university of groningen / faculty of science and engineering

UNIVERSITY OF GRONINGEN

BACHELOR'S THESIS

Measuring Parent Centrality in Java Packages

Author:
Theodosios
ATHANASAKIS

Supervisors:
Prof. dr. ir. Paris
AVGERIOU,
Darius SAS

A thesis submitted in fulfillment of the requirements for the degree of Bachelor Degree in Computing Science

in the

Faculty of Science and Engineering

Department of Computing Science

June 4, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As time passes, software programs grow ever more complicated and harder to be monitored manually. This means that in programs containing millions of lines of code, it gets harder to perform regular maintenance activities such as finding and fixing bugs, or extending the program with new features. To simplify this process, software developers strive to keep the code tidy and documented. In Java programs, one very common activity is to group the source code files that share similar responsibilities, called classes, into directories, called packages. Classes, and therefore the packages containing them, to properly function, use the functionalities provided by other classes and packages. When a class A uses another class B, we call this relationship a dependency between class A and class B, namely A depends on B. Software design principles suggest that "The dependencies between packages must not form cycles" [1]. This means that the packages should form a *Directed Acyclic Graph (DAG)*. Even if this is the main guidance for quality programs, most projects contain such cycles. This happens either because of the large number of programmers that work on a module or program or because of the deliberate (or not) oversight of these principles. These abnormalities are called "Architectural Smells". Furthermore, the dependencies that form circles, are known as "Cyclic Dependencies". To measure the severity of a smell like this in the code, we use certain metrics.

However, *are all Cyclic Dependencies crucial, and how common are the non-critical Cyclic Dependencies in a program*? To answer the question, we will implement one of the metrics applying in such smells, called *Parent Centrality*, inside an existing tool, ASTracker [2].

The implementation of the metric is based on the definition of the paper "On the Shape of Circular Dependencies in Java Programs" by H. A. Al-Mutawa et al. [3], who state: "For a given fixed tangle (Cyclic Dependency) (V, E), [...] we define two sets of edges. Let E_p be the set of edges [...] pointing from a child to a parent package. Formally, let E_p^+ be the set of edges pointing from a child to parent package, and where the parent package has a higher Betweenness Centrality than the child package. We then call the ratio $E_p = E_p^+ / E_p$ the Parent Centrality of a tangle (Cyclic Dependency).". In other words, it measures at what amount a parent package is a hub among its children inside the smell.

The importance of the metric is, as H. A. Al-Mutawa et al. suggest in the same paper, "[...] we can consider [...] (parent-children shape) as non-critical. If we were able to show that there is an abundance of tangles (smells) like this, this would explain why so many widely-used real-world programs are riddled with Cyclic Dependencies.". This means that if a program has a high Parent Centrality value, many of the Architectural Smells of that program may not be critical. Moreover, in conjunction with 21 real-world projects, we are trying to analyze the results and, also, discover if Parent Centrality correlates with other metrics (see Chapter 3).

Chapter 2

Related Work and Background

This chapter describes the background knowledge necessary to understand the key concepts mentioned in the thesis, as well as some metrics similar to Parent Centrality. First, Section 2.1 introduces the definition of a graph and some of its characteristics. Next, Section 2.2 investigates the term "Dependency Graph" and the concepts that the paper uses, like "Cyclic Dependencies". Also, Section 2.3 focuses on other similar metrics in such dependencies. Moreover, Section 2.4 brings in the basic idea behind the tool that is used to extract this metric, "ASTracker". Lastly, Section 2.5 analyzes the mathematical definition of the Parent Centrality metric.

2.1 Graph

The science behind graphs is called *Graph Theory*. Before we explain the term "graph", though, we have to define the terms "vertex" and "edge". To do that, we will use the definition of John W. Essam and Michael E. Fisher [4] where they determine:

> A *vertex* set **V** is a set of objects a, b, c, [...] called *vertices*. [...]

and

> An *edge* [...] is an unordered pair of distinct vertices from a vertex set **V**. The edge **(i,j)** [...] with the vertices i and j [...] connect them. An edge may be represented by a continuous line connecting the points i and j.

With the above descriptions of "vertex" and "edge" in mind, we can easily define now the term "graph" as:

> An undirected graph $G = (V, E)$ is a vertex set **V**, having at least one member, together with an associated edge set **E**.

A good way to understand graphs is to think of them as towns connected by roads, representing the vertices and edges, respectively. In this paper, we only use directed graphs, which means that the edges have direction, i.e. one-way roads.

One interesting characteristic in graphs is the presence, or absence, of cycles inside a connected sub-graph [5]. Cycles can manifest themselves in multiple shapes, as described in the next section.

2.1.1 Shapes of Cycles

There are two types of cyclic shapes, symmetric and asymmetric [5]. The symmetric ones are the: (i) tiny (Section 2.1.1.1), (ii) clique (Section 2.1.1.2), (iii) circle (Section 2.1.1.3), (iv) chain (Section 2.1.1.4) and (v) star (Section 2.1.1.5). Also, the asymmetric ones are the: (i) multi-hub and (ii) semi-clique. In this paper, only the symmetric ones are used in the analysis of the results in Section 3.4 because ASTracker tool recognize only those. For that reason, we will focus on them. All symmetrical shapes can be found in Figure 2.1, originally used by H. A. Al-Mutawa et al. [3].

2.1.1.1 Tiny

This shape is the simplest and consists of two vertices where both have an edge towards the other.

2.1.1.2 Clique

This shape is an expansion of the tiny one that described before (see Section 2.1.1.1). There are more than two vertices, and the maximum shortest path between a pair of them is 1. This means that every vertex has an edge towards every other vertex in the topology.

2.1.1.3 Circle

In this shape, there are only edges that compose a cyclic path, i.e. start and endpoint is the same vertex.

2.1.1.4 Chain

A shape like this reminds us of a real chain's design. This means that it is a sequential structure where the vertices are connected in both directions. In some cases, the first and last vertices are, also, connected [3]. In other words, it is like having towns (vertices) in a line, and every town has a road (edge) towards every one of its neighbors.

2.1.1.5 Star

This shape has the characteristic of one central vertex. Every other vertex forms a tiny circle with the central. Furthermore, the only path between two vertices includes the central one.

