  |     |  |  |  |--MSEProperty
|  |     |  |  |  |     |  |  |  \--MSEMeasurement
|  |     |  |  |  |     |  |  \--O--cluster 124
|  |     |  |  |  |     |  |     |--O--cluster 71
|  |     |  |  |  |     |  |     |  |--FAMIXInheritanceDefinition
|  |     |  |  |  |     |  |     |  \--FAMIXAccess
|  |     |  |  |  |     |  |     \--FAMIXMethod
|  |     |  |  |  |     |  |--VisualWorksPackageImporter
|  |     |  |  |  |     \--O--cluster 169
|  |     |  |  |  |        |--O--cluster 153
|  |     |  |  |  |        |  |--O--cluster 144
|  |     |  |  |  |        |  |  |--O--cluster 96
|  |     |  |  |  |        |  |  |  |--FAMIXNamespace
|  |     |  |  |  |        |  |  |  \--MSEMetric
|  |     |  |  |  |        |  |  \--FAMIXAttribute
|  |     |  |  |  |        |  \--ImporterFacade
|  |     |  |  |  |        \--O--cluster 131
|  |     |  |  |  |     |  |--O--cluster 81 Illustrated on page 58
|  |     |  |  |  |           |  |--O--cluster 65
|  |     |  |  |  |           |  |  |--MGPreferences
|  |     |  |  |  |           |  |  \--MSEModelAttributeDescriptor
|  |     |  |  |  |           |  \--MSEModelInformation
|  |     |  |  |  |           \--VisualWorksParseTreeMetricCalculator
|  |     |  |  |  \--O--cluster 178
|  |     |  |  |     |--O--cluster 168
|  |     |  |  |     |  |--SmalltalkClassMetricsOperator
|  |     |  |  |     |  \--MSEAbstractOperator
```

Table C.5: Dendogramm of the clusters found in Moose 2.84

```
|   |     |  |  |     \--O--cluster 173
|   |     |  |  |        |--O--cluster 165
|   |     |  |  |        |  |--O--cluster 109
|   |     |  |  |        |  |  |--VisualWorksImporterFacade
|   |     |  |  |        |  |  \--O--cluster 84
|   |     |  |  |        |  |     |--VisualWorksParseTreeAnnotator
|   |     |  |  |        |  |     \--O--cluster 60
|   |     |  |  |        |  |        |--DSMetricOperator
|   |     |  |  |        |  |        \--CandidateInvocationsOperator
|   |     |  |  |        |  \--O--cluster 158
|   |     |  |  |        |     |--O--cluster 143
|   |     |  |  |        |     |  |--O--cluster 137
|   |     |  |  |        |     |  |  |--O--cluster 122
|   |     |  |  |        |     |  |  |  |--O--cluster 110
|   |     |  |  |        |     |  |  |  |  |--O--cluster 56
|   |     |  |  |        |     |  |  |  |  |  |--O--cluster 38
|   |     |  |  |        |     |  |  |  |  |  |  |--MGPreferencesUI
|   |     |  |  |        |     |  |  |  |  |  |  \--MGMetricsUI
|   |     |  |  |        |     |  |  |  |  |  \--VisualWorksImporterUI
|   |     |  |  |        |     |  |  |  |  \--O--cluster 104
|   |     |  |  |        |     |  |  |  |     |--O--cluster 76
|   |     |  |  |        |     |  |  |  |     |  |--MGOverviewUI
|   |     |  |  |        |     |  |  |  |     |  \--MGFlawDetectorCombinatorUI
|   |     |  |  |        |     |  |  |  |     \--O--cluster 91
|   |     |  |  |        |     |  |  |  |        |--VisualWorksParseTree
|   |     |  |  |        |     |  |  |  |        \--O--cluster 73
|   |     |  |  |        |     |  |  |  |           |--MGBarChartUI
|   |     |  |  |        |     |  |  |  |           \--MSEAbstractSchemaSaver
|   |     |  |  |        |     |  |  |  \--O--cluster 80
|   |     |  |  |        |     |  |  |     |--MSEToolManager
|   |     |  |  |        |     |  |  |     \--O--cluster 58
|   |     |  |  |        |     |  |  |        |--XMIReaderHandler
|   |     |  |  |        |     |  |  |        \--ModelManager
|   |     |  |  |        |     |  |  \--O--cluster 59
|   |     |  |  |        |     |  |     |--XMLSaver
|   |     |  |  |        |     |  |     \--MSECDIFSaver
|   |     |  |  |        |     |  \--O--cluster 101
|   |     |  |  |        |     |     |--Group
|   |     |  |  |        |     |     \--MSEAbstractImporter
|   |     |  |  |        |     \--O--cluster 149
|   |     |  |  |        |        |--O--cluster 119
|   |     |  |  |        |        |  |--FileIOFacade
|   |     |  |  |        |        |  \--SmalltalkMetricOperator
|   |     |  |  |        |        \--O--cluster 128
|   |     |  |  |        |           |--O--cluster 106
|   |     |  |  |        |           |  |--MGItemPropertyTableModel
|   |     |  |  |        |           |  \--MSECDIFScanner
|   |     |  |  |        |           \--MGItemCollectionUI
|   |     |  |  |        \--MSESingleValueConvertor
|   |     |  |  \--O--cluster 175
|   |     |  |     |--O--cluster 116
|   |     |  |     |  |--MSEImporter
|   |     |  |     |  \--MSEUUIDGenerator
|   |     |  |     \--O--cluster 147
|   |     |  |        |--MSEAbstractGroup
|   |     |  |        \--FAMIXModelRoot
|   |     |  \--O--cluster 185
```

Table C.6: Dendogramm of the clusters found in Moose 2.84

```
|  |     |     |--O--cluster 171
|  |     |     |  |--O--cluster 164
|  |     |     |  |  |--O--cluster 155
|  |     |     |  |  |  |--O--cluster 146
|  |     |     |  |  |  |  |--O--cluster 135
|  |     |     |  |  |  |  |  |--O--cluster 112 Illustrated on page 59
|  |     |     |  |  |  |  |  |  |--O--cluster 49
|  |     |     |  |  |  |  |  |  |  |--O--cluster 47
|  |     |     |  |  |  |  |  |  |  |  |--FAMIXAbstractObject
|  |     |     |  |  |  |  |  |  |  |  \--FAMIXAbstractScopable
|  |     |     |  |  |  |  |  |  |  \--FAMIXPackage
|  |     |     |  |  |  |  |  |  \--O--cluster 70
|  |     |     |  |  |  |  |  |     |--FAMIXAbstractStructuralEntity
|  |     |     |  |  |  |  |  |     \-_FAMIXInvocation
|  |     |     |  |  |  |  |  \--MooseGagerUI
|  |     |     |  |  |  |  \--MGMetricsFacade
|  |     |     |  |  |  \--O--cluster 140
|  |     |     |  |  |     |--O--cluster 102
|  |     |     |  |  |     |  |--PluggableTableAdaptor
|  |     |     |  |  |     |  \--O--cluster 83
|  |     |     |  |  |     |     |--MSECDIFImporter
|  |     |     |  |  |     |     \--MSEMooseLauncher
|  |     |     |  |  |     \--O--cluster 111
|  |     |     |  |  |        |--GroupUI
|  |     |     |  |  |        \--ModelLoaderUI
|  |     |     |  |  \--AbstractEntity
|  |     |     |  \--O--cluster 162
|  |     |     |     |--O--cluster 151
|  |     |     |     |  |--O--cluster 94
|  |     |     |     |  |  |--FAMIXAbstractBehaviouralEntity
|  |     |     |     |  |  \--MSEMetricManager
|  |     |     |     |  \--O--cluster 107
|  |     |     |     |     |--FAMIXClass
|  |     |     |     |     \--MSEModel
|  |     |     |     \--ImportingContext
|  |     |     \--O--cluster 152
|  |     |        |--O--cluster 139 Illustrated on page 61
|  |     |        |  |--O--cluster 117
|  |     |        |  |  |--VisualWorksImporter
|  |     |        |  |  \--LanguageIndependentMetricsOperator
|  |     |        |  \--O--cluster 123
|  |     |        |     |--VisualWorksParseTreeEnumerator
|  |     |        |     \--MSEEntityView
|  |     |        \--XMIDTDProducer
|  |     \--ContextualMetricOperator
|  \--O--cluster 189
|     |--O--cluster 132
|     |  |--O--cluster 66
|     |  |  |--MOFTypeAlias
|     |  |  \--FamixModelSmalltalkQueryFacade
|     |  \--MGRoot
|     \--MSEMinimalCDIFReadStream
\--MSEApplicationModel
```

Table C.7: Dendogramm of the clusters found in Moose 2.84

Appendix D

SmallWiki 1.303 Results

Class name	(|#HorizontalRule|)	(|#MimeView|)	(|#UnorderedList|)	(|#ListItem|)	(|#TemplateHead|)	(|#OrderedList|)	(|#Paragraph|)
Size Pattern	’Single’	’Single’	’Single’	’Single’	’Single’	’Single’	’Single’
No. Method Clumps	’0’	’0’	’0’	’0’	’0’	’0’	’0’
AdderExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	0.5	0.0	0.0
AdderOverrider	0.5	0.5	0.5	0.5	0.0	0.5	0.5
AllState1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Interface	0.5	0.5	0.5	0.5	0.5	0.5	0.5
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0	0.0	1.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table D.1: SmallWiki 1.303 Profiles

Class name	(|#Preformatted|)	(|#Document|)	(|#TableCell|)	(|#ResourceHistory|)	(|#PreviousStructure|)	(|#TemplateHeadNavigation|)
Size Pattern	’Single’	’Single’	’Single’	’Micro’	’Micro’	’Micro’
No. Method Clumps	’0’	’0’	’0’	’0’	’0’	’0’
AdderExtender	0.0	0.0	0.0	0.5	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	0.0	0.0
AdderOverrider	0.5	0.5	0.5	0.0	0.5	0.5
AllState1	0.0	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0	0.0
Interface	0.5	0.5	0.5	0.5	0.5	0.5
MicroSpecialExtender	0.0	0.0	0.0	1.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	0.0	1.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table D.2: SmallWiki 1.303 Profiles

Class name	(|#PageView|)	(|#NextStructure|)	(|#ParentStructure|)	(|#ParentStructure|)	(|#TemplateBodyPath|)	(|#Role|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	0.0	0.0
AdderOverrider	0.5	0.5	0.5	0.5	0.5	0.0
AllState1	0.0	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0	0.0
Interface	0.5	0.5	0.5	0.5	0.5	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0	1.0	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table D.3: SmallWiki 1.303 Profiles

Class name	(|#VisitorRecentChanges|)	(|#ErrorUnauthorized|)	(|#ErrorNotFound|)	(|#TemplateBodyReferences|)	(|#PageHistory|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.5	0.0	0.0	0.0	0.5
AdderNormal	0.0	0.0	0.0	0.0	0.0
AdderOverrider	0.5	1.0	1.0	0.5	0.0
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.333333	0.333333
Interface	1.0	1.0	1.0	0.5	0.5
MicroSpecialExtender	1.0	0.0	0.0	0.0	1.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.5	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table D.4: SmallWiki 1.303 Profiles

