Passau University

Department of Informatics and Mathematics

Master's Thesis

On the Relation of Type Errors and Static Attributes of Feature-Oriented Product Lines

Author:
Judith Roth

First Supervisor:
Dr.-Ing. Sven Apel
Second Supervisor:
Prof. Dr. Christian Lengauer

August 21, 2014

Abstract

Feature-oriented software product lines are built using features that implement a functionality or configuration option. Thus code is reused which leads to reduced development costs. The variability of features can lead to product-line-specific type errors, which occur if a feature requires another feature without that relation being described in the feature model. Hence a valid product can contain references to not included features and consequently contain type errors. The relation between such type errors and static attributes is examined in this thesis in order to increase understanding of them. 29 feature-oriented software product lines are evaluated with 17 measures belonging to different approaches that try to capture different aspects of the reasons for those type errors: Variability, coupling, possible feature interactions and code fragmentation. For the variability measures, which are based solely on the feature model, no correlations to type errors have been found. Approaches which considered the references between features (coupling) and the number of possible feature interactions showed weak to moderate correlation for about a third of the subject product lines. Fragmentation of classes correlated to the type errors for half of the subject systems, most moderately or strongly. Measures based on classes correlated stronger to type errors than measures based on features.

Contents

1 Introduction

1.1 Motivation

Software is part of our daily lives, still it is almost never finished completely - it is work in progress, evolving version for version. This brings certain challenges for developers.

On the one hand, software systems tend to grow into software-ecosystems, like Eclipse and Firefox, which provide a huge range of configuration possibilities e.g. with plugins that enable users to adjust the software according to their own requirements through adding extra functions and features. On the other hand, more and more platforms for running software are available. A good example are all those different platforms for mobile devices like Android, iOS, Windows Phone, Sailfish OS, Ubuntu Touch, Bada and even more.

Therefore a programming paradigm was necessary to enable developers to keep pace with those trends and not having to develop the same software over and over again in different configurations or for different platforms.

This is what *software product lines* and *feature-oriented programming* are created for. Instead of developing one self-contained and inflexible software, rather a family of programs belonging to a certain domain is developed. This is achieved through reuse and variability of *features*, which represent software parts in terms of a functionality or a configuration option. Thus software product lines facilitate mass-customization [1]. Through immense reuse of code lower time-to-market and cost reduction is possible for large software systems when implemented as product line [11].

A huge drawback of larger software systems is that they get harder to maintain and enhance as the complexity usually grows with the size. This is also true for software product lines, because of the additional difficulty of optional software parts (features) that are potentially not completely isolated from each other and hence can require or even disturb each other. That is why errors can arise only for certain products of the product line which contain or miss specific features. Those are *product-line-specific type errors*, as they can not occur in software without optional parts. Those errors are therefore harder to find than errors in conventional software, because either all products, which can be up-to exponential to the number of features, need to be tested or variability-aware testing that has to take into account relationships between features is necessary [17].

Given that feature-oriented software product line development is a rather new paradigm, the understanding of what causes product-line-specific type errors is fairly low. Experienced developers often have a good intuition about

which code parts of a software they work on are likely to cause errors. There are several measures for conventional software which capture different aspects of that intuition, for example measures that indicate structuredness or complexity of code components. However, as feature-oriented development is quite young, there are so far no measures for indicating which code parts or features are likely to contain those product-line-specific type errors.

Therefore the goal of this thesis is to explore the reasons for product-line-specific type errors in order to find measures which correlate with them. That knowledge and measures can then be used by developers to understand what kind of components they have to pay extra attention to for avoiding type errors. The focus for that matter lies on the characteristics of features that contain type errors.

1.2 Outline of the Thesis

This thesis is organized in four large parts.

In Chapter 2 Background fundamentals of software product lines and feature-oriented development are covered as well as principles of feature models and feature interactions. Also information about software measures and type checking for product lines is given.

Chapter 3 Design and Implementation is divided in two parts: The first covers how type errors are counted in this thesis and the way they can be represented as a graph. The second part elaborates on the measures which are used to quantify static attributes of feature-oriented software product lines. Most of the measures operate on code-level, therefore quantify characteristics regarding the structure of code. Some measures also operate on the feature model, therefore capturing the conceptual structure of a product line.

The relation between those measures and type errors is examined in Chapter 4 Evaluation after the introduction of the subject product lines which are used in this thesis.

The results are then summarized in Chapter 5 Conclusion.

2 Background

2.1 Feature-Oriented Software Product Lines

Software product lines (SPLs) are modular software systems, each of them belonging to a certain domain [14]. They are build of different components that implement single functions or requirements and are preferably isolated from each other. These components are called *features* [14]. Apel and Kästner define a feature as *"a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option."* [3]

Features can be combined in different ways to build a *product*. Therefore, every product is a concrete software with certain characteristics and some products may share some of these characteristics.

There are different ways in which software product lines can be implemented. Some of them are using preprocessor annotations, building frameworks, aspect orientation and feature orientation [1][27]. In contrast to other approaches, *feature orientation* prioritizes the organization of all software artifacts in features. That way all implementation fragments of one feature are united in a single module, called *collaboration*. This leads to a better overview on which code belongs to which feature hence tracing and debugging are simplified [1]. *Collaboration diagrams* are used to illustrate the association between features and classes. The part of a class that was introduced by a certain feature is called *role*. An example of a collaboration diagram is shown in Figure 1 with an excerpt of GPL, a graph library that is implemented as product line.

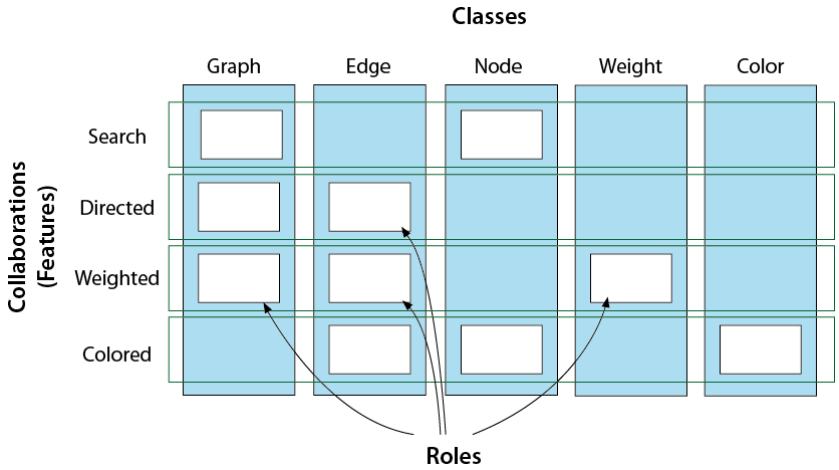

Figure 1: Extract of the collaboration diagram of GPL (taken from [24])

2.2 Feature Model

The way features can be combined to build valid products is defined in a *feature model* [8]. Therefore the feature model constitutes the variability of its features [1]. There are different ways, in which a feature model can be represented and in many representations it is possible to define features, which contain no code but are used for grouping and therefore help to increase understandability of the model. These features are called *abstract features*. In contrast, all other features are called *concrete features*.

One simple method to represent the feature model is using **feature-lists**. For each valid product there is a file which lists the names of features included in this product. This is only suitable for small product lines, as the dependencies are only implicitly modeled and for bigger product lines there is also the problem of explosion of possible variants, up-to exponential in the number of features [6].

Feature models can also be depicted as **feature-diagrams**, which are trees whose nodes represent features [1]. Figure 2 shows the feature-diagram of EPL, a software product line for evaluating expressions.

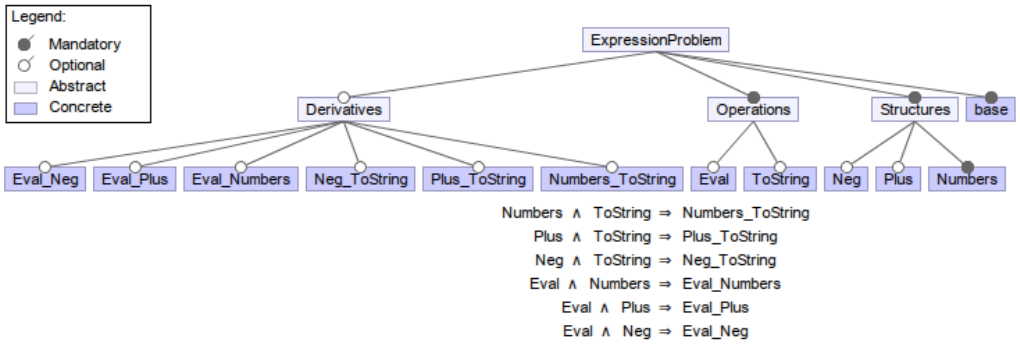

Figure 2: Feature-diagram of EPL

Hierarchical structures can be modeled as parent-child relations which are easy to discern in the graphical representation and additional propositional formulas can be used to define dependencies that do not fit into the structure of a tree. The following different dependencies between parent and child/children are possible [8][1]:

Mandatory Parent and child can only be selected together (logical equivalence)

Optional The child can only be selected if the parent is selected, whereas the parent can be selected independently from the child (logical

implication)

And All child features must be selected

Or One or more of the children can be selected

Alternative Exactly one of the children must be selected

Another way to describe the information of the feature model is to **list all features pairwise** and state their relation. There are three possible relations:

Never-dependency: Feature 1 can never be selected together with feature 2 and vice versa.

Always-dependency: Feature 1 is always selected if feature 2 is selected (which does not automatically lead to feature 2 is always selected if feature 1 is selected).

Maybe-dependency: Feature 1 may be selected if feature 2 is selected (which as well does not assume it is the same the other way around, as it is also possible that feature 2 is always selected if feature 1 is selected)

Further possibilities to represent the feature model are using grammars or propositional formulas. It is possible to automatically transform feature-diagrams into grammars and vice versa, as the diagram is just another notation for the grammar [25]. A propositional formula representing the feature model can also be derived from the feature diagram. In this thesis FeatureIDE [1] was used for generating feature-diagrams and converting feature models to their different representations.

2.3 Feature Interaction

In an ideal world, each feature of a software product line is an closed, independent component, which is therefore easy to add to or remove from a configuration which builds a product. It only adds a defined behaviour. When the behaviour of a feature changes due to the presence of one or more other features a *feature interaction* takes place [3][5]. Sometimes feature interactions are wanted, for example to ensure communication and cooperation between features in order to accomplish a task or to ensure a requirement is met [5].

[1] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

A very descriptive example for unintended feature interaction is the combination of features "flood control" and "fire control" in facility management. If, in case of a fire, "flood control" does turn off the sprinkling system before the fire is extinguished, unwanted feature interaction takes place [24].

2.4 Type Checking

An important topic in computer-science is ensuring that systems behave the way they were specified. There are different formal methods to help ensuring the correct behaviour of programs. One formal and lightweight method are type systems [23]. Pierce defines a type system as *"a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute."* [23]

The process of analyzing if a program is well-typed, therefore containing no type errors according to the type system defined by the programming language, is called type checking [26].

The modularity of product lines make type checking even more challenging as type errors can occur only for certain products, which contain or miss specific features. There are different ways in which type checking on software product lines can be done.

One approach is to build all valid products and then type check each of them. This is called *product-based* type checking. But due to the problem of explosion of possible variants it is possible only for small product lines to compose and check all valid products [7].

Another method is to type check every feature in isolation, therefore called *feature-based* type check. With this approach the effort is linear to the number of features instead of exponential as it is with product-based type checking. But as the information of the feature model is not examined, the analysis might be incomplete, because issues across two or more features are not considered [26].

Therefore the efficient and comprehensive way to type check product lines is to check on the entire codebase and to take into account information about feature-dependencies and feature-variability provided by the feature model [7][26]. This approach is called *family-based* type checking [18].

This thesis uses information about type errors found with Fuji [2], an extensible compiler for feature-oriented programming in Java. Fuji is capable of all three introduced variants of type checking, but in this thesis family-based type checking was used due to it being efficient and comprehensive. For further differentiation, type errors that can only occur in software product lines

[2] http://fosd.de/fuji/

```
1 package tmp;
2
3 public class Neg {
4   public int eval(){
5     return -x.eval();
6   }
7 }
```

Listing 1: Class Neg of feature Eval_Neg

```
1 package tmp;
2
3 public interface Exp {
4   int eval();
5 }
```

Listing 2: Class Exp of feature Eval

as they are based on information in two or more features, will be referred to as SPL type errors and all others as feature-local (FL) type errors. SPL type errors occur, if one feature references or uses information provided by another feature, without reflecting this relationship between the two features in the feature model. Therefore SPL type errors are unwanted feature interactions. An Example of this is shown in Listings 1 and 2. According to the feature model of EPL (see Figure 2 in Chapter 2.2), feature Eval_Neg may be selected if feature Eval is selected, but it is possible to select feature Eval_Neg without selecting feature Eval. This leads to the SPL type error shown in Listing 3.

```
1 ~/fuji/examples/EPL/Eval_Neg/tmp/Neg.java:5:
2   Semantic Error: MAYBE dependency:
3 Feature Eval_Neg accesses the method
4   int eval();
5 of feature Eval.
6 Feature Eval may not be present in every valid selection.
```

Listing 3: SPL type error of EPL found with Fuji

SPL type errors can be further divided by the relationship between the engaged features. They can occur between features that share a maybe-dependency or a never-dependency, because only with this dependencies it is possible that information used in one feature and introduced in another feature is missing in a product that includes the first but excludes the second feature.

2.5 Software Measures

Measuring of software systems is done to ensure quality and to track the development progress [16]. It is possible to distinguish between two kinds of software measures. On the one hand, there are measures on internal attributes of software, such as lines of code, cohesion or coupling. They deal

with the structure of software. On the other hand, there are measures for external attributes such as performance or usability, which require to consider the environment, in which the software is running, besides the software itself [22]. Only measures on the internal attributes of software are considered in this thesis.

Cohesion and coupling are measures used to determine structuredness of software [2]. Cohesion refers to the degree to which code elements like fields or methods of a class depend on other elements of the same class. One characteristic of highly cohesive classes is that they can not be splitted easily in two or more classes [10]. This indicates that they implement only one concern, which is good for understanding and reusing of code [15][21]. Coupling on the contrary refers to the degree to which elements of a class refer to elements of other classes [2]. Therefore it is the counterpart to cohesion. Low coupling is preferable as it means that a component is mostly isolated and therefore can be exchanged with less difficulty than components with strong ties to other components. The concepts of coupling and cohesion apply to features as well as classes [2].

3 Design and Implementation

3.1 Representation and Counting of SPL Type Errors

As mentioned before, in this thesis the data regarding type errors is gathered using Fuji [3] with family-based type checking.

3.1.1 Counting Type Errors for Features

Interactions between features and thus SPL type errors as well (see Chapters 2.3 and 2.4) can be modelled as a graph: $G_{SPLTE} = (V, E)$, where V is a set of vertices representing features and $E \subseteq V \times V$ is a set of edges containing an edge for every pair of features that take part in a SPL type error together. An edge weight $w : E \to \mathbb{Z}$ is used to record the number of errors that both features take part in. Accordingly vertex attributes $a_{SPL} : V \to \mathbb{Z}$ and $a_{FL} : V \to \mathbb{Z}$ are used to count the number of SPL and feature-local type errors a feature is involved in. How the number of errors for vertices (features) and edges are calculated is described in the following.

3.1.1.1 SPL Type Error between two Features

SPL type errors can arise due to different constellations, which need to be reflected differently in G_{SPLTE}. The simplest constellation consists of a reference from one feature to another without this relationship being reflected in the feature model. This corresponds to a directed edge from the first feature to the second, which is referenced. An example for this is given in Chapter 2.4, Listings 1 to 3. The edge-weight w is set to the number of occurrences of the reference in the code (or increased by the number if the edge already exists) as well as a_{SPL} for the feature that implements the reference. For the feature that is referenced, a_{SPL} is increased by 1 no matter how many code lines implement the reference, because it is one relationship missing in the feature model and from the point of view of the referenced feature, the type error(s) are always caused by the same (missing) part.

3.1.1.2 SPL Type Error between more than two Features

If more than two features are involved in a SPL type error, there are two different possibilities that need to be considered.

The first is quite similar to the case described in Chapter 3.1.1.1, but instead of one feature that is referenced, the reference could go to several features without having the relation to one of them reflected in the feature model. An example extracted from GPL is given in Listing 4, where

[3] http://fosd.de/fuji/

method EdgeConstructor is called in feature WeightedWithEdges, which is implemented in the features UndirectedWithEdges and DirectedWithEdges (both not shown). The relationship from WeightedWithEdges to UndirectedWithEdges or DirectedWithEdges is not reflected in the feature model thus leading to the type error shown in Listing 5. Again, for the feature that implements the reference, a_{SPL} is increased according to the number of code lines that implement the reference while for the features that are referenced a_{SPL} is increased by one, because the error is always caused by the same (missing) part. From the feature that references to every feature that can be used by the reference a edge in G_{SPLTE} is added with weight w according to the number of code lines that implement the reference.

```
1 public class Edge {
2     private int weight;
3
4     public void EdgeConstructor( Vertex the_start,  Vertex
          the_end, int the_weight ) {
5         EdgeConstructor( the_start,the_end );
6         weight = the_weight;
7     }
8     ...
9 }
```

Listing 4: Excerpt of class Edge of feature WeightedWithEdges from GPL

```
1 ~/fuji/examples/GPL/WeightedWithEdges/GPL/Edge.java:10:
2   Semantic Error: 2 optional targets (there may be a valid
        selection where none of these targets is present):
3 Feature WeightedWithEdges accesses:
4 - the method
5     public void EdgeConstructor(Vertex the_start, Vertex the_end
          ) { ... }
6   of feature UndirectedWithEdges
7 - the method
8     public void EdgeConstructor(Vertex the_start, Vertex the_end
          ) { ... }
9   of feature DirectedWithEdges
```

Listing 5: SPL type error of GPL found with Fuji

The other possibility that involves more than two features is if feature A references feature B which misses subtype-information (that is necessary in A e.g. because of an implicit cast) which is provided by feature C (or several other features). The type error occurs if the dependency between B and C (which is in this case needed for the inheritance-information) is not reflected in the feature model. An example of an type error of MobileMedia8, a product line for manipulating photos, videos and music on

mobile devices, is given in Listings 6 to 9. In class MediaController of feature includePhotoAlbum_x_CopyPhotoOrSMS object *controller* of class PhotoViewController uses the method *setNextController* (see Listing 6, line 5 and 6). Class PhotoViewController is implemented by features includePhotoAlbum_x_CopyPhotoOrSMSOrCapturePhoto (see Listing 7) and feature x_CopyPhotoOrSMSOrCapturePhoto (see Listing 8). Only feature x_CopyPhotoOrSMSOrCapturePhoto provides the information that PhotoViewController is a subtype of AbstractController (see Listing 8), which implements method *setNextController*.

Therefore, features includePhotoAlbum_x_CopyPhotoOrSMS and includePhotoAlbum_x_CopyPhotoOrSMSOrCapturePhoto can only be used in combination with feature x_CopyPhotoOrSMSOrCapturePhoto for a bug-free product. This relationship is again not modelled in the feature model, leading to the type error shown in Listing 9.

```
1  package lancs.mobilemedia.core.ui.controller;
2  public class MediaController {
3  @MethodObject static class MediaController_showImage {
4      protected void hook20(){
5          controller=new PhotoViewController(_this.midlet,_this.
               getAlbumData(),(AlbumListScreen)_this.
               getAlbumListScreen(),name);
6          controller.setNextController(nextcontroller);
7          canv.setCommandListener(controller);
8          nextcontroller=controller;
9          original();
10     }
11   }
12 }
```

Listing 6: Class MediaController of feature includePhotoAlbum_x_CopyPhotoOrSMS

```
1  package lancs.mobilemedia.core.ui.controller;
2  import lancs.mobilemedia.core.ui.screens.PhotoViewScreen;
3  public class PhotoViewController {
4  }
```

Listing 7: Class PhotoViewController of feature includePhotoAlbum_x_CopyPhotoOrSMSOrCapturePhoto from MobileMedia8

```
1 package lancs.mobilemedia.core.ui.controller;
2 ...
3 public class PhotoViewController extends AbstractController {
4     ...
5 }
```

Listing 8: Excerpt from class PhotoViewController of feature x_CopyPhotoOrSMSOrCapturePhoto from MobileMedia8

For this case, G_{SPLTE} is updated with several edges:

- From feature A (in the example feature includePhotoAlbum_x_CopyPhotoOrSMS) to all other features that take part in the error with weight w according to the number of code-parts that implement the error.

- From feature B to feature C (in the example from feature includePhotoAlbum_x_CopyPhotoOrSMSOrCapturePhoto to feature x_CopyPhotoOrSMSOrCapturePhoto) with weight w increased by 1, again because it is one missing information that causes the error between those two features.

The counting of errors for the features (vertices) is again similar to the other cases. For the feature that contains the type error and triggers the error-message, a_{SPL} is increased by the number of lines that contain the error, for all other participating features it is increased by 1.

```
1 ~/fuji/examples/MobileMedia8-fuji-compilable/
    includePhotoAlbum_x_CopyPhotoOrSMS/lancs/mobilemedia/core/ui/
    controller/MediaController.java:6:
2   Semantic Error: MAYBE dependency:
3 Class PhotoViewController of feature
    includePhotoAlbum_x_CopyPhotoOrSMSOrCapturePhoto accesses the
     method
4   public void setNextController(ControllerInterface
      nextController) { ... }
5 which is accessible via class AbstractController.
6 Class PhotoViewController of feature
    x_CopyPhotoOrSMSOrCapturePhoto extends class
    AbstractController.
7 The information that class PhotoViewController extends class
    AbstractController is only present in feature
    x_CopyPhotoOrSMSOrCapturePhoto.
8 Feature x_CopyPhotoOrSMSOrCapturePhoto may not be present in
    every valid selection.
```

Listing 9: SPL type error of MobileMedia8 found with Fuji

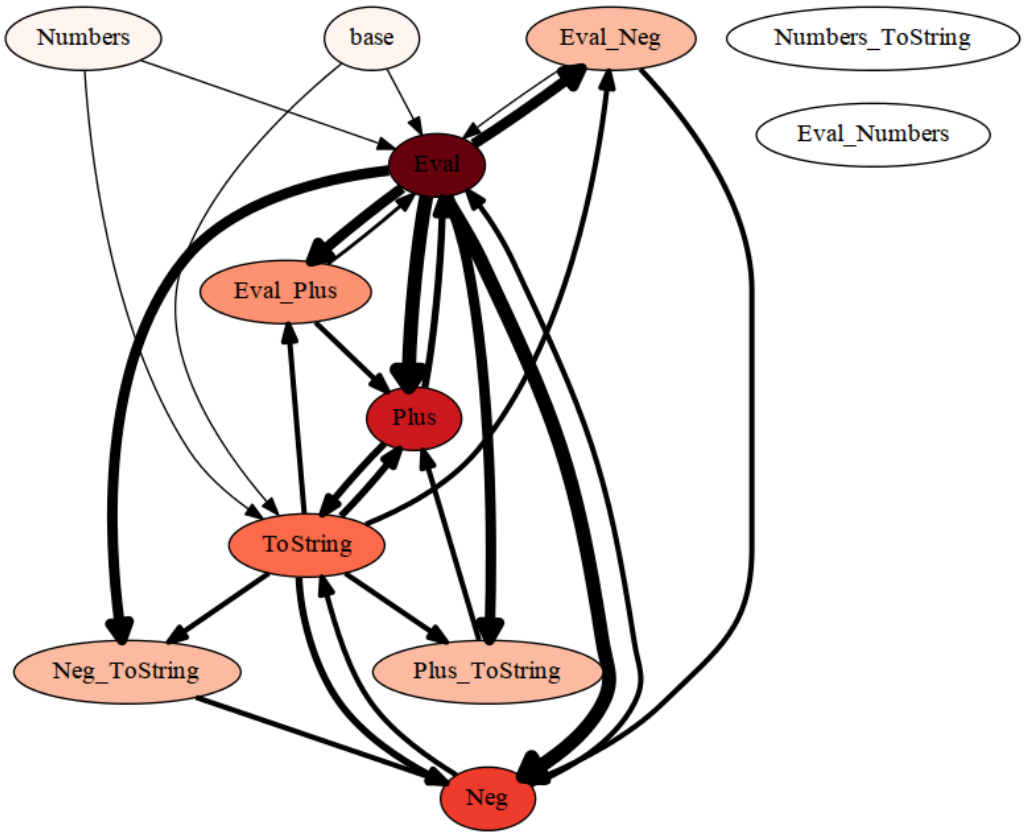

Figure 3: SPL type errors of EPL visualized as graph G_{SPLTE} (line width denotes the edge weight w (number of SPL type errors) and red color denotes number of type errors, white = no SPL type errors, light red = some SPL type errors and dark red = many SPL type errors)

In Figure 3 G_{SPLTE} for EPL is shown as an example. The vertices are colored according to the number of SPL type errors the corresponding feature takes part in. White means no SPL type errors and the more SPL type errors the feature is involved in the darker the red color. The width of the edges corresponds to the edge weight w, which represents the number of type errors that the connected features are involved in.

3.1.1.3 Feature-local Type Errors

Feature-local type errors involve only the feature they occur in, that is why a_{FL} of the feature that implements the error is increased by the number of appearances of the type error.

3.1.2 Counting Type Errors for Classes

Only the information in which class an error occurs is given. Hence the counting is straightforward: For every class it is counted how many SPL type errors are caused by it and if one SPL type error occurs in multiple lines it is counted once for each line. Feature-local type errors are not considered, because the main focus of this thesis are SPL type errors and feature-local type errors correspond to type errors in common software (which is not build with the software product line approach) and the relation between those and characteristics of classes are therefore well examined already.

3.2 Measures

The goal of this thesis was to examine reasons for type errors and to find measures that can be used as indicators for error proneness of software product line components like features. Hence the measures used and created in this thesis were meant to catch assumed causes of SPL type errors from different points of view. On the one hand, the complexity arising from the conceptual structure of a certain product line, which is documented in the feature model and therefore reflected in its variability, was examined. On the other hand a closer look on the structure of the code of product lines was taken, in order to find measures and that can make "code smells" quantifiable.

3.2.1 Variability

The variability of each feature of a software product line depends on the dependencies it has to other features, which are described in the feature model (see Chapter 2.2). SPL type errors occur, if a feature has a structural dependency, e.g. reference to a method of another feature without that dependency being modeled in the feature model. Because of that a connection between the characteristics of a feature in the feature model and SPL type errors appears possible. The information on variability is extracted from feature diagrams which were created using FeatureIDE [4].