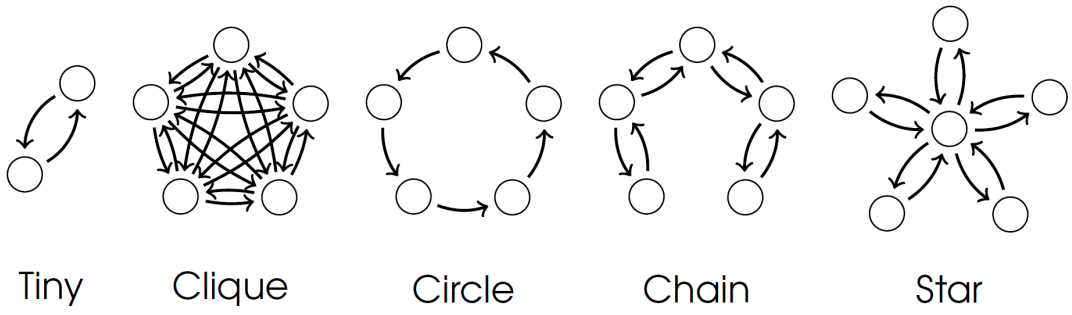


FIGURE 2.1: Symmetric Shapes. Originally used by H. A. Al-Mutawa et al. [3]

2.1.2 Betweenness Centrality

There exist several metrics that can be computed inside a graph. The Betweenness Centrality metric measures the importance of a vertex by the view of trails. This means that a vertex has high Betweenness Centrality when it appears in a large number of shortest paths between pair of vertices inside the graph [6].

2.2 Dependency Graph

A *Dependency Graph* is a type of graph for representing the dependency relationship between software artifacts. Specifically, when a class or package A uses another class or package B, we call this relationship a dependency between class A and class B, namely A depends on B. In the dependency graph, every vertex represents a class or a package, and the edges form the dependencies between them. If one class/package A depends on another class/package B, there is an edge from A to B.

2.2.1 Architectural Smells

Architectural Smells: issues in the architecture that often cause extra maintenance effort [7]. They have some characteristics and shapes inside the graph. They are not bugs, but they increase the cognitive complexity, change proneness, and cohesion of the affected artifacts. Some of them are: "Unstable Dependencies", "Hublike Dependencies" and "Cyclic Dependencies" [2]. In this thesis, we only focus on Cyclic Dependencies.

2.2.2 Cyclic Dependencies

Cyclic Dependencies are architectural smells that create circular shapes among several components [2]. Software engineers divide these shapes into different categories. Section 2.1.1 describes some of the categories that Al-Mutawa et al. identifies [3] and this paper uses.

2.3 Related Work

There are lots of studies analyzing architectural smells and circular dependencies and their characteristics. Some of them confirm the reason why cyclic dependencies are harmful and should be avoided from a program [1], [8], [9], [10], or explore the relation between different circular smells inside Java projects [11]. Also, many papers are trying to make a catalogue of bad smells and how to identify them [12], [13].

Many papers introduce tools for detecting or analyzing kinds of smells [14], [15], [16], [17]. A study by Lenhard Jörg et al. in 2017, though, suggests that existing code smell detection techniques, as implemented in contemporary tools, are not sufficiently accurate for classifying whether a class contains architectural inconsistencies, even when combining categories of code smells [18].

One of the existing tools analyzing and calculating the metric about smells is ASTracker [2]. More information about the tool can be found in Section 2.4. As ASTracker grows, more and more metrics are included. This thesis, if met, will contribute to the implementation of the "Parent Centrality" metric, which can then be added as a new feature to ASTracker.

2.4 ASTracker

ASTracker[1] is a tool that is designed for calculating several metrics computed on with Architectural Smells. The input of the tool is the GraphML files of all the versions of a project produced by Arcan [15]. Then, for every smell from a version, the tool calculates the smell characteristics and maps every smell in each version to its closest successor in the next version. Finally, it returns the results as CSV and GraphML files [2] (see Figure 2.2).

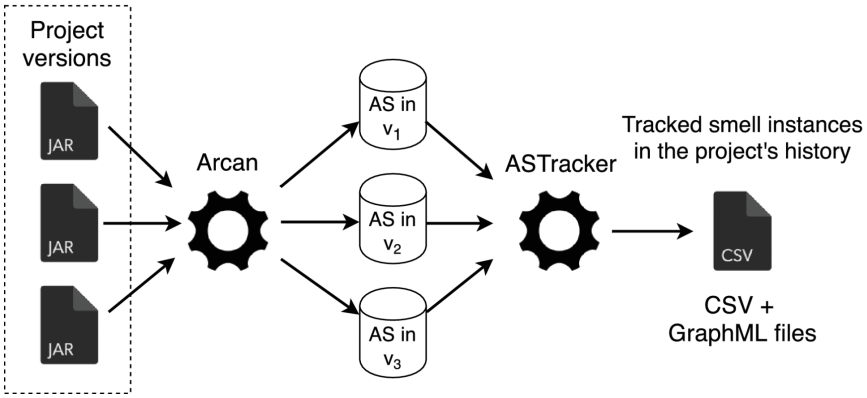


FIGURE 2.2: Data collection process and tooling. The data of each individual project was then merged in a single data set. Originally used by D. Sas et al. [2]

[1] Visit `https://github.com/darius-sas/astracker` to access the tool.

2.5 Parent Centrality Metric

Parent Centrality (PC) is a metric that applies to Circular Dependencies between software artifacts. To calculate the metric, we use the Dependency Graph and the Architectural Smells contained in it. In fact, the PC metric shows whether a parent package is a hub or not. H. A. Al-Mutawa et al. [3] give the mathematical definition:

> For a given fixed tangle (Cyclic Dependency) (V, E), we compute the Betweenness Centrality for each vertex. We then define two sets of edges. Let E_p be the set of edges where the source vertex is a (direct or indirect) sub-package of the target vertex. These are the edges pointing from a child to a parent package. Formally, let E_p^+ be the set of edges pointing from a child to parent package, and where the parent package has a higher Betweenness Centrality than the child package. We then call the ratio in Equation 2.1 the Parent Centrality of a tangle (Cyclic Dependency).

$$P_c = \frac{|E_p^+|}{|E_p|} \tag{2.1}$$

To make it more clear, we will describe some examples. For that reason, we have created three sets of graphs, and we will examine their P_c value. The names of the vertices follow the namespace package structure of Java language. Also, in the Figures 2.3, 2.4 and 2.5, the red line symbolizes the edges that belong to E_p set and the blue line symbolizes the edges that belong to E_p^+ set. Lastly, the index right to the name of the package (vertex) represents its Betweenness Centrality.

2.5.1 Example 1

In our first example (see Figure 2.3), we have a chain shape graph with one parent and two children.