Class name	(|#Table|)	(|#RedirectAction|)	(|#TableRow|)	(|#ViewAction|)	(|#TemplateBodyTitle|)	(|#Logout|)	(|#LinkExternal|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AdderNormal	1.0	0.0	0.5	0.0	0.0	0.0	0.0
AdderOverrider	0.5	0.5	0.5	0.5	0.5	0.666667	0.666667
AllState1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.5	1.0	0.5	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.666667	0.0	0.0	0.0	0.25	0.0
Interface	1.0	0.5	1.0	1.0	1.0	0.666667	0.666667
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0	1.0	1.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	1.0	0.0	0.0	0.0	0.5	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.5	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table D.5: SmallWiki 1.303 Profiles

<table>
<tr><td>Class name</td><td>(|#RecentChanges|)</td><td>(|#WikiItem|)</td><td>(|#SecurityInformation|)</td><td>(|#LinkMailTo|)</td><td>(|#SessionAction|)</td><td>(|#Header|)</td></tr>
<tr><td>Size Pattern</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td></tr>
<tr><td>No. Method Clumps</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td></tr>
<tr><td>AdderExtender</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AdderNormal</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AdderOverrider</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.666667</td><td>0.0</td><td>0.333333</td></tr>
<tr><td>AllState1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllState2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllState3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AllStateClean1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ConstantDefiner</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>DataStorage</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.25</td><td>0.5</td></tr>
<tr><td>Delegater</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>Funnel</td><td>0.25</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.5</td><td>0.0</td></tr>
<tr><td>Interface</td><td>0.666667</td><td>0.333333</td><td>0.333333</td><td>0.666667</td><td>0.333333</td><td>0.333333</td></tr>
<tr><td>MicroSpecialExtender</td><td>0.5</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.5</td><td>0.0</td></tr>
<tr><td>MicroSpecialOverrider</td><td>0.5</td><td>0.0</td><td>0.0</td><td>1.0</td><td>0.0</td><td>0.5</td></tr>
<tr><td>SharingEntries4</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries5</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.5</td><td>0.0</td></tr>
<tr><td>SingleEntry</td><td>0.5</td><td>0.0</td><td>0.5</td><td>0.0</td><td>0.5</td><td>0.0</td></tr>
<tr><td>Singleton</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>StructuredFlow</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ThreeLayers</td><td>0.333333</td><td>0.333333</td><td>0.333333</td><td>0.333333</td><td>1.0</td><td>1.0</td></tr>
<tr><td>Wide Interface</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
</table>

Table D.6: SmallWiki 1.303 Profiles

<table>
<tr><td>Class name</td><td>(|#AdminRole|)</td><td>(|#InvisibleAction|)</td><td>(|#TemplateBodyContents|)</td><td>(|#Text|)</td><td>(|#VisitorReferences|)</td><td>(|#HistoryAction|)</td></tr>
<tr><td>Size Pattern</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td></tr>
<tr><td>No. Method Clumps</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td></tr>
<tr><td>AdderExtender</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AdderNormal</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.25</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AdderOverrider</td><td>0.333333</td><td>0.333333</td><td>0.666667</td><td>0.25</td><td>0.5</td><td>0.4</td></tr>
<tr><td>AllState1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllState2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AllState3</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ConstantDefiner</td><td>1.0</td><td>0.0</td><td>0.5</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>DataStorage</td><td>0.0</td><td>0.25</td><td>0.0</td><td>0.5</td><td>0.5</td><td>0.0</td></tr>
<tr><td>Delegater</td><td>1.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>Funnel</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.142857</td></tr>
<tr><td>Interface</td><td>1.0</td><td>0.666667</td><td>1.0</td><td>0.5</td><td>0.5</td><td>0.4</td></tr>
<tr><td>MicroSpecialExtender</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>MicroSpecialOverrider</td><td>0.5</td><td>0.5</td><td>1.0</td><td>0.333333</td><td>0.666667</td><td>0.666667</td></tr>
<tr><td>SharingEntries4</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries5</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>SingleEntry</td><td>0.0</td><td>0.5</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.4</td></tr>
<tr><td>Singleton</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>StructuredFlow</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.5</td></tr>
<tr><td>ThreeLayers</td><td>0.333333</td><td>1.0</td><td>0.333333</td><td>1.0</td><td>1.0</td><td>0.333333</td></tr>
<tr><td>Wide Interface</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
</table>

Table D.7: SmallWiki 1.303 Profiles

<table>
<tr><td>Class name</td><td>(|#ErrorAction|)</td><td>(|#TemplateHeadTitle|)</td><td>(|#DocumentComposite|)</td><td>(|#Code|)</td><td>(|#SwazooServer|)</td><td>(|#Storage|)</td></tr>
<tr><td>Size Pattern</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td><td>’Micro’</td></tr>
<tr><td>No. Method Clumps</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td><td>’0’</td></tr>
<tr><td>AdderExtender</td><td>0.0</td><td>0.0</td><td>0.2</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AdderNormal</td><td>0.2</td><td>0.0</td><td>0.0</td><td>0.2</td><td>0.0</td><td>0.333333</td></tr>
<tr><td>AdderOverrider</td><td>0.4</td><td>0.4</td><td>0.0</td><td>0.2</td><td>0.166667</td><td>0.0</td></tr>
<tr><td>AllState1</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AllState2</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.333333</td></tr>
<tr><td>AllState3</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AllStateClean1</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean2</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ConstantDefiner</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>DataStorage</td><td>0.0</td><td>0.5</td><td>0.25</td><td>0.0</td><td>0.0</td><td>0.5</td></tr>
<tr><td>Delegater</td><td>0.0</td><td>0.0</td><td>0.5</td><td>0.0</td><td>0.5</td><td>0.0</td></tr>
<tr><td>Funnel</td><td>0.142857</td><td>0.0</td><td>0.0</td><td>0.166667</td><td>0.0</td><td>0.2</td></tr>
<tr><td>Interface</td><td>0.8</td><td>0.4</td><td>0.4</td><td>0.4</td><td>0.333333</td><td>0.333333</td></tr>
<tr><td>MicroSpecialExtender</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>MicroSpecialOverrider</td><td>0.666667</td><td>0.666667</td><td>0.0</td><td>0.333333</td><td>0.25</td><td>0.0</td></tr>
<tr><td>SharingEntries4</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries5</td><td>0.0</td><td>0.0</td><td>0.75</td><td>0.0</td><td>0.5</td><td>0.2</td></tr>
<tr><td>SingleEntry</td><td>1.0</td><td>0.0</td><td>0.0</td><td>0.666667</td><td>0.0</td><td>0.5</td></tr>
<tr><td>Singleton</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>StructuredFlow</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ThreeLayers</td><td>0.333333</td><td>1.0</td><td>1.0</td><td>0.333333</td><td>0.666667</td><td>1.0</td></tr>
<tr><td>Wide Interface</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
</table>

Table D.8: SmallWiki 1.303 Profiles

Class name	(|#WikiScanner|)	(|#Link|)	(|#TemplateBodySession|)	(|#VisitorSearch|)	(|#Permission|)	(|#VisitorCollector|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'1'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.142857	0.142857	0.142857	0.142857
AdderNormal	0.333333	0.166667	0.0	0.0	0.285714	0.0
AdderOverrider	0.0	0.0	0.142857	0.428571	0.142857	0.0
AllState1	0.0	0.0	0.0	0.0	0.0	0.333333
AllState2	0.0	0.0	0.0	0.0	0.333333	0.0
AllState3	0.0	0.0	0.0	0.666667	0.333333	0.0
AllStateClean1	0.0	0.0	0.666667	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.333333
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.5	0.25	0.0	0.0	0.0	0.0
DataStorage	0.0	0.5	1.0	0.5	0.5	1.0
Delegater	0.0	0.0	0.0	0.0	0.333333	0.0
Funnel	0.125	0.0	0.0	0.5	0.142857	0.0
Interface	0.666667	0.5	0.285714	0.571429	0.714286	0.142857
MicroSpecialExtender	0.0	0.0	0.2	0.2	0.2	0.2
MicroSpecialOverrider	0.0	0.0	0.2	0.6	0.2	0.0
SharingEntries4	0.0	0.166667	0.333333	0.0	0.285714	0.333333
SharingEntries5	0.0	0.0	0.0	0.166667	0.285714	0.0
SingleEntry	0.666667	0.0	0.0	0.5	0.5	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table D.9: SmallWiki 1.303 Profiles

Class name	(|#SwazooSite|)	(|#VisitorRenderer|)	(|#TemplateBody|)	(|#TemplateBodyActions|)	(|#TemplateBodyCustom|)	(|#FifoCache|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.111111	0.111111	0.111111
AdderNormal	0.125	0.0	0.111111	0.0	0.0	0.0
AdderOverrider	0.0	0.25	0.222222	0.111111	0.111111	0.111111
AllState1	0.0	0.0	0.0	0.0	0.0	0.0
AllState2	0.333333	0.333333	0.0	0.0	0.0	0.333333
AllState3	0.333333	0.0	0.0	0.0	0.0	0.333333
AllStateClean1	0.0	0.333333	0.666667	0.333333	0.333333	0.666667
AllStateClean2	0.0	0.333333	0.0	0.0	0.0	0.0
AllStateClean3	0.333333	0.0	0.0	0.0	0.666667	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.5	0.5	0.666667	0.333333	0.666667	0.666667