In the following the two measures based solely on the information of the feature model that were developed in this thesis are described.

3.2.1.1 Dependencies to other Features

SPL type errors can occur if two features have a maybe- or never-dependency (see Chapter 2.4). Hence a positive correlation between the number of certain dependencies a feature F has to other features and the number of SPL type

[4] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

errors that F takes part in seems possible. Each ordered pair of features is mapped according to their dependency to one of the sets *always*, *never* and *maybe*. For computing a measure the dependencies are weighted.

As SPL type errors can never occur if one feature has *always* to be selected if another feature is selected, those types of dependencies are less important for finding features that are likely to be part of an SPL type error. Thus they are weighted lowest (1).

Never-dependencies are rather uncommon, because they are only derived if alternative-groups are used in the feature model or constraints are defined to ensure that two features can not be used in a valid product together. Because of this they are probably also easier to remember for developers. Hence those dependencies are weighted medium (5).

The dependency that appears most often and is therefore accountable for most SPL type errors is the *maybe*-dependency. So it is weighted highest (10).

With this weighting we define a dependency-measure *dep* for each feature F, where (u, v) is a pair of features:

$$\begin{aligned} dep(F) = &|\{(u,v) \in maybe \wedge u = F\}| * 10 \\ &+ |\{(u,v) \in never \wedge u = F\}| * 5 \\ &+ |\{(u,v) \in always \wedge u = F\}| \end{aligned}$$

3.2.1.2 Number of Constraints

As mentioned in Chapter 2.2, a feature diagram can have additional constraints in form of propositional logic formulas. Constraints affect the variability just as the tree-structure of the feature diagram does. For example, if one feature implies another feature, the variability of the first feature is reduced as all products that contain the first but not the second feature are no longer valid. Additionally, those constraints need to be considered while developing, hence they impede comprehensibility of the code.

Therefore a positive correlation between the number of constraints a feature F takes part in to SPL type errors is possible. However, as additional constraints can be used to reflect dependencies between features on code-level in the feature diagram (and thus prevent SPL type errors), a negative correlation seems possible as well.

For each feature F, the number of constraints it takes part in is counted:

$$\#constraints(F) = |\{constraints | F \text{ takes part in } constraint\}|$$

3.2.2 Cohesion and Coupling

Cohesion and coupling are measures for structuredness of software (see Chapter 2.5). As well-structured software components are easier to understand than poorly structured software components and SPL type errors arise due to references between features that are not reflected in the feature model, SPL type errors seem more likely in poor structured features, hence features with high coupling and low cohesion. Therefore a positive correlation between SPL type errors and the measures for coupling is expected and a negative correlation for the measure for cohesion.

Four measures for cohesion and coupling were examined in this thesis. *Internal-ratio Feature Dependency* and *External-ratio Feature Dependency* are measures for cohesion and coupling of features and were introduced by Apel and Beyer in 2011 [2]. *Coupling between Object Classes* measures cohesion between classes. It was introduced by Chidamber and Kemerer in 1994 [13]. *Coupling between Features* is based on the same principles but measures cohesion for features.

3.2.2.1 Internal-ratio Feature Dependency

Internal-ratio Feature Dependency (IFD) is a measure for the cohesion of a feature. It is based on the dependency graph $G = (V, E)$ which represents a software system. The set V of vertices represents the elements (e.g. fields, methods, classes) of the system and the set $E \subseteq V \times V$ the dependency relation between those elements. Furthermore, a mapping between features and elements is used to identify which elements were introduced by which feature F, called $elems(F)$.
For IFD the number of actual internal dependencies (id) of a feature F is measured in relation to the number of possible internal dependencies (which is $|elems(F)|^2$ as self-references are included):

$$IFD(F) = \frac{|id(F)|}{|elems(F)|^2}$$

with $id(F) = \{(v, w) | (v, w) \in E \wedge v \in elems(F) \wedge w \in elems(F)\}$.
We get $IFD(F) = 1$ if feature F is maximally cohesive, therefore all possible internal dependencies are actually present. On the other hand, if a feature F has no internal dependencies, $IFD(F) = 0$, indicating it is not cohesive [2].

3.2.2.2 External-ratio Feature Dependency

External-ratio Feature Dependency (EFD) is as well based on the dependency graph and is a measure for coupling of features:

$$EFD(F) = \frac{|id(F)|}{|d(F)|}$$

with $d(F) = \sum\{(v, w)|(v, w) \in E \wedge v \in elems(F)\}$
For highly coupled features that depend only on elements of other features and have no internal dependencies, $EFD(F) = 0$. On the contrary, if a feature depends only on elements it introduces, $EFD(F) = 1$ [2].

3.2.2.3 Coupling between Object Classes

Coupling between Object Classes (CBO) is as well a measure for coupling, but for classes. This measure counts to how many other classes a class C is coupled, with some restrictions on how coupling is defined:

$$CBO(C) = |\textit{classes coupled to } C|$$

Two classes are coupled, if one of them acts on the other e.g. through using of methods or instance variables of the other class. Thereby use and used-by relationships are examined, but every class is counted only once, no matter how often it is accessed. Several types of relations are not counted at all: Using of constants of another class, calls to API declarations, handling of events, use of user-defined types and object instantiations. If polymorphy is used, every class to which a call can go is counted [13]. Therefore a low CBO-value means low coupling while a high CBO refers to high coupling.

3.2.2.4 Coupling between Features

Coupling between Features (CBF) is based on CBO. It counts how many other features a feature F is coupled to:

$$CBO(F) = |\textit{features coupled to } F|$$

The rules of CBO on what types of relations are counted are applied for CBF as well. If a code element is introduced by more than one feature, all features are counted when it is referenced. A low CBF-value refers to low coupling and a high value to high coupling.

3.2.3 Possible Feature Interactions

Looking only at coupling is not sufficient to reason about type errors in software product lines, as implications can also arise if two or more features enhance the same class (see second example in Chapter 3.1.1.2, Listings 6 to 9). Following the argumentation that highly coupled code components are more error-prone due to higher complexity and less structuredness, it seems likely that features which can interact with a lot of other features are more error-prone than features that can interact with less other features as well. Hence in this thesis a model was designed for capturing the structure of possible feature interactions of a feature-oriented software product line.

Feature interactions are possible if one feature references or uses a code fragment which was introduced or enhanced by another feature (this corresponds to coupling). They are also possible if two or more features introduce or enhance the same code fragment. Other kinds of possible interactions between features are not considered in this thesis because only those interactions based on references or introduces can lead to SPL type errors.

For this thesis, the information about which code elements are introduced by which features (from now on called "introduces" / "intros") and feature-references (from now on called "references" / "refs") are gathered using Fuji.

3.2.3.1 Representation as Graph

Introduces and references can be represented as graphs. A graph for references is $G_{refs} = (V, E)$ with V being a set of vertices that represent features and $E \subseteq V \times V$ a set of edges that contains an edge between two features if there is a reference from the first feature to a code element that is introduced by the second feature. The number of the references from one feature to another is assigned to each edge as weight: $w : E \to \mathbb{Z}$. G_{refs} of EPL is given in Figure 4 as an example. Again, vertices with white background correspond to features without SPL type errors and the darker the red color, the more type errors the corresponding feature is involved in. For clear arrangement of the graph, the edge weight was not visualized.

$G_{intros} = (V, E)$ is a graph with V being a set of vertices representing features and $E \subseteq V \times V$ a set of edges containing an edge if both features in question introduce the same code element. For each two features introducing the same code element two directed edges between them are added to G_{intros} (each in one direction). The number of the same code elements that two features introduce is used as edge-weight: $w : E \to \mathbb{Z}$. Figure 5 illustrates G_{intros} for EPL.

G_{refs} and G_{intros} of a product line can be combined into one single graph of possible feature interactions (PFI): $G_{PFI} = (V, E)$ with $V = V_{refs} \cup V_{intros}$

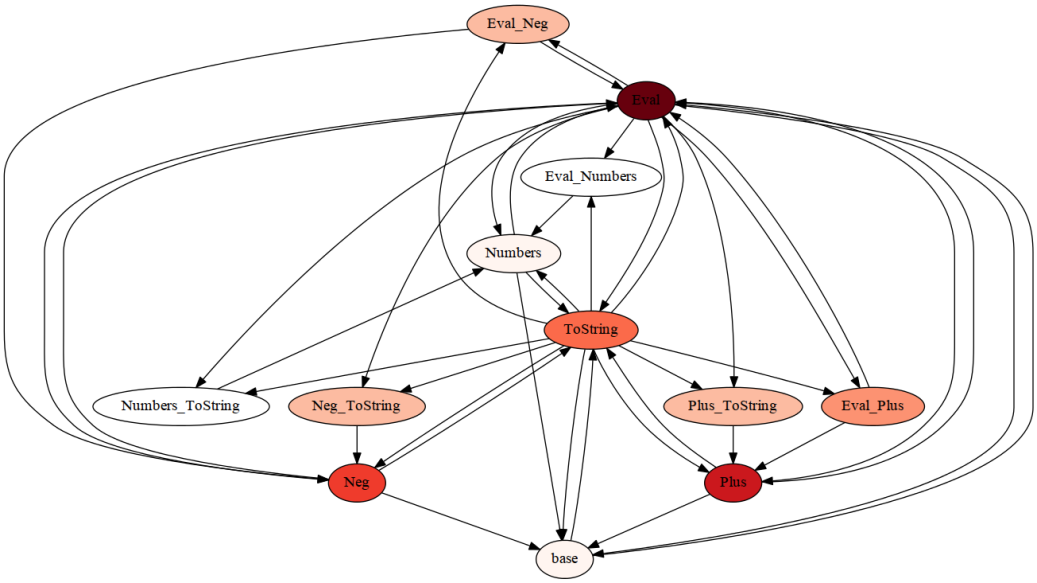

Figure 4: G_{refs} for EPL (red color denotes number of type errors, white = no SPL type errors, light red = some SPL type errors and dark red = many SPL type errors)

, $E = E_{refs} \cup E_{intros}$ and $w = w_{refs} + w_{intros}$. Figure 6 shows G_{PFI} for EPL. Graph measures can be used in order to measure the relative importance of a vertex (which represents a feature) in the graph (which represents the product line). The degree of a vertex represents the number of other features the corresponding feature can possibly interact with. Thus it is examined in different variations of the graphs, which can be enhanced with information based on the feature model. More complicated graph measures like betweenness centrality and closeness centrality of vertices are not considered, because they are based on shortest paths in the graph which is less related to SPL type errors than the number of direct connections to other features (therefore the degree). Only edge betweenness centrality is considered, because there is no measure like the degree regarding the importance of an edge. Finding a correlation between characteristics of an edge in the graph and its likeliness for type errors is especially interesting, as the edge represents a connection between two features and those connections are prerequisites for SPL type errors. Positive correlation between these graph measures and SPL type errors is assumed.

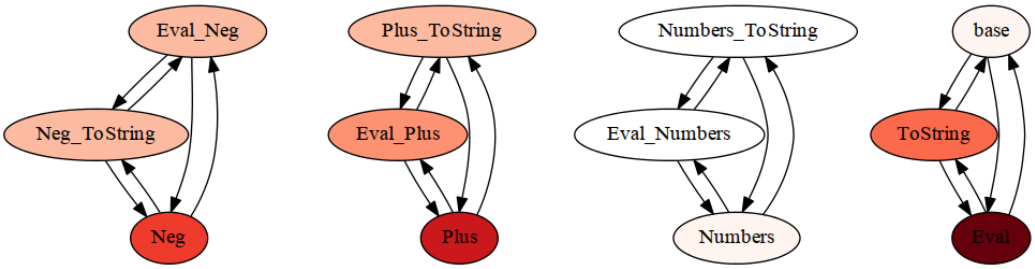

Figure 5: G_{intros} for EPL (red color denotes number of type errors, white = no SPL type errors, dark red = many SPL type errors)

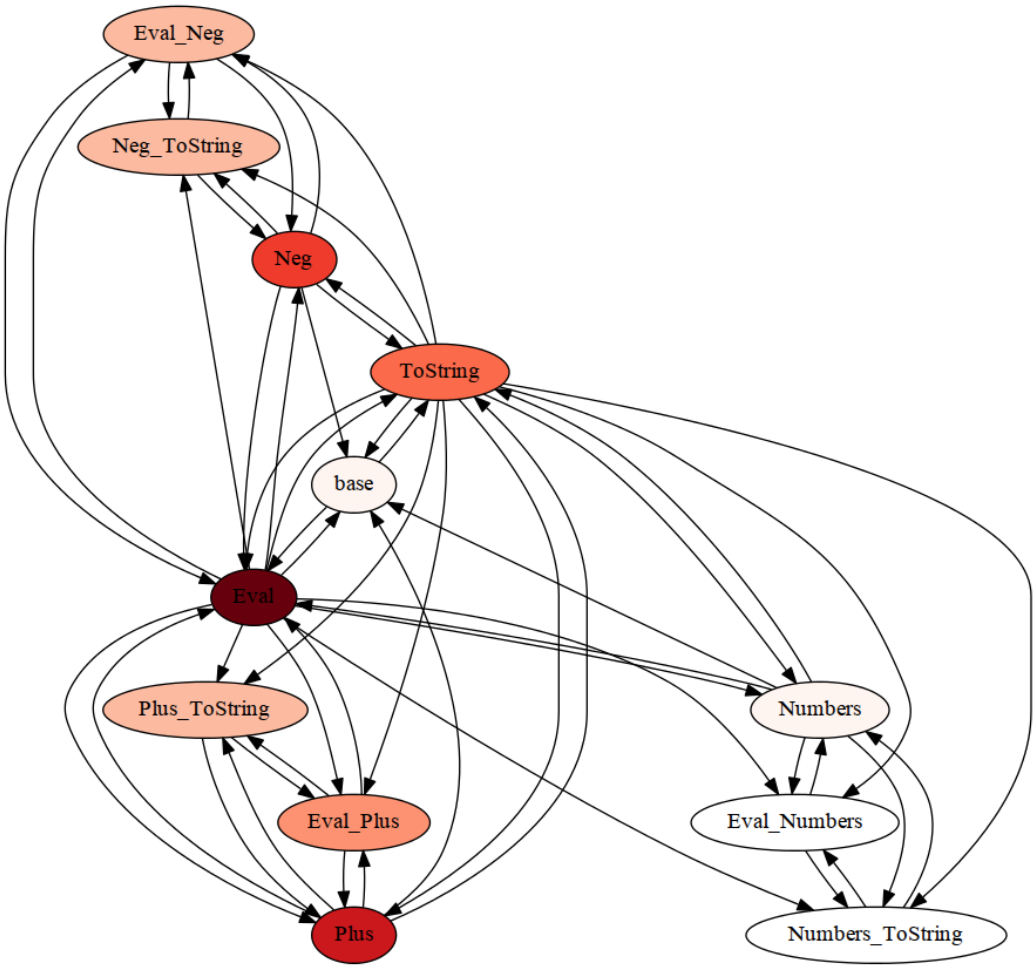

Figure 6: G_{PFI} for EPL (red color denotes number of type errors, white = no SPL type errors, dark red = many SPL type errors)

3.2.3.2 Degree Centrality

For every graph, the degree of each vertex can be computed. For G_{PFI} the degree of a vertex corresponds to the number of features with which a feature F that is represented by a vertex can possibly interact. As SPL type errors are unwanted feature interactions, it is possible that features that have more possibilities to interact with other features are more likely to be involved in SPL type errors.

$$deg(F) = |\{(u, v) \in E | u = F \vee v = F\}|$$

Instead of only taking into account how many features a feature F can possibly interact with, the weight of the edges can be considered as well. The weight of an edge depends on the number of references and introduces between the two features. We define deg_w:

$$deg_w(F) = \sum_{(u,v) \in E \wedge (u=F \vee v=F)} w$$

3.2.3.3 Degree Centrality with Dependency-Weight

The information of possible interactions stored in the graphs can be further enhanced with information based on the feature model in order to combine two different approaches code-based and variability-based. Hence a mapping is defined for every edge to one of the sets *always*, *maybe* or *never*, depending on the dependency between the two features in question.

The weighting is the same as for "Dependencies to other Features" (see Chapter 3.2.1): Always-dependencies are weighted lowest (1), never-dependencies are weighted medium (5) and maybe-dependencies, which are accountable for most SPL type errors are counted highest (10).
As with the edge-weight, for each vertex (corresponding to a feature F) it is now counted how many edges of the different kinds are attached to that vertex and then weighted accordingly. So we define deg_{dep} as:

$$\begin{aligned} deg_{dep}(F) = &|\{(u, v) \in E \wedge u = F \wedge (u, v) \in always\}| \\ &+ |\{(u, v) \in E \wedge u = F \wedge (u, v) \in never\}| * 5 \\ &+ |\{(u, v) \in E \wedge u = F \wedge (u, v) \in maybe\}| * 10 \end{aligned}$$

Again, instead of only taking into account the type of dependency an edge represents, it is now also possible to consider how strong the tie between the two features in question is. The weight of the edge that corresponds to the number of references from one feature to another or the number of code elements that both features introduce is further weighted (multiplied) by the

weight of the dependency between the two features. Therefore the definition of $deg_{w,dep}$ is:

$$\begin{aligned} deg_{w,dep}(F) = &\sum_{(u,v)\in E \wedge (u=F \vee v=F \wedge (u,v)\in always)} w \\ &+ \sum_{(u,v)\in E \wedge (u=F \vee v=F \wedge (u,v)\in never)} w * 5 \\ &+ \sum_{(u,v)\in E \wedge (u=F \vee v=F \wedge (u,v)\in maybe)} w * 10 \end{aligned}$$

3.2.3.4 Betweenness Centrality for Edges

As SPL type errors are feature interactions, it is particularly interesting, if there is a measure that can be used to indicate for a pair of features if they are likely to take part in SPL type errors together. This is why betweenness centrality for edges, which is a measure for the relative importance of edges in a graph, is considered in this thesis:

$$c_B(e) = \sum_{s,t\in V} \frac{\sigma(s,t|e)}{\sigma(s,t)}$$

where $\sigma(s,t)$ is the number of shortest paths between $s \in V$ and $t \in V$ and $\sigma(s,t|e)$ is the number of shortest paths passing through edge e [12].

3.2.4 Code Fragmentation

Looking at a collaboration diagram (see Chapter 2.1, Figure 1) it is clear that features and classes split code of a product line in regard to different abstractions. The more fragmented a feature or respectively a class is, the harder it is to comprehend. Hence more fragmentation may lead to more type errors.

For each class and respectively for each feature the number of roles in the collaboration diagram is counted (see Figure 7). For a feature F this corresponds to the number of classes which it introduces or refines and for a class C to the number of features that contribute in it and therefore crosscut it:

$$frag_f(F) = |Roles\ of\ F|$$
$$frag_c(C) = |Roles\ of\ C|$$

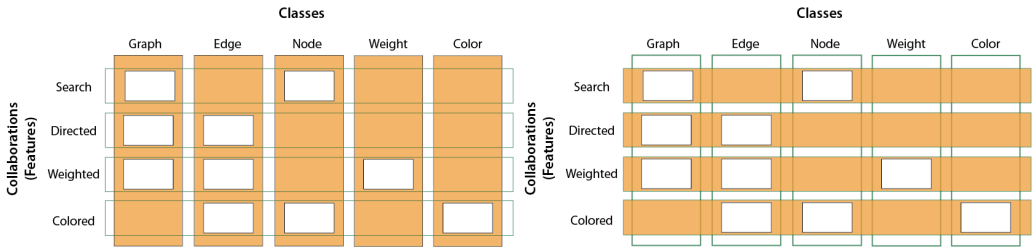

Figure 7: Different views on the collaboration diagram which illustrate fragmentation of classes (left) and fragmentation of features (right) (excerpt of the collaboration diagram of GPL)

4 Evaluation

The relation between SPL type errors and the measures that were introduced in the previous Chapter is examined in the following. 28 subject product lines are evaluated.

4.1 Subject Product Lines

In this thesis 28 Product lines are considered. They are all based on Java, being developed using AHEAD and FeatureHouse. AHEAD *(Algebraic Hierarchical Equations for Application Design)* is a feature-oriented methodology for creating software product lines [9] and FeatureHouse is a framework for software composition [4].

The subject product lines are of different sizes. Because most measures that were developed or used in this thesis work on features also the classification of the size of a product line was based on the number of features. Statistical significance is influenced by the size of the vectors that are correlated, insofar the classification of size is important for the evaluation of the correlation results. There are three big case studies, each of them containing more than 40 features. Then there are three product lines of medium size and all other case studies are small product lines which contain less than 20 features. Of those there are eight very small product lines consisting of less than ten features. See Tables 1 and 2.

Features	Size	Subject Product Lines
>40	big	BerkeleyDB, MobileMedia8, Violet
20-39	medium	GPL, GUIDSL, TankWar
10-19	small	EPL, Notepad-Robinson, PokerSPL, Prop4J, Vistex, Chat-Benduhn, Chat-Harbich, Chat-Ludwig, Chat-Lueddecke, Chat-Meye, Chat-Otte, Chat-Urbanek, Chat-Weigelt, Chat-Zaske
<10	very small	BankAccountTP, Prevayler, Sudoku, UnionFind, Chat-Beck, Chat-Huber, Chat-Schulze, Chat-Vielsmaier

Table 2: Sizes of the subject product lines in comparison

The case studies contain systems that were developed from scratch as software product lines while others were refactored from existing software systems (see Table 1).

All of the subject product lines are either student projects (13 Chat-Systems, Notepad-Robinson, Vistex) or were developed/refactored at universities during research projects.

Name	LOC	#C	#F	#SPL TE	#FL TE	Development process
BankAccountTP	132	3	8	5	13	S
BerkeleyDB	45000	283	99	198	273	S
Chat-Beck	423	6	8	11	74	S
Chat-Benduhn	759	14	11	4	32	S
Chat-Harbich	630	20	14	1	16	S
Chat-Huber	1270	24	5	14	88	S
Chat-Ludwig	975	9	10	16	19	S
Chat-Lueddecke	912	14	11	20	65	S
Chat-Meye	812	9	11	5	42	S
Chat-Otte	655	9	10	13	27	S
Chat-Schulze	923	11	9	79	19	S
Chat-Urbanek	963	31	11	4	45	S
Chat-Vielsmaier	957	21	7	54	55	S
Chat-Weigelt	877	11	10	39	39	S
Chat-Zaske	3093	41	16	31	216	S
EPL	111	12	12	42	0	S
GPL	1940	16	20	16	55	S
GUIDSL	11529	144	26	59	238	S
MobileMedia8	4189	51	45	142	1075	R
Notepad-Robinson	800	9	10	3	0	R
PokerSPL	283	8	10	1	18	R
Prevayler	5268	138	6	15	63	R
Prop4J	1531	14	14	490	443	R
Sudoku	1422	26	7	17	0	R
TankWar	4845	22	30	66	560	S
UnionFind	210	4	8	19	46	S
Violet	7194	67	88	117	60	R
Vistex	1608	8	16	12	31	S

Table 1: Overview of the subject product lines (LOC: Lines of code, #F: Number of concrete features containing Java-code, #C: Number of classes, #SPL TE: Number of software product line specific type errors, #FL TE: Number of feature-local type errors, S: Developed from scratch as product line, R: Refactored)

The subject systems belong to different domains. There are three editors of different kinds: Notepad-Robinson, Violet and Vistex. Vistex and GPL both deal with graphs. Then there are three games, namely TankWar, Sudoku and PokerSPL. Prevayler and BerkeleyDB are databases that can be embedded in applications. EPL, Prop4J and UnionFind implement algorithms in different variants. Then there is a product line for managing a bank account, BankAccountTP, and a tool for specifying AHEAD models and verifying feature selections, namely GUIDSL.

Further descriptions of the subject product lines are available in Appendix A.1.

4.2 Statistical Analysis Method

As SPL type errors are not normally distributed for most of the subject product lines (p-values of Shapiro-Wilk test smaller than 0.05; see Table 14 in Appendix A.2), Spearman's rank correlation coefficient was used to determine if a measure relates to type errors and therefore to find out if a measure is suitable for indicating error-proneness. To ensure comparability between results, Spearman's rank correlation coefficient was also used for those case studies with normally distributed errors. The p-values of Spearman's rank correlation coefficient, which show statistical significance, are displayed using the following star notation:

$0.05 \geq$ p-value > 0.01:	*
$0.01 \geq$ p-value > 0.001:	**
$0.001 \geq$ p-value:	***

Correlation results in this thesis are shown as tables, correlation matrices or line charts. Despite the data being discrete, line charts are used for better readability in comparison to charts that only use marks.

In the next Chapters only those results that are discussed are shown. All correlation results are available in Appendix A.3.

4.3 Correlation between Variability Measures and Type Errors

The next two Chapters will take a closer look on the relationship between SPL type errors and the variability-measures that were introduced in Chapter 3.2.1.

4.3.1 Dependencies to other Features

Weight of dependencies is a measure for the relations a feature has to all other features. It was introduced in Chapter 3.2.1.1 and is now set in correlation to the number of SPL type errors that a feature is involved in. In Table 3 and Figure 8 it is shown that only 6 out of 28 subject product lines correlate significantly, half of them strongly negative (Chat-Benduhn, Chat-Huber and Vistex) and the other half positive (MobileMedia8, Notepad-Robinson and Violet). The three that correlate positively have a rather simple and more symmetrical feature diagram while the feature diagram of Chat-Benduhn and Vistex are more asymmetrical. Those differences are illustrated in Figures 9 and 10 with the feature models of Chat-Benduhn and NotepadRobinson. Chat-Huber is also more symmetrical but uses constraints, which of course affect the dependencies between the involved features. With almost symmetric feature models and no constraints the weight of dependencies of most features is the same and only different for those few features outside the symmetry (often base-features). Therefore, if the features outside the symmetry have less SPL type errors, the correlation is positive.