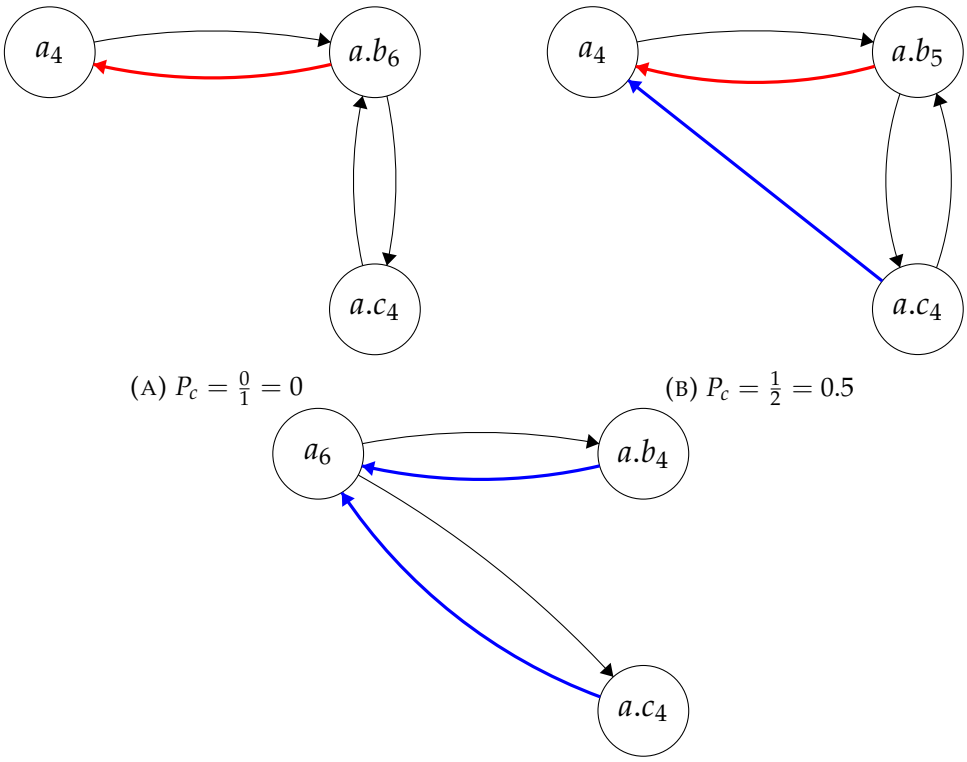
In Figure 2.3a, the center vertex is one of the children, which means that we expect the smell have zero Parent Centrality.

In Figure 2.3b, we add an edge from the other child, the one that did not have a connection with its parent, toward the parent vertex. This means that the Parent Centrality increases but the package *a.b* still has bigger Betweenness Centrality than its parent.

Finally, we make a star shape smell, and at the center of that, we place the parent. The previous metric reaches 1, as the parent package is a very important piece of the smell.

2.5.2 Example 2

The previous experiment was only between ancestors or descendants. In this one, we attach to the prior packages one more, but without relation with the



(A) $P_c = \frac{0}{1} = 0$ (B) $P_c = \frac{1}{2} = 0.5$

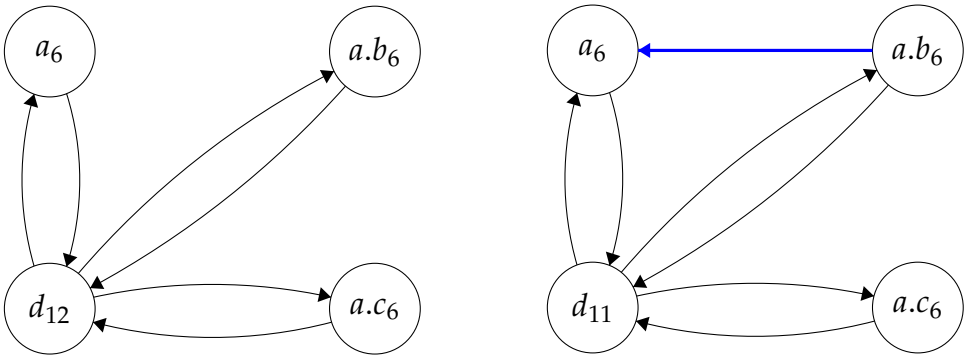
(C) $P_c = \frac{2}{2} = 1$

FIGURE 2.3: Chain like shape example

others. This new package takes the place of the center of the star (see Figure 2.4).

In Figure 2.4a, there is no edge from child to parent. Without a doubt, we do not have a countable result for this smell, and we call it *undefined*.

Next, in Figure 2.4b, we add an edge from a child towards the parent. In this case, the Betweenness Centrality of the parent and the child is the same, and $P_c = 1$. If we had a more complicated smell, the result could have been different.



(A) $P_c = undefined$ (B) $P_c = \frac{1}{1} = 1$

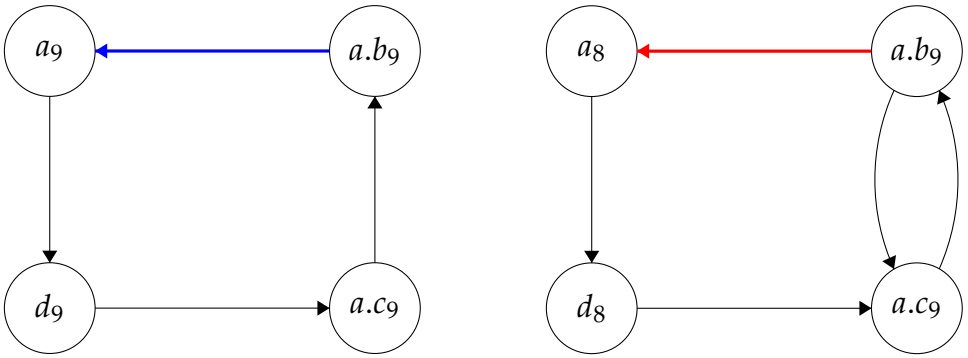
FIGURE 2.4: Star like shape example

2.5.3 Example 3

In the last example, we will examine the behavior of a circle smell. There is no need to present an example in which there is no child to parent edges, as we have already seen a similar situation in Figure 2.4a.

In Figure 2.5a, we have a classic circle, where only one edge is from a child towards a parent package. In such a circle, every vertex has the same Betweenness Centrality, and therefore, $P_c = 1$.

However, this value can change, if we add the vertex which missing between the children (see Figure 2.5b). With this addition, we reinforce the centrality of the children, and the P_c value drops.



(A) $P_c = \frac{1}{1} = 1$ (B) $P_c = \frac{0}{1} = 0$

FIGURE 2.5: Circle like shape example

Chapter 3

Implementation and Study Design

This chapter is organized as follows: Section 3.1 and Section 3.2 describe the implementation of Parent Centrality and the technique being used to validate the algorithm. After that, Section 3.3 reports on the nature of the data later analysis uses. Finally, Section 3.4, reviews extensively the results of the metric from several views, such as the difference in smell shapes and investigate the correlation between Parent Centrality and some other metrics ASTracker tool produces.