Delegater	0.0	0.5	0.0	0.0	0.0	0.0
Funnel	0.125	0.0	0.0	0.125	0.2	0.2
Interface	0.125	0.375	0.333333	0.222222	0.222222	0.222222
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.125	0.166667	0.6	0.375	0.4	0.2
SharingEntries5	0.25	0.166667	0.0	0.0	0.0	0.2
SingleEntry	0.285714	0.0	0.0	0.166667	1.0	0.333333
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.0	0.0	1.0	0.0	0.0
ThreeLayers	1.0	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table D.10: SmallWiki 1.303 Profiles

Class name	(|#LinkInternal|)	(|#TemplateBodyW3C|)	(|#TemplateBodySearch|)	(|#Cache|)	(|#Page|)	(|#PropertyManager|)	(|#Template|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.111111	0.111111	0.111111	0.090909	0.090909	0.0
AdderNormal	0.111111	0.0	0.0	0.222222	0.090909	0.363636	0.583333
AdderOverrider	0.333333	0.111111	0.111111	0.0	0.090909	0.090909	0.0
AllState1	0.0	0.0	0.0	0.333333	0.333333	0.333333	0.0
AllState2	0.0	0.0	0.0	0.333333	0.333333	0.333333	0.0
AllState3	0.0	0.0	0.0	0.333333	0.0	0.333333	0.0
AllStateClean1	0.0	1.0	1.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.333333	1.0	1.0	0.0	0.333333	0.333333	0.0
Delegater	0.0	0.0	0.0	0.5	0.0	0.75	0.0
Funnel	0.375	0.0	0.0	0.125	0.0	0.125	0.25
Interface	0.444444	0.222222	0.222222	0.555556	0.272727	0.818182	0.583333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.125	0.333333	0.333333	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.5	0.125	1.0	0.0
SingleEntry	0.666667	0.0	0.0	0.2	0.0	0.5	0.2
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	1.0	0.0	0.0	0.5	0.0	0.0	0.0
ThreeLayers	1.0	1.0	1.0	0.666667	1.0	1.0	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table D.11: SmallWiki 1.303 Profiles

Class name	(|#SIXXStorage|)	(|#ResourceEdit|)	(|#BasicRole|)	(|#PageEdit|)	(|#VisitorRendererWiki|)	(|#FolderEdit|)	(|#Login|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.333333	0.166667	0.166667	0.25	0.0	0.0
AdderNormal	0.0833333	0.0833333	0.166667	0.166667	0.0	0.0769231	0.0
AdderOverrider	0.0833333	0.0833333	0.0833333	0.0833333	0.75	0.0769231	0.230769
AllState1	0.333333	0.0	0.333333	0.0	0.333333	0.0	0.0
AllState2	0.333333	0.333333	0.333333	0.666667	0.333333	0.0	0.0
AllState3	0.0	0.333333	0.0	0.666667	0.333333	0.333333	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.666667	0.0	0.333333	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.5	0.5	0.125	0.25	0.0	0.125	0.375
Delegater	0.0	0.0	0.8	0.0	0.0	0.0	0.0
Funnel	0.25	0.25	0.0	0.25	0.875	0.25	0.5
Interface	0.166667	0.5	0.75	0.416667	1.0	0.153846	0.230769
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.125	0.25	0.25	0.125	0.0	0.0	0.375
SharingEntries5	0.125	0.125	0.625	0.5	0.25	0.25	0.125
SingleEntry	0.428571	1.0	0.0	0.5	0.666667	0.230769	0.3
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.0	0.0	0.5	0.0	1.0	1.0
ThreeLayers	1.0	1.0	1.0	1.0	0.666667	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table D.12: SmallWiki 1.303 Profiles

Class name	(|#EditAction|)	(|#User|)	(|#Search|)	(|#TemplateHeadMeta|)	(|#ExpiringCache|)	(|#Folder|)	(|#VisitorRendererHtml|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0769231	0.125	0.0588235	0.0	0.117647	0.0555556	0.0555556
AdderNormal	0.0769231	0.25	0.0588235	0.0	0.0	0.222222	0.0
AdderOverrider	0.153846	0.0625	0.0588235	0.117647	0.0588235	0.277778	0.833333
AllState1	0.0	0.333333	0.0	0.0	0.0	0.333333	0.0
AllState2	0.0	0.333333	0.333333	0.0	1.0	0.333333	0.333333
AllState3	0.666667	0.333333	0.0	0.0	1.0	0.333333	0.333333
AllStateClean1	0.0	0.0	1.0	1.0	1.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0833333	0.0
DataStorage	0.125	0.4	0.8	1.0	0.8	0.0833333	0.166667
Delegater	0.0	0.5	0.0	0.0	0.0	0.0	0.0
Funnel	0.25	0.25	0.125	0.25	0.25	0.375	0.125
Interface	0.307692	0.5625	0.176471	0.117647	0.176471	0.611111	0.888889
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.25	0.375	0.25	0.125	0.0	0.0
SharingEntries5	0.25	0.75	0.125	0.0	0.375	0.75	0.375
SingleEntry	0.454545	0.2	0.571429	0.5	0.333333	0.285714	0.333333
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.5	0.0	0.0	0.5	1.0	0.5
ThreeLayers	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table D.13: SmallWiki 1.303 Profiles

Class name	(|#Visitor|)	(|#SnapshotStorage|)	(|#Resource|)	(|#TemplateEdit|)	(|#Request|)	(|#Action|)	(|#Response|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.047619	0.0434783	0.0	0.0	0.0	0.0
AdderNormal	0.904762	0.047619	0.347826	0.333333	0.30303	0.181818	0.194444
AdderOverrider	0.0	0.047619	0.0434783	0.0740741	0.0	0.0	0.0
AllState1	0.0	1.0	1.0	0.0	0.0	0.0	1.0
AllState2	0.0	0.333333	0.333333	0.0	0.4	0.333333	0.333333
AllState3	0.0	0.333333	0.333333	0.333333	0.4	1.0	0.666667
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.333333	0.333333	0.0	0.333333
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0909091	0.0
DataStorage	0.0	0.285714	0.357143	0.222222	0.590909	0.272727	0.375
Delegater	0.0	0.0	0.0	0.0	0.0	0.2	0.0
Funnel	1.0	0.375	0.25	0.375	0.125	0.25	0.25
Interface	0.904762	0.142857	0.434783	0.407407	0.30303	0.272727	0.194444
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.25	0.125	0.5	0.25	0.5	0.25
SharingEntries5	0.0	0.5	0.625	0.25	0.5	0.625	0.25
SingleEntry	0.285714	0.125	0.285714	0.166667	0.454545	0.190476	0.0714286
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	1.0	1.0	0.5	1.0	1.0	1.0	0.0
ThreeLayers	0.333333	1.0	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table D.14: SmallWiki 1.303 Profiles

Class name	(|#WikiParser|)	(|#Server|)	(|#HtmlWriteStream|)	(|#Structure|)
Size Pattern	'Normal'	'Normal'	'Giant'	'Giant'
No. Method Clumps	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0133333	0.025641
AdderNormal	0.767442	0.155556	0.613333	0.205128
AdderOverrider	0.0	0.0	0.0	0.0128205
AllState1	0.333333	0.333333	0.666667	0.6
AllState2	0.333333	0.333333	0.0	0.6
AllState3	0.333333	0.333333	0.666667	0.6
AllStateClean1	0.0	0.333333	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.333333
AllStateClean3	0.0	0.0	0.0	0.666667
ConstantDefiner	0.103448	0.0	0.0	0.0384615
DataStorage	0.0714286	0.366667	0.04	0.269231
Delegater	0.0	0.0	0.0	0.0
Funnel	0.625	0.125	1.0	1.0
Interface	0.837209	0.177778	0.626667	0.269231
MicroSpecialExtender	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.125	0.0	0.75
SharingEntries5	1.0	0.625	0.5	0.875
SingleEntry	0.153846	0.0666667	0.075	0.0833333
Singleton	0.0	0.0	0.0	0.0
StructuredFlow	1.0	1.0	1.0	1.0
ThreeLayers	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0

Table D.15: SmallWiki 1.303 Profiles

Appendix E

Moose 2.84 Results

Class name	(|#SortExpressionVisitor|)	(|#VisualWorksNamespaceImporter|)	(|#MSEApplicationModel|)	(|#SmallInteger|)	(|#MOFTypeAlias|)
Size Pattern	'Single'	'Single'	'Single'	'Single'	'Single'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.5	0.0	0.0
AdderNormal	0.5	0.0	0.0	0.5	0.0
AdderOverrider	0.0	0.5	0.0	0.0	0.0
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0
Interface	0.5	0.5	0.5	0.5	0.0
MicroSpecialExtender	0.0	0.0	1.0	0.0	0.0
MicroSpecialOverrider	0.0	1.0	0.0	0.0	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.1: Moose version 2.84 Profiles

Class name	(|#DSAndOperator|)	(|#DSBottomValuesOperator|)	(|#DSHigherThanOperator|)	(|#DSLowerThanOperator|)	(|#DSTopValuesOperator|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	0.0
AdderOverrider	0.5	0.5	0.5	0.5	0.5
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.5	0.5	0.5	0.5	0.5
DataStorage	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0
Interface	1.0	1.0	1.0	1.0	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.2: Moose version 2.84 Profiles

Class name	(|#DSOrOperator|)	(|#FAMIXModelSmalltalkQueryFacade|)	(|#MGRoot|)	(|#MGMethodCollectionUI|)	(|#SelectExpressionVisitor|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	0.5
AdderOverrider	0.5	0.0	0.0	1.0	0.0
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.5	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.333333	0.333333
Interface	1.0	0.0	0.0	1.0	0.5
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	0.0	0.0	1.0	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.5	1.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.0	0.0	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.3: Moose version 2.84 Profiles

Class name	(|#MOFTypedElement|)	(|#FAMIXSourceFile|)	(|#ItemLabel|)	(|#MOFNamespace|)	(|#FAMIXAbstractAssociation|)	(|#ItemChildren|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.5	0.0	0.0
AdderOverrider	0.0	0.5	1.0	0.0	0.0	1.0
AllState1	0.0	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.333333	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.5	0.5	0.0	1.0	0.0
DataStorage	0.5	0.0	0.0	0.25	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0	0.0
Interface	0.0	1.0	1.0	0.5	1.0	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	1.0	1.0	0.0	0.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.5	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.666667	0.333333	0.333333	1.0	0.333333	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table E.4: Moose version 2.84 Profiles

Class name	(|#MGAndCombinedFlawDetector|)	(|#DSRelativeOperator|)	(|#MGSingleClassMetricFlawDetector|)	(|#MGSingleMethodMetricFlawDetector|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0
AdderOverrider	0.5	1.0	1.0	1.0
AllState1	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0
ConstantDefiner	0.5	0.0	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0
Funnel	0.0	0.333333	0.0	0.0