Dependencies between all features are considered in this measure no mat-

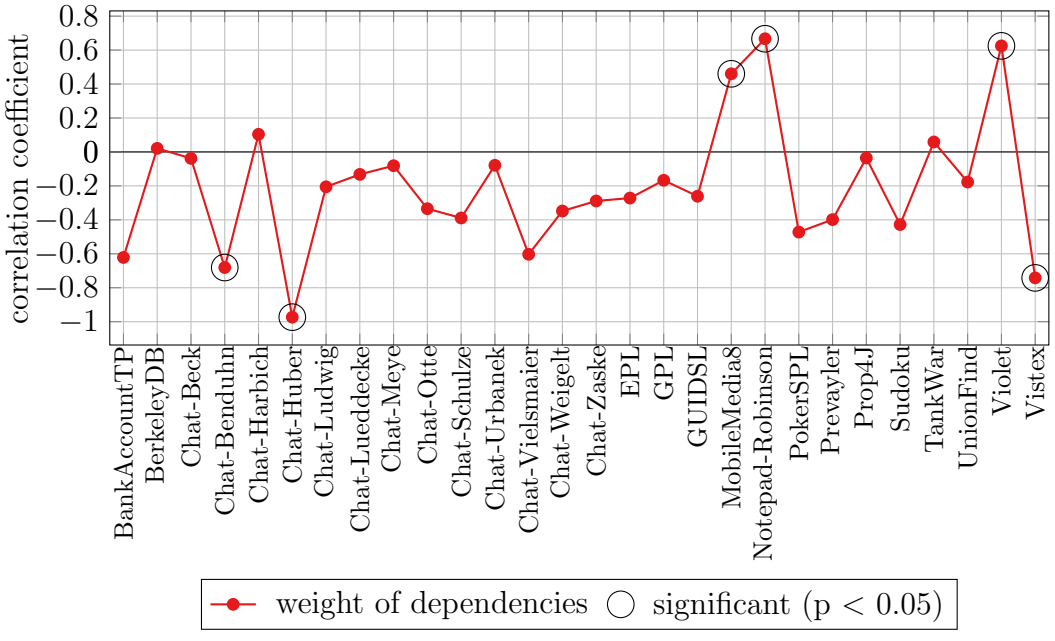

Figure 8: Correlation between SPL type errors and the weighted dependencies of a feature

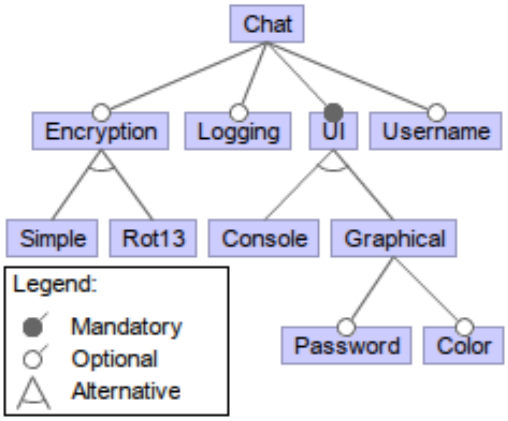

Figure 9: Feature model of Chat-Benduhn

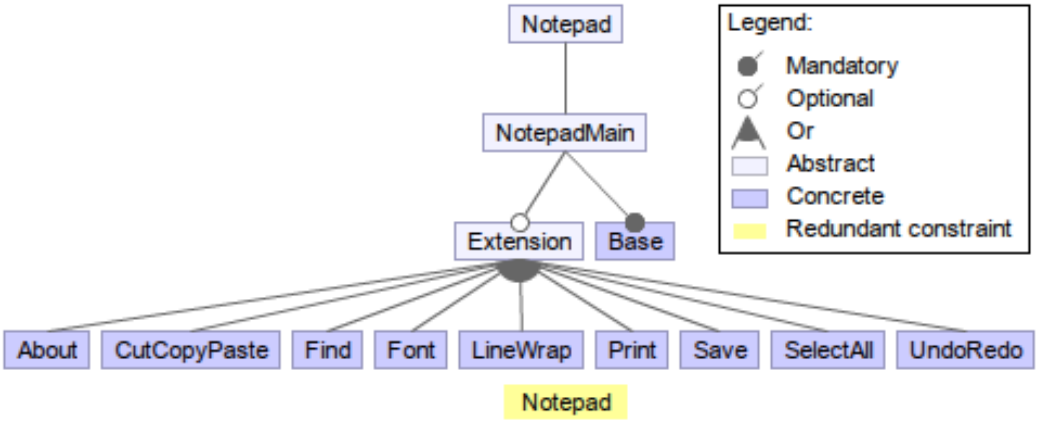

Figure 10: Feature model of NotepadRobinson (all concrete features except Base have the same weight of dependencies)

Chat-Benduhn	Chat-Huber	MobileMedia8	Notepad-Robinson	Violet	Vistex
-0.681*	**-0.973****	0.460**	**0.667***	**0.625*****	**-0.742*****

Table 3: Correlation between SPL type errors and the weighted dependencies of a feature (only significant results shown)

ter if there are actual relations between those features on code-level. The results of the correlation to SPL type errors shows that this is not enough information to find features that are likely to contain type errors.

4.3.2 Number of Constraints

11 out of 28 subject product lines use propositional constraints in their feature diagram. The number of constraints a feature takes part in (see Chapter 3.2.1.2) was correlated to the number of SPL type errors of that feature in order to find out whether the additional complexity of constraints makes an impact on the number of type errors. On the one hand, the additional complexity of constraints could lead to more SPL type errors, on the other hand, code-level relations can be reflected in the feature-diagram using constraints, hence constraints can be used to eliminate SPL type errors. When considering all constraints a feature takes part in, only two correlate significantly, EPL (0.675*, 6 constraints) and Violet (0.321**, 22 constraints) (see Figure 11).

As only two case studies correlate significantly, both assumptions about how constraints could affect features in regard to the number of SPL type errors were not confirmed. Therefore this measure is also not suitable as predictor for type errors.

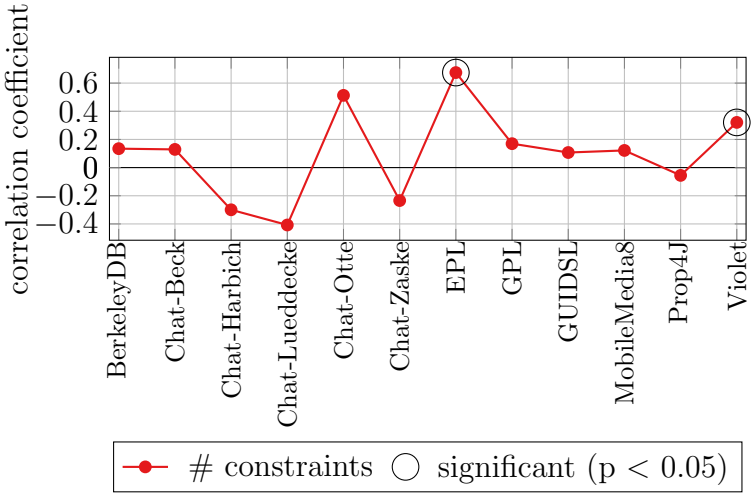

Figure 11: Correlation between SPL type errors and *#constraints*

4.4 Correlation between Cohesion and Coupling and Type Errors

The four measures IFD, EFD, CBO and CBF that were described in Chapter 3.2.2 are examined in regard to their relation to SPL type errors in the next Chapters.

4.4.1 Internal-ratio Feature Dependency

Only the correlation to SPL type errors is examined as IFD measures cohesion for features (see Chapter 3.2.2.1). In Figure 12 the correlation is visualised. Only six subject product lines show a significant correlation be-

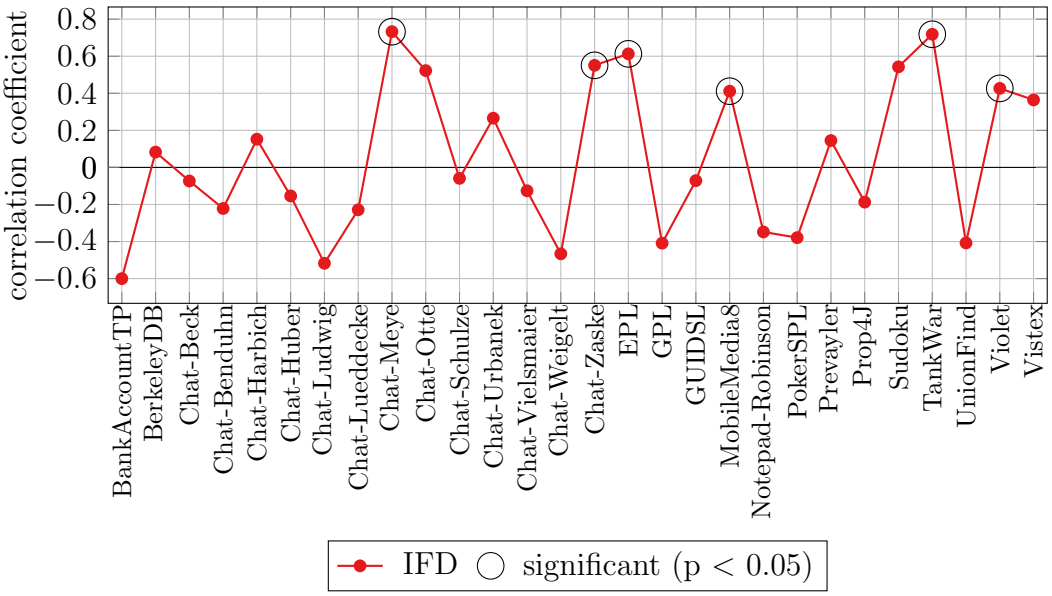

Figure 12: Correlation between SPL type errors and IFD

tween IFD and SPL type errors. All of them correlate positively, which was not expected, as high cohesion usually indicates well-structuredness. And well-structured features typically contain less type errors than unstructured ones. (see Table 4).

4.4.2 External-ratio Feature Dependency

Only five of the subject product lines show a significant correlation between SPL type errors and EFD, which was introduced in Chapter 3.2.2.2. Two of them show a strong or very strong negative correlation and three a weak

Chat-Meye	Chat-Zaske	EPL	MobileMedia8	TankWar	Violet
0.732*	0.551*	0.613*	0.411**	0.718***	0.426***

Table 4: Correlation between SPL type errors and IFD (only significant correlations)

or moderate positive correlation (see Table 5 or Figure 13 for a comparision to the other measures for coupling). Those that correlate positively (BerkeleyDB, MobileMedia8 and Violet) are all big product lines with more than 40 features (that contain java code) each. The negative correlating ones are small product lines, Prevalyer with six features and Chat-Benduhn with eleven features. Hence no clear connection between SPL type errors and EFD exists for the subject systems in this thesis.

BerkeleyDB	Chat-Benduhn	MobileMedia8	Prevayler	Violet
0.241*	-0.621*	0.458**	-0.899*	0.504***

Table 5: Correlation between SPL type errors and EFD (only significant correlations)

All values for correlation between EFD and SPL type errors are visualized in Figure 13.

4.4.3 Coupling between Object Classes

CBO is a measure for coupling of classes (see Chapter 3.2.2.3) therefore it was correlated to the number of type errors of classes. Because CBO is not related to software product lines, feature-local type errors and both kinds of type errors together ("all") were considered as well. All results for correlation between CBO and SPL type errors are visualized in Figure 13 (as well as the correlation results for the other measures for coupling). In addition, the results that correlate significantly are shown in Table 6. Sudoku has no feature-local type errors. For most of the subject product lines the correlation is either significant for all three (SPL, FL and all kinds of type errors) or not significant at all. This leads to the conclusion that the likeliness for a highly coupled class to contain feature-local type errors is the same than for SPL

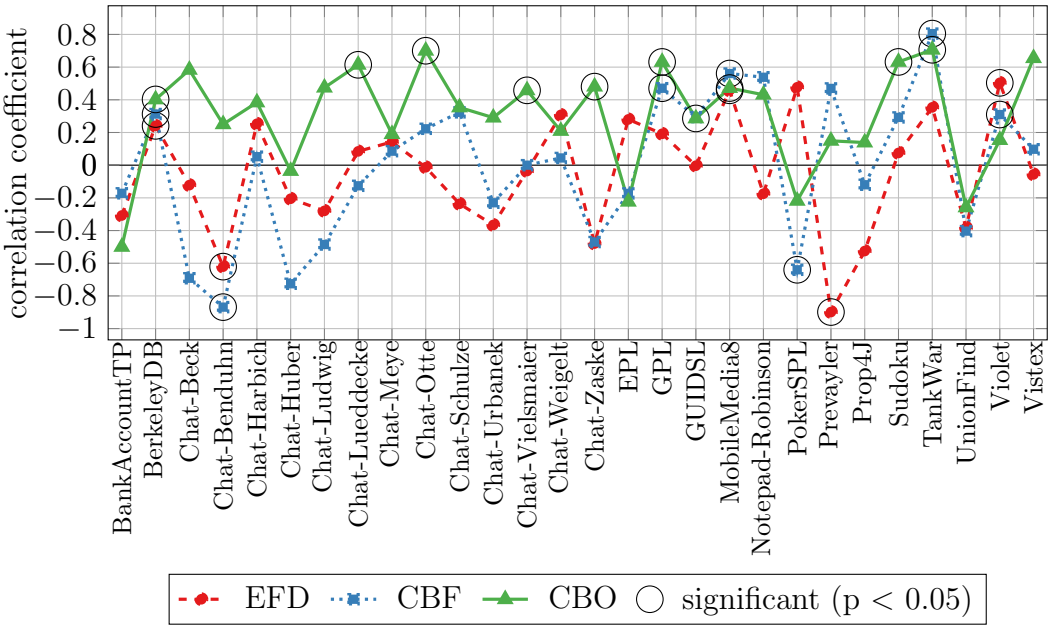

Figure 13: Correlation between SPL type errors and EFD, CBF and CBO

type errors. With an exception for GUIDSL the correlations to SPL type errors that are significant are all medium to strong positive. There does not seem to be a coherence between the significance of the correlation and the domains or sizes of the product lines.

But most of the subject systems that do not correlate significantly have balanced CBO values, therefore not having one (or several) central classes that communicate with a lot of other classes and are involved in most functionalities.

4.4.4 Coupling between Features

CBF measures Coupling between Features (see Chapter 3.2.2.4) and is similar to EFD. And like EFD it also shows no clear negative or positive correlation to SPL type errors for those subject systems that correlate significantly (seven in total, see Table 7 or Figure 13).

Except for GUIDSL, all big and medium sized subject product lines correlate significantly and positively to SPL type errors. And except for Violet, they also correlate positively and significantly for feature-local type errors. The two product lines that correlate significantly and negatively (PokerSPL and Chat-Benduhn) to SPL type errors are small ones.

Because EFD shows as well positive correlation for big product lines and

Name	SPL type errors	FL type errors	All type errors
BerkeleyDB	**0.402** ***	**0.484** ***	**0.510** ***
Chat-Lueddecke	**0.614** *	**0.634** *	**0.670** **
Chat-Otte	**0.700** *	0.449	0.651
Chat-Urbanek	0.291	**0.440** *	**0.440** *
Chat-Vielsmaier	**0.458** *	**0.678** ***	**0.613** **
Chat-Zaske	**0.481** **	0.267	0.271
GPL	**0.632** **	**0.665** **	**0.658** **
GUIDSL	**0.286** ***	**0.349** ***	**0.463** ***
MobileMedia8	**0.472** ***	**0.380** **	**0.443** **
Sudoku	**0.632** ***	-	**0.632** ***
TankWar	**0.706** ***	0.196	**0.495** *

Table 6: Correlation between type errors and CBO (only systems with at least one significant correlation shown)

Name	SPL type errors	FL type errors	All type errors
BerkeleyDB	**0.313** **	**0.304** **	**0.351** ***
Chat-Benduhn	**-0.868** ***	**-0.723** *	**-0.758** **
GPL	**0.471** *	**0.762** ***	**0.640** **
GUIDSL	0.294	0.319	**0.477** *
MobileMedia8	**0.561** ***	**0.640** ***	**0.672** ***
PokerSPL	**-0.640** *	0.250	-0.101
Prevayler	0.470	**0.877** *	**0.926** **
TankWar	**0.805** ***	**0.444** *	**0.845** ***
Violet	**0.310** **	**-0.633** ***	**-0.211** *

Table 7: Correlation between type errors and CBF (only systems with at least one significant correlation shown)

negative correlation for small product lines, there could be a connection. But as both correlate significantly only for few of the subject product lines no clear statement can be made.

4.5 Correlation between Possible Feature Interactions and Type Errors

In Chapter 3.2.3 measures on the graphs G_{intros}, G_{refs} and G_{PFI} are introduced. The relation between those measures and SPL type errors is examined in the next Chapters with a focus on G_{PFI}, because this graph contains all relations that can lead to SPL type errors, while the others (G_{intros} and G_{refs}) only capture parts of those relations.

4.5.1 Degree Centrality

Figure 14 and Table 8 show the correlation between SPL type errors and the degree ($deg(F)$) of each feature F for graphs G_{intros}, G_{refs} and G_{PFI}. The degree and the graphs were introduced in Chapters 3.2.3.1 and 3.2.3.2. Except for three small or very small product lines, namely BankAccountTP,

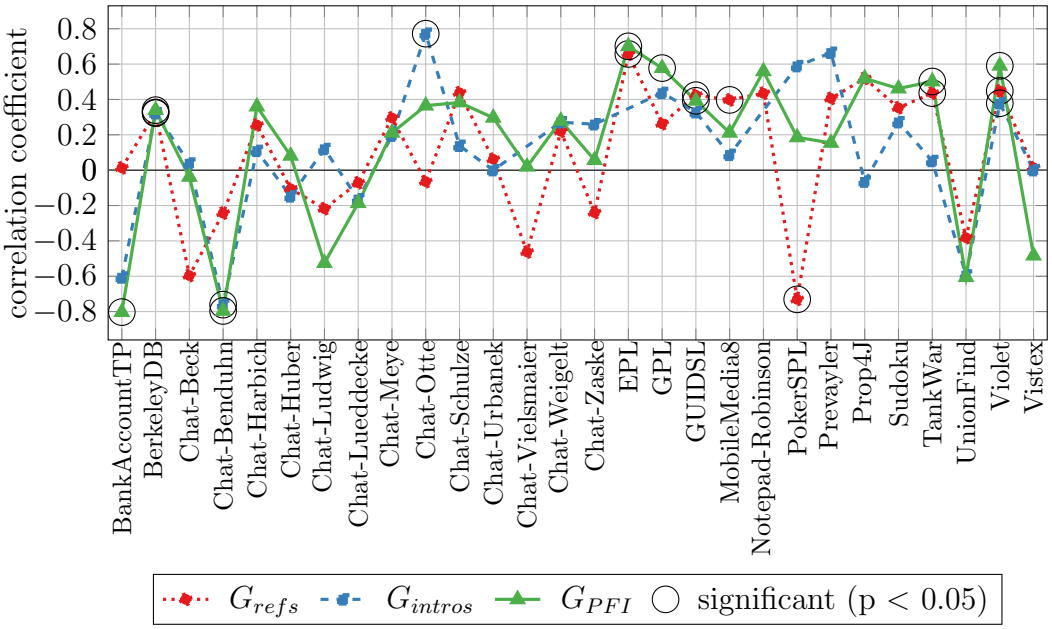

Figure 14: Correlation between number SPL type errors and degree (deg) in the graphs G_{intros}, G_{refs} and G_{PFI}

Chat-Benduhn and PokerSPL, all significant correlations are positive. For G_{refs} more values are significant than for G_{intros}, which was to be expected, because more SPL type errors of the subject product lines are caused by references not reflected in the feature model than by missing information in one feature that is present in another feature which introduces the same code element (see Chapter 3.1).

Five out of eight systems with significant positive correlations for G_{PFI} show stronger correlation between SPL type errors and $deg(F)$ for G_{PFI} than for G_{intros} and G_{refs}. Those systems are BerekeleyDB, EPL, GPL, TankWar and Violet, except for EPL all medium or big product lines. This leads to the conclusion that for certain systems G_{PFI} is more suitable for detecting features that are likely to be involved in SPL type errors while for some others the additional information about introduces seems to weaken the expressiveness of the graph. As mentioned before, this and the next chapter focus on G_{PFI}, because it contains all information that can lead to SPL type errors.

Comparing the correlation between SPL type errors and *deg* as well as SPL type errors and deg_w (see Chapter 3.2.3.2) shows that taking into account the number of references and introduces two features share brings no clear advantage over only taking into account the number of features a feature can possibly interact with. Of those systems that correlate significantly and positively three show stronger correlation for deg_w (Berkeley DB, Notepad-Robinson and TankWar) and four stronger correlation for *deg* (EPL, GPL, GUIDSL and Violet). The comparison is shown in Figure 15 and Table 9.

4.5.2 Degree Centrality with Dependency-Weight

A comparison between the correlation of SPL type errors and degree as well as degree with weighted dependencies and their variations is shown in figures 16 and 17 with the numeric values in Table 9. Deg_{dep} is a weighting of the degree based on the mapping of edges of G_{PFI} to the sets *never*, *maybe* and *always* (see Chapter 3.2.3.3).

Figure 16 shows the differences between *dep*, where the relations between all features in the feature model are considered (see Chapters 3.2.1.1 and 4.3.1), and deg_{dep}, where only those relations are considered that can actually lead to type errors because there are elements referenced or shared between the features in question (see Chapter 3.2.3.3). *Dep* shows significant correlation to SPL type errors for six of the subject systems, half of them positive and the other half negative. For *deg* eight subject product lines correlate significantly and only two of them negatively (see Table 9). Therefore it seems that the information provided by the feature model alone is less

	G_{intros}	G_{refs}	G_{PFI}
BankAccountTP	-0.607	0.013	**-0.803** *
BerkeleyDB	**0.323** **	**0.326** ***	**0.339** ***
Chat-Benduhn	**-0.761** **	-0.243	**-0.796** **
Chat-Otte	**0.772** **	-0.065	0.365
EPL	-	**0.657** *	**0.701** *
GPL	0.437	0.263	**0.577** **
GUIDSL	0.328	**0.424** *	**0.393** *
MobileMedia8	0.087	**0.397** **	0.211
PokerSPL	0.588	**-0.731** *	0.186
TankWar	0.052	**0.436** *	**0.502** **
Violet	**0.379** ***	**0.448** ***	**0.591** ***

Table 8: Correlation between number SPL type errors and degree (deg) in the graphs G_{intros}, G_{refs} and G_{PFI} (only product lines with at least one significant value)

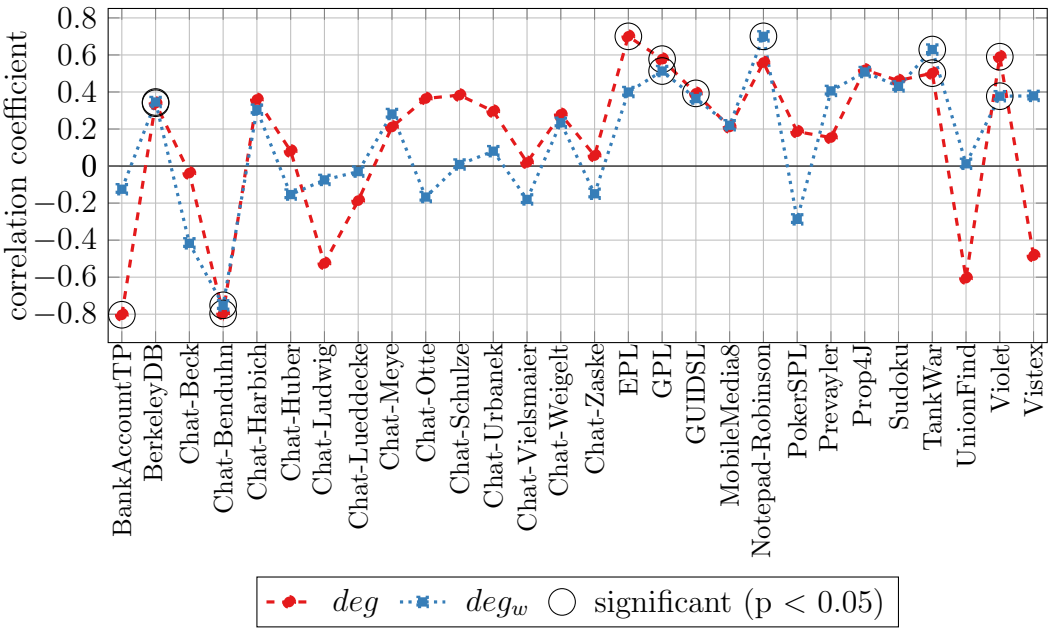

Figure 15: Correlation between number of SPL type errors and deg as well as deg_w in graph G_{PFI}

	deg	deg_w	deg_{dep}	$deg_{w,dep}$
BankAccountTP	**-0.803** *	-0.125	-0.478	-0.025
BerkeleyDB	**0.339** ***	**0.346** ***	**0.318** **	**0.351** ***
Chat-Benduhn	**-0.796** **	**-0.752** **	**-0.665** *	**-0.618** *
EPL	**0.701** *	0.400	**0.968** ***	**0.775** **
GPL	**0.577** **	**0.513** *	0.091	**0.569** **
GUIDSL	**0.393** *	0.365	0.276	**0.443** *
MobileMedia8	0.211	0.220	**0.308** *	**0.298** *
Notepad-Robinson	0.559	**0.701** *	0.435	**0.696** *
Prop4J	0.519	0.508	**0.659** *	**0.705** **
Sudoku	0.462	0.430	**0.905** **	0.599
TankWar	**0.502** **	**0.629** ***	**0.462** *	**0.653** ***
UnionFind	-0.605	0.012	**-0.877** **	-0.135
Violet	**0.591** ***	**0.377** ***	**0.604** ***	**0.486** ***

Table 9: Correlation between number of SPL type errors and different variations of degree in graph G_{PFI} (only subject systems with at least one significant correlation value)

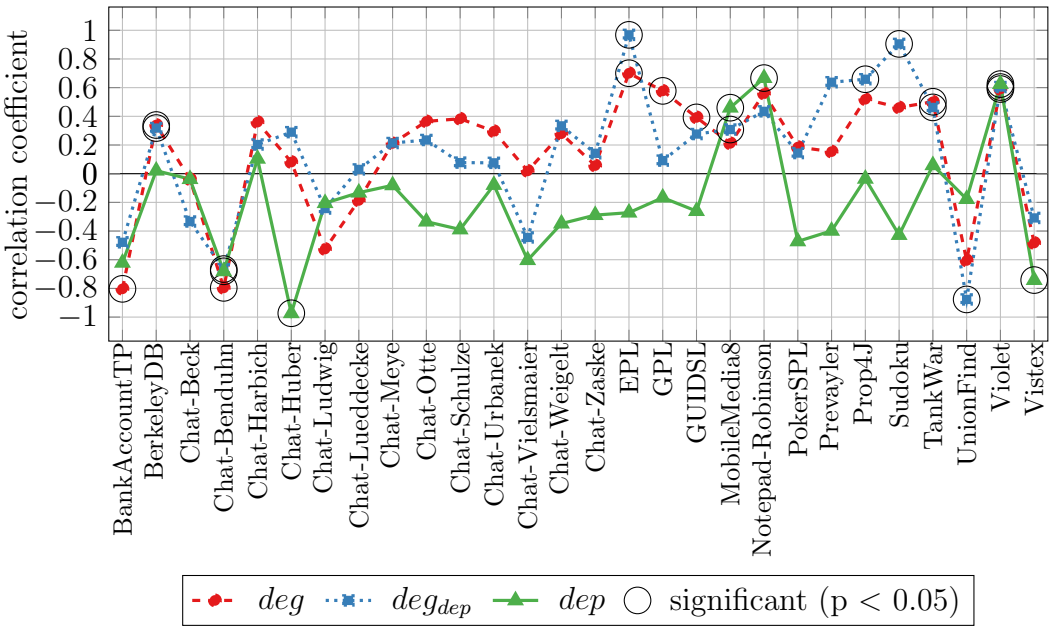

Figure 16: Correlation between SPL type errors and deg, deg_{dep} and dep

effective for finding features which are likely to contain SPL type errors than the combination of the information about possible feature interactions on code-level and the information provided by the feature model.