3.1 Implementation of the Metric

The implementation of the Parent Centrality metric mentioned before is split into 3 main parts. For the algorithm to be completely correct, and before the procedure starts, it checks if the architectural smell is between packages as this is the type of cycle this metric targets. Otherwise, it returns *null*.

In the first section of the algorithm, we extract the sub-graph of the Architectural Smell that the metric will be applied. The purpose of this is to take only the vertices and edges, that are affected by the smell, and apply the metric only to that specific smell for the calculation (see Figure 3.1).

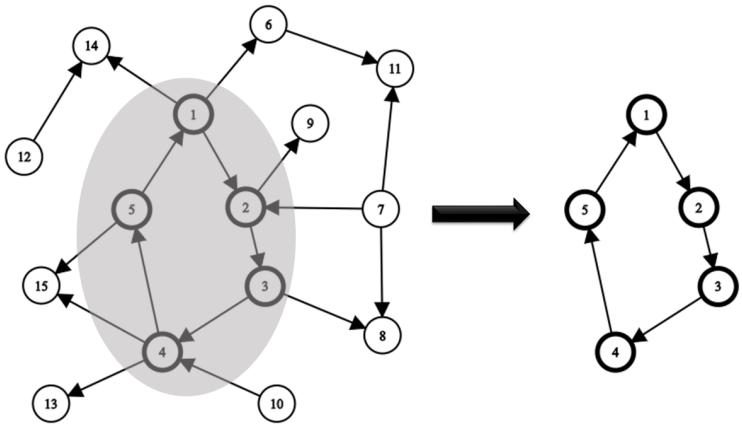


FIGURE 3.1: Graph and sub-graph (smell)

The second step is to compute the Betweenness Centrality of every vertex of the sub-graph that we extracted before.

Now that all the necessary data have been collected, we just calculate the cardinality of E_p and E_p^+, as H. A. Al-Mutawa et al. specify [3]. More precisely, for the E_p, we count the edges from child package to parent, and for the E_p^+, we take the previous set and count only the edges that the parent's Betweenness Centrality is greater than the child's one.

Lastly, we divide and return the value. If $|E_p| = 0$, the result is *undefined*.

The pseudo-code of the implementation that we introduced, can be found in Algorithm 1.

3.2 Testing

After the implementation has finished, we also want to check if the solution is correct. For that reason, we conduct unit tests using four known graphs in combination with the correct results, and we check every stage of the implementation. The stages that we want to check are:

1. *sub-graph extraction* by comparing if the new sub-graph has the right vertices and edges,
2. *Betweeneess Centrality calculation* by comparing if the vertices have an attribute for this reason with the correct value,
3. *Parent Centrality calculation* by comparing the actual and the calculated, by the algorithm, result and,
4. *the function that the system uses*, in the same way, that we check parent centrality calculation.

Also, for increased correctness, we conduct system tests in two versions of *ant* and two versions of *antlr*, two tools that are used for defining languages and generate compilers, respectively. At these tests, we only conduct *Test 4* from the stages mentioned before.

3.3 Data Collection

Now that we ensured that the proposed algorithm yields the correct results, we analyze the results of the Parent Centrality metric. These results have been extracted from a data-set, composed of the Dependency Graphs of 21 real-world open-source Java projects, with at least 74 versions each, and a total of 3039 versions. These graphs have been previously used by D. Sas et al. [2]. It is the output of the Arcan tool[1], which assembled the artifacts affected by architectural smells for each version of a project directly from source code files. These data give us the opportunity to test the metric and

[1] Visit `https://bit.ly/2tY1uDA` to access the tool.

Algorithm 1 Parent Centrality Algorithm

```
▷ graph: Dependency Graph
▷ smell: Architectural Smell
procedure PC METRIC(graph, smell)
    if Smell Between Packages then
        Initialize sub-graph

        for Every vertex from graph affected by smell do
            Copy vertex to sub-graph
        end for
        for Every edge from graph between two affected vertices do
            Copy edge to sub-graph
        end for

        for Every vertex from sub-graph do
            Find Betweennness Centrality of vertex
            Create attribute and add the previous value to vertex
        end for

        Count E_p value as H. A. Al-Mutawa et al. describe
        Count E_p^+ value as H. A. Al-Mutawa et al. describe
        if E_p = 0 then
            return "undefined"
        end if
        PC := E_p^+ / E_p

        return PC
    end if

    return null
end procedure
```

its results on a wide range of projects and the evolution that every one of them undergoes throughout its lifetime. After the computation of Parent Centrality for every Cyclic Dependency smell in our data set, we end up with the results presented in the next section.

3.4 Data Analysis and Results

For our analysis, we collected data from 21 real-world projects, 3039 versions, and 271739 smells in total. After the execution of the ASTracker tool, we extracted only the smells that referred to Cyclic Dependencies between packages. Also, we took into consideration the Parent Centrality metrics that had a numeric result and not *undefined*. Although, some general outcomes about *undefined* values are shown in Section 3.4.1. Lastly, following on from these steps, we realized that one entire project (guava) and some in-between versions of the other ones do not have smells complying with the previous criteria. So, we excluded them, too. Finally, we are left with 20 projects, 2653 versions, and 114714 smells (78510 unique).

We present some noteworthy and interesting results, as it is not realistic to fit all data and results in a single paper. Also, we run the Pearson Method for Parent Centrality, in correlation with one of the metrics: Size of Smell (see Section 3.4.3), Number of Edges (see Section 3.4.4) or Page Rank (see Section 3.4.5). This correlation will show us if, depending on the value of another metric, we can predict the Parent Centrality value. Especially for the correlation between Parent Centrality and Page Rank, if the assumption is true, it may mean that both measure more or less the same thing.

Pearson Method is a well-known measurement for linear correlation between two variables. It calculates a rho (ρ) value, which is within the range $[-1, 1]$ taking into account the covariance and the standard deviation of the two variables. The method defines positive or negative linear correlation when $\rho = 1$ or $\rho = -1$, respectively, and no linear correlation when $\rho = 0$ [2].

Last but not least, there are plots about the evolution of Parent Centrality through versions in Appendix A. Due to the less importance in comparison with the above, we will not analyze them, and their existence in the Appendix is just for plenitude.