Interface	1.0	1.0	1.0	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.5	0.0	0.0
Singleton	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0

Table E.5: Moose version 2.84 Profiles

Class name	(|#MSEModelMVAttributeDescriptor|)	(|#DSAbsoluteOperator|)	(|#MGOrCombinedFlawDetector|)	(|#MGNOPOverrideDetector|)	(|#ItemEditor|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.5	0.0	0.0	0.0	0.0
AdderOverrider	0.5	1.0	0.5	1.0	0.333333
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.5	0.0	0.5	0.0	0.0
DataStorage	0.0	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.333333	0.0	0.0	0.0
Interface	1.0	1.0	1.0	1.0	0.333333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	1.0	1.0	1.0	1.0	0.5
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0	0.0
SingleEntry	0.0	0.5	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.333333	0.333333	0.333333	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.6: Moose version 2.84 Profiles

Class name	(|#MOFConstant|)	(|#CompiledMethod|)	(|#MOFAttribute|)	(|#MSESchema|)	(|#ItemService|)	(|#MGClassCollectionUI|)	(|#XMIReader|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.666667	0.0	0.666667	1.0	0.0	1.0
AdderOverrider	0.0	0.0	0.0	0.0	0.0	0.666667	0.0
AllState1	0.0	0.0	0.0	0.333333	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.333333	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DataStorage	0.5	0.0	0.5	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.0	0.0	0.0	0.25	0.0
Interface	0.0	0.666667	0.0	0.666667	1.0	0.666667	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	1.0	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	1.0	0.0	0.0	0.0
SingleEntry	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.666667	0.666667	0.666667	0.666667	0.333333	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table E.7: Moose version 2.84 Profiles

Class name	(|#SourceImporter|)	(|#VisualWorksImporterAbstractPanel|)	(|#MGMisplacedMethodDetector|)	(|#CandidateInvocationsWithBaseOperator|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.333333	0.0	0.0
AdderOverrider	0.333333	0.0	0.666667	0.666667
AllState1	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.0	0.0
AllState3	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0
DataStorage	0.0	0.5	0.0	0.0
Delegater	0.0	0.0	0.0	0.0
Funnel	0.25	0.0	0.2	0.0
Interface	0.333333	0.666667	0.666667	0.666667
MicroSpecialExtender	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.5	0.0	1.0	1.0
SharingEntries4	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.0	0.0
SingleEntry	0.5	0.0	1.0	0.0
Singleton	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.0	0.0	0.0
ThreeLayers	0.333333	1.0	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0

Table E.8: Moose version 2.84 Profiles

Class name	(|#DSAdapterOperator|)	(|#FAMIXExpressionArgument|)	(|#MSECDIFFilteringSaver|)	(|#SmalltalkAnnotatorOperator|)	(|#SingleValueToXMLConverter|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	1.0
AdderOverrider	0.333333	0.0	0.666667	0.75	0.25
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.333333	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.5	0.0	0.0	0.0
DataStorage	0.0	0.5	0.5	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.5	0.0	0.0	0.0	0.166667
Interface	0.333333	0.333333	0.666667	0.75	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.5	0.0	1.0	1.0	0.333333
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	1.0	0.0	0.0
SingleEntry	0.5	0.0	0.0	0.0	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	1.0	1.0	0.333333	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.9: Moose version 2.84 Profiles

Class name	(|#FAMIXLocalVariable|)	(|#MSEPropertyOperator|)	(|#DSNamedExpression|)	(|#DSOperand|)	(|#MSEMinimalCDIFReadStream|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.25	0.0	0.0	0.0	0.0
AdderNormal	0.0	0.0	0.0	0.0	0.0
AdderOverrider	0.5	0.5	0.25	0.5	0.0
AllState1	0.0	0.0	0.0	0.333333	0.333333
AllState2	0.0	0.0	0.0	0.333333	0.333333
AllState3	0.0	0.333333	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.333333	0.333333	0.0	0.0	0.0
DataStorage	0.0	0.0	0.5	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	1.0
Funnel	0.0	0.4	0.0	0.0	0.0
Interface	1.0	0.75	0.25	0.5	0.75
MicroSpecialExtender	0.333333	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.666667	0.666667	0.333333	0.666667	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.0	0.2	0.0	1.0	0.75
SingleEntry	0.0	0.5	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	0.666667	1.0	0.666667	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.10: Moose version 2.84 Profiles

Class name	(|#DSCompositionOperator|)	(|#MOFReference|)	(|#DSFilteringOperator|)	(|#FAMIXAccessArgument|)	(|#MOFImport|)	(|#MOFParameter|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.2	0.0	0.0
AdderNormal	0.0	0.4	0.0	0.0	0.4	0.0
AdderOverrider	0.5	0.0	0.2	0.0	0.0	0.0
AllState1	0.0	0.0	0.666667	0.0	0.0	0.0
AllState2	0.0	0.333333	0.333333	0.0	0.333333	0.0
AllState3	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.333333	0.0	0.0
DataStorage	0.0	0.5	0.5	0.5	0.5	1.0
Delegater	0.0	0.0	0.0	0.0	0.0	0.0
Funnel	0.333333	0.0	0.0	0.0	0.0	0.0
Interface	0.5	0.4	0.4	0.4	0.4	0.0
MicroSpecialExtender	0.0	0.0	0.0	0.333333	0.0	0.0
MicroSpecialOverrider	0.666667	0.0	0.333333	0.0	0.0	0.0
SharingEntries4	0.0	0.0	0.0	0.333333	0.0	0.0
SharingEntries5	0.0	0.5	0.5	0.0	0.5	0.0
SingleEntry	0.5	0.0	0.0	0.0	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.0	0.0
ThreeLayers	0.333333	1.0	1.0	1.0	1.0	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table E.11: Moose version 2.84 Profiles

Class name	(|#VisualWorksImporterParcelPanel|)	(|#MOFDataType|)	(|#MOFTag|)	(|#MSECDIFFilteringImporter|)	(|#OperatorManagerUI|)
Size Pattern	’Micro’	’Micro’	’Micro’	’Micro’	’Micro’
No. Method Clumps	’0’	’0’	’0’	’0’	’0’
AdderExtender	0.0	0.0	0.0	0.0	0.166667
AdderNormal	0.2	0.4	0.0	0.4	0.166667
AdderOverrider	0.2	0.0	0.0	0.2	0.0
AllState1	0.333333	0.0	0.0	0.0	0.333333
AllState2	0.0	0.333333	0.0	0.0	0.0
AllState3	0.333333	0.0	0.0	0.0	0.333333
AllStateClean1	0.0	0.0	0.0	0.333333	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.5	1.0	0.5	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.4	0.0	0.0	0.0	0.4
Interface	0.4	0.4	0.0	0.6	0.333333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.25
MicroSpecialOverrider	0.333333	0.0	0.0	0.333333	0.0
SharingEntries4	0.0	1.0	0.0	1.0	0.0
SharingEntries5	0.2	1.0	0.0	0.0	0.2
SingleEntry	0.5	0.0	0.0	0.0	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	0.5
ThreeLayers	0.666667	1.0	0.666667	1.0	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.12: Moose version 2.84 Profiles

Class name	(|#MGCorrelationChart|)	(|#MGBypassedAccessorDetector|)	(|#MGCombinedFlawDetector|)	(|#FAMIXInclude|)	(|#MGStackedBarChartUI|)
Size Pattern	’Micro’	’Micro’	’Micro’	’Micro’	’Micro’
No. Method Clumps	’0’	’0’	’0’	’0’	’0’
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.333333	0.0	0.333333	0.0	0.333333
AdderOverrider	0.0	0.333333	0.166667	0.0	0.0
AllState1	0.0	0.0	0.333333	0.333333	0.0
AllState2	0.0	0.0	0.0	0.0	0.333333
AllState3	0.333333	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.25	0.25	0.0
DataStorage	0.0	0.0	0.25	0.5	0.25
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.142857	0.125	0.2	0.0	0.0
Interface	0.333333	0.333333	0.666667	0.166667	0.333333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.5	0.25	0.0	0.0
SharingEntries4	0.0	0.0	0.6	0.0	0.0
SharingEntries5	0.142857	0.0	0.0	0.0	0.166667
SingleEntry	0.5	0.75	0.5	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.5	0.0	0.0	0.0
ThreeLayers	0.666667	0.333333	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.13: Moose version 2.84 Profiles

Class name	(|#VisualWorksImporterNamespacePanel|)	(|#MSESingleValueToCDIFConverter|)	(|#KeyValuePair|)	(|#VisualLauncher|)	(|#MSEXMIDTDProducer|)
Size Pattern	’Micro’	’Micro’	’Micro’	’Micro’	’Micro’
No. Method Clumps	’0’	’0’	’0’	’0’	’0’
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.166667	1.0	0.0	0.333333	0.166667
AdderOverrider	0.166667	0.166667	0.0	0.0	0.0
AllState1	0.333333	0.0	0.666667	0.0	0.0
AllState2	0.0	0.0	0.333333	0.0	0.0
AllState3	0.333333	0.0	0.0	0.0	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.5	0.0	0.0
Delegater	0.0	0.0	1.0	0.0	0.0
Funnel	0.333333	0.125	0.0	0.0	0.285714
Interface	0.333333	1.0	0.5	0.333333	0.166667
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.25	0.25	0.0	0.0	0.0
SharingEntries4	0.0	0.0	0.666667	0.0	0.0
SharingEntries5	0.166667	0.0	1.0	0.0	0.142857
SingleEntry	0.5	0.5	0.0	0.0	0.25
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.0	1.0
ThreeLayers	0.666667	0.333333	1.0	0.666667	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.14: Moose version 2.84 Profiles

Class name	(|#FAMIXAbstractLocalEntity|)	(|#FAMIXUnknownVariable|)	(|#COBOLMetricOperator|)	(|#ModelViewerUISubcanvas|)	(|#MOFConstraint|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.166667	0.333333	0.0	0.142857	0.0
AdderNormal	0.166667	0.0	0.0	0.714286	0.428571
AdderOverrider	0.0	0.0	0.333333	0.0	0.0
AllState1	0.333333	0.0	0.0	0.0	0.0
AllState2	0.333333	0.0	0.0	0.666667	0.333333
AllState3	0.0	0.0	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.333333	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.5	0.5	0.0	0.0	0.5
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.2	0.142857	0.0	0.0
Interface	0.333333	0.333333	0.333333	0.857143	0.428571
MicroSpecialExtender	0.25	0.5	0.0	0.2	0.0
MicroSpecialOverrider	0.0	0.0	0.5	0.0	0.0