Figure 16 also shows the differences between deg and deg_{dep} for G_{PFI}. The three subject systems that correlate negatively and significantly to SPL type errors for at least one of the two measures are all very small or small (BankAccountTP, Chat-Benduhn and UnionFind). But the medium to big product lines all show positive and significant correlation for at least one of the two measures. Comparing deg and deg_{dep} shows no definite improvement, as only five of the subject systems correlate stronger for deg_{dep} than for deg and four of the systems the other way around (see Table 9).

When comparing deg_w and $deg_{w,dep}$ a tendency to stronger correlations is shown for $deg_{w,dep}$ (see Figure 17). Again, small or very small systems tend to show no significant correlation and if they do, it is mostly positive (EPL, Notepad-Robinson, Prop4J) but also negative for one of the systems (Chat-Benduhn). The medium to big systems that correlate significantly and positively show a stronger correlation for $deg_{w,dep}$ than for deg except for BerkeleyDB and Notepad-Robinson.

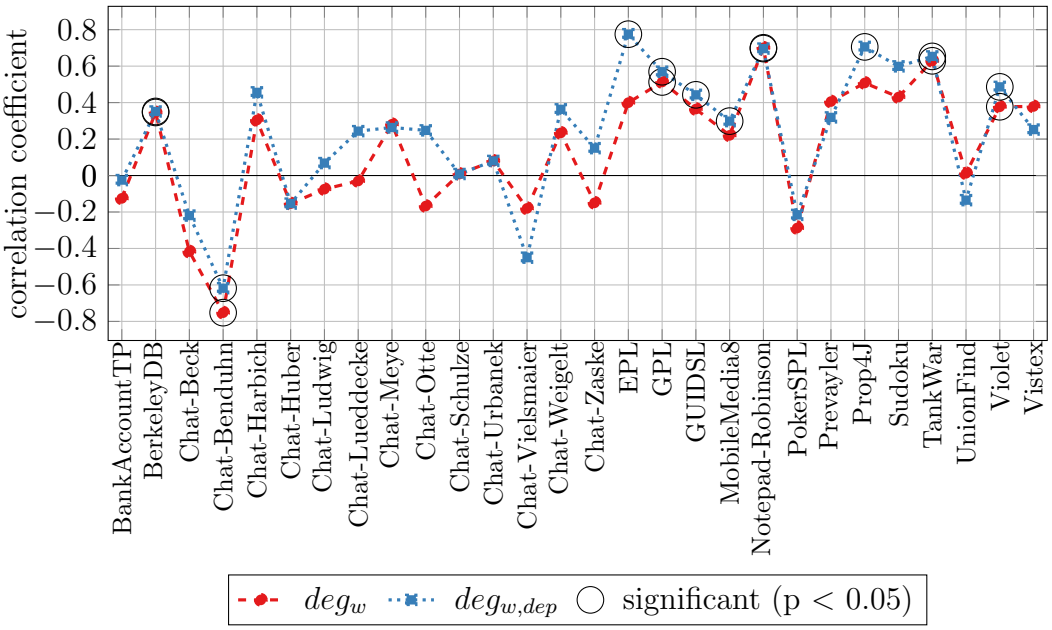

Figure 17: Correlation between SPL type errors and deg_w as well as $deg_{w,dep}$

Finally by comparing deg_{dep} and $deg_{w,dep}$ again only a small improvement by taking into account the number of references and introduces is shown for

the subject product lines. Of those systems that correlate significantly and positively four show a stronger correlation for deg_{dep} than for $deg_{w,dep}$ and six a stronger correlation for $deg_{w,dep}$ than for deg_{dep} (see Table 9). It has to be mentioned that all medium and big subject product lines considered in this thesis correlate significantly and positively for $deg_{w,dep}$, while for the other measures always at least one of the medium and big subject systems does not correlate significantly. Therefore $deg_{w,dep}$ works slightly better for predicting SPL type errors than the other degree-based measures.

4.5.3 Betweenness Centrality for Edges

For each subject system the number of SPL type errors for its edges in G_{SPLTE} and the edge-betweenness-value (see Chapter 3.2.3.4) in the graphs G_{intros}, G_{refs} and G_{PFI} is correlated. In these graphs an edge represents a combination of two features which introduce the same code element and/or where one of the features references the other, depending on the graph that is considered. Finding characteristics of pairs of features that are likely to take part in SPL type errors together is as desirable as finding characteristics of features that are likely to be error-prone.

Of the 28 subject product lines that are considered in this thesis, 20 show

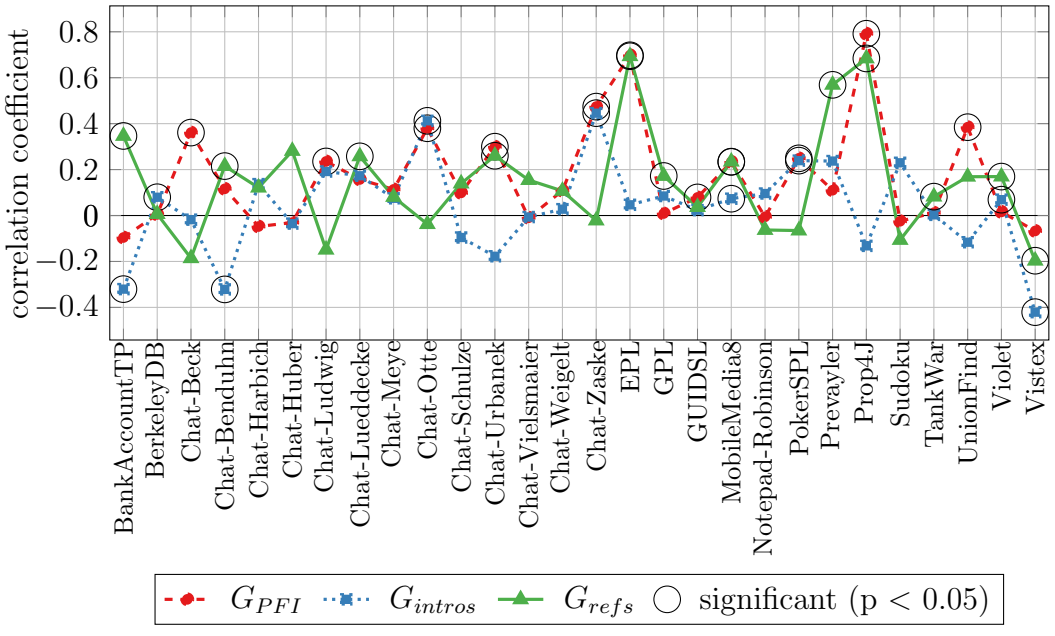

Figure 18: Correlation between SPL type errors and edge betweenness centrality (c_B) for graphs G_{intros}, G_{refs} and G_{PFI}

	G_{PFI}	G_{intros}	G_{refs}
BankAccountTP	-0.096	**-0.321** *	**0.346** **
BerkeleyDB	0.005	**0.080** ***	0.006
Chat-Beck	**0.361** **	-0.018	-0.186
Chat-Benduhn	0.116	**-0.322** ***	**0.215** *
Chat-Ludwig	**0.235** *	0.192	-0.149
Chat-Lueddecke	0.157	0.173	**0.258** **
Chat-Otte	**0.380** ***	**0.412** ***	-0.038
Chat-Urbanek	**0.300** **	-0.177	**0.260** **
Chat-Zaske	**0.474** ***	**0.446** ***	-0.022
EPL	**0.698** ***	0.047	**0.694** ***
GPL	0.009	0.085	**0.172** ***
GUIDSL	**0.079** *	0.023	0.037
MobileMedia8	**0.234** ***	**0.073** **	**0.235** ***
PokerSPL	**0.249** *	**0.239** *	-0.066
Prevayler	0.110	0.237	**0.569** **
Prop4J	**0.792** ***	-0.132	**0.684** ***
TankWar	0.012	0.003	**0.082** *
UnionFind	**0.385** **	-0.116	0.169
Violet	0.016	**0.069** ***	**0.169** ***
Vistex	-0.067	**-0.421** ***	**-0.197** **

Table 10: Correlation between SPL type errors and edge betweenness centrality (c_B) in G_{PFI}, G_{refs} and G_{intros} (only subject systems with at least one significant correlation)

at least for one of the three graphs G_{intros}, G_{refs} and G_{PFI} a significant correlation between SPL type errors of edges and edge betweenness centrality (see Table 10 and Figure 18). Of those only three small to very small systems show significant negative correlation (namely BankAccountTP, Chat-Benduhn and Vistex), all other significant correlation values are positive.

9 out of the 20 systems that show a significant correlation for edge betweenness centrality for at least one of the graphs correlate strongest in G_{PFI} (Chat-Beck, Chat-Luwdig, Chat-Urbanek, Chat-Zaske, EPL, GUIDSL, PokerSPL, Prop4J and UnionFind) and 8 correlate strongest for G_{refs} (BankAccountTP, Chat-Benduhn, Chat-Lueddecke, GPL, MM8, Prevayler, TankWar and Violet). Only two show the strongest correlation for edge betweenness in G_{intros}.

For edge betweenness centrality and degree centrality the subject systems

show different results for G_{intros}, G_{refs} and G_{PFI} in regard to which of the three graphs shows strongest correlation between the measure and SPL type errors for the subject system. Overall, G_{intros} shows the weakest correlations of all three graphs and G_{PFI} shows slightly better results than G_{refs}. As mentioned before, as most SPL type errors have connection to references this result supports the intuition.

4.6 Correlation between Code Fragmentation and Type Errors

In the next Chapters the two code fragmentation measures that were introduced in Chapter 3.2.4 are evaluated.

4.6.1 Fragmentation of Classes

For each subject system the number of features that crosscut a class (see Chapter 3.2.4) and the number of the type errors of the class (SPL as well as FL type errors) were correlated. Every class of EPL is crosscut by exactly three features, which means that there is no correlation to SPL type

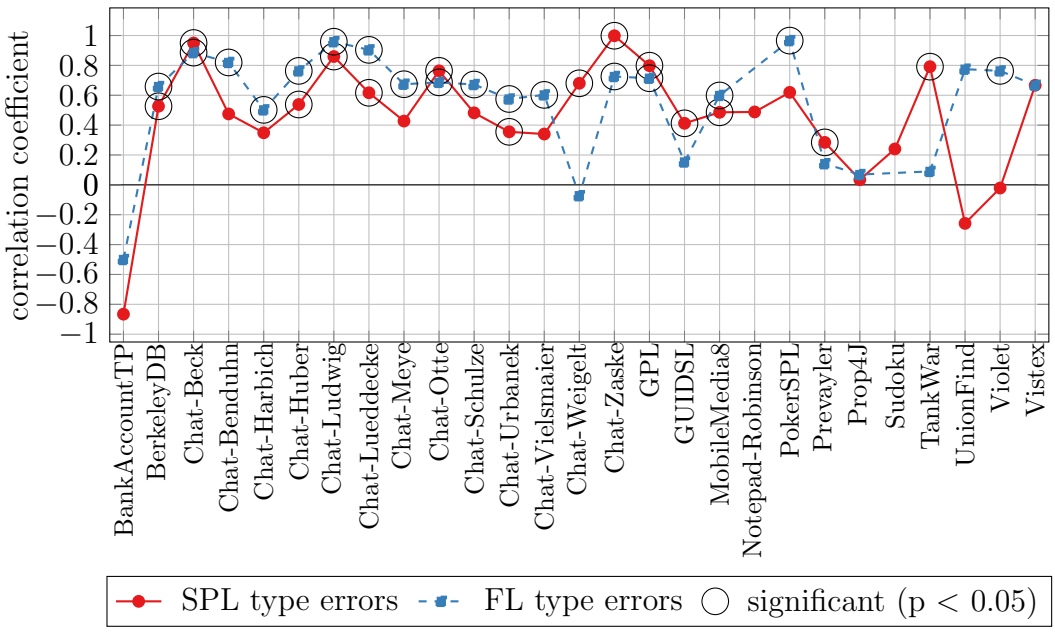

Figure 19: Fragmentation of classes ($frag_c$) in correlation to SPL and FL type errors

errors, because their quantity differs between features. As no correlation can be computed with a non-variable vector, the correlation values for EPL are omitted in the following. Notepad-Robinson, Sudoku and EPL don't have feature-local type errors and are therefore not considered in this regard. Results are shown in Figure 19 and Table 11.

For both kinds of type errors, all systems that correlate signficantly also correlate positively, meaning that stronger fragmented classes are more error-prone. For SPL-type errors half of the subject systems correlate significantly and for feature-local type errors even more (17 out of 25 subject systems with feature local type errors).

No significant correlation between SPL type errors and fragmentation of classes is shown for all three Editors (Violet, Vistex and Notepad-Robinson) and two (Sudoku and PokerSPL) of the three games in the subject product lines. EPL, Prop4J and UnionFind, which do neither correlate significantly

Name	SPL type errors	FL type errors	All type errors
BerkeleyDB	**0.526** ***	**0.658** ***	**0.671** ***
Chat-Beck	**0.950** **	**0.885** *	**0.885** *
Chat-Benduhn	0.475	**0.820** ***	**0.827** ***
Chat-Harbich	0.349	**0.502** *	**0.506** *
Chat-Huber	**0.538** **	**0.763** ***	**0.808** ***
Chat-Ludwig	**0.859** **	**0.959** ***	**0.982** ***
Chat-Lueddecke	**0.617** *	**0.904** ***	**0.904** ***
Chat-Meye	0.427	**0.674** *	**0.688** *
Chat-Otte	**0.764** *	**0.686** *	**0.844** **
Chat-Schulze	0.482	**0.672** *	0.559
Chat-Urbanek	**0.355** *	**0.575** ***	**0.575** ***
Chat-Vielsmaier	0.340	**0.603** **	**0.509** *
Chat-Weigelt	**0.680** *	-0.073	0.454
Chat-Zaske	**0.999** ***	**0.726** ***	**0.740** ***
GPL	**0.798** ***	**0.714** **	**0.837** ***
GUIDSL	**0.412** ***	0.153	**0.355** ***
MobileMedia8	**0.486** ***	**0.599** ***	**0.651** ***
PokerSPL	0.619	**0.966** ***	**0.966** ***
Prevayler	**0.284** ***	0.142	0.126
TankWar	**0.791** ***	0.090	**0.473** *
Violet	-0.021	**0.763** ***	**0.460** ***

Table 11: Correlation between type errors and fragmentation of classes ($frag_c$) (only systems with at least one significant correlation shown)

to SPL type errors nor to feature-local type errors all implement some kind of an algorithm. Apart from that, no connection between the correlation and the domains of the subject product lines was found.

It has to be noticed that this measure correlates strongly for many of the subject systems with feature-local type errors, although they are not affected by the fragmentation. One explanation could be that highly fragmented classes are more complex and hence more error-prone in general.

4.6.2 Fragmentation of Features

For all features the number of type errors (feature-local as well as software product line specific type errors) was correlated to the number of classes the feature introduces/enhances (and by which it therefore is fragmented) (see Chapter 3.2.4). The results are shown in Figure 20 and Table 12. Aside from PokerSPL, all correlations to SPL type errors that are significant are positive. Of those, PokerSPL is the only small product line with less than 20 features.

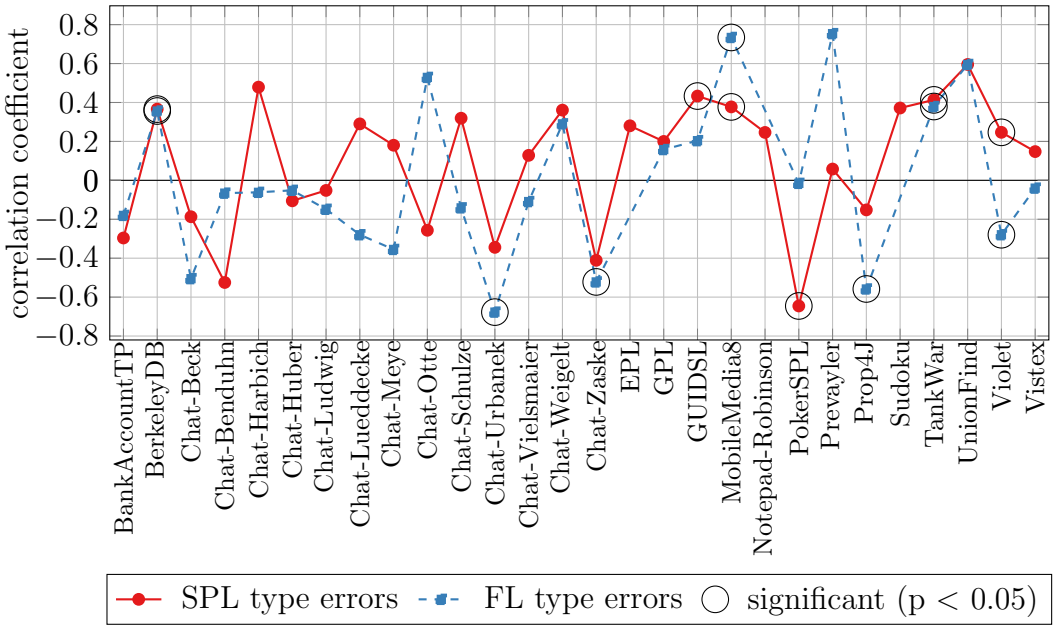

Figure 20: Fragmentation of features ($frag_f$) in correlation to SPL type errors and FL type errors

Name	SPL type errors	FL type errors	All type errors
BerkeleyDB	**0.366** ***	**0.356** ***	**0.393** ***
Chat-Urbanek	-0.344	**-0.677** *	**-0.648** *
Chat-Zaske	-0.411	**-0.522** *	-0.492
GUIDSL	**0.433** *	0.204	**0.473** *
MobileMedia8	**0.377** *	**0.734** ***	**0.662** ***
PokerSPL	**-0.645** *	-0.016	-0.316
Prop4J	-0.152	**-0.558** *	-0.253
TankWar	**0.413** *	**0.379** *	**0.582** ***
Violet	**0.247** *	**-0.280** **	0.007

Table 12: Correlation between fragmentation of features ($frag_f$) and type errors (only systems with at least one significant correlation shown)

4.7 Comparison of the Approaches

4.7.1 Measures based on Classes compared to Measures based on Features

In this thesis two measures were examined both based on classes as well as based on features: CBO/CBF and fragmentation of features/fragmentation of classes. In both cases more significant correlations were found for the measures based on classes (see Tables 6 and 7 in Chapter 4.4 as well as Tables 11 and 12 in Chapter 4.6). One reason for this that seems plausible is that there are often more classes than features because features often represent a higher level of abstraction than classes (a functionality instead of a code unit) and therefore most systems have more classes than features. This means a finer classification of type errors and a larger vector that is correlated and thus a higher probability of significance.

For Coupling of Features and Coupling of Object Classes this explanation seems likely, as only one of those three subject systems (MobileMedia8, TankWar and Violet) where CBF correlates stronger to SPL type errors than CBO has more classes than features and of the seven where CBO is better than CBF (BerkeleyDB, Chat-Lueddecke, Chat-Otte, Chat-Vielsmaier, Chat-Zaske, GPL, GUIDSL), only two (Chat-Otte and GPL) have more features than classes.

But it is different for fragmentation. Of those systems that have more features than classes (Chat-Beck, Chat-Luwdig, Chat-Meye, Chat-Otte, GPL, Notepad-Robinson, PokerSPL, TankWar, UnionFind, Violet and Vistex) only two show a stronger correlation for fragmentation of features than for fragmentation of classes. Five of them (Chat-Beck, Chat-Ludwig, Chat-Otte,

GPL and TankWar) even show a stronger correlation between SPL type errors and fragmentation of classes than for fragmentation of features, which is a counterexample for this explanation.

Another possible explanation is that with feature oriented software development all assets are organised in features, therefore it is easier to keep track of a highly fragmented feature than a highly fragmented class, especially if no tools that provide different views on the code are used. But this does not explain, why feature-local type errors also correlate stronger to fragmentation of classes than to fragmentation of features, as those are equal easy or hard to find in both views on the code.

What is to be considered in this regard as well is the difference in the counting of SPL type errors between classes and features. For classes only the information where the type error emerges is given, while for features the information which features are involved besides the feature where the type error arises is available and used for counting (see Chapter 3.1).

4.7.2 Measures based on the Structure of Code versus Measures based on the Feature Model

Comparing results between measures based only on information of the feature model (see Chapter 4.3) and measures based only on structure of the code (see Chapters 4.4 and 4.6), shows that the information of the feature model as it was used in this thesis is not specific enough to find characteristics of features that are likely to take part in SPL type errors. The measures based only on structure of the code seem more promising. In Chapter 4.5.2 a combination of both approaches is evaluated with the result that the combination can strengthen the correlation slightly, compared to using only the code-based measure (see Figure 16 in Chapter 4.5.2).

4.7.3 Coupling and Possible Feature Interactions in Comparison

When only considering CBF and EFD, the measures for coupling that are computed for each feature, in comparison to the degree in G_{refs}, which is a graph for coupling of features, it is to notice that except for GPL, all systems that correlate significantly and positively for CBF or EFD also correlate significantly and positively for the degree in G_{refs} (see Table 13). Hence the information regarding how likely a feature is to have SPL type errors when it is highly coupled is nearly the same for all three measures.

The additional information about which features introduce the same code elements in G_{PFI} changes the correlation to SPL type errors only a little. For five of the subject systems (BerkeleyDB, EPL, GPL, TankWar, Violet) the

	EFD	CBF	*deg* of G_{refs}	*deg* of G_{PFI}	$deg_{w,dep}$ of G_{PFI}
BankAccountTP	-0.308	-0.172	0.013	**-0.803** *	-0.025
BerkeleyDB	**0.241** *	**0.313** **	**0.326** ***	**0.339** ***	**0.351** ***
Chat-Benduhn	**-0.621** *	**-0.868** ***	-0.243	**-0.796** **	**-0.618** *
EPL	0.280	-0.173	**0.657** *	**0.701** *	**0.775** **
GPL	0.192	**0.471** *	0.263	**0.577** **	**0.569** **
GUIDSL	-0.003	0.294	**0.424** *	**0.393** *	**0.443** *
MobileMedia8	**0.458** **	**0.561** ***	**0.397** **	0.211	**0.298** *
Notepad-Robinson	-0.175	0.538	0.435	0.559	**0.696** *
PokerSPL	0.477	**-0.640** *	**-0.731** *	0.186	-0.216
Prevayler	**-0.899** *	0.470	0.406	0.154	0.319
Prop4J	-0.525	-0.120	0.519	0.519	**0.705** **
TankWar	0.354	**0.805** ***	**0.436** *	**0.502** **	**0.653** ***
Violet	**0.504** ***	**0.310** **	**0.448** ***	**0.591** ***	**0.486** ***

Table 13: Comparison of correlations between SPL type errors and measures for coupling and possible feature interactions (only systems with at least one significant value shown)

correlation to SPL type errors is stronger for degree in G_{PFI} than for degree in G_{refs}. Only two systems (GUIDSL, MobileMedia8) show a stronger correlation between SPL type errors and degree in G_{refs} than to degree in G_{PFI}. Even when enhancing the graph with information about the dependencies between features and taking into account the number of references and introduces that connect two features, the correlation to SPL type errors is only somewhat stronger than the correlation to EFD and CBF (see Table 13).

4.7.4 Overall Comparison

By putting all measures in one graph (see Figure 21) it is harder to see how a certain measure performs, but the overall picture is more visible.

Four of the subject systems always correlate positively, those four are also the biggest (in regard to the number of features) subject product lines examined in this thesis: BerkeleyDB, MobileMedia8, TankWar and Violet. All subject product lines that show significant negative correlation to SPL type errors for any of the measures are small (10-19 features) or very small (<10 features) ones, namely BankAccountTP, Chat-Benduhn, Chat-Huber, PokerSPL, Prevayler, UnionFind and Vistex. There are two systems that show no significant correlation to SPL type errors for any of the measures, Chat-Harbich and Chat-Schulze, again both small or very small product lines. All in all, the values of the different measures lie closer together for bigger systems and wider apart for smaller systems.

One measure that often sticks out in Figure 21 is fragmentation of classes. Every significant correlation between the measure and SPL or feature local type errors is positive. For SPL type errors, half of the subject systems show a significant correlation and for 17 out of 28 subject product lines there is also a significant correlation between fragmentation of classes and feature-local type errors (see Chapter 4.6.1). Therefore it is the measure with the most significant correlations examined in this thesis.