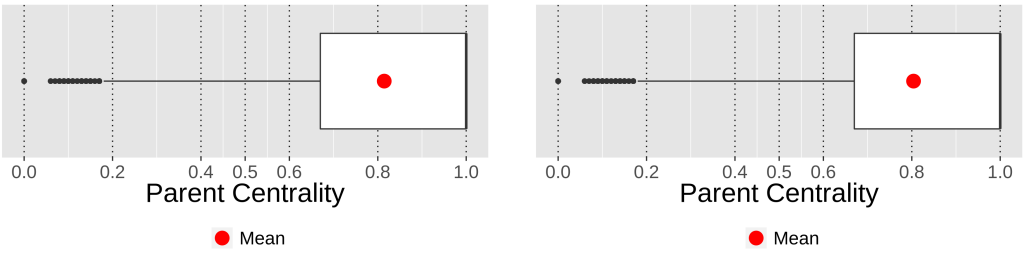
3.4.1 General Statistics on Parent Centrality

The *undefined* values of Parent Centrality comprise 27% of the Cyclic Dependencies between packages. This means that one out of four smells does not have a direct connection from a child to a parent. In other words, in one out of four Cyclic Dependencies that form in a project, there is no child package that depends directly on its parent or ancestor. This may happen for two reasons, (i) the smell is between siblings or/and irrelevant packages, or (ii) there

[2] Visit `https://libguides.library.kent.edu/SPSS/PearsonCorr` for more information about Pearson Method.

is a parent, but no child in the smell depends on it. The difference between $P_c = undefined$ and $P_c = 0$ is the fact that in the latter, there is, at least, one direct dependency from a child towards its parent, but the child package is more central inside the smell than its parent. However, *undefined* values are not of interest in the Parent Centrality metric, and so we exclude them for the rest of the analysis.

As we see in Figure 3.2a, the mean value of Parent Centrality is slightly over 0.8, and the median value is 1. This reveals that parent packages are likely to be the center of their children. Also, in at least half of the smells, parent packages are hubs relative to children's packages referencing them.



(A) With Tiny and Clique shapes (B) Without Tiny and Clique shapes

FIGURE 3.2: Parent Centrality of all projects combined

3.4.2 Shapes

The interesting part, about the shapes of the smells, is the distribution of Parent Centrality among them (see Figure 3.3). Tiny and clique shapes are complete graphs, and so the Betweenness Centrality is the same for all the vertices. Therefore, either the Parent Centrality is 1 if the smell has a parent package, or "undefined" otherwise. On the other hand, chain shape value, depends, mostly, on the position of parent packages inside the structure. This means that the more central the parent package is, the higher the value. We can surely understand that if a project has a big number of tiny or/and clique smells, its Parent Centrality is probably very high. However, if we extract all tiny and clique shapes, mean and median values do not change significantly (see Figure 3.2b). This might happen because the unclassified shapes, i.e the unknown structure shapes, are over 87% of the total smells. Also, their mean value is around the same as the total one. So the participation of the other shapes is not significant.

A typical example arises if we examine the distribution of the smells between the projects (see Table 3.1). Ant-ivy and Cassandra projects have around 95% unclassified smell, and around the same distribution among the circle, clique and tiny smells combined. So, we expect to have the same mean and median value, too. However, if we look at these values in Figure 3.5, though, they are very different.

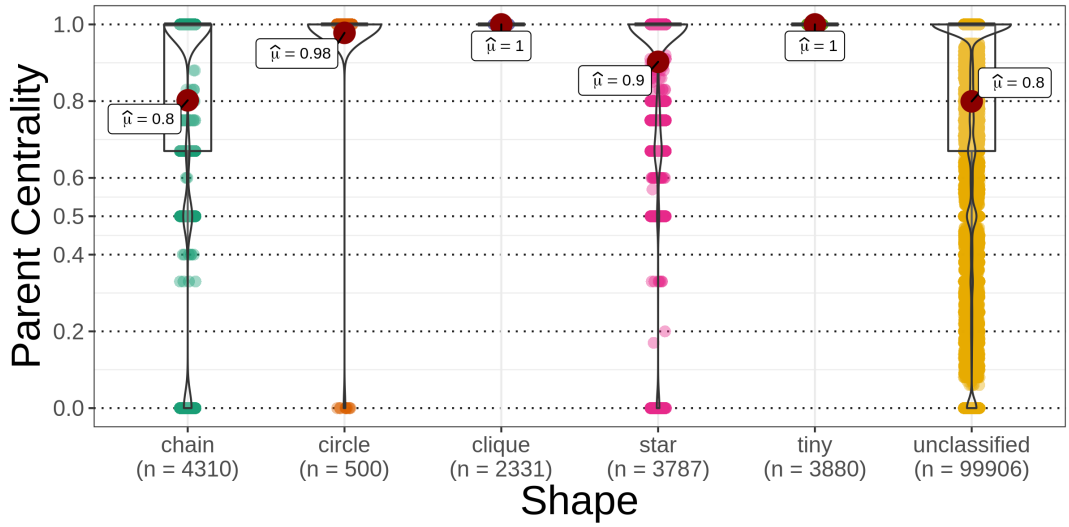


FIGURE 3.3: Parent Centrality of every shape

For that reason, we run the same analysis, excluding the unclassified values (see Figure 3.4). By looking at the plot, we can understand that tiny (32.6%), chain (27.9%) and star (23.9%) shapes are the most common ones, in this order.

Last but not least is that only half of the projects develop at least one circle smell, and in no more than two of them, the distribution is over 2%.

3.4.3 Size of Smells

The Size of a Smell is the number of vertices that are affected by the smell. The bigger the size, the more complicated it becomes to measure Parent Centrality. In our study, the size range is between 2 and 39.

As we see in Figure 3.6, most of the smells have a size between 2 and 10. Moreover, the Pearson Method reveals that there is no correlation between the Size of Smell and Parent Centrality as the rho value is $\rho = 0.03$. Even if we extract tiny and clique shapes, as we know their value, rho remains almost the same (see Table 3.2). However, there are no smells with size over 25 and Parent Centrality lower than 0.5. We can understand that parent packages in bigger smells have bigger Betweenness Centrality (i.e. package is a part of many shortest paths inside smell) than most of their children depending on them, as the definition of Parent Centrality suggests (see Section 2.5).