SharingEntries4	0.0	0.4	0.0	0.0	0.0
SharingEntries5	0.666667	0.0	0.0	0.625	0.333333
SingleEntry	0.0	0.5	0.333333	0.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.5	0.0	0.0
ThreeLayers	1.0	1.0	0.333333	0.666667	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.15: Moose version 2.84 Profiles

Class name	(|#ModelViewerUI|)	(|#MGChartDataGenerator|)	(|#PreferencesUI|)	(|#MeasurementOperator|)	(|#VisualWorksImporterCategoryPanel|)
Size Pattern	'Micro'	'Micro'	'Micro'	'Micro'	'Micro'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.142857	0.0	0.0	0.0	0.0
AdderNormal	0.142857	0.571429	0.142857	0.0	0.142857
AdderOverrider	0.0	0.0	0.0	0.285714	0.142857
AllState1	0.333333	0.0	0.0	0.0	0.333333
AllState2	0.0	0.0	0.0	0.0	0.0
AllState3	0.333333	0.0	0.333333	0.333333	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.2	0.0
DataStorage	0.0	0.5	0.0	0.25	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.285714	0.285714	0.2	0.25	0.25
Interface	0.285714	0.571429	0.142857	0.428571	0.285714
MicroSpecialExtender	0.2	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.4	0.2
SharingEntries4	0.0	0.571429	0.0	0.0	0.0
SharingEntries5	0.142857	0.0	0.2	0.125	0.125
SingleEntry	0.5	0.5	1.0	0.5	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.5	0.0	0.5	0.5
ThreeLayers	0.666667	1.0	0.666667	1.0	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.16: Moose version 2.84 Profiles

Class name	(|#MSEConstants|)	(|#VisualWorksImporterPackagePanel|)	(|#MGFlawDetector|)	(|#MSEComputedGroup|)	(|#MGTableUI|)
Size Pattern	'Micro'	'Micro'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.142857	0.285714	0.25	0.25	0.125
AdderOverrider	0.0	0.142857	0.125	0.25	0.0
AllState1	0.0	0.333333	0.0	0.0	1.0
AllState2	0.0	0.0	0.0	0.333333	0.333333
AllState3	0.0	0.333333	0.0	0.333333	0.0
AllStateClean1	0.0	0.0	0.333333	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.333333	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	1.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	1.0	0.75	0.5
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.125	0.375	0.0	0.2	0.0
Interface	0.857143	0.428571	0.375	0.5	0.125
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.2	0.0	0.0	0.0
SharingEntries4	0.0	0.0	0.5	0.2	0.0
SharingEntries5	0.0	0.125	0.0	0.2	0.2
SingleEntry	0.5	0.333333	0.0	0.5	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.5	0.0	0.0	0.0
ThreeLayers	0.666667	0.666667	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.17: Moose version 2.84 Profiles

Class name	(|#MSEEnumeratedGroup|)	(|#CppMetricOperator|)	(|#FAMIXGlobalVariable|)	(|#FAMIXReporter|)	(|#MGSingleMetricFlawDetector|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.222222	0.0	0.0
AdderNormal	0.0	0.0	0.111111	0.111111	0.111111
AdderOverrider	0.25	0.222222	0.222222	0.0	0.0
AllState1	0.0	0.0	0.0	0.333333	0.0
AllState2	0.0	0.0	0.0	0.333333	0.0
AllState3	0.0	0.333333	0.0	0.333333	0.0
AllStateClean1	0.333333	0.0	0.0	0.0	0.0
AllStateClean2	0.333333	0.0	0.333333	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.166667	0.166667	0.0
DataStorage	0.5	0.0	0.333333	0.0	1.0
Delegater	0.5	0.0	0.0	0.0	0.0
Funnel	0.2	0.25	0.125	0.125	0.0
Interface	0.375	0.222222	0.666667	0.222222	0.111111
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.8	0.0	0.625	0.0	0.333333
SharingEntries5	0.0	0.125	0.0	0.25	0.0
SingleEntry	0.5	0.5	0.5	1.0	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.5	0.0	0.0	0.0
ThreeLayers	1.0	0.666667	1.0	0.666667	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.18: Moose version 2.84 Profiles

Class name	(|#MGTableDataGenerator|)	(|#FAMIXFormalParameter|)	(|#FAMIXAbstractArgument|)	(|#MGItemMetricsUI|)	(|#MGClassSelectorUI|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.222222	0.111111	0.0	0.0
AdderNormal	0.555556	0.111111	0.0	0.222222	0.222222
AdderOverrider	0.0	0.222222	0.0	0.0	0.0
AllState1	0.0	0.0	0.666667	0.666667	0.0
AllState2	0.0	0.0	0.666667	0.333333	0.0
AllState3	0.0	0.0	0.0	0.0	0.333333
AllStateClean1	0.0	0.333333	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.166667	0.333333	0.0	0.0
DataStorage	0.333333	0.333333	0.666667	0.333333	0.333333
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.5	0.0	0.0	0.0	0.25
Interface	0.555556	0.666667	0.333333	0.222222	0.222222
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.875	0.142857	0.0	0.166667	0.25
SharingEntries5	0.0	0.0	0.25	0.166667	0.125
SingleEntry	0.5	0.0	0.0	0.0	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.0	0.0	0.0	0.0
ThreeLayers	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.19: Moose version 2.84 Profiles

Class name	(|#DSExpression|)	(|#MGCorrelationMetricsChooserUI|)	(|#Preferences|)	(|#MOFAssociationEnd|)	(|#MOFStructuralFeature|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.111111	0.111111	0.222222	0.0	0.0
AdderOverrider	0.0	0.0	0.0	0.0	0.0
AllState1	0.666667	0.0	0.666667	0.0	0.666667
AllState2	0.333333	0.0	0.333333	0.0	0.0
AllState3	0.333333	0.333333	0.0	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.5	0.0	0.666667	1.0	1.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.25	0.125	0.0	0.0	0.0
Interface	0.222222	0.111111	0.222222	0.0	0.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.25	0.0	0.0	0.0	0.0
SharingEntries5	0.25	0.125	0.25	0.0	0.0
SingleEntry	0.0	0.5	0.0	0.0	0.0
Singleton	0.0	0.0	1.0	0.0	0.0
StructuredFlow	0.5	0.5	0.0	0.0	0.0
ThreeLayers	1.0	0.666667	1.0	0.666667	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.20: Moose version 2.84 Profiles

Class name	(|#MOFAssociation|)	(|#FAMIXImplicitVariable|)	(|#MGSingleMetricDetectorUI|)	(|#MGModel|)	(|#EntityCommentEditor|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.222222	0.0	0.0	0.111111
AdderNormal	0.222222	0.0	0.333333	0.777778	0.333333
AdderOverrider	0.0	0.222222	0.0	0.0	0.0
AllState1	0.666667	0.0	0.0	0.0	0.666667
AllState2	0.333333	0.0	0.333333	0.0	0.666667
AllState3	0.0	0.0	0.333333	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.333333	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.166667	0.0	0.0	0.0
DataStorage	0.833333	0.5	0.0	0.0	0.333333
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.375	0.125	0.0	0.25
Interface	0.222222	0.555556	0.333333	0.777778	0.444444
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.25	0.0	0.0	0.0
SharingEntries5	0.5	0.0	0.125	0.0	0.375
SingleEntry	0.0	0.5	0.5	0.0	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.5	0.0	0.0
ThreeLayers	1.0	1.0	0.666667	0.666667	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.21: Moose version 2.84 Profiles

Class name	(|#MGQueryFacade|)	(|#MSEModelSaverUI|)	(|#SmalltalkClassMetricsOperator|)	(|#VisualWorksImporterFacade|)	(|#ProgressUI|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.666667	0.2	0.0	0.5	0.3
AdderOverrider	0.0	0.0	0.2	0.1	0.0
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.333333	0.0	0.0	0.0
AllState3	0.0	0.333333	0.0	0.0	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.666667
AllStateClean2	0.0	0.0	0.0	0.0	0.333333
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.333333	0.0	0.0	0.0	0.666667
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.25	0.142857	0.125	0.25	0.166667
Interface	0.666667	0.2	0.2	0.6	0.3
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.375	0.0	0.0	0.0	0.333333
SharingEntries5	0.0	0.285714	0.0	0.0	0.166667
SingleEntry	0.5	0.25	0.75	0.25	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	1.0	0.5
ThreeLayers	1.0	0.666667	0.333333	0.333333	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.22: Moose version 2.84 Profiles

Class name	(|#FAMIXFunction|)	(|#VisualWorksParseTreeAnnotator|)	(|#MGValueSliderUI|)	(|#MOFGeneralizableElement|)	(|#OperatorManager|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.181818	0.181818	0.0	0.0	0.0
AdderNormal	0.0909091	0.0	0.363636	0.454545	0.416667
AdderOverrider	0.181818	0.0909091	0.0	0.0	0.0
AllState1	0.333333	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.333333	0.333333	0.0
AllState3	0.0	0.0	0.333333	0.0	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.333333
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.142857	0.0	0.0	0.0	0.0
DataStorage	0.0	0.0	0.333333	0.666667	0.25
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.125	0.125	0.285714	0.0	0.25
Interface	0.545455	0.272727	0.363636	0.454545	0.416667
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.625	0.0	0.0	0.0	0.5
SharingEntries5	0.0	0.0	0.142857	0.2	0.0
SingleEntry	0.5	1.0	0.5	0.0	0.5
Singleton	0.0	0.0	0.0	0.0	1.0
StructuredFlow	0.0	1.0	0.5	0.0	0.5
ThreeLayers	1.0	0.333333	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.23: Moose version 2.84 Profiles

Class name	(|#DSMetricOperator|)	(|#MSEAbstractTool|)	(|#MOFPackage|)	(|#FAMIXAbstractNamedEntity|)	(|#AbstractNamedEntity|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0833333	0.0	0.0	0.0833333	0.0
AdderNormal	0.0	0.25	0.416667	0.25	0.25
AdderOverrider	0.166667	0.0	0.0	0.0	0.0
AllState1	0.0	0.0	1.0	0.333333	0.0
AllState2	0.0	0.333333	0.333333	0.0	0.0
AllState3	0.0	0.0	0.0	0.0	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.333333	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.25	0.25
DataStorage	0.0	0.25	0.625	0.375	0.375
Delegater	0.0	0.5	0.0	0.0	0.0
Funnel	0.125	0.125	0.0	0.125	0.125
Interface	0.25	0.333333	0.416667	0.5	0.416667
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.5	0.0	0.25	0.125
SharingEntries5	0.0	0.125	0.2	0.0	0.125
SingleEntry	1.0	0.5	0.0	0.5	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	1.0	1.0	0.0	0.0	0.0