4.8 Threats to Validity

The subject product lines used in this thesis are of different sizes (see Chapter 4) and most of the measures are calculated per feature in order to correlate them to the type errors of the features. Statistical significance depends on the size of the vectors that are correlated as well as the strength of the correlation. As there are various reasons for type errors and the measures mostly capture only parts of them, the correlation is often moderate at most. Insofar most of the small and very small product lines are not likely to show a significant correlation between type errors and the measures. In order to

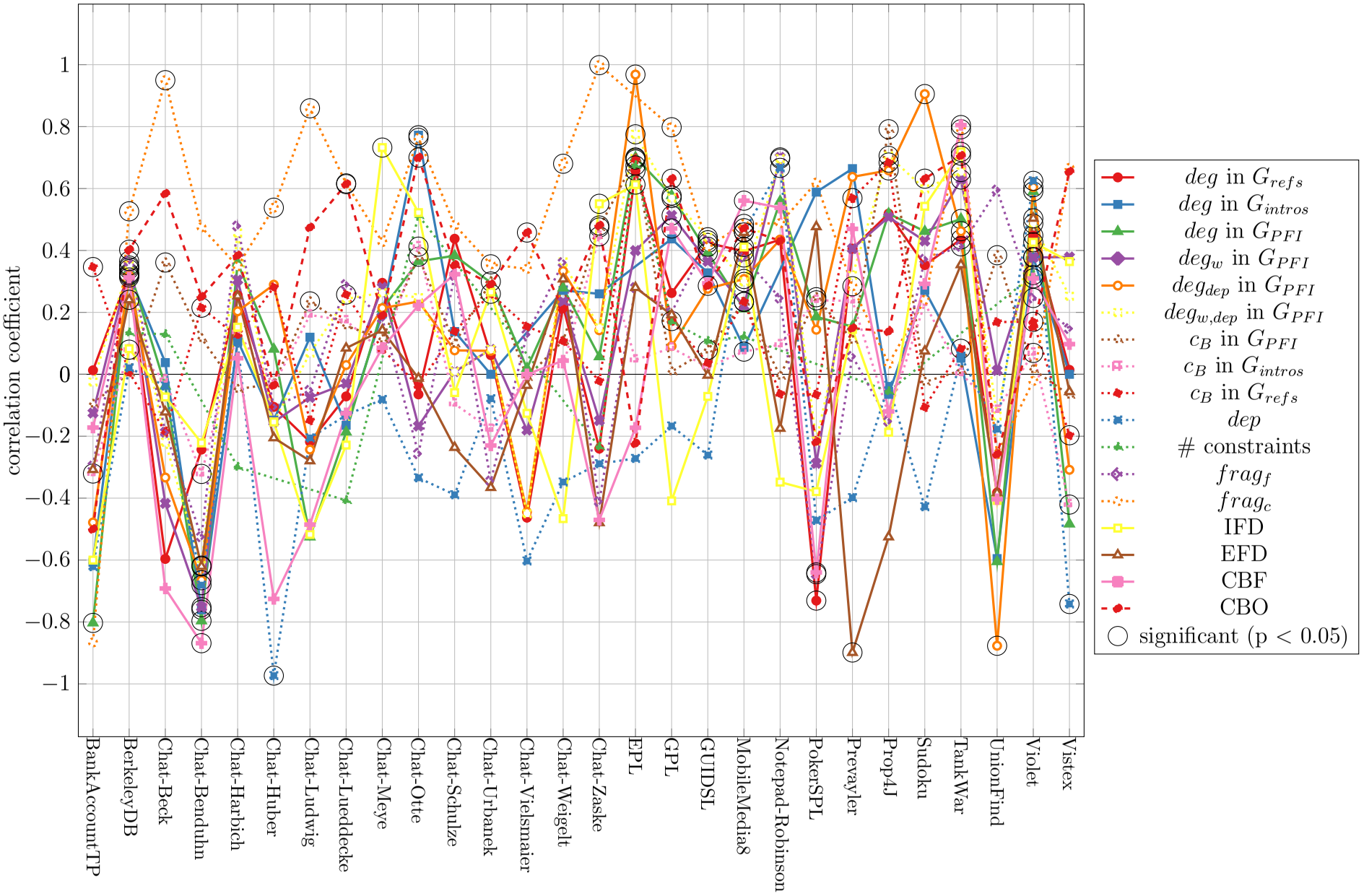

Figure 21: Correlation between SPL type errors and the measures used in this thesis

make the results of this thesis more generalizable, additional product lines of larger size need to be examined.

It also must be considered that most of the measures which are based on the structure of the code are implicitly influenced by size (lines of code) of the feature or class they are calculated for. For example, measures that take into account the number of relations between features - of course big features or classes can contain more references than small ones. The same is true for the number of introduces that two features share or the fragmentation of features or classes.

5 Conclusion

The goal of this thesis was to examine reasons for SPL type errors and to find a measure that can be used as an indicator for them. Therefore different measures were developed and examined which were meant to capture aspects of the different reasons for SPL type errors. In this regard the conceptual structure, therefore the feature model, of 28 product lines was analysed as well as their code and the relations between code elements. 13 measures were assessed in total and can be classified into four approaches: Variability, cohesion and coupling, possible feature interactions and code fragmentation.

The measures for variability, which are based only on the feature model, were the number of constraints of a feature and a weighting of the dependencies to other features. Both were not suitable as an indicator for error-proneness of features for the subject product lines.

On code level, cohesion and coupling were examined as well as possible feature interactions and code fragmentation. While the measures for cohesion and coupling that were based on features (IFD, EFD, CBF) also showed no correlation to SPL type errors for most of the subject systems, CBO, which is a measure for coupling of classes, showed a significant and positive correlation for a third of the subject product lines.

Possible feature interactions were analysed based on graphs that visualise them and computation of graph measures like degree centrality and edge betweenness centrality. With all additional information the degree ($deg_{w,dep}$) of one third of the subject systems correlated significantly and positively to SPL type errors. Using information about the relations between features in the possible feature interaction graphs improved the correlation between SPL type errors and the degree insignificantly, because there were also always subject systems that correlated less with those additional information. Edge betweenness centrality of the basic possible feature interactions graph (G_{PFI}) correlated significantly and positively to SPL type errors for 11 of the 28 subject product lines.

Finally code fragmentation was evaluated for features and classes. Again the measure based on classes showed more significant correlations to SPL type errors than the measure for features. With half of the subject systems correlating significantly and positively, fragmentation of classes is better than the other evaluated measures for predicting which features contain SPL type errors for the subject product lines that were analysed in this thesis. In contrast to some of the other examined measures, code fragmentation is a measure that is easily and intuitively applicable for developers in order to find out which code-parts they have to pay extra attention to for avoiding SPL type errors.

References

[1] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013.

[2] Sven Apel and Dirk Beyer. Feature cohesion in software product lines: An exploratory study. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 421–430, New York, NY, USA, 2011. ACM.

[3] Sven Apel and Christian Kästner. An overview of feature-oriented software development. *Journal of Object Technology*, 8:49–84, 2009.

[4] Sven Apel, Christian Kastner, and Christian Lengauer. Featurehouse: Language-independent, automated software composition. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 221–231, Washington, DC, USA, 2009. IEEE Computer Society.

[5] Sven Apel, Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, and Brady Garvin. Exploring feature interactions in the wild: The new feature-interaction challenge. In *Proceedings of the 5th International Workshop on Feature-Oriented Software Development*, FOSD '13, pages 1–8, New York, NY, USA, 2013. ACM.

[6] Sven Apel, Alexander von Rhein, Philipp Wendler, Armin Größlinger, and Dirk Beyer. Strategies for product-line verification: Case studies and experiments. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 482–491, Piscataway, NJ, USA, 2013. IEEE Press.

[7] Sven Apel, Wolfgang Scholz, Christian Lengauer, and Christian Kästner. Language-independent reference checking in software product lines. In *Proceedings of the 2Nd International Workshop on Feature-Oriented Software Development*, FOSD '10, pages 65–71, New York, NY, USA, 2010. ACM.

[8] Don Batory. Feature models, grammars, and propositional formulas. In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pages 7–20, Berlin, Heidelberg, 2005. Springer Publishing Company, Incorporated.

[9] Don Batory. A tutorial on feature oriented programming and the ahead tool suite. In *Proceedings of the 2005 International Conference on*

Generative and Transformational Techniques in Software Engineering, GTTSE'05, pages 3–35, Berlin, Heidelberg, 2006. Springer Publishing Company, Incorporated.

[10] James M. Bieman and Byung-Kyoo Kang. Cohesion and reuse in an object-oriented system. In *Proceedings of the 1995 Symposium on Software Reusability*, SSR '95, pages 259–262, New York, NY, USA, 1995. ACM.

[11] Günter Böckle, Jesus Bermejo Munoz, Peter Knauber, Charles W. Krueger, Julio Cesar Sampaio do Prado Leite, Frank van der Linden, Linda M. Northrop, Michael Stark, and David M. Weiss. Adopting and institutionalizing a product line culture. In Gary J. Chastek, editor, *Software Product Lines, Second International Conference, SPLC 2*, volume 2379 of *Lecture Notes in Computer Science*, pages 49–59, San Diego, CA, USA, August 2002. Springer.

[12] Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *SOCIAL NETWORKS*, 30(2), 2008.

[13] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994.

[14] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[15] Bruno Carreiro da Silva, Cláudio Sant'Anna, Christina Chavez, and Alessandro Garcia. Concern-based cohesion: Unveiling a hidden dimension of cohesion measurement. In Dirk Beyer, Arie van Deursen, and Michael W. Godfrey, editors, *ICPC*, pages 103–112, 2012.

[16] Christof Ebert and Reiner Dumke. *Software Measurement*. Springer Publishing Company, Incorporated, 2007.

[17] Christian Kästner, Alexander von Rhein, Sebastian Erdweg, Jonas Pusch, Sven Apel, Tillmann Rendel, and Klaus Ostermann. Toward variability-aware testing. In *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, FOSD '12, pages 1–8, New York, NY, USA, 2012. ACM.

[18] Sergiy Kolesnikov, Alexander von Rhein, Claus Hunsen, and Sven Apel. A comparison of product-based, feature-based, and family-based type

checking. In *Proceedings of the 12th International Conference on Generative Programming: Concepts & Experiences*, GPCE '13, pages 115–124, New York, NY, USA, 2013. ACM.

[19] Roberto E. Lopez-Herrejon and Don S. Batory. A standard problem for evaluating product-line methodologies. In *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, GCSE '01, pages 10–24, London, UK, UK, 2001. Springer Publishing Company, Incorporated.

[20] Roberto E. Lopez-Herrejon and Don S. Batory. The expression problem as a product line and its implementation in ahead. Technical report, University of Texas at Austin. Department of Computer Sciences, 2004.

[21] Andrian Marcus, Denys Poshyvanyk, and Rudolf Ferenc. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Trans. Softw. Eng.*, 34(2):287–300, March 2008.

[22] Sandro Morasca. Fundamental aspects of software measurement. *Software Engineering International Summer Schools*, 2013.

[23] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, MA, USA, 2002.

[24] Gunter Saake Sven Apel, Christian Kästner. Software product-line engineering. Lecture notes, University of Passau, 2013.

[25] Sahil Thaker, Don Batory, David Kitchin, and William Cook. Safe composition of product lines. In *Proceedings of the 6th International Conference on Generative Programming and Component Engineering*, GPCE '07, pages 95–104, New York, NY, USA, 2007. ACM.

[26] Thomas Thüm, Sven Apel, Christian Kästner, Martin Kuhlemann, Ina Schaefer, and Gunter Saake. Analysis strategies for software product lines. Technical Report FIN-04-2012, Otto-von-Guericke-University Magdeburg, April 2012.

[27] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Sci. Comput. Program.*, 79:70–85, January 2014.

List of Figures

List of Tables

A Appendix

A.1 Description of the Subject Product Lines

BankAccountTP is a software for managing a bank account, which was developed from scratch as a product line.

BerkeleyDB (Java Edition) is an open source database engine that can be embedded into applications as a library. In this thesis the Fuji-compilable version is examined.

13 Chat-Systems are product lines developed by students at courses on software product line development.

EPL is a small product line that implements the expression problem and therefore evaluates expressions. It was implemented as an example for using and research on feature-oriented software development [20].

GPL is a graph library implemented as product line. Like EPL it was implemented as example and for research on feature-oriented software development [19].

GUIDSL is a tool for configuration of product lines which use a feature model in GUIDSL format. The GUIDSL format is a grammar with the possibility of further constraints in form of propositional formulas. It was implemented using feature-oriented programming methods.

MobileMedia8 is a software for manipulating photo, music and video on mobile devices. It was developed from scratch as software product line at Lancaster University. In this thesis the Fuji-compilable version is examined..

Notepad-Robinson was developed, like the chat systems, by a student as part of a course.

PokerSPL is a game which was refactored as product line at Otto-von-Guericke University Magdeburg.

Prevayler is an open source object persistence library for Java[5] which was refactored using feature-oriented software development.

Prop4J is a library for arbitrary propositional formulas which is used in FeatureIDE.

[5] http://prevayler.org/

Sudoku is an game which was as well refactored as feature-oriented software product line.

TankWar is a open source game which was developed from scratch feature-oriented.

UnionFind is a product line of UnionFind algorithms. It was developed at Otto-von-Guericke University Magdeburg as a case study.

Violet is a simple UML Editor which was as well recreated as feature-oriented product line through decomposing the open source software Violet [6].

Vistex was as well developed as a student project. It is a software product line for graphical manipulation of graphs and was designed to be easily extensible.

[6] http://sourceforge.net/projects/violet/

A.2 P-Values for Shapiro-Wilk Test

	Classes			Features		
	SPL	FL	all	SPL	FL	all
BankAccountTP	-9.11e-07	**4.63e-01**	**5.10e-01**	1.26e-02	1.91e-02	2.20e-02
BerkeleyDB	2.72e-32	3.87e-33	1.61e-31	1.75e-19	1.91e-21	9.70e-21
Chat-Beck	2.67e-02	**6.10e-02**	**5.51e-02**	**5.22e-01**	**4.94e-01**	**9.61e-01**
Chat-Benduhn	2.19e-07	6.55e-04	3.68e-04	7.17e-03	5.67e-06	2.36e-04
Chat-Harbich	2.69e-09	1.88e-05	1.44e-05	1.71e-06	1.01e-03	9.00e-04
Chat-Huber	1.46e-09	8.96e-06	3.18e-06	**4.73e-01**	**1.11e-01**	**2.49e-01**
Chat-Ludwig	4.29e-05	1.29e-04	2.59e-04	**2.02e-01**	1.33e-04	**1.41e-01**
Chat-Lueddecke	2.89e-07	2.72e-04	4.28e-06	1.80e-04	1.86e-03	9.95e-03
Chat-Meye	4.29e-05	3.14e-03	3.85e-03	2.89e-05	2.31e-02	9.37e-03
Chat-Otte	5.22e-06	6.73e-05	1.14e-05	**1.04e-01**	8.19e-04	1.52e-02
Chat-Schulze	9.45e-04	1.58e-05	4.20e-03	7.41e-03	3.13e-04	2.86e-02
Chat-Urbanek	4.64e-12	1.29e-09	5.15e-10	5.42e-03	**1.54e-01**	**1.01e-01**
Chat-Vielsmaier	1.05e-06	9.79e-07	2.05e-06	**2.63e-01**	**7.89e-01**	**9.24e-01**
Chat-Weigelt	3.37e-06	**9.32e-02**	4.83e-05	3.36e-02	**9.99e-02**	**1.13e-01**
Chat-Zaske	1.65e-13	1.13e-12	9.65e-13	1.51e-03	1.21e-02	1.59e-02
EPL	**4.71e-01**	**1.00**	**4.71e-01**	**2.15e-01**	**1.00**	**2.15e-01**
GPL	6.89e-06	8.96e-07	2.47e-06	1.59e-05	1.63e-05	1.08e-04
GUIDSL	5.80e-24	3.36e-25	1.41e-24	2.74e-06	9.13e-10	2.38e-08
MobileMedia8	5.23e-14	2.31e-09	8.75e-10	1.69e-10	2.02e-12	1.06e-11
Notepad-Robinson	3.22e-07	**1.00**	3.22e-07	4.67e-06	**1.00**	4.67e-06
PokerSPL	1.05e-06	1.39e-02	7.93e-03	1.69e-04	7.10e-04	2.14e-03
Prevayler	1.43e-25	3.24e-24	2.35e-24	**1.73e-01**	3.61e-02	**1.71e-01**
Prop4J	2.67e-06	3.99e-06	2.55e-06	1.67e-04	3.99e-06	1.24e-05
Sudoku	6.85e-09	**1.00**	6.85e-09	**7.63e-02**	**1.00**	**7.63e-02**
TankWar	4.51e-07	1.98e-08	3.61e-08	1.06e-07	4.32e-10	1.75e-09
UnionFind	**1.97e-01**	**8.65e-01**	**2.75e-01**	**9.69e-02**	3.37e-02	**1.40e-01**
Violet	3.58e-17	2.82e-16	1.17e-16	1.54e-19	3.52e-16	3.98e-19
Vistex	1.05e-06	1.05e-06	1.05e-06	2.57e-05	3.98e-04	5.68e-04

Table 14: P-Values for Shapiro-Wilk test on the distribution of type errors of classes and features (bold values are greater than 0.05) (SPL: software product line specific type errors, FL: Feature-local type errors, all: both kinds of type errors together)

A.3 Correlation Results

A.3.1 Results for Correlation between Variability Measures and Type Errors

A.3.1.1 Dependencies to other Features

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.621	-0.688	-0.704
BerkeleyDB	0.021	0.125	0.036
Chat-Beck	-0.038	-0.578	-0.235
Chat-Benduhn	**-0.681** *	-0.495	-0.515
Chat-Harbich	0.104	**-0.564** *	-0.528
Chat-Huber	**-0.973** **	**-0.949** *	**-0.949** *
Chat-Ludwig	-0.205	-0.111	-0.224
Chat-Lueddecke	-0.132	-0.555	-0.453
Chat-Meye	-0.081	-0.082	-0.139
Chat-Otte	-0.334	-0.473	-0.568
Chat-Schulze	-0.389	-0.273	-0.387
Chat-Urbanek	-0.079	0.122	0.074
Chat-Vielsmaier	-0.603	-0.538	-0.482
Chat-Weigelt	-0.348	-0.412	-0.373
Chat-Zaske	-0.289	-0.460	-0.433
EPL	-0.272	-	-0.272
GPL	-0.167	**-0.453** *	-0.295
GUIDSL	-0.261	-0.132	-0.218
MobileMedia8	**0.460** **	**0.371** *	**0.533** ***
Notepad-Robinson	**0.667** *	-	**0.667** *
PokerSPL	-0.472	0.576	0.185
Prevayler	-0.399	0.664	0.655
Prop4J	-0.036	-0.481	-0.273
Sudoku	-0.428	-	-0.428
TankWar	0.059	-0.285	-0.136
UnionFind	-0.177	-0.203	-0.176
Violet	**0.625** ***	**-0.805** ***	-0.058
Vistex	**-0.742** ***	**-0.674** **	**-0.719** **

Table 15: Correlation between type errors and weighted dependencies

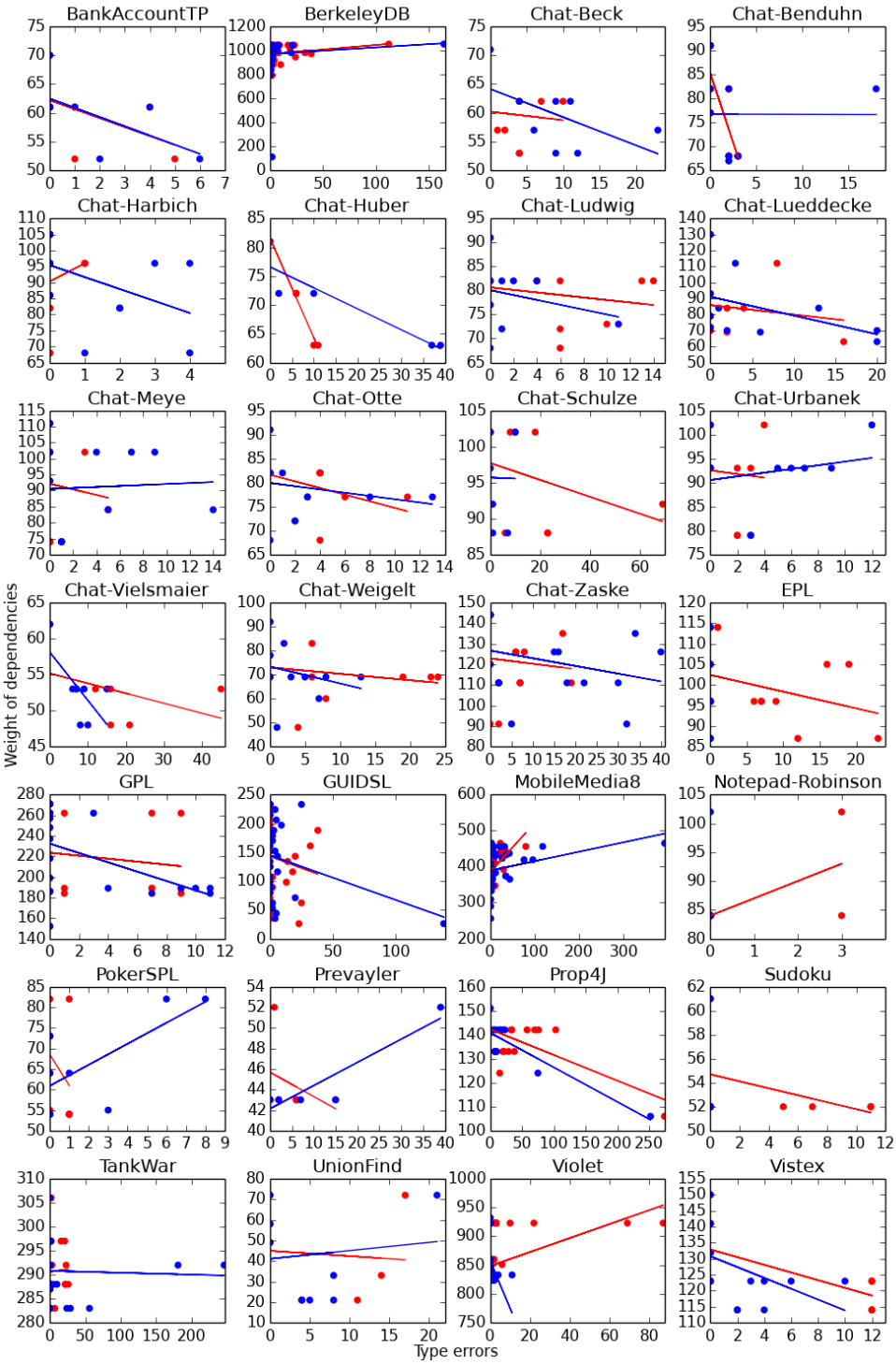

Figure 22: Correlation between type errors and the weighted dependencies of a feature (red = SPL type errors, blue = feature local type errors)

A.3.1.2 Number of Constraints

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-	-	-
BerkeleyDB	0.134	**0.225** *	0.161
Chat-Beck	0.129	0.317	0.000
Chat-Benduhn	-	-	-
Chat-Harbich	-0.300	-0.100	-0.137
Chat-Huber	-	-	-
Chat-Ludwig	-	-	-
Chat-Lueddecke	-0.408	-0.268	-0.343
Chat-Meye	-	-	-
Chat-Otte	0.513	**0.705** *	**0.822** **
Chat-Schulze	-	-	-
Chat-Urbanek	-	-	-
Chat-Vielsmaier	-	-	-
Chat-Weigelt	-	-	-
Chat-Zaske	-0.234	-0.349	-0.349
EPL	**0.675** *	-	**0.675** *
GPL	0.170	0.141	0.169
GUIDSL	0.107	-0.100	0.020
MobileMedia8	0.122	0.133	0.088
Notepad-Robinson	-	-	-
PokerSPL	-	-	-
Prevayler	-	-	-
Prop4J	-0.055	0.282	0.186
Sudoku	-	-	-
TankWar	-	-	-
UnionFind	-	-	-
Violet	**0.321** **	0.171	**0.396** ***
Vistex	-	-	-

Table 16: Correlation between type errors and the number of constraints a feature takes part in

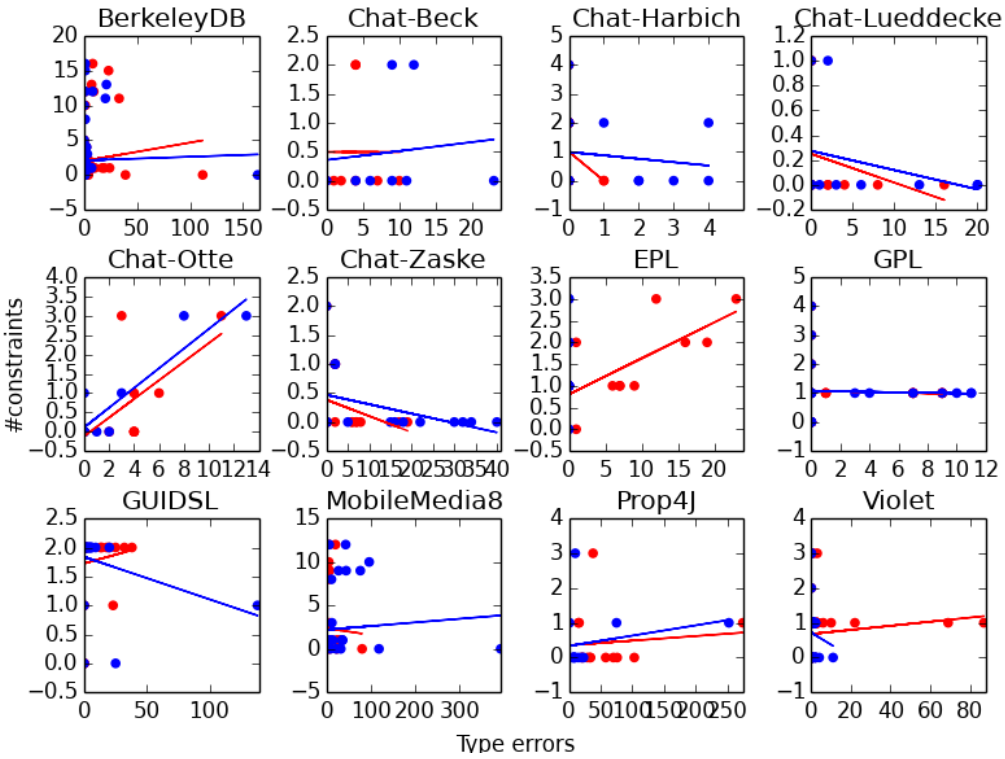

Figure 23: Correlation between type errors and #constraints of a feature (red = SPL type errors, blue = feature local type errors)

A.3.2 Results for Correlation between Cohesion and Coupling and Type Errors

A.3.2.1 Internal-ratio Feature Dependency

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.600	-0.439	-0.585
BerkeleyDB	0.083	-0.148	-0.009
Chat-Beck	-0.073	0.132	0.333
Chat-Benduhn	-0.222	0.264	0.034
Chat-Harbich	0.152	0.139	0.120
Chat-Huber	-0.154	-0.200	-0.200
Chat-Ludwig	-0.517	-0.023	-0.418
Chat-Lueddecke	-0.229	0.075	0.056
Chat-Meye	**0.732** *	0.328	0.476
Chat-Otte	0.522	0.065	0.506
Chat-Schulze	-0.059	**-0.774** *	-0.412
Chat-Urbanek	0.266	0.378	0.409
Chat-Vielsmaier	-0.126	0.393	-0.108
Chat-Weigelt	-0.466	-0.425	-0.449
Chat-Zaske	**0.551** *	**0.512** *	**0.525** *
EPL	**0.613** *	-	**0.613** *
GPL	-0.409	**-0.497** *	**-0.492** *
GUIDSL	-0.071	-0.214	-0.111
MobileMedia8	**0.411** **	**0.603** ***	**0.594** ***
Notepad-Robinson	-0.348	-	-0.348
PokerSPL	-0.379	0.051	-0.277
Prevayler	0.145	-0.754	-0.714
Prop4J	-0.187	0.168	-0.167
Sudoku	0.543	-	0.543
TankWar	**0.718** ***	**0.452** *	**0.754** ***
UnionFind	-0.408	-0.370	-0.405
Violet	**0.426** ***	**-0.473** ***	0.004
Vistex	0.364	0.455	0.393