If we take the analysis a bit further and examine the Size of Smell and Parent Centrality of every shape, some results are very interesting (see Table 3.3). As expected, tiny and clique shapes are always 1. Also, circle shapes tend to have a descent of the Parent Centrality value, as Size of Smell increases, namely a negative correlation ($\rho_{circle} = -0.59$). More information can be

Project	**Shape Distribution (%)**					
	Chain	**Circle**	**Clique**	**Star**	**Tiny**	**Unclassified**
accumulo	4.00	0.00	0.22	3.68	4.38	87.72
activemq	2.77	0.04	2.06	2.56	4.72	87.84
ant-ivy	1.65	2.10	0.19	0.53	0.46	95.07
cassandra	1.42	0.00	0.96	1.16	0.70	95.76
chukwa	0.95	2.86	0.00	0.64	17.01	78.54
druid	10.17	0.00	2.98	0.80	15.12	70.93
httpcompo nents-client	7.23	0.00	0.00	0.00	77.11	15.66
jackrabbit	3.14	0.14	1.75	8.37	4.68	81.91
jackson-databind	8.04	0.00	21.12	5.67	1.08	64.09
jena	4.04	0.32	1.86	2.16	2.85	88.77
jsqwiki	5.27	0.21	5.84	6.65	4.80	77.23
lucene-solr	5.30	0.00	2.36	4.30	6.13	81.92
mina	10.63	0.00	0.17	9.80	11.30	68.11
pdfbox	2.33	2.20	1.20	2.55	0.86	90.86
pgjdbc	0.00	0.00	0.00	8.76	47.68	43.56
poi	8.02	0.04	4.99	5.67	7.00	74.28
struts	18.34	0.00	0.00	11.46	8.12	62.08
testng	9.90	0.00	2.97	9.99	0.27	76.88
tika	29.60	0.00	0.00	6.80	20.00	43.60
xerces2-j	4.46	0.36	0.00	11.95	0.98	82.25
Average	6.86	0.41	2.43	5.18	11.76	73.35

TABLE 3.1: Shape Distribution in every Project

found in Table 3.3. One noteworthy fact is that symmetric shapes do not exceed the size of 10. All the other smells have an unknown shape.

With all the above results in mind, we can surely comprehend that the Size of Smell of a certain shape is uncorrelated with its Parent Centrality value. The exception to that rule is the circle shape, which seems to have an inversely proportional correlation between Size and Parent Centrality. However, the observations are minimal and in a small range of Size of Smell to have a reliable result.

The last thing that is interesting in this correlation is the range of the rho value if we run Pearson Method in every project (see Table 3.4). This range is $\rho = [-0.56, 0.27]$. Apparently, this range does not follow the general rho value, but further analysis of the reasons is beyond this paper.

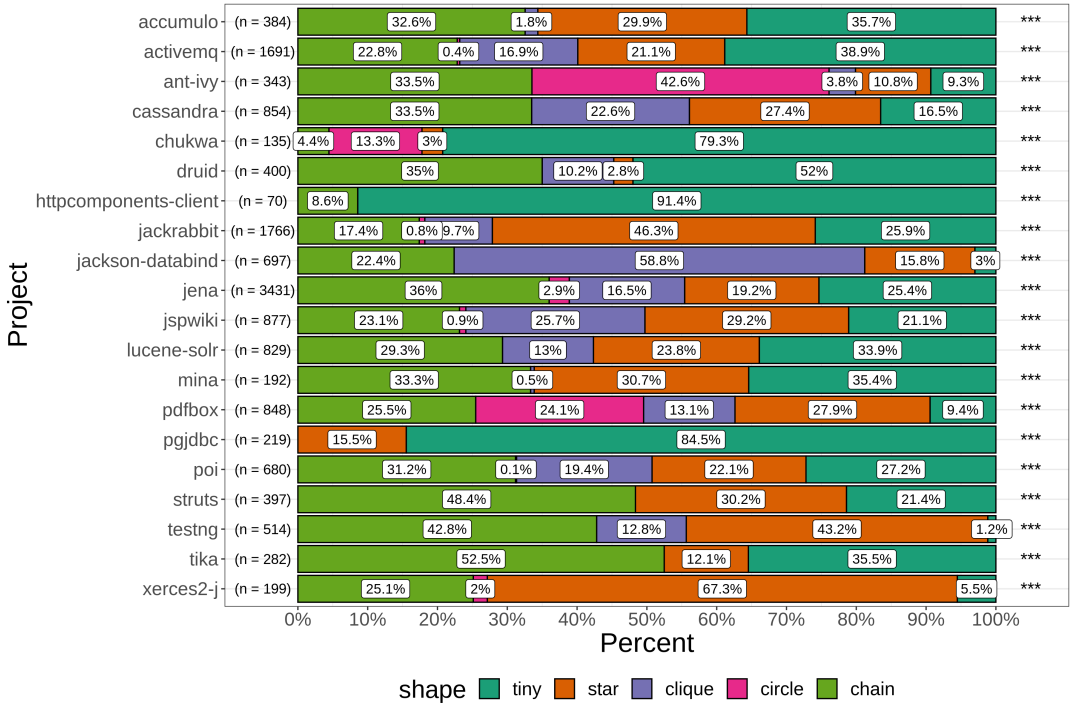


FIGURE 3.4: Shape distribution in every project (Without Unclassified shape)

	Correlation between Parent Centrality and...		
	Size	**Edges**	**Page Rank**
With Tiny & Clique	0.03	0.11	-0.01
Without Tiny & Clique	0.07	0.13	-0.02

TABLE 3.2: Pearson Method (ρ)

3.4.4 Number of Edges of Smell

Number of Edges is a metric that counts the total number of incoming and outgoing edges between the vertices of the smell. Inside our observations, the range is between 2 and 4450, with most of them be below 300 edges.

The correlation between Parent Centrality and this metric, according to Pearson Method (see Figure 3.7), is around zero ($\rho = 0.11$) and seems that the two metrics are more or less uncorrelated. Even if we extract tiny and clique shapes, as we know their value, rho remains almost the same (see Table 3.2). However, the interesting part here is that a very large Number of Edges (> 2200), leads to smell with $P_c \geq 0.8$.

The same pattern occurs if we group the same observations by shape. There is no relevance between the Number of Edges and Parent Centrality metrics, as Table 3.3 displays. Moreover, we report the absence of every symmetrical shape smells for the Number of Edges over 725.