ThreeLayers	0.333333	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.24: Moose version 2.84 Profiles

Class name	(|#MGPreferencesUI|)	(|#DetectionStrategy|)	(|#MSEModelClassDescriptor|)	(|#MSEAbstractOperator|)	(|#MGOverviewUI|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'1'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0833333	0.0	0.0833333
AdderNormal	0.5	0.0833333	0.166667	0.0833333	0.25
AdderOverrider	0.0	0.0	0.0	0.0833333	0.0
AllState1	1.0	1.0	0.0	0.0	0.0
AllState2	1.0	1.0	0.0	0.0	0.333333
AllState3	0.333333	0.333333	0.0	0.0	0.333333
AllStateClean1	0.0	0.0	0.666667	0.0	0.0
AllStateClean2	0.0	0.0	0.333333	0.333333	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.0	0.625	1.0	0.25	0.125
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.5	0.0	0.125	0.375
Interface	0.5	0.0833333	0.25	0.166667	0.333333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.0	1.0	0.125	0.25
SharingEntries5	0.375	0.5	0.0	0.0	0.125
SingleEntry	0.0	0.5	0.0	0.75	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	1.0	0.5
ThreeLayers	0.666667	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.25: Moose version 2.84 Profiles

Class name	(|#FAMIXAbstractImporter|)	(|#MSEProperty|)	(|#CandidateInvocationsOperator|)	(|#MOFClass|)	(|#MGFlawDetectorCombinatorUI|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0769231	0.0714286	0.0	0.0
AdderNormal	0.615385	0.0	0.0	0.357143	0.285714
AdderOverrider	0.0	0.0	0.214286	0.0	0.0
AllState1	0.333333	0.333333	0.0	1.0	0.0
AllState2	0.333333	0.0	0.333333	0.333333	0.0
AllState3	0.333333	0.0	0.333333	0.0	0.333333
AllStateClean1	0.0	0.333333	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.444444	0.0	0.0	0.0
DataStorage	0.25	0.625	0.0	0.625	0.25
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.125	0.0	0.375	0.0	0.75
Interface	0.615385	0.384615	0.285714	0.357143	0.285714
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.166667	0.0	0.0	0.25
SharingEntries5	0.75	0.0	0.375	0.2	0.125
SingleEntry	0.5	0.0	0.5	0.0	1.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.0	1.0	0.0	1.0
ThreeLayers	1.0	1.0	0.666667	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.26: Moose version 2.84 Profiles

Class name	(|#MOFModelElement|)	(|#MSEToolManager|)	(|#FAMIXModelQueryFacade|)	(|#MSEAbstractRoot|)	(|#MGMetricsUI|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.133333	0.0
AdderNormal	0.428571	0.357143	1.0	0.466667	0.333333
AdderOverrider	0.0	0.0	0.0	0.0	0.0
AllState1	0.0	0.333333	0.0	0.0	1.0
AllState2	0.333333	0.333333	0.0	0.0	0.0
AllState3	0.0	0.333333	0.0	0.0	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.75	0.0	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.5	1.0	0.25	0.25
Interface	0.428571	0.357143	1.0	0.6	0.333333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.0	0.0	0.0	0.0
SharingEntries5	0.285714	0.875	0.0	0.0	0.125
SingleEntry	0.0	1.0	0.333333	0.5	0.5
Singleton	0.0	1.0	1.0	0.0	0.0
StructuredFlow	0.0	1.0	0.0	0.0	0.0
ThreeLayers	1.0	0.666667	0.333333	0.333333	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.27: Moose version 2.84 Profiles

Class name	(|#MSEMeasurement|)	(|#FAMIXInheritanceDefinition|)	(|#VisualWorksImporterUI|)	(|#FAMIXNamespace|)	(|#VisualWorksPackageImporter|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0666667	0.133333	0.0666667	0.0	0.266667
AdderNormal	0.0	0.0	0.333333	0.8125	0.0
AdderOverrider	0.0	0.0	0.0	0.125	0.0666667
AllState1	0.333333	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.333333	0.0	0.0
AllState3	0.0	0.0	0.333333	0.0	0.333333
AllStateClean1	0.0	1.0	0.0	0.333333	0.0
AllStateClean2	0.0	0.666667	0.0	0.0	0.333333
AllStateClean3	0.0	0.0	0.0	0.0	0.333333
ConstantDefiner	0.4	0.1	0.0	0.0909091	0.0
DataStorage	0.4	0.9	0.0	0.2	0.5
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	0.0	0.125	0.0	0.25
Interface	0.333333	0.2	0.4	1.0	0.333333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.125	0.2	0.0	1.0	0.375
SharingEntries5	0.0	0.0	0.375	0.0	0.125
SingleEntry	0.0	0.0	0.5	0.0	1.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	1.0	0.0	1.0
ThreeLayers	1.0	1.0	0.666667	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.28: Moose version 2.84 Profiles

Class name	(|#VisualWorksParseTree|)	(|#MGBarChartUI|)	(|#FAMIXAccess|)	(|#FileIOFacade|)	(|#Group|)	(|#XMIReaderHandler|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.117647	0.0	0.0	0.0
AdderNormal	0.375	0.588235	0.117647	0.352941	0.333333	0.444444
AdderOverrider	0.0	0.0	0.0	0.0	0.0555556	0.0
AllState1	0.0	0.0	0.0	0.0	0.666667	0.666667
AllState2	0.0	0.0	0.0	0.0	0.333333	1.0
AllState3	0.333333	0.333333	0.333333	0.0	0.333333	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.666667	0.0	0.666667	0.0	0.333333	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0909091	0.0	0.0	0.0
DataStorage	0.5	0.1	0.9	0.0	0.416667	0.0833333
Delegater	0.333333	0.142857	0.0	0.0	0.0	0.0
Funnel	0.25	0.5	0.125	0.25	0.5	0.25
Interface	0.4375	0.647059	0.294118	0.352941	0.388889	0.444444
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.375	0.0	0.25	0.0	0.25	0.0
SharingEntries5	0.25	0.125	0.125	0.0	0.875	0.875
SingleEntry	0.5	0.25	0.666667	0.25	0.75	1.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	1.0	0.5	0.0	1.0	0.5	0.5
ThreeLayers	1.0	1.0	1.0	0.333333	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table E.29: Moose version 2.84 Profiles

Class name	(|#MSEAbstractImporter|)	(|#MSEAbstractSchemaSaver|)	(|#MSEMetric|)	(|#ModelManager|)	(|#MGItemPropertyTableModel|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.0	0.0	0.0
AdderNormal	0.222222	0.611111	0.611111	0.333333	0.444444
AdderOverrider	0.0	0.0	0.0	0.0	0.0
AllState1	0.333333	1.0	0.666667	0.666667	0.0
AllState2	0.333333	1.0	0.666667	0.666667	0.0
AllState3	0.333333	0.0	0.333333	0.333333	0.0
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.333333	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	0.0
DataStorage	0.333333	0.0833333	0.333333	0.0833333	0.333333
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.25	0.125	0.125	0.5	0.5
Interface	0.277778	0.611111	0.611111	0.333333	0.444444
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.25	0.0	0.625	0.0	1.0
SharingEntries5	0.25	0.125	0.375	0.625	0.0
SingleEntry	0.75	1.0	0.666667	1.0	0.75
Singleton	0.0	0.0	0.0	1.0	0.0
StructuredFlow	1.0	0.0	0.0	1.0	0.5
ThreeLayers	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.30: Moose version 2.84 Profiles

Class name	(|#MSEImporter|)	(|#FAMIXAbstractObject|)	(|#XMLSaver|)	(|#SmalltalkMetricOperator|)	(|#MSESingleValueConverter|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0526316	0.105263	0.0	0.0
AdderNormal	0.222222	0.736842	0.263158	0.15	0.0952381
AdderOverrider	0.0	0.0	0.421053	0.15	0.0
AllState1	0.0	0.0	0.0	0.0	0.0
AllState2	0.0	0.0	0.333333	0.0	0.0
AllState3	0.0	0.0	0.333333	0.333333	0.333333
AllStateClean1	0.0	0.0	0.0	0.0	0.0
AllStateClean2	0.0	0.333333	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0769231	0.0	0.0	0.0
DataStorage	0.0	0.333333	0.0	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.125	0.125	0.25	0.125	0.125
Interface	0.222222	0.842105	0.789474	0.3	0.142857
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	1.0	0.0	0.0	0.0
SharingEntries5	0.0	0.0	0.5	0.75	0.75
SingleEntry	0.5	0.5	0.5	0.75	1.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.5	0.0	0.5	0.5	0.0
ThreeLayers	0.666667	1.0	0.666667	0.666667	0.666667
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.31: Moose version 2.84 Profiles

Class name	(|#MGItemCollectionUI|)	(|#MSECDIFSaver|)	(|#MSECDIFScanner|)	(|#FAMIXMethod|)	(|#MSEAbstractGroup|)	(|#MGPreferences|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'1'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0952381	0.0	0.0909091	0.0434783	0.0
AdderNormal	0.142857	0.190476	0.636364	0.181818	0.130435	0.0
AdderOverrider	0.0	0.428571	0.0	0.0454545	0.0	0.0
AllState1	1.0	1.0	0.0	0.0	0.0	0.0
AllState2	0.333333	0.666667	0.0	0.0	0.0	0.0
AllState3	0.333333	0.333333	0.0	0.333333	0.0	0.0
AllStateClean1	0.0	0.0	0.0	1.0	0.333333	1.0
AllStateClean2	0.0	0.0	0.0	0.0	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0666667	0.533333	0.0
DataStorage	0.142857	0.0714286	0.0	0.428571	0.142857	0.928571
Delegater	0.0	0.0	0.0	0.0	0.833333	0.0
Funnel	0.375	0.25	0.625	0.625	1.0	0.0
Interface	0.142857	0.714286	0.636364	0.363636	1.0	0.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0	0.0
SharingEntries4	1.0	0.0	0.0	0.5	0.0	0.0
SharingEntries5	0.25	0.5	0.0	0.125	0.0	0.0
SingleEntry	0.5	0.5	0.75	0.5	0.5	0.0
Singleton	0.0	0.0	0.0	0.0	0.0	0.0
StructuredFlow	1.0	0.5	0.5	0.5	0.0	0.0
ThreeLayers	1.0	1.0	0.333333	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0	0.0

Table E.32: Moose version 2.84 Profiles

Class name	(|#MSEUUIDGenerator|)	(|#VisualWorksImporter|)	(|#FAMIXAbstractScopable|)	(|#PluggableTableAdaptor|)	(|#FAMIXModelRoot|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.0	0.0	0.08	0.0	0.0
AdderNormal	0.347826	0.12	0.52	0.4	0.185185
AdderOverrider	0.0	0.08	0.0	0.0	0.0
AllState1	0.0	0.333333	0.333333	0.666667	0.0
AllState2	0.333333	0.333333	0.0	1.0	0.0
AllState3	0.0	0.0	0.0	0.666667	0.0
AllStateClean1	0.0	0.0	0.333333	0.0	0.0
AllStateClean2	0.0	0.0	0.333333	0.0	0.0
AllStateClean3	0.0	0.0	0.0	0.0	0.0
ConstantDefiner	0.0	0.0	0.0	0.0	1.0
DataStorage	0.0	0.0625	0.4375	0.0	0.0
Delegater	0.0	0.0	0.0	0.0	0.0
Funnel	0.0	1.0	0.125	1.0	0.0
Interface	0.347826	0.2	0.6	0.4	1.0
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	0.0	0.0	1.0	0.0	0.0