Table 17: Correlation between type errors and IFD

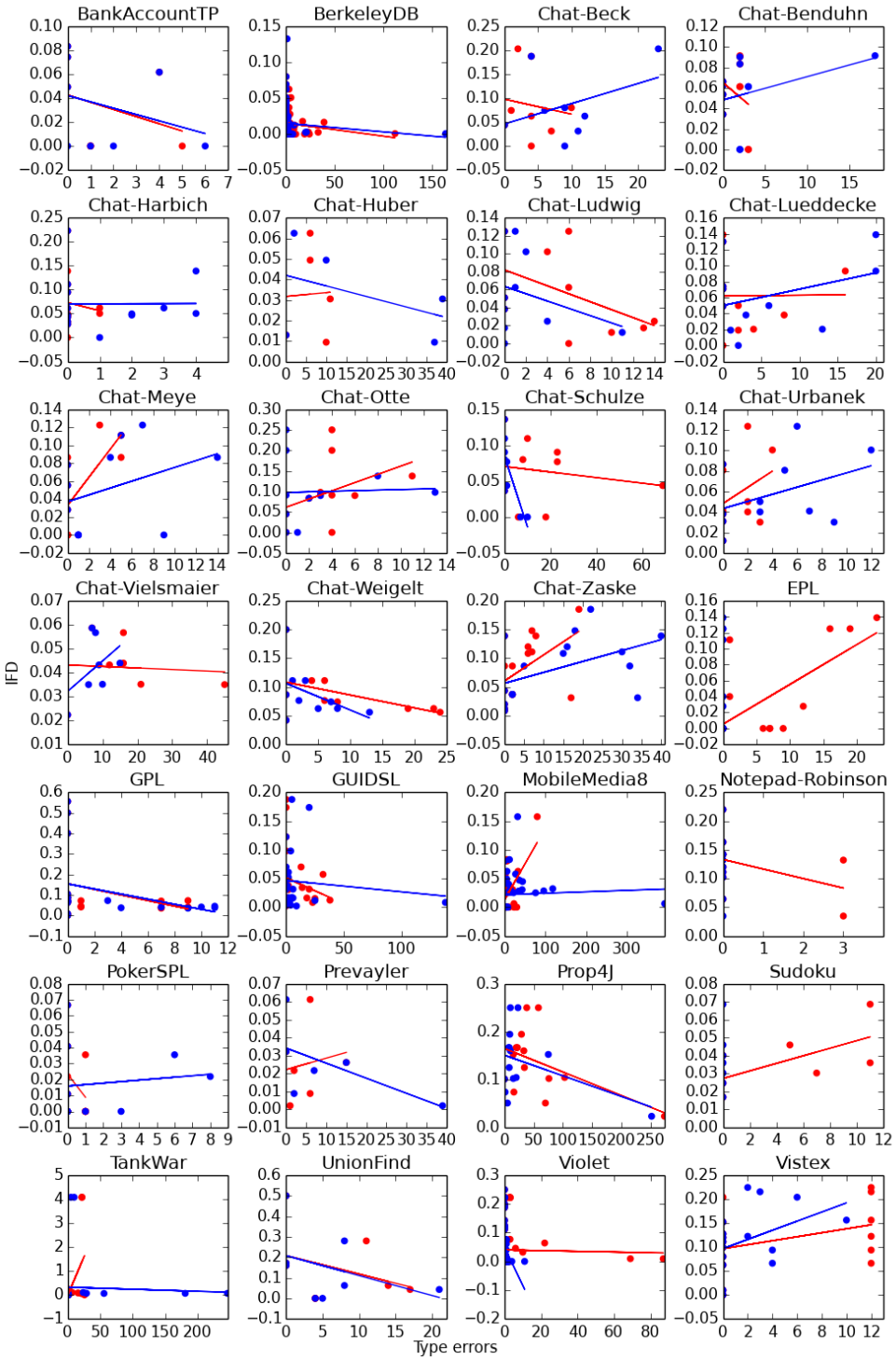

Figure 24: Correlation between type errors and IFD (red = SPL type errors, blue = feature local type errors)

A.3.2.2 External-ratio Feature Dependency

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.308	-0.572	-0.450
BerkeleyDB	**0.241** *	0.127	0.190
Chat-Beck	-0.122	-0.527	-0.262
Chat-Benduhn	**-0.621** *	-0.254	-0.437
Chat-Harbich	0.253	-0.015	0.015
Chat-Huber	-0.205	-0.100	-0.100
Chat-Ludwig	-0.280	-0.409	-0.407
Chat-Lueddecke	0.084	-0.522	-0.363
Chat-Meye	0.143	-0.289	-0.204
Chat-Otte	-0.013	-0.578	-0.381
Chat-Schulze	-0.236	-0.645	-0.445
Chat-Urbanek	-0.366	-0.457	-0.446
Chat-Vielsmaier	-0.036	-0.250	0.072
Chat-Weigelt	0.308	0.202	0.301
Chat-Zaske	-0.479	**-0.592** *	**-0.583** *
EPL	0.280	-	0.280
GPL	0.192	-0.043	0.137
GUIDSL	-0.003	-0.061	0.057
MobileMedia8	**0.458** **	**0.490** ***	**0.559** ***
Notepad-Robinson	-0.175	-	-0.175
PokerSPL	0.477	**-0.652** *	-0.399
Prevayler	**-0.899** *	-0.145	-0.257
Prop4J	-0.525	-0.339	-0.341
Sudoku	0.075	-	0.075
TankWar	0.354	0.245	**0.421** *
UnionFind	-0.383	-0.333	-0.380
Violet	**0.504** ***	**-0.463** ***	0.088
Vistex	-0.056	-0.124	-0.129

Table 18: Correlation between type errors and EFD

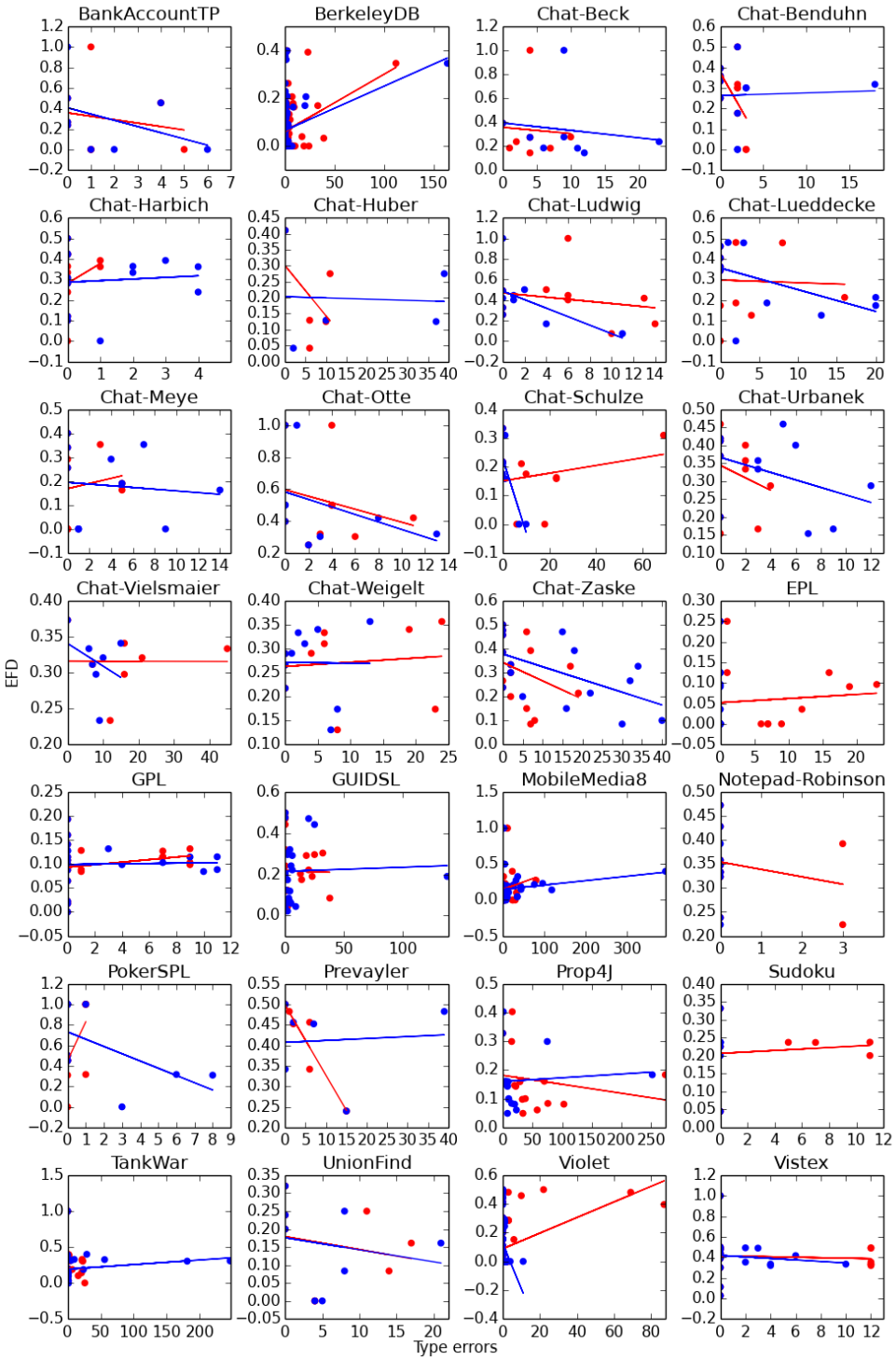

Figure 25: Correlation between type errors and EFD (red = SPL type errors, blue = feature local type errors)

A.3.2.3 Coupling between Object Classes

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.500	0.000	0.000
BerkeleyDB	**0.402** ***	**0.484** ***	**0.510** ***
Chat-Beck	0.583	0.295	0.295
Chat-Benduhn	0.250	0.479	0.472
Chat-Harbich	0.384	0.101	0.104
Chat-Huber	-0.035	0.247	0.217
Chat-Ludwig	0.474	0.617	0.621
Chat-Lueddecke	**0.614** *	**0.634** *	**0.670** **
Chat-Meye	0.190	0.361	0.240
Chat-Otte	**0.700** *	0.449	0.651
Chat-Schulze	0.353	0.241	0.353
Chat-Urbanek	0.291	**0.440** *	**0.440** *
Chat-Vielsmaier	**0.458** *	**0.678** ***	**0.613** **
Chat-Weigelt	0.209	0.026	0.278
Chat-Zaske	**0.481** **	0.267	0.271
EPL	-0.224	-	-0.224
GPL	**0.632** **	**0.665** **	**0.658** **
GUIDSL	**0.286** ***	**0.349** ***	**0.463** ***
MobileMedia8	**0.472** ***	**0.380** **	**0.443** **
Notepad-Robinson	0.431	-	0.431
PokerSPL	-0.218	0.201	0.201
Prevayler	0.149	-0.075	-0.043
Prop4J	0.139	0.002	0.098
Sudoku	**0.632** ***	-	**0.632** ***
TankWar	**0.706** ***	0.196	**0.495** *
UnionFind	-0.258	0.775	0.544
Violet	0.154	-0.235	0.007
Vistex	0.655	0.655	0.655

Table 19: Correlation between type errors and CBO

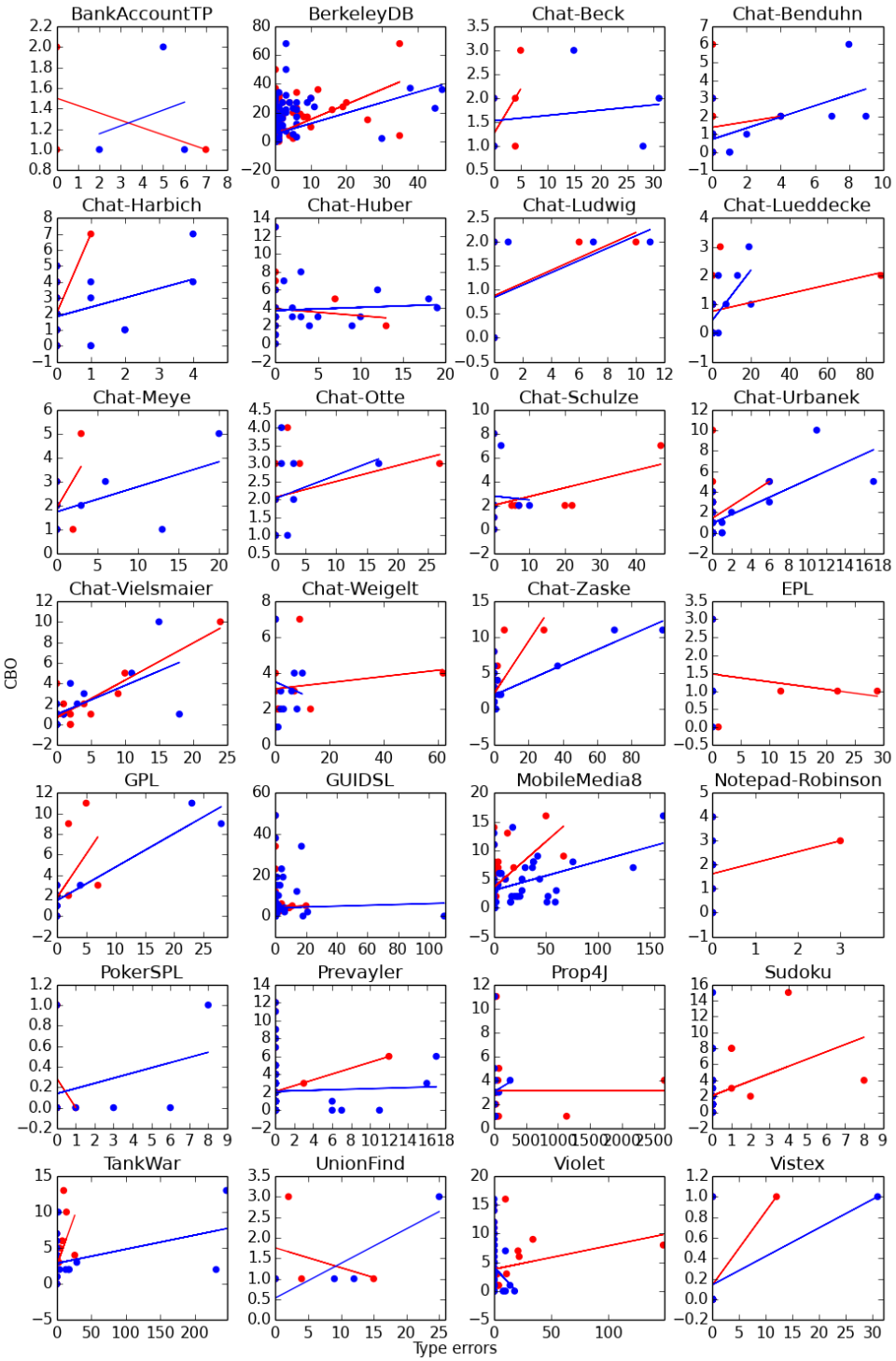

Figure 26: Correlation between type errors and CBO (red = SPL type errors, blue = feature local type errors)

A.3.2.4 Coupling between Features

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.172	-0.058	-0.168
BerkeleyDB	**0.313** **	**0.304** **	**0.351** ***
Chat-Beck	-0.692	-0.296	-0.246
Chat-Benduhn	**-0.868** ***	**-0.723** *	**-0.758** **
Chat-Harbich	0.052	-0.124	-0.139
Chat-Huber	-0.725	-0.707	-0.707
Chat-Ludwig	-0.486	-0.337	-0.502
Chat-Lueddecke	-0.126	-0.307	-0.236
Chat-Meye	0.086	-0.506	-0.447
Chat-Otte	0.223	0.495	0.350
Chat-Schulze	0.323	0.207	0.373
Chat-Urbanek	-0.230	-0.017	-0.094
Chat-Vielsmaier	0.000	0.289	0.000
Chat-Weigelt	0.044	0.082	0.044
Chat-Zaske	-0.470	-0.351	-0.333
EPL	-0.173	-	-0.173
GPL	**0.471** *	**0.762** ***	**0.640** **
GUIDSL	0.294	0.319	**0.477** *
MobileMedia8	**0.561** ***	**0.640** ***	**0.672** ***
Notepad-Robinson	0.538	-	0.538
PokerSPL	**-0.640** *	0.250	-0.101
Prevayler	0.470	**0.877** *	**0.926** **
Prop4J	-0.120	-0.097	-0.055
Sudoku	0.292	-	0.292
TankWar	**0.805** ***	**0.444** *	**0.845** ***
UnionFind	-0.403	-0.356	-0.401
Violet	**0.310** **	**-0.633** ***	**-0.211** *
Vistex	0.097	-0.040	0.040

Table 20: Correlation between type errors and CBF

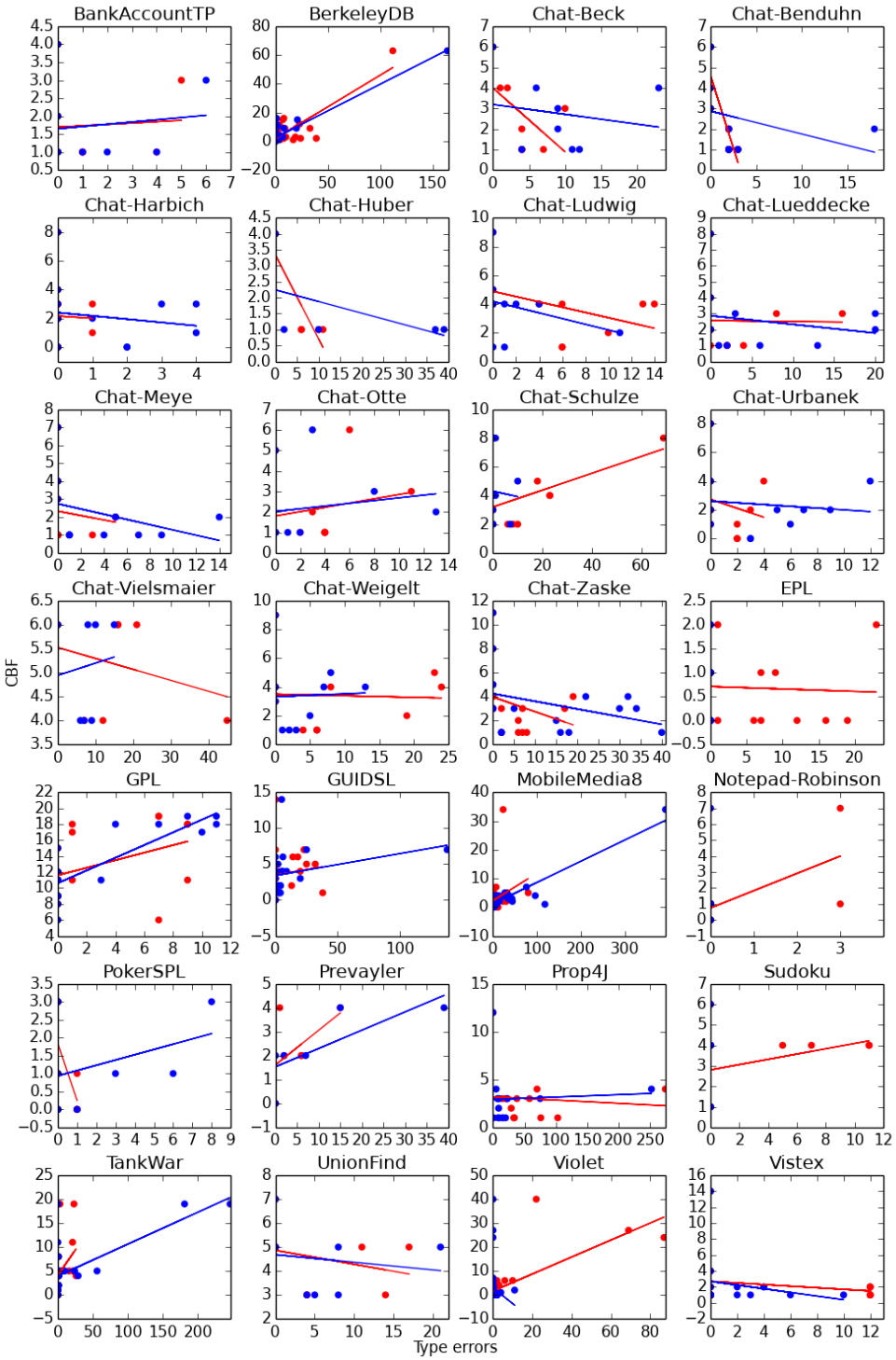

Figure 27: Correlation between type errors and CBF (red = SPL type errors, blue = feature local type errors)

A.3.3 Results for Correlation between Possible Feature Interactions and Type Errors

A.3.3.1 Degree Centrality

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.607	-0.615	-0.592
BerkeleyDB	**0.323** **	**0.288** **	**0.356** ***
Chat-Beck	0.038	-0.550	-0.381
Chat-Benduhn	**-0.761** **	-0.253	-0.497
Chat-Harbich	0.109	-0.103	-0.068
Chat-Huber	-0.148	0.000	0.000
Chat-Ludwig	0.120	-0.042	0.059
Chat-Lueddecke	-0.165	-0.263	-0.314
Chat-Meye	0.195	-0.472	-0.359
Chat-Otte	**0.772** **	-0.111	0.284
Chat-Schulze	0.138	-0.040	0.101
Chat-Urbanek	0.000	-0.596	-0.467
Chat-Vielsmaier	-	-	-
Chat-Weigelt	0.271	0.224	0.270
Chat-Zaske	0.260	0.290	0.290
EPL	-	-	-
GPL	0.437	0.415	**0.488** *
GUIDSL	0.328	**0.633** ***	**0.500** **
MobileMedia8	0.087	0.236	0.162
Notepad-Robinson	-	-	-
PokerSPL	0.588	-0.322	-0.072
Prevayler	0.664	0.531	0.655
Prop4J	-0.065	-0.433	-0.194
Sudoku	0.271	-	0.271
TankWar	0.052	**0.598** ***	0.278
UnionFind	-0.595	-0.595	-0.592
Violet	**0.379** ***	**-0.888** ***	**-0.372** ***
Vistex	0.000	0.000	0.000

Table 21: Correlation between type errors and *deg* in G_{intros}

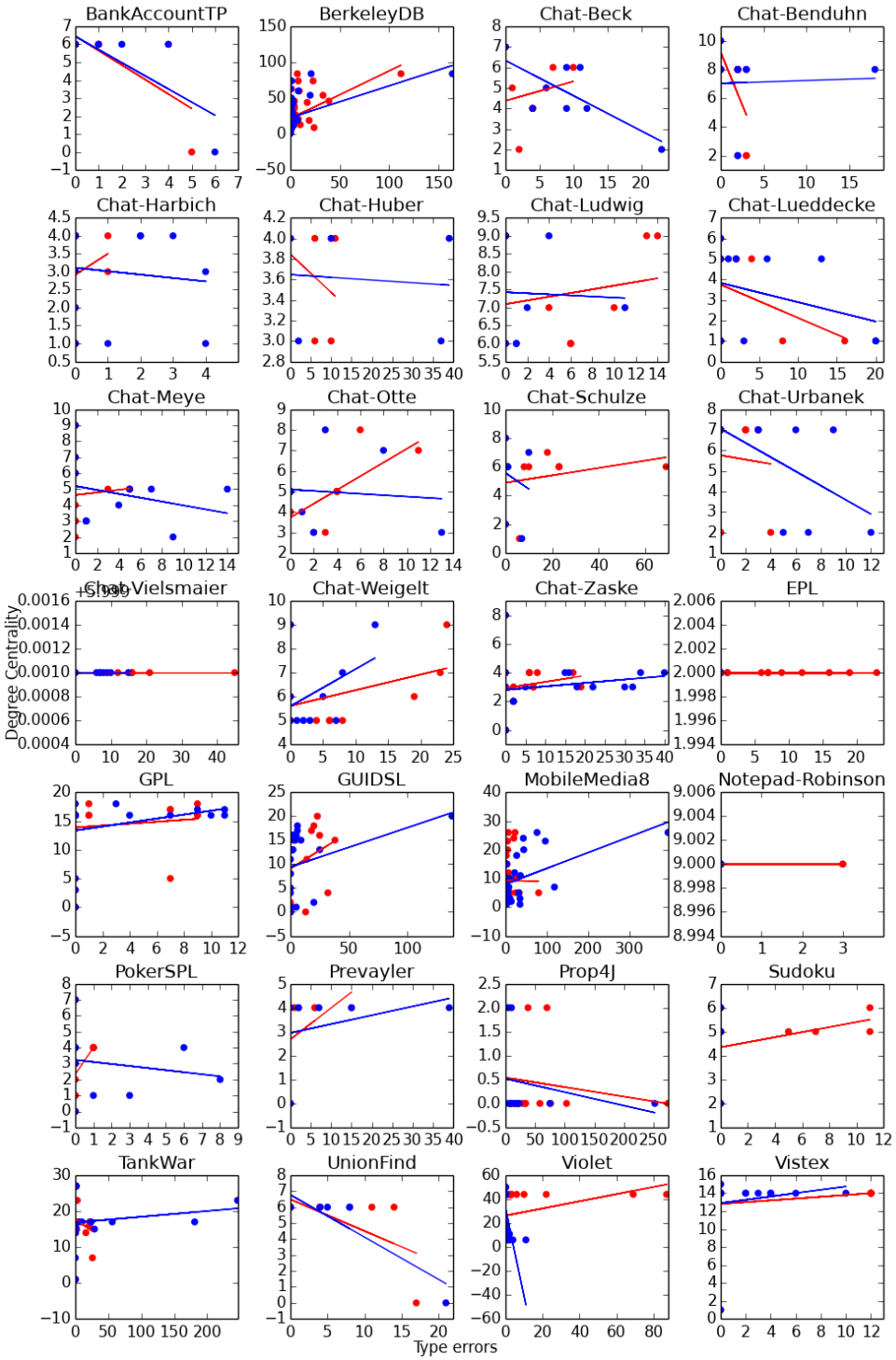

Figure 28: Correlation between type errors and *deg* in G_{intros} (red = SPL type errors, blue = feature local type errors)