However, if we aggregate the data by the project (see Table 3.4), we can say that there is a wide range of ρ values. The most significant ones are $\rho_{struts} =$

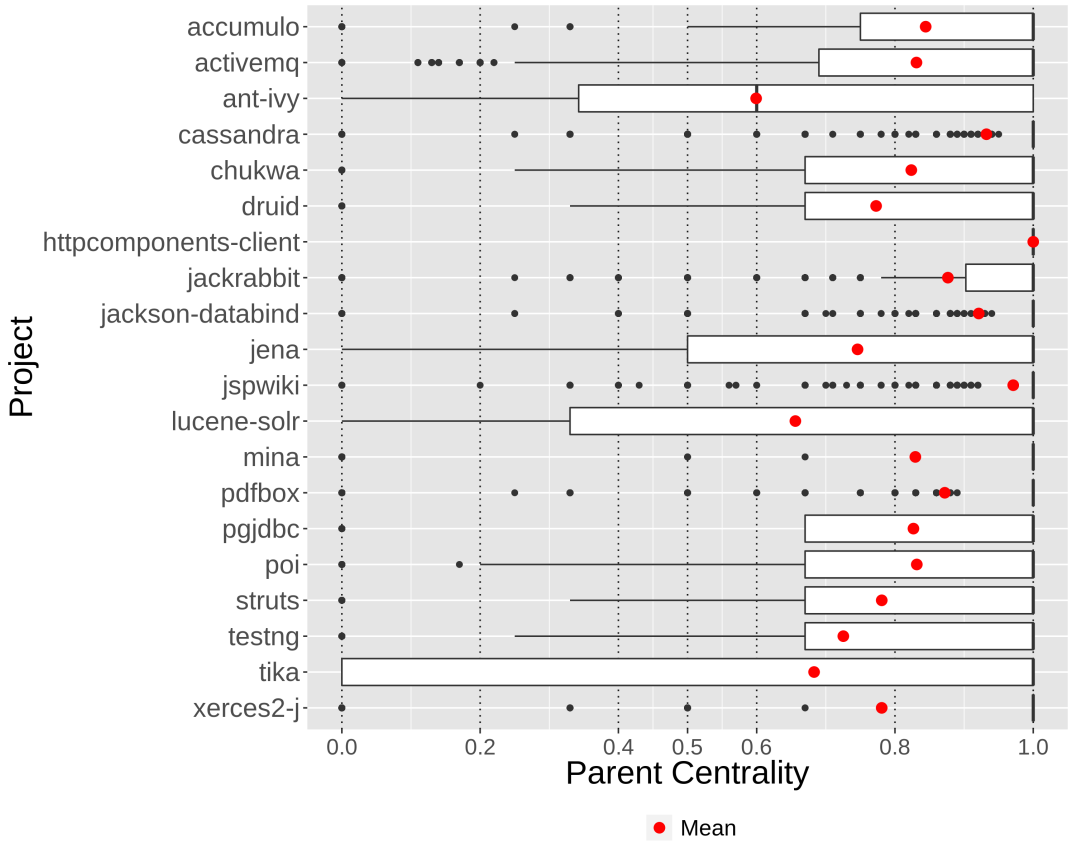


FIGURE 3.5: Parent Centrality of every project

-0.70, $\rho_{pgjdbc} = -0.46$ and $\rho_{tika} = -0.44$, which tend to have an inversely proportional trend between Parent Centrality and Number of Edges metrics. These projects, also, had the same tendency with the Size of smell metric, as the same table reports. More details will be discussed in Chapter 4.

3.4.5 Page Rank of Smell

Page Rank metric measures the centrality of the smell, in a different way than Parent Centrality does. However, it is more correct to say that it measures the importance of a node inside the graph. In a nutshell, "importance" is determined by how many edges and other "important" vertices point to the vertex. In our observations, the values vary from 0 to 0.19.

Even though the two metrics measure centrality, the rho value of Pearson Method ($\rho = -0.01$) does not show the expected results, namely directly proportional relationship (see Figure 3.8). Additionally, there is no difference in rho value, when extracting tiny and clique shapes from the analysis, as we know their value beforehand (see Table 3.2). However, we come across a phenomenon that appeared both in Size of Smell (see Section 3.4.3) and in Number of Edges (see Section 3.4.4) analyses. When these metrics exceed a certain number, there is a lack of small Parent Centrality values. Actually, when Page Rank is over 0.1, $P_c = 1$.

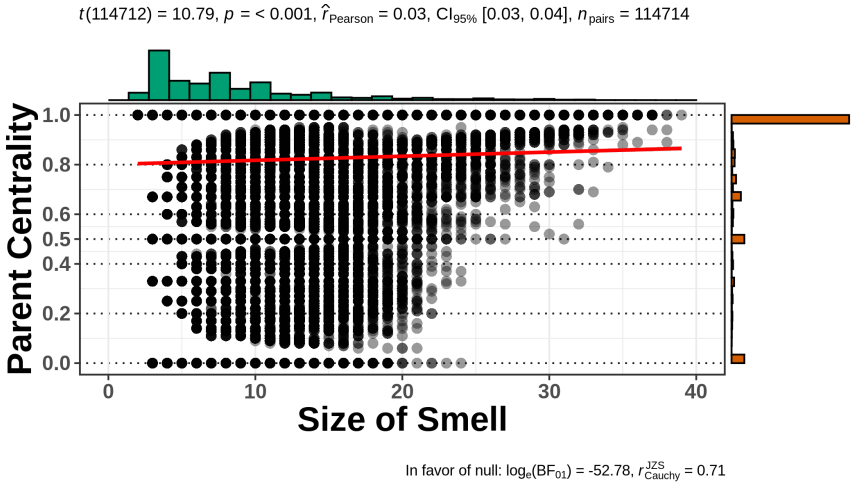


FIGURE 3.6: Correlation between Parent Centrality and Size of Smell (Pearson Method)

Shape	**Correlation between Parent Centrality and...**		
	Size	**Edges**	**Page Rank**
Chain	0.03	0.07	-0.13
Circle	-0.59	0.08	0.07
Clique	NA	NA	NA
Star	0.12	-0.04	0.02
Tiny	NA	NA	NA
Unclassified	0.08	0.15	-0.01
Average	-0.09	0.07	-0.01

TABLE 3.3: Pearson Method (ρ) grouped by Shape

The same pattern (i.e. no relevance) appears by dividing the results by shape in Table 3.3. Parent Centrality and Page Rank seams to measure different things, as $P_c \approx 0$ for all shapes.

When it comes to dividing the data by project and do the correlation, the only outline to the general idea of no correlation between Parent Centrality and Page Rank is the Testng project with $\rho_{testng} = -0.51$ and inversely proportional correlation, as Table 3.4 suggests.

The above results do not keep up with our initial assumption, as Parent Centrality and Page Rank are uncorrelated. We conclude that they measure centrality in a totally different way. Section 4.2 provide more information for this observation.