SharingEntries5	0.125	0.25	0.0	0.5	0.0
SingleEntry	0.0	0.25	0.25	0.75	0.0
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	1.0	0.0	1.0	0.0
ThreeLayers	0.666667	1.0	1.0	0.666667	0.333333
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.33: Moose version 2.84 Profiles

Class name	(|#FAMIXPackage|)	(|#VisualWorksParseTreeMetricCalculator|)	(|#ImporterFacade|)	(|#FAMIXAttribute|)	(|#MSEModelInformation|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'0'	'0'	'0'
AdderExtender	0.037037	0.296296	0.0	0.0714286	0.0
AdderNormal	0.740741	0.037037	0.37037	0.392857	0.233333
AdderOverrider	0.037037	0.0740741	0.0	0.0714286	0.0
AllState1	0.0	0.25	0.666667	0.0	0.0
AllState2	0.0	0.5	0.333333	0.0	0.0
AllState3	0.0	0.25	0.0	0.333333	0.0
AllStateClean1	0.666667	1.0	0.0	0.333333	1.0
AllStateClean2	0.333333	0.333333	0.0	0.0	0.2
AllStateClean3	0.0	0.333333	0.333333	0.333333	0.0
ConstantDefiner	0.0555556	0.0	0.0	0.0526316	0.0
DataStorage	0.333333	0.777778	0.222222	0.222222	1.0
Delegater	0.0	0.0	1.0	0.0	0.0
Funnel	0.125	0.125	0.125	0.625	0.125
Interface	0.851852	0.407407	0.814815	0.571429	0.233333
MicroSpecialExtender	0.0	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0	0.0
SharingEntries4	1.0	0.125	0.125	0.75	0.625
SharingEntries5	0.0	0.5	1.0	0.125	0.0
SingleEntry	0.333333	1.0	0.333333	0.5	0.5
Singleton	0.0	0.0	0.0	0.0	0.0
StructuredFlow	0.0	0.0	0.0	0.5	0.0
ThreeLayers	1.0	1.0	1.0	1.0	1.0
Wide Interface	0.0	0.0	0.0	0.0	0.0

Table E.34: Moose version 2.84 Profiles

Class name	(|#MSEModelAttributeDescriptor|)	(|#MSECDIFImporter|)	(|#LanguageIndependentMetricsOperator|)	(|#GroupUI|)
Size Pattern	'Normal'	'Normal'	'Normal'	'Normal'
No. Method Clumps	'0'	'0'	'1'	'0'
AdderExtender	0.03125	0.0	0.0	0.0285714
AdderNormal	0.09375	0.424242	0.0	0.4
AdderOverrider	0.0	0.0	0.0588235	0.0
AllState1	0.25	0.666667	0.0	1.0
AllState2	0.0	0.666667	0.0	0.166667
AllState3	0.0	1.0	0.0	0.166667
AllStateClean1	1.0	0.0	0.0	0.333333
AllStateClean2	0.0	0.0	0.0	0.333333
AllStateClean3	0.0	0.0	0.0	0.0
ConstantDefiner	0.047619	0.0	0.0	0.0434783
DataStorage	0.8	0.0909091	0.0	0.272727
Delegater	0.0	0.0	0.0	0.111111
Funnel	0.0	0.375	0.25	0.25
Interface	0.15625	0.424242	0.0588235	0.485714
MicroSpecialExtender	0.0	0.0	0.0	0.0
MicroSpecialOverrider	0.0	0.0	0.0	0.0
SharingEntries4	0.25	0.0	0.0	0.75
SharingEntries5	0.0	1.0	0.0	0.25
SingleEntry	0.0	0.75	0.5	0.25
Singleton	0.0	0.0	0.0	0.0
StructuredFlow	0.0	1.0	1.0	0.5
ThreeLayers	1.0	1.0	0.333333	1.0
Wide Interface	0.0	0.0	0.0	0.0

Table E.35: Moose version 2.84 Profiles

<table>
<tr><td>Class name</td><td>(|#FAMIXAbstractStructuralEntity|)</td><td>(|#MSEMooseLauncher|)</td><td>(|#ModelLoaderUI|)</td><td>(|#VisualWorksParseTreeEnumerator|)</td></tr>
<tr><td>Size Pattern</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td></tr>
<tr><td>No. Method Clumps</td><td>'0'</td><td>'0'</td><td>'1'</td><td>'0'</td></tr>
<tr><td>AdderExtender</td><td>0.0285714</td><td>0.0833333</td><td>0.0</td><td>0.025641</td></tr>
<tr><td>AdderNormal</td><td>0.857143</td><td>0.722222</td><td>0.789474</td><td>0.153846</td></tr>
<tr><td>AdderOverrider</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0769231</td></tr>
<tr><td>AllState1</td><td>0.0</td><td>0.666667</td><td>0.0</td><td>0.333333</td></tr>
<tr><td>AllState2</td><td>0.0</td><td>0.333333</td><td>0.125</td><td>0.0</td></tr>
<tr><td>AllState3</td><td>0.0</td><td>0.333333</td><td>0.125</td><td>0.333333</td></tr>
<tr><td>AllStateClean1</td><td>1.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean2</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllStateClean3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ConstantDefiner</td><td>0.0434783</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>DataStorage</td><td>0.363636</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>Delegater</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>Funnel</td><td>0.125</td><td>0.625</td><td>1.0</td><td>0.5</td></tr>
<tr><td>Interface</td><td>0.914286</td><td>0.805556</td><td>0.789474</td><td>0.25641</td></tr>
<tr><td>MicroSpecialExtender</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>MicroSpecialOverrider</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries4</td><td>1.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries5</td><td>0.0</td><td>0.875</td><td>0.125</td><td>0.25</td></tr>
<tr><td>SingleEntry</td><td>0.5</td><td>0.75</td><td>0.666667</td><td>1.0</td></tr>
<tr><td>Singleton</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>StructuredFlow</td><td>0.0</td><td>0.5</td><td>0.0</td><td>1.0</td></tr>
<tr><td>ThreeLayers</td><td>1.0</td><td>0.666667</td><td>0.666667</td><td>0.666667</td></tr>
<tr><td>Wide Interface</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
</table>

Table E.36: Moose version 2.84 Profiles

<table>
<tr><td>Class name</td><td>(|#MSEEntityView|)</td><td>(|#XMIDTDProducer|)</td><td>(|#MooseGagerUI|)</td><td>(|#MGMetricsFacade|)</td><td>(|#FAMIXInvocation|)</td><td>(|#ContextualMetricOperator|)</td></tr>
<tr><td>Size Pattern</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td></tr>
<tr><td>No. Method Clumps</td><td>'0'</td><td>'1'</td><td>'1'</td><td>'0'</td><td>'0'</td><td>'1'</td></tr>
<tr><td>AdderExtender</td><td>0.0</td><td>0.0</td><td>0.0222222</td><td>0.0</td><td>0.0444444</td><td>0.0</td></tr>
<tr><td>AdderNormal</td><td>0.1</td><td>0.0697674</td><td>0.6</td><td>0.666667</td><td>0.688889</td><td>0.0</td></tr>
<tr><td>AdderOverrider</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0652174</td></tr>
<tr><td>AllState1</td><td>0.666667</td><td>0.333333</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllState2</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>AllState3</td><td>0.333333</td><td>0.666667</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AllStateClean1</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.333333</td><td>1.0</td><td>0.333333</td></tr>
<tr><td>AllStateClean2</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td></tr>
<tr><td>AllStateClean3</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ConstantDefiner</td><td>0.037037</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0333333</td><td>0.0</td></tr>
<tr><td>DataStorage</td><td>0.153846</td><td>0.0</td><td>0.233333</td><td>0.133333</td><td>0.4</td><td>0.133333</td></tr>
<tr><td>Delegater</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>Funnel</td><td>0.25</td><td>0.5</td><td>0.5</td><td>1.0</td><td>0.375</td><td>0.25</td></tr>
<tr><td>Interface</td><td>0.125</td><td>0.0697674</td><td>0.622222</td><td>0.666667</td><td>0.755556</td><td>0.0652174</td></tr>
<tr><td>MicroSpecialExtender</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>MicroSpecialOverrider</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries4</td><td>0.875</td><td>0.0</td><td>1.0</td><td>1.0</td><td>1.0</td><td>0.375</td></tr>
<tr><td>SharingEntries5</td><td>0.25</td><td>1.0</td><td>0.25</td><td>0.0</td><td>0.125</td><td>0.0</td></tr>
<tr><td>SingleEntry</td><td>0.25</td><td>0.5</td><td>0.25</td><td>0.75</td><td>0.25</td><td>0.25</td></tr>
<tr><td>Singleton</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>StructuredFlow</td><td>1.0</td><td>1.0</td><td>1.0</td><td>1.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>ThreeLayers</td><td>1.0</td><td>0.666667</td><td>1.0</td><td>1.0</td><td>1.0</td><td>1.0</td></tr>
<tr><td>Wide Interface</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
</table>

Table E.37: Moose version 2.84 Profiles

<table>
<tr><td>Class name</td><td>(|#AbstractEntity|)</td><td>(|#FAMIXAbstractBehaviouralEntity|)</td><td>(|#MSEMetricManager|)</td><td>(|#ImportingContext|)</td><td>(|#FAMIXClass|)</td><td>(|#MSEModel|)</td></tr>
<tr><td>Size Pattern</td><td>'Normal'</td><td>'Normal'</td><td>'Normal'</td><td>'Giant'</td><td>'Giant'</td><td>'Giant'</td></tr>
<tr><td>No. Method Clumps</td><td>'0'</td><td>'0'</td><td>'0'</td><td>'0'</td><td>'0'</td><td>'0'</td></tr>
<tr><td>AdderExtender</td><td>0.0</td><td>0.0151515</td><td>0.0</td><td>0.0</td><td>0.0194175</td><td>0.0</td></tr>
<tr><td>AdderNormal</td><td>0.293103</td><td>0.757576</td><td>0.971831</td><td>0.431818</td><td>0.854369</td><td>0.65873</td></tr>
<tr><td>AdderOverrider</td><td>0.0</td><td>0.030303</td><td>0.0</td><td>0.0</td><td>0.0291262</td><td>0.00793651</td></tr>
<tr><td>AllState1</td><td>0.25</td><td>1.0</td><td>1.0</td><td>0.8</td><td>0.0</td><td>0.25</td></tr>
<tr><td>AllState2</td><td>0.25</td><td>1.0</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.25</td></tr>
<tr><td>AllState3</td><td>0.25</td><td>0.25</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.25</td></tr>
<tr><td>AllStateClean1</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td><td>1.0</td><td>0.666667</td></tr>
<tr><td>AllStateClean2</td><td>0.333333</td><td>0.333333</td><td>0.0</td><td>0.333333</td><td>0.333333</td><td>0.666667</td></tr>
<tr><td>AllStateClean3</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td></tr>
<tr><td>ConstantDefiner</td><td>0.153846</td><td>0.0227273</td><td>0.0212766</td><td>0.0</td><td>0.0144928</td><td>0.0</td></tr>
<tr><td>DataStorage</td><td>0.0789474</td><td>0.227273</td><td>0.0434783</td><td>0.172414</td><td>0.176471</td><td>0.0833333</td></tr>
<tr><td>Delegater</td><td>0.0416667</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.333333</td></tr>
<tr><td>Funnel</td><td>1.0</td><td>0.5</td><td>1.0</td><td>1.0</td><td>0.5</td><td>1.0</td></tr>