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	0.013	0.222	0.075
BerkeleyDB	**0.326** ***	**0.214** *	**0.320** **
Chat-Beck	-0.596	-0.201	-0.158
Chat-Benduhn	-0.243	-0.532	-0.445
Chat-Harbich	0.255	-0.358	-0.331
Chat-Huber	-0.105	-0.051	-0.051
Chat-Ludwig	-0.218	-0.131	-0.201
Chat-Lueddecke	-0.071	-0.409	-0.425
Chat-Meye	0.295	-0.281	-0.196
Chat-Otte	-0.065	0.596	0.174
Chat-Schulze	0.438	-0.098	0.282
Chat-Urbanek	0.062	-0.048	-0.077
Chat-Vielsmaier	-0.463	-0.312	-0.611
Chat-Weigelt	0.220	0.154	0.210
Chat-Zaske	-0.240	-0.162	-0.140
EPL	**0.657** *	-	**0.657** *
GPL	0.263	**0.525** *	0.389
GUIDSL	**0.424** *	**0.493** *	**0.515** **
MobileMedia8	**0.397** **	**0.536** ***	**0.520** ***
Notepad-Robinson	0.435	-	0.435
PokerSPL	**-0.731** *	0.270	-0.150
Prevayler	0.406	0.783	**0.829** *
Prop4J	0.519	-0.074	0.245
Sudoku	0.353	-	0.353
TankWar	**0.436** *	**0.416** *	**0.511** **
UnionFind	-0.382	-0.349	-0.379
Violet	**0.448** ***	**-0.837** ***	**-0.253** *
Vistex	0.014	-0.112	-0.034

Table 22: Correlation between type errors and *deg* in G_{refs}

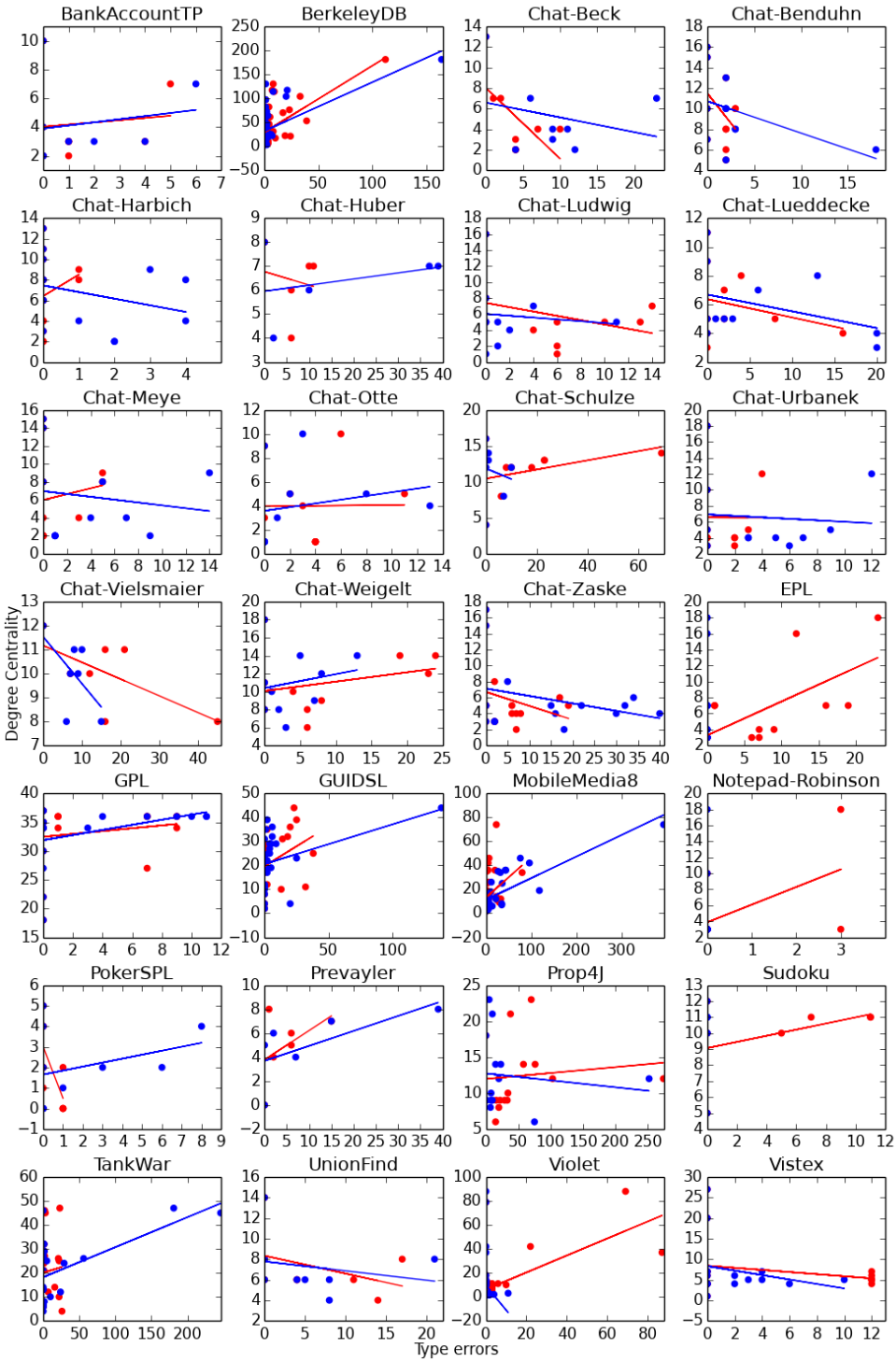

Figure 29: Correlation between type errors and *deg* in G_{refs} (red = SPL type errors, blue = feature local type errors)

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	**-0.803** *	**-0.712** *	**-0.783** *
BerkeleyDB	**0.339** ***	**0.227** *	**0.340** ***
Chat-Beck	-0.038	-0.255	-0.037
Chat-Benduhn	**-0.796** **	-0.392	-0.569
Chat-Harbich	0.360	-0.353	-0.311
Chat-Huber	0.081	0.158	0.158
Chat-Ludwig	-0.525	-0.135	-0.456
Chat-Lueddecke	-0.186	-0.475	-0.481
Chat-Meye	0.211	-0.464	-0.350
Chat-Otte	0.365	0.060	0.020
Chat-Schulze	0.383	0.195	0.398
Chat-Urbanek	0.296	-0.214	-0.139
Chat-Vielsmaier	0.019	**-0.810** *	-0.117
Chat-Weigelt	0.279	0.188	0.263
Chat-Zaske	0.057	0.136	0.147
EPL	**0.701** *	-	**0.701** *
GPL	**0.577** **	**0.663** **	**0.641** **
GUIDSL	**0.393** *	**0.545** **	**0.509** **
MobileMedia8	0.211	**0.426** **	**0.348** *
Notepad-Robinson	0.559	-	0.559
PokerSPL	0.186	-0.203	-0.209
Prevayler	0.154	0.739	**0.820** *
Prop4J	0.519	-0.074	0.245
Sudoku	0.462	-	0.462
TankWar	**0.502** **	**0.499** **	**0.565** **
UnionFind	-0.605	-0.605	-0.601
Violet	**0.591** ***	**-0.858** ***	-0.151
Vistex	-0.484	**-0.536** *	**-0.521** *

Table 23: Correlation between type errors and *deg* in G_{PFI}

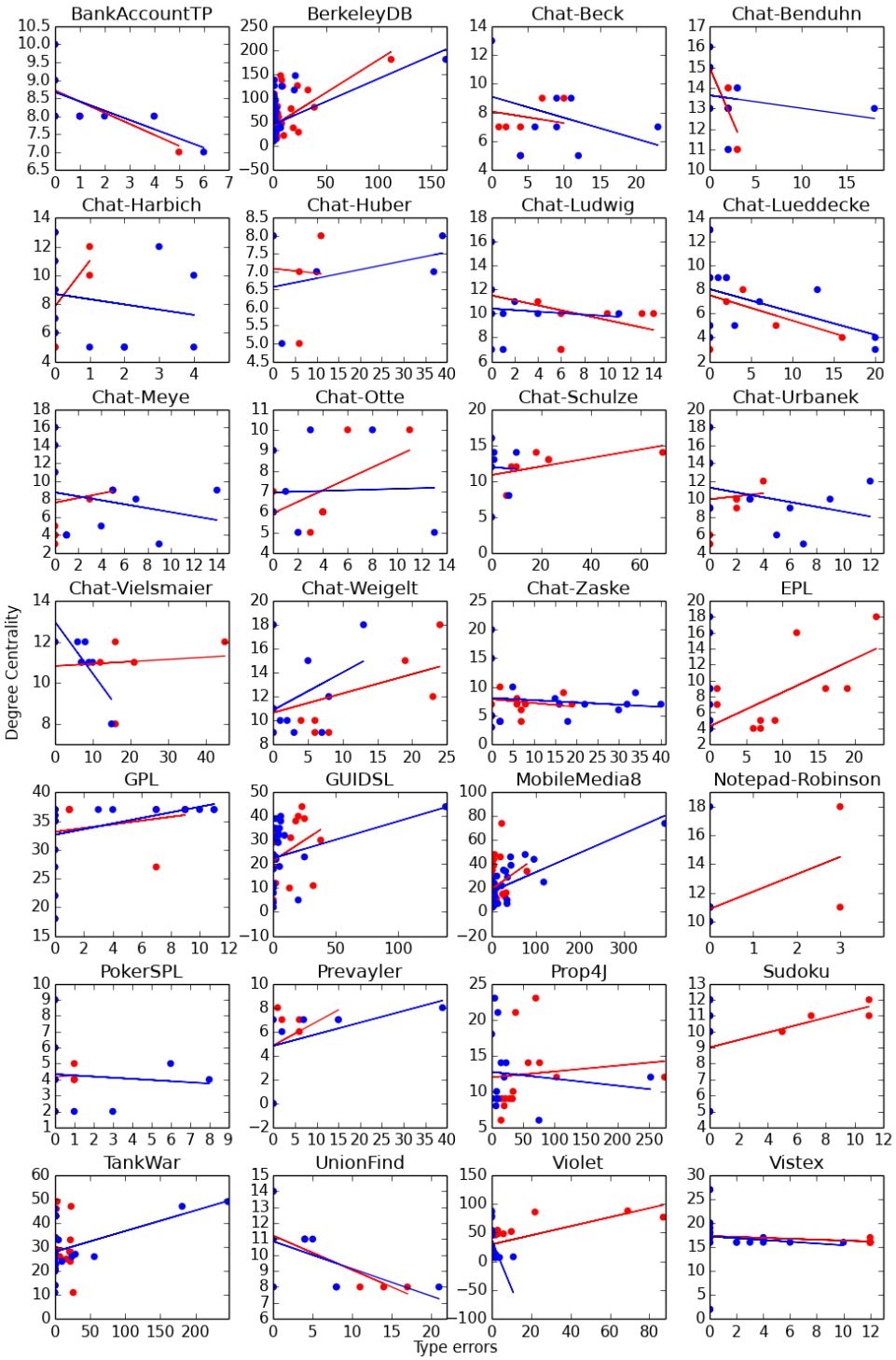

Figure 30: Correlation between type errors and *deg* in G_{PFI} (red = SPL type errors, blue = feature local type errors)

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.125	-0.038	-0.146
BerkeleyDB	**0.346** ***	**0.391** ***	**0.434** ***
Chat-Beck	-0.417	-0.446	-0.287
Chat-Benduhn	**-0.752** **	-0.297	-0.422
Chat-Harbich	0.304	-0.235	-0.208
Chat-Huber	-0.154	-0.100	-0.100
Chat-Ludwig	-0.075	0.026	-0.043
Chat-Lueddecke	-0.030	-0.256	-0.223
Chat-Meye	0.281	-0.365	-0.258
Chat-Otte	-0.167	0.531	0.159
Chat-Schulze	0.008	-0.128	0.000
Chat-Urbanek	0.080	-0.374	-0.315
Chat-Vielsmaier	-0.180	-0.357	-0.342
Chat-Weigelt	0.234	0.215	0.239
Chat-Zaske	-0.150	0.058	0.048
EPL	0.400	-	0.400
GPL	**0.513** *	**0.779** ***	**0.667** **
GUIDSL	0.365	0.228	**0.414** *
MobileMedia8	0.220	**0.646** ***	**0.530** ***
Notepad-Robinson	**0.701** *	-	**0.701** *
PokerSPL	-0.288	0.007	-0.167
Prevayler	0.406	0.783	**0.829** *
Prop4J	0.508	0.304	**0.543** *
Sudoku	0.430	-	0.430
TankWar	**0.629** ***	**0.430** *	**0.683** ***
UnionFind	0.012	0.012	0.024
Violet	**0.377** ***	**-0.847** ***	**-0.329** **
Vistex	0.379	0.194	0.325

Table 24: Correlation between type errors and deg_w in G_{PFI}

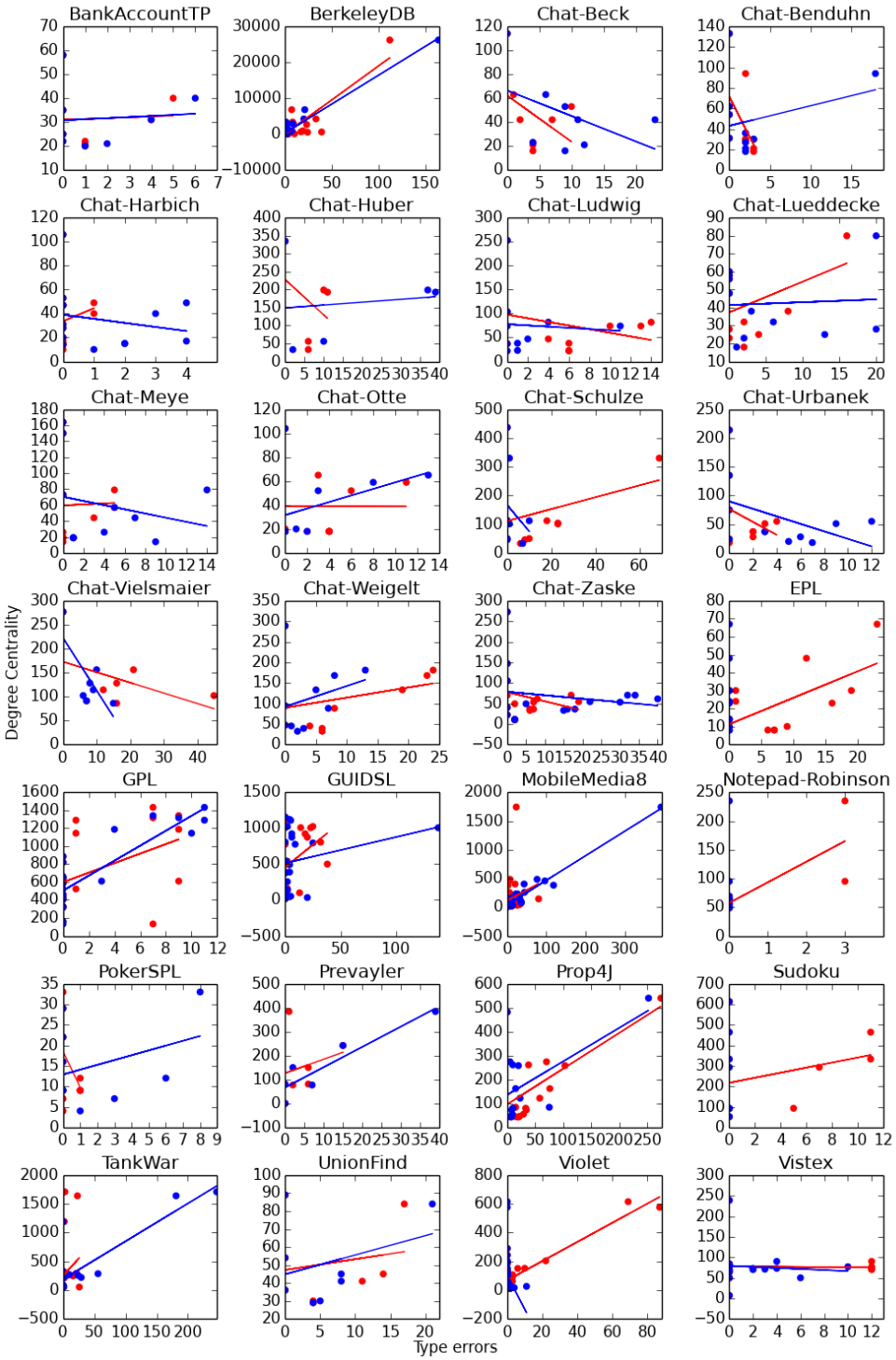

Figure 31: Correlation between type errors and deg_w in G_{PFI} (red = SPL type errors, blue = feature local type errors)

A.3.3.2 Degree Centrality with Dependency-Weight

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.478	-0.447	-0.491
BerkeleyDB	**0.318** **	**0.219** *	**0.313** **
Chat-Beck	-0.333	-0.479	-0.398
Chat-Benduhn	**-0.665** *	-0.350	-0.523
Chat-Harbich	0.203	-0.482	-0.442
Chat-Huber	0.289	0.359	0.359
Chat-Ludwig	-0.243	-0.205	-0.294
Chat-Lueddecke	0.030	-0.312	-0.307
Chat-Meye	0.214	-0.384	-0.304
Chat-Otte	0.235	-0.046	-0.137
Chat-Schulze	0.077	0.023	0.094
Chat-Urbanek	0.076	-0.296	-0.241
Chat-Vielsmaier	-0.445	0.055	-0.556
Chat-Weigelt	0.333	0.265	0.308
Chat-Zaske	0.142	0.118	0.141
EPL	**0.968** ***	-	**0.968** ***
GPL	0.091	0.015	0.070
GUIDSL	0.276	**0.602** **	**0.517** **
MobileMedia8	**0.308** *	**0.487** ***	**0.439** **
Notepad-Robinson	0.435	-	0.435
PokerSPL	0.144	0.055	0.032
Prevayler	0.638	0.725	0.771
Prop4J	**0.659** *	0.054	0.385
Sudoku	**0.905** **	-	**0.905** **
TankWar	**0.462** *	**0.586** ***	**0.600** ***
UnionFind	**-0.877** **	**-0.889** **	**-0.871** **
Violet	**0.604** ***	**-0.832** ***	-0.111
Vistex	-0.309	-0.311	-0.288

Table 25: Correlation between type errors and deg_{dep} in G_{PFI}

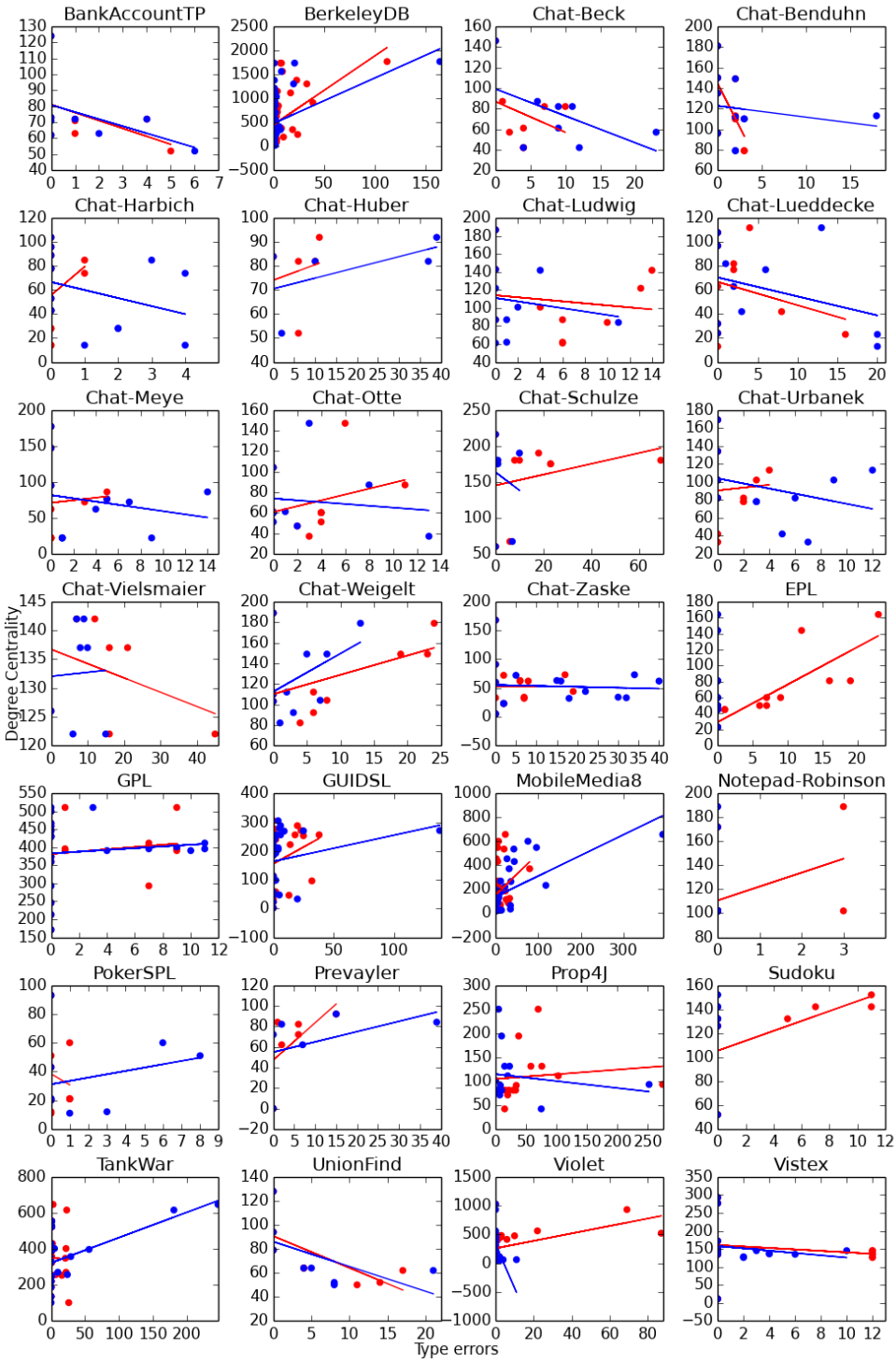

Figure 32: Correlation between type errors and deg_{dep} in G_{PFI} (red = SPL type errors, blue = feature local type errors)

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.025	-0.114	-0.122
BerkeleyDB	**0.351** ***	**0.324** **	**0.409** ***
Chat-Beck	-0.220	-0.431	-0.190
Chat-Benduhn	**-0.618** *	-0.167	-0.311
Chat-Harbich	0.456	-0.242	-0.193
Chat-Huber	-0.154	-0.100	-0.100
Chat-Ludwig	0.068	-0.013	0.012
Chat-Lueddecke	0.244	0.042	0.074
Chat-Meye	0.264	-0.215	-0.173
Chat-Otte	0.248	0.229	0.242
Chat-Schulze	0.008	-0.128	0.000
Chat-Urbanek	0.080	-0.354	-0.296
Chat-Vielsmaier	-0.450	-0.143	-0.577
Chat-Weigelt	0.363	0.362	0.374
Chat-Zaske	0.150	0.214	0.229
EPL	**0.775** **	-	**0.775** **
GPL	**0.569** **	**0.829** ***	**0.714** ***
GUIDSL	**0.443** *	0.186	**0.462** *
MobileMedia8	**0.298** *	**0.652** ***	**0.582** ***
Notepad-Robinson	**0.696** *	-	**0.696** *
PokerSPL	-0.216	0.083	-0.077
Prevayler	0.319	0.667	0.714
Prop4J	**0.705** **	0.348	**0.644** *
Sudoku	0.599	-	0.599
TankWar	**0.653** ***	**0.479** **	**0.694** ***
UnionFind	-0.135	-0.160	-0.122
Violet	**0.486** ***	**-0.839** ***	**-0.221** *
Vistex	0.252	0.112	0.222

Table 26: Correlation between type errors and $deg_{w,dep}$ in G_{PFI}

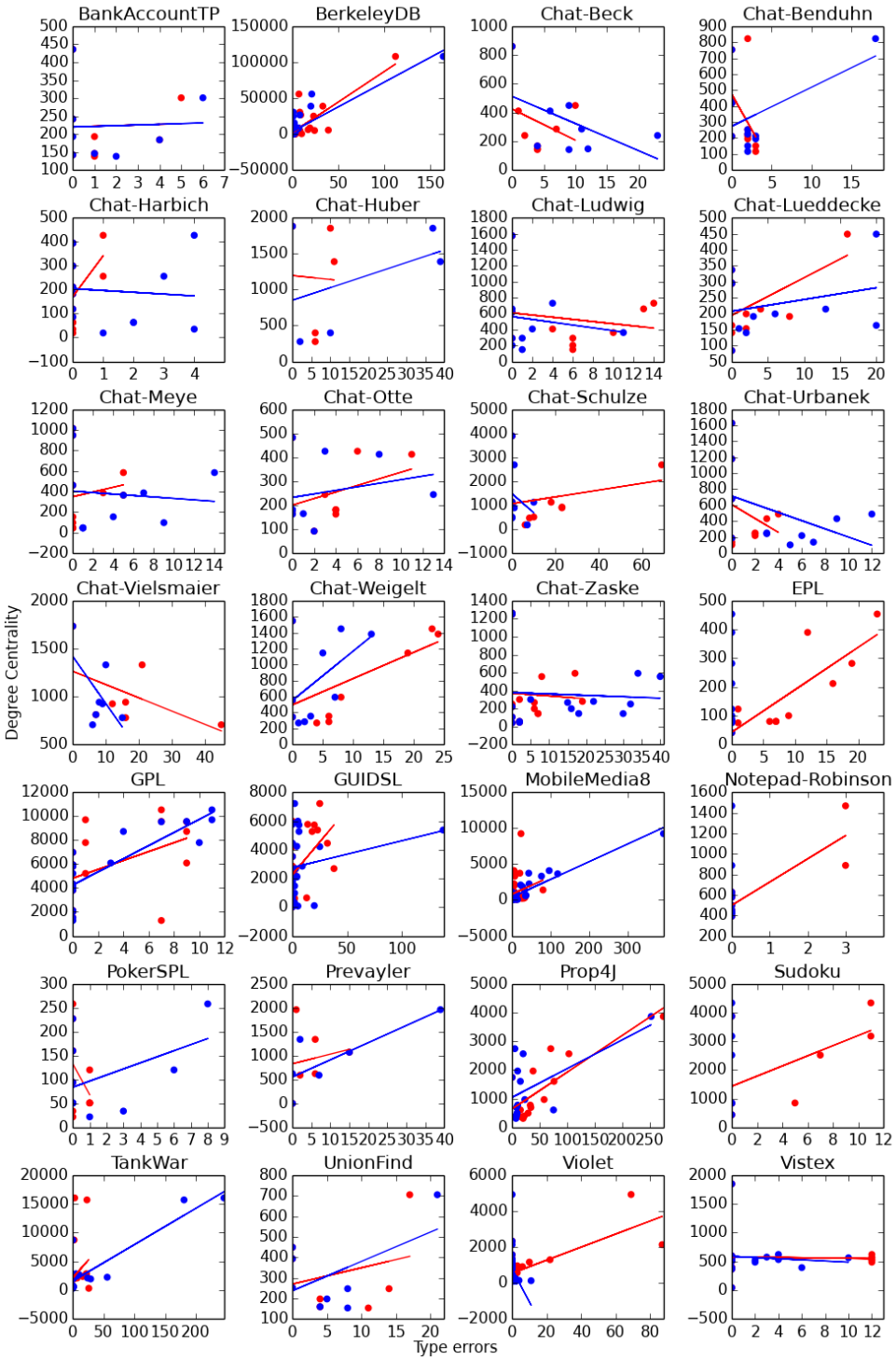

Figure 33: Correlation between type errors and $deg_{w,dep}$ in G_{PFI} (red = SPL type errors, blue = feature local type errors)