<tr><td>Interface</td><td>0.431034</td><td>0.863636</td><td>0.985915</td><td>0.431818</td><td>0.912621</td><td>0.825397</td></tr>
<tr><td>MicroSpecialExtender</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>MicroSpecialOverrider</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>SharingEntries4</td><td>1.0</td><td>0.5</td><td>0.25</td><td>1.0</td><td>1.0</td><td>1.0</td></tr>
<tr><td>SharingEntries5</td><td>0.25</td><td>1.0</td><td>1.0</td><td>0.0</td><td>0.5</td><td>0.25</td></tr>
<tr><td>SingleEntry</td><td>0.5</td><td>0.5</td><td>0.75</td><td>1.0</td><td>0.5</td><td>0.75</td></tr>
<tr><td>Singleton</td><td>1.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr>
<tr><td>StructuredFlow</td><td>1.0</td><td>0.5</td><td>0.5</td><td>0.0</td><td>0.5</td><td>0.5</td></tr>
<tr><td>ThreeLayers</td><td>1.0</td><td>1.0</td><td>1.0</td><td>1.0</td><td>1.0</td><td>1.0</td></tr>
<tr><td>Wide Interface</td><td>0.0</td><td>0.863636</td><td>0.985915</td><td>0.431818</td><td>0.912621</td><td>0.825397</td></tr>
</table>

Table E.38: Moose version 2.84 Profiles

Bibliography

[AKUT 93] T. Akutsu. *A Polynomial Time Algorithm for Finding a Largest Common Subgraph of almost Trees of Bounded Degree*. IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, vol. E76-A, no. 9, 1993. (p 35)

[ARÉV 04] G. Arévalo, F. Buchli, and O. Nierstrasz. Software Pattern Detection using Formal Concept Analysis. March 2004. (p 2)

[BERT 74] J. Bertin. Graphische Semiologie. Walter de Gruyter, 1974. (p 9)

[BUNK 02] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and Vento. *A Comparison of Algorithms for Maximum Common Subgraph on Randomly Connected Graphs*. In Structural, Syntactic and Statistical Pattern Recognition, volume 2396, pages 123–132, 2002. (pp 31, 32)

[CASA 98] E. Casais. *Re-Engineering Object-Oriented Legacy Systems*. Journal of Object-Oriented Programming, vol. 10, no. 8, pages 45–52, January 1998. (p 5)

[CHIK 90] E. J. Chikofsky and J. H. Cross, II. *Reverse Engineering and Design Recovery: A Taxonomy*. IEEE Software, pages 13–17, January 1990. (p 5)

[CONS 93] M. P. Consens and A. O. Mendelzon. *Hy+: A Hygraph-based Query and Visualisation System*. In Proceeding of the 1993 ACM SIGMOD International Conference on Management Data, SIGMOD Record Volume 22, No. 2, pages 511–516, 1993. (p 23)

[CROS 98] J. H. Cross II, S. Maghsoodloo, and D. Hendrix. *Control Structure Diagrams: Overview and Evaluation*. Journal of Empirical Software Engineering, vol. 3, no. 2, pages 131–158, 1998. (p 23)

[DAVI 95] A. M. Davis. 201 Principles of Software Development. McGraw-Hill, 1995. (p 5)

[DEME 01] S. Demeyer, S. Tichelaar, and S. Ducasse. *FAMIX 2.1 — The FAMOOS Information Exchange Model*. Research report, University of Bern, 2001. (p 7)

[DEME 02] S. Demeyer, S. Ducasse, and O. Nierstrasz. Object-Oriented Reengineering Patterns. Morgan Kaufmann, 2002. (pp 5, 16)

[DUCA 00] S. Ducasse, M. Lanza, and S. Tichelaar. *Moose: an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems*. In Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000), June 2000. (pp 23, 57)

[EICK 92] S. G. Eick, J. L. Steffen, and S. Eric E., Jr. *SeeSoft—A Tool for Visualizing Line Oriented Software Statistics*. IEEE Transactions on Software Engineering, vol. 18, no. 11, pages 957–968, November 1992. (p 23)

[FUNA 99] N. FUNABIKI and J. KITAMICHI. *A Two-Stage Discrete Optimization Method for Largest Common Subgraph Problems*. In IEICE Transactions, volume E82-D, pages 1145–1153, 1999. (p 35)

[GAMM 95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Reading, Mass., 1995. (p 16)

[HEND 02] D. Hendrix, J. H. Cross II, and S. Maghsoodloo. *The Effectiveness of Control Structure Diagrams in Source Code Comprehension Activities*. IEEE Transactions on Software Engineering, vol. 28, no. 5, pages 463–477, Mai 2002. (p 23)

[HIGG 87] D. A. Higgins and N. Zvegintzov. Data Structured Software Maintenance: The Warnier/Orr Approach. Dorset House, January 1987. (p 23)

[JERD 97] D. J. Jerding, J. T. Stansko, and T. Ball. *Visualizing Interactions in Program Executions*. In Proceedings of ICSE '97, pages 360–370, 1997. (p 23)

[KAZM 99] R. Kazman and S. J. Carriere. *Playing detective: Reconstructing software architecture from available evidence.* Automated Software Engineering, April 1999. (p 23)

[KLEY 88] M. F. Kleyn and P. C. Gingrich. *GraphTrace — Understanding Object-Oriented Systems Using Concurrently Animated Views*. In Proceedings OOPSLA '88, volume 23, pages 191–205, November 1988. (p 23)

[KLIM 96] E. J. Klimas, S. Skublics, and D. A. Thomas. Smalltalk with Style. Prentice-Hall, 1996. (p 12)

[LANG 95] D. B. Lange and Y. Nakamura. *Interactive Visualization of Design Patterns can help in Framework Understanding*. In Proceedings of OOPSLA '95, pages 342–357. ACM Press, 1995. (p 23)

[LANZ] M. Lanza. *CodeCrawler*. http://www.iam.unibe.ch/∼lanza/CodeCrawler/codecrawler.html. (pp 23, 57)

[LANZ 03] M. Lanza. *Object-Oriented Reverse Engineering — Coarse-grained, Fine-grained, and Evolutionary Software Visualization*. PhD thesis, University of Berne, Mai 2003. (pp 1, 2, 11, 44, 47)

[LIEB 89] K. J. Lieberherr and A. J. Riel. *Contributions to Teaching Object Oriented Design and Programming*. In Proceedings OOPSLA '89, ACM SIGPLAN Notices, volume 24, pages 11–22, October 1989. (p 16)

[LITT 96] D. Littman, J. Pinto, S. Letovsky, and E. Soloway. *Mental Models and Software Maintenance*. In Soloway and Iyengar, editors, Empirical Studies of Programmers, First Workshop, pages 80–98, 1996. (p 24)

[MCGR 82] J. J. McGregor. *Backtrack Search Algorithms and the Maximal Common Subgraph Problem*. Software Practice and Experience, vol. 12, pages 23–34, 1982. (p 30)

[MEND 95] A. Mendelzon and J. Sametinger. *Reverse Engineering by Visualizing and Querying*. Software — Concepts and Tools, vol. 16, pages 170–182, 1995. (p 23)

[MÜ 86] H. A. Müller. *Rigi — A Model for Software System Construction, Integration, and Evaluation based on Module Interface Specifications*. PhD thesis, Rice University, 1986. (p 6)

[NASS 73] I. Nassi and B. Shneiderman. *Flowchart Techniques for Structured Programming*. SIGPLAN Notices, vol. 8, no. 8, August 1973. (p 23)

[NIER 02] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. *Towards pattern-based design recovery*. In Proceedings of the 24th international conference on Software engineering, pages 338–348. ACM Press, 2002. (p 1)

[PAUW 93] W. D. Pauw, R. Helm, D. Kimelman, and J. Vlissides. *Visualizing the Behavior of Object-Oriented Systems*. In Proceedings OOPSLA ’93, pages 326–337, October 1993. (p 23)

[PAUW 99] W. D. Pauw and G. Sevitsky. *Visualizing Reference Patterns for Solving Memory Leaks in Java*. In R. Guerraoui, editor, Proceedings ECOOP ’99, volume 1628 of *LNCS*, pages 116–134, Lisbon, Portugal, June 1999. Springer-Verlag. (p 23)

[PRIC 93] B. A. Price, R. M. Baecker, and I. S. Small. *A Principled Taxonomy of Software Visualization*. Journal of Visual Languages and Computing, vol. 4, no. 3, pages 211–266, 1993. (p 23)

[REIS 90] S. P. Reiss. *Interacting with the FIELD environment*. Software — Practice and Experience, vol. 20, pages 89–115, 1990. (p 23)

[RICH 99] T. Richner and S. Ducasse. *Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information*. In H. Yang and L. White, editors, Proceedings ICSM ’99 (International Conference on Software Maintenance), pages 13–22. IEEE Computer Society Press, September 1999. (pp 6, 23)

[RUGA 98] S. Rugaber and J. White. *Restoring a Legacy: Lessons Learned*. IEEE Software, vol. 15, no. 4, pages 28–33, July 1998. (p 5)

[SOMM 00] I. Sommerville. Software Engineering. Addison Wesley, Sixth edition, 2000. (p 5)

[STAS 90] J. T. Stasko. *TANGO: A Framework and System for Algorithm Animation*. IEEE Computer, vol. 23, no. 9, pages 27–39, September 1990. (p 23)

[STOR 95] M.-A. D. Storey and H. A. Müller. *Manipulating and documenting software structures using SHriMP views*. In Proceedings of the 1995 International Conference on Software Maintenance, 1995. (pp 6, 23)

[TILL 94] S. R. Tilley, K. Wong, M.-A. D. Storey, and H. A. Müller. *Programmable Reverse Enginnering*. International Journal of Software Engineering and Knowledge Engineering, vol. 4, no. 4, pages 501–520, 1994. (p 23)

[TONE 99] P. Tonella and G. Antoniol. *Object Oriented Design Pattern Inference*. In Proceedings ICSM ’99, pages 230–238, October 1999. (p 2)

[TUFT 90] E. R. Tufte. Envisioning Information. Graphics Press, 1990. (p 9)

[VON 96] A. von Mayrhauser and A. Vans. *Identification of Dynamic Comprehension Processes During Large Scale Maintenance*. IEEE Transactions on Software Engineering, vol. 22, no. 6, pages 424–437, June 1996. (p 24)

[WANG 97] Wang, Fan, and Horng. *Genetic-Based Search for Error-Correcting Graph Isomorphism*. IEEETSMC: IEEE Transactions on Systems, Man, and Cybernetics, vol. 27, 1997. (pp 35, 71)

[WHIT 32] H. Whitney. *Congruent graphs and the connectivity of graphs*. Am. J. Math., vol. 54, pages 150–168, 1932. (p 29)

[WILD 92] N. Wilde and R. Huitt. *Maintenance Support for Object-Oriented Programs*. IEEE Transactions on Software Engineering, vol. SE-18, no. 12, pages 1038–1044, December 1992. (p 5)

[Sta 98] J. T. Stasko, J. Domingue, M. H. Brown, and B. A. Price, editors. Software Visualization — Programming as a Multimedia Experience. The MIT Press, 1998. (p 23)