A.3.3.3 Betweenness Centrality for Edges

	G_{PFI}	G_{intros}	G_{refs}
BankAccountTP	-0.096	**-0.321** *	**0.346** **
BerkeleyDB	0.005	**0.080** ***	0.006
Chat-Beck	**0.361** **	-0.018	-0.186
Chat-Benduhn	0.116	**-0.322** ***	**0.215** *
Chat-Harbich	-0.048	0.137	0.121
Chat-Huber	-0.031	-0.035	0.281
Chat-Ludwig	**0.235** *	0.192	-0.149
Chat-Lueddecke	0.157	0.173	**0.258** **
Chat-Meye	0.114	0.075	0.081
Chat-Otte	**0.380** ***	**0.412** ***	-0.038
Chat-Schulze	0.099	-0.096	0.139
Chat-Urbanek	**0.300** **	-0.177	**0.260** **
Chat-Vielsmaier	-0.010	-0.007	0.154
Chat-Weigelt	0.108	0.028	0.107
Chat-Zaske	**0.474** ***	**0.446** ***	-0.022
EPL	**0.698** ***	0.047	**0.694** ***
GPL	0.009	0.085	**0.172** ***
GUIDSL	**0.079** *	0.023	0.037
MobileMedia8	**0.234** ***	**0.073** **	**0.235** ***
Notepad-Robinson	-0.006	0.096	-0.063
PokerSPL	**0.249** *	**0.239** *	-0.066
Prevayler	0.110	0.237	**0.569** **
Prop4J	**0.792** ***	-0.132	**0.684** ***
Sudoku	-0.025	0.230	-0.106
TankWar	0.012	0.003	**0.082** *
UnionFind	**0.385** **	-0.116	0.169
Violet	0.016	**0.069** ***	**0.169** ***
Vistex	-0.067	**-0.421** ***	**-0.197** **

Table 27: Correlation between SPL type errors and edge betweenness centrality (c_B) in G_{intros}, G_{refs} and G_{PFI}

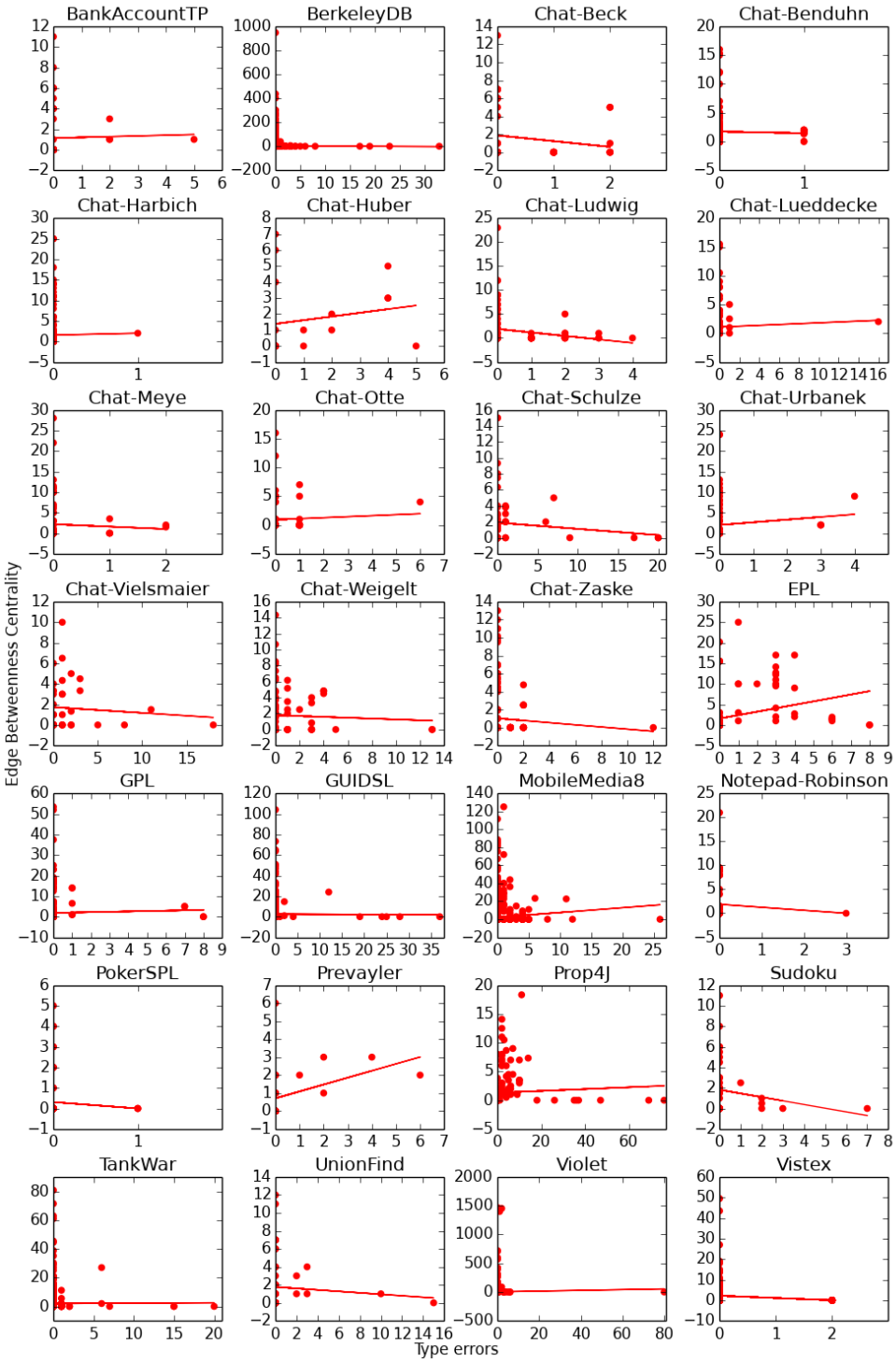

Figure 34: Correlation between type errors and edge betweenness centrality (c_B) in G_{refs}

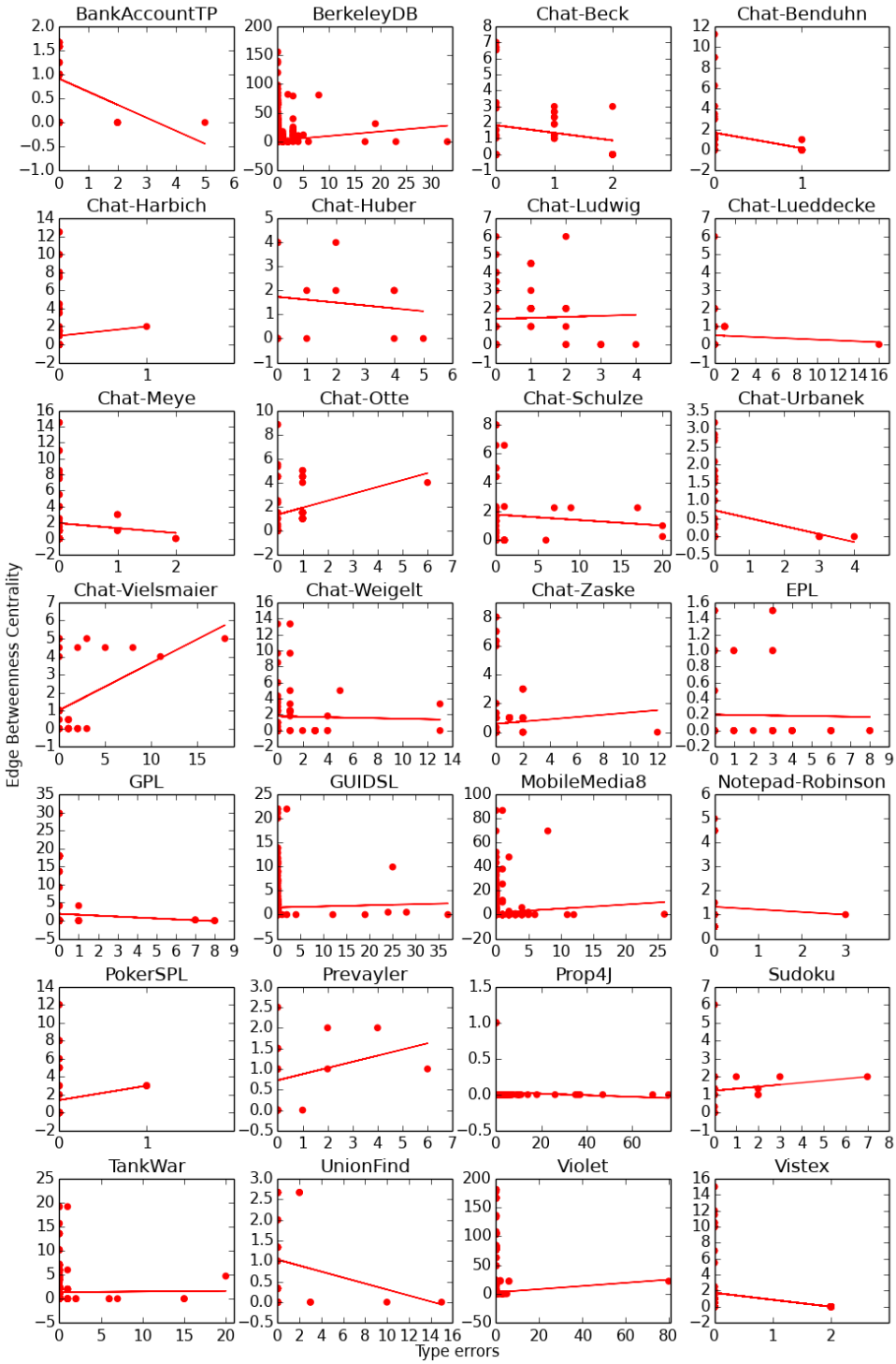

Figure 35: Correlation between type errors and edge betweenness centrality (c_B) in G_{intros}

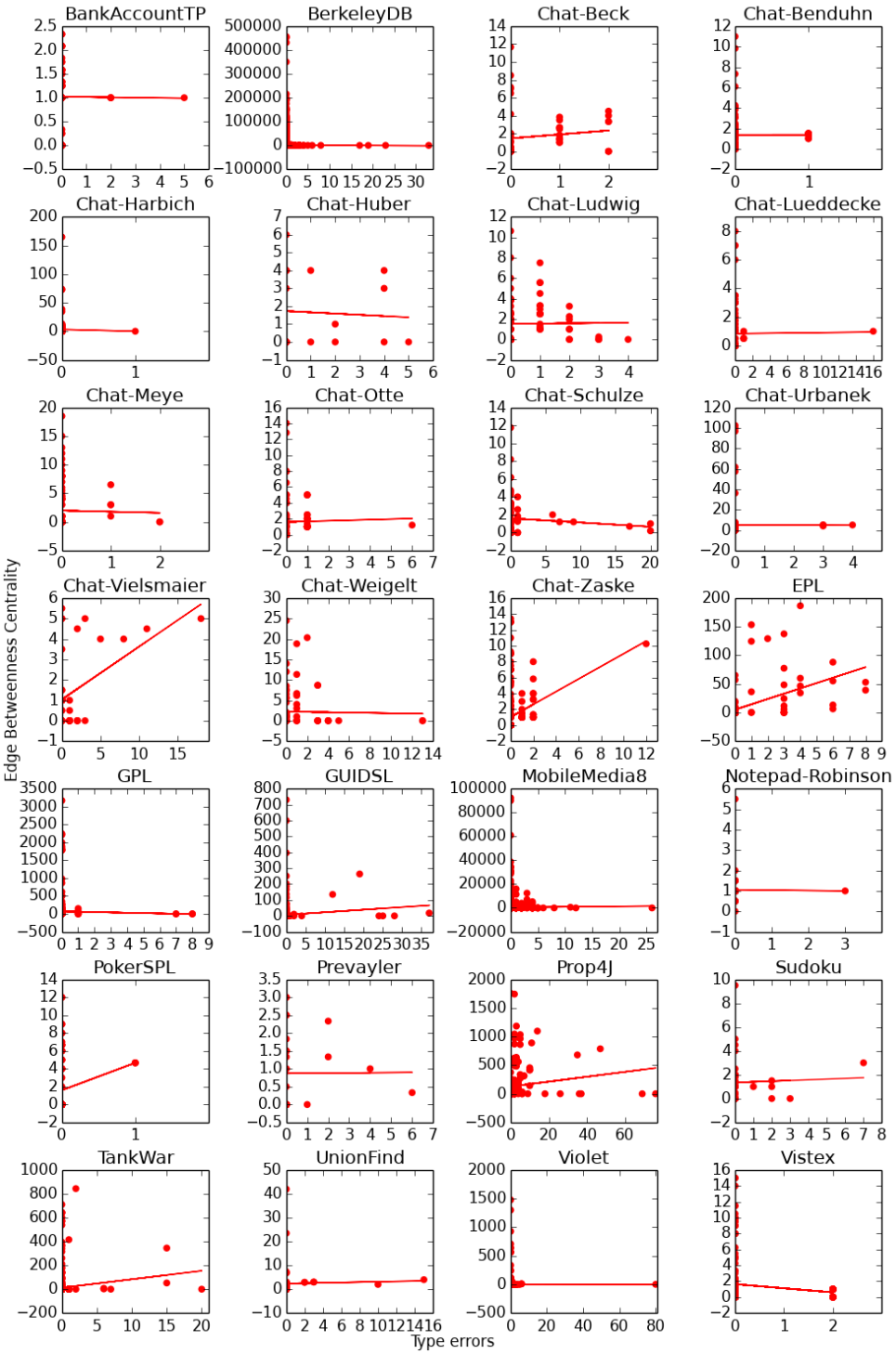

Figure 36: Correlation between type errors and edge betweenness centrality (c_B) in G_{PFI}

A.3.4 Results for Correlation between Code Fragmentation and Type Errors

A.3.4.1 Fragmentation of Classes

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.866	-0.500	-0.500
BerkeleyDB	**0.526** ***	**0.658** ***	**0.671** ***
Chat-Beck	**0.950** **	**0.885** *	**0.885** *
Chat-Benduhn	0.475	**0.820** ***	**0.827** ***
Chat-Harbich	0.349	**0.502** *	**0.506** *
Chat-Huber	**0.538** **	**0.763** ***	**0.808** ***
Chat-Ludwig	**0.859** **	**0.959** ***	**0.982** ***
Chat-Lueddecke	**0.617** *	**0.904** ***	**0.904** ***
Chat-Meye	0.427	**0.674** *	**0.688** *
Chat-Otte	**0.764** *	**0.686** *	**0.844** **
Chat-Schulze	0.482	**0.672** *	0.559
Chat-Urbanek	**0.355** *	**0.575** ***	**0.575** ***
Chat-Vielsmaier	0.340	**0.603** **	**0.509** *
Chat-Weigelt	**0.680** *	-0.073	0.454
Chat-Zaske	**0.999** ***	**0.726** ***	**0.740** ***
EPL	-	-	-
GPL	**0.798** ***	**0.714** **	**0.837** ***
GUIDSL	**0.412** ***	0.153	**0.355** ***
MobileMedia8	**0.486** ***	**0.599** ***	**0.651** ***
Notepad-Robinson	0.488	-	0.488
PokerSPL	0.619	**0.966** ***	**0.966** ***
Prevayler	**0.284** ***	0.142	0.126
Prop4J	0.034	0.069	-0.034
Sudoku	0.240	-	0.240
TankWar	**0.791** ***	0.090	**0.473** *
UnionFind	-0.258	0.775	0.544
Violet	-0.021	**0.763** ***	**0.460** ***
Vistex	0.667	0.667	0.667

Table 28: Correlation between type errors and fragmentation of classes ($frag_c$)

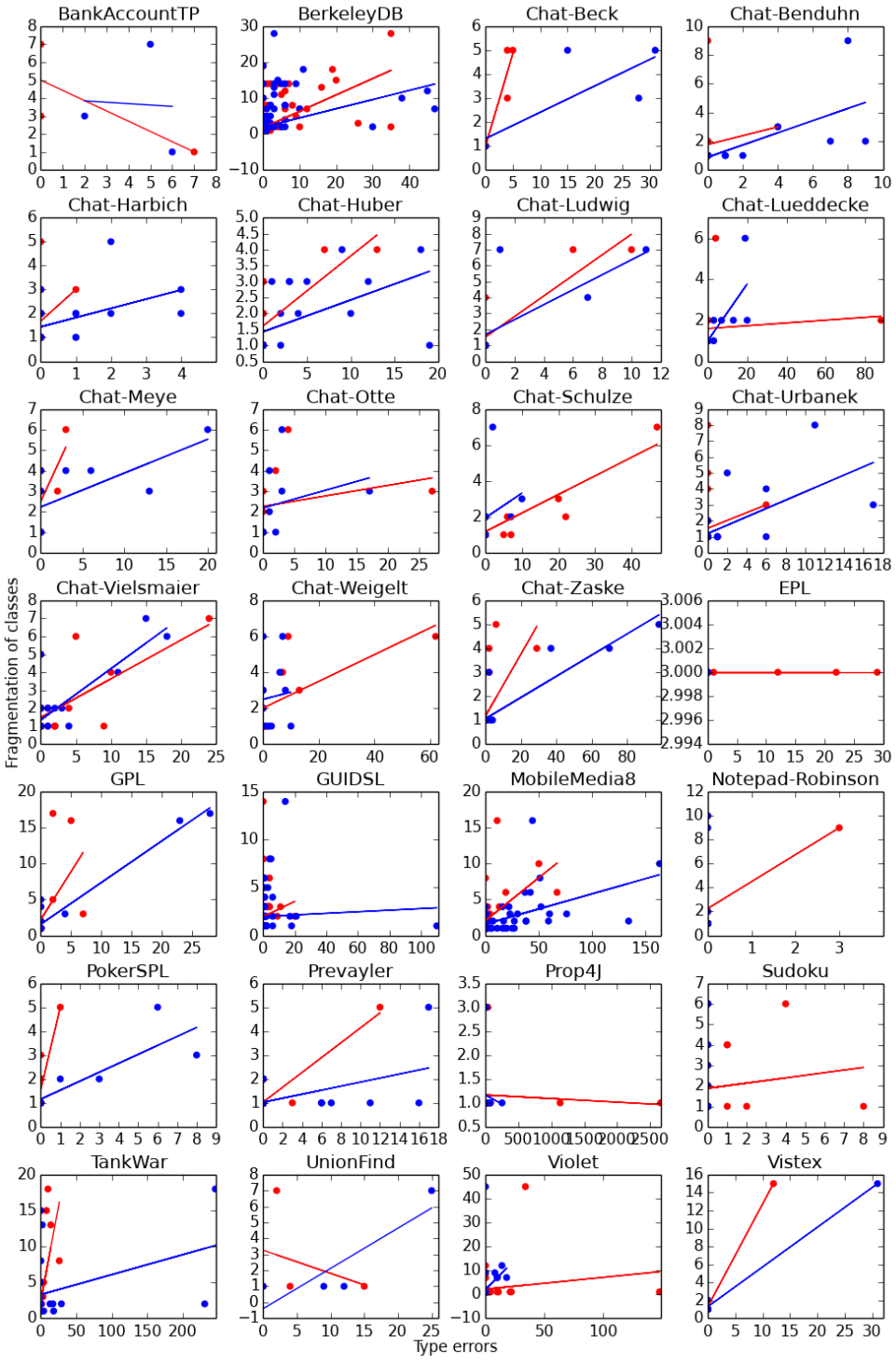

Figure 37: Correlation between type errors and fragmentation of classes ($frag_c$) (red = SPL type errors, blue = feature local type errors)

A.3.4.2 Fragmentation of Features

Name	SPL type errors	FL type errors	All type errors
BankAccountTP	-0.296	-0.180	-0.289
BerkeleyDB	**0.366** ***	**0.356** ***	**0.393** ***
Chat-Beck	-0.187	-0.505	-0.365
Chat-Benduhn	-0.525	-0.066	-0.228
Chat-Harbich	0.479	-0.062	-0.010
Chat-Huber	-0.105	-0.051	-0.051
Chat-Ludwig	-0.052	-0.149	-0.071
Chat-Lueddecke	0.290	-0.279	-0.157
Chat-Meye	0.180	-0.355	-0.288
Chat-Otte	-0.257	0.529	0.122
Chat-Schulze	0.319	-0.141	0.189
Chat-Urbanek	-0.344	**-0.677** *	**-0.648** *
Chat-Vielsmaier	0.128	-0.109	0.138
Chat-Weigelt	0.361	0.290	0.360
Chat-Zaske	-0.411	**-0.522** *	-0.492
EPL	0.280	-	0.280
GPL	0.200	0.158	0.227
GUIDSL	**0.433** *	0.204	**0.473** *
MobileMedia8	**0.377** *	**0.734** ***	**0.662** ***
Notepad-Robinson	0.246	-	0.246
PokerSPL	**-0.645** *	-0.016	-0.316
Prevayler	0.058	0.754	0.771
Prop4J	-0.152	**-0.558** *	-0.253
Sudoku	0.372	-	0.372
TankWar	**0.413** *	**0.379** *	**0.582** ***
UnionFind	0.595	0.595	0.592
Violet	**0.247** *	**-0.280** **	0.007
Vistex	0.148	-0.040	0.046

Table 29: Correlation between type errors and and fragmentation of features ($frag_f$)

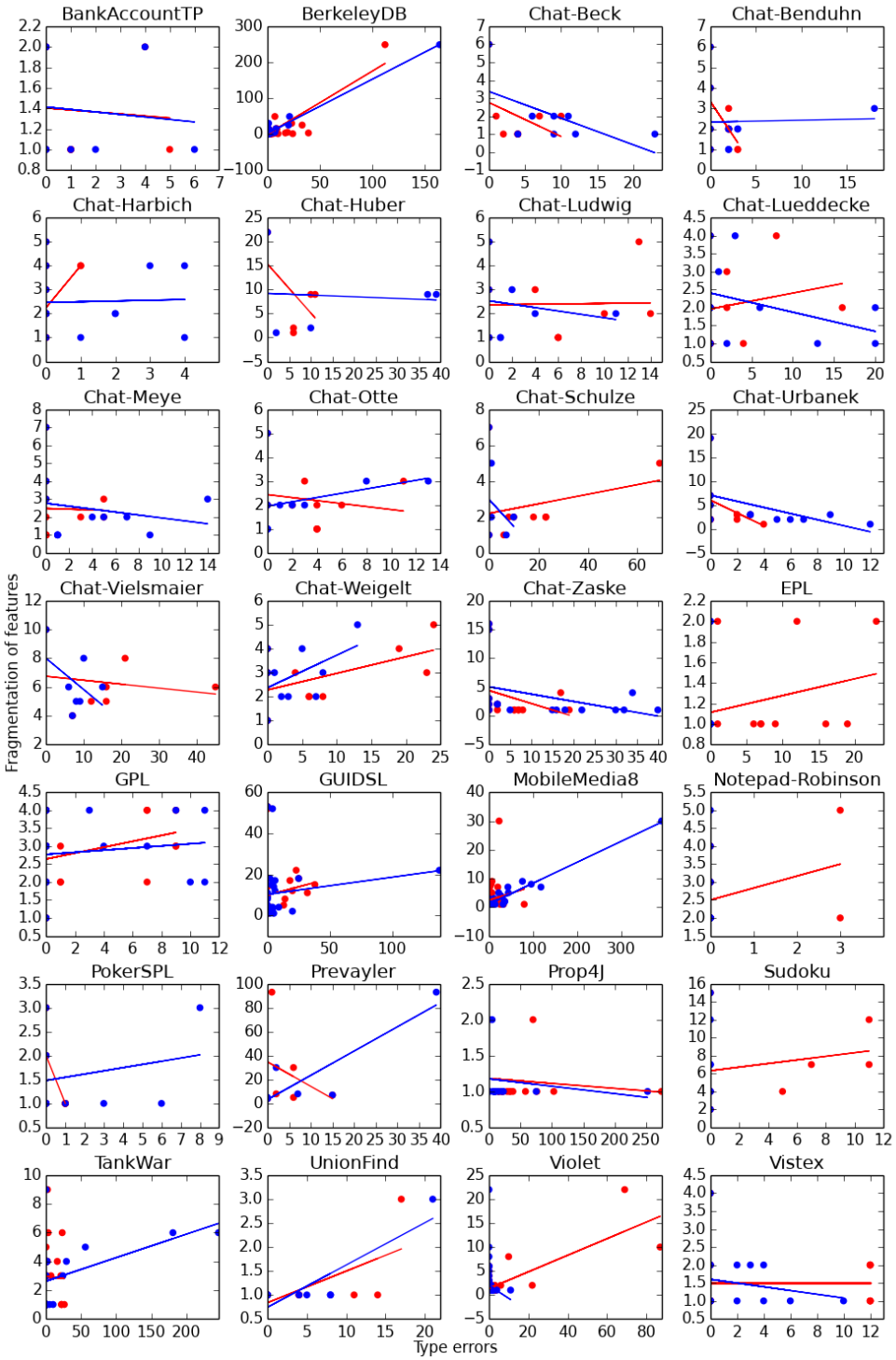

Figure 38: Correlation between type errors and fragmentation of features ($frag_f$) (red = SPL type errors, blue = feature local type errors)

Eidesstattliche Erklärung: Hiermit versichere ich an Eides statt, dass ich diese Masterarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Judith Roth,
Passau, 21. August 2014