Project	**Correlation between Parent Centrality and...**		
	Size	**Edges**	**Page Rank**
accumulo	-0.02	-0.01	0.08
activemq	-0.14	-0.15	-0.27
ant-ivy	-0.07	-0.04	0.16
cassandra	0.20	0.19	0.10
chukwa	0.04	-0.26	-0.08
druid	-0.27	0.02	0.15
httpcomponents-client	NA	NA	NA
jackrabbit	0.09	0.14	0.06
jackson-databind	-0.03	0.03	0.25
jena	-0.25	-0.19	0.10
jsqwiki	0.06	0.03	-0.04
lucene-solr	-0.46	-0.34	0.29
mina	-0.10	-0.19	-0.20
pdfbox	0.09	0.06	-0.10
pgjdbc	-0.56	-0.45	0.27
poi	-0.26	-0.03	-0.24
struts	-0.47	-0.70	0.08
testng	0.27	0.13	-0.51
tika	-0.33	-0.44	-0.35
xerces2-j	-0.20	-0.10	-0.30
Average	-0.13	-0.12	-0.03

TABLE 3.4: Pearson Method (ρ) grouped by Project

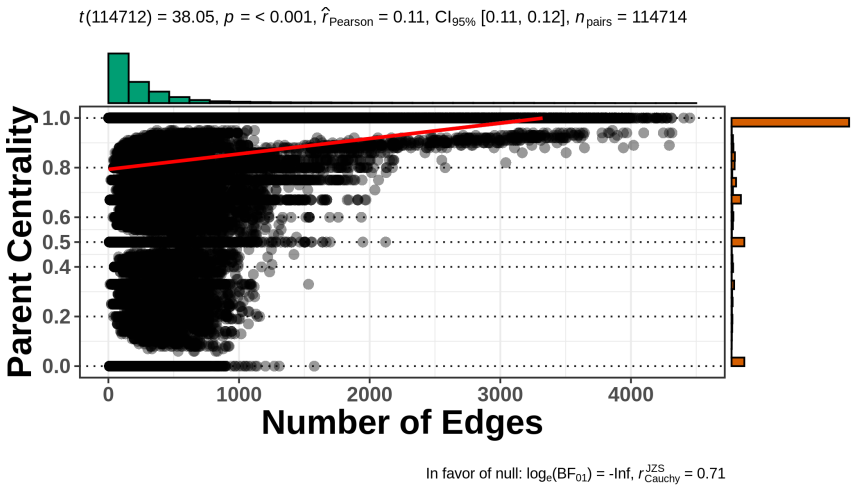


FIGURE 3.7: Correlation between Parent Centrality and Number of Edges of Smell (Pearson Method)

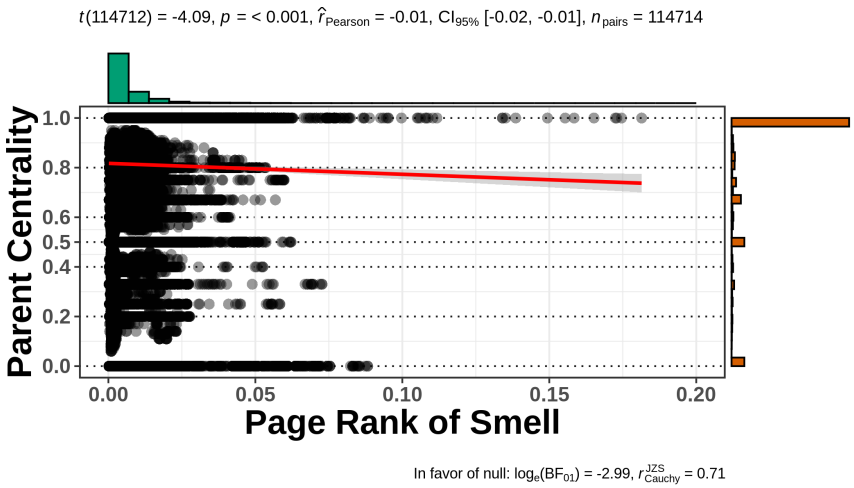


FIGURE 3.8: Correlation between Parent Centrality and Page Rank of Smell (Pearson Method)

Chapter 4

Discussion

In this chapter, we will discuss some of the results of Section 3.4. In Section 4.1, we will compare the results of our analysis in correspondence with the results of H. A. Al-Mutawa et al. in the paper "On the Shape of Circular Dependencies in Java Programs" [3]. Also, in Section 4.2, we will discuss some of the most noteworthy outcomes presented in Chapter 3.4.

4.1 Comparison with Related Work

The overall mean and median values of the analysis of H. A. Al-Mutawa et al. and ours are more or less the same. We measured a mean value equal to 0.8, whereas H. A. Al-Mutawa et al.'s was 0.76 (see Figure 3.2). The median values were equal to 1 and 1 as measured by the two studies, respectively.

Furthermore, the shapes that we outline in our case are chain, circle, clique, star, and tiny (see Section 2.1.1). H. A. Al-Mutawa et al. suggest that "There are a large number of tiny tangles (shapes) [...]. Symmetric shapes like chains, circles, cliques, and stars are rarer, [...]". Our analysis in Figure 3.3 shows that chain-shaped cycles, in absolute numbers, are more. Additionally, tiny and star shapes are approximately the same cardinality. However, this analysis is misleading because it depends on the size of the projects. As we see in Table 3.1, the average percentage of tiny shape smells between projects is almost doubled the percentage of chain ones, which matches the analysis of H. A. Al-Mutawa et al.

Lastly, the *undefined* Parent Centrality percentage in our analysis hovers around the same numbers with H. A. Al-Mutawa el al.'s paper, 27% in comparison with 22%, correspondingly. The meaning of these findings has been discussed in Section 3.4.1.

The above results confirm our analysis of Parent Centrality in respect to the paper "On the Shape of Circular Dependencies in Java Programs" [3]. This means that many of the Cyclic Dependencies of a program are non-critical. These are the smells with Parent Centrality close to 1.

4.2 Results Interpretation

As we saw in Sections 3.4.3, 3.4.4 and 3.4.5, there is no correlation between Parent Centrality and the other characteristics investigated. However, some considerations are necessary to better understand why this is happening.

The first one is that the values of tiny and clique shapes are known, in advance, as Section 3.4.2 explains. Even if you extract these values, the results about the rho values of the Pearson Method remain the same. The most logical reason is the quantity of these shapes in comparison to the total number of smells.

If we look deeper into the correlations, we find that the Page Rank metric measures centrality, just like the Parent Centrality metric does. However, no relevance between them appears by our analysis in Section 3.4.5. This is happening because, whereas Parent Centrality takes into account only the connections from child to parent, Page Rank is more general, considering the importance of every vertex. This might mean that, ultimately, they measure different things.

Another consideration, that it is not perceptible at first glance, is related to the correlation between Parent Centrality and the Size of Smell or Number of Edges. In Table 3.4, most of the projects report a close rho value between the previously mentioned metrics, namely Size of Smell and Number of Edges. This is probably happening because of the positive correlation between these metrics ($\rho = 0.87$), as the Pearson method shows. As we observe in Figure 4.1, as the Size of Smell grows, the Number of Edges is also getting larger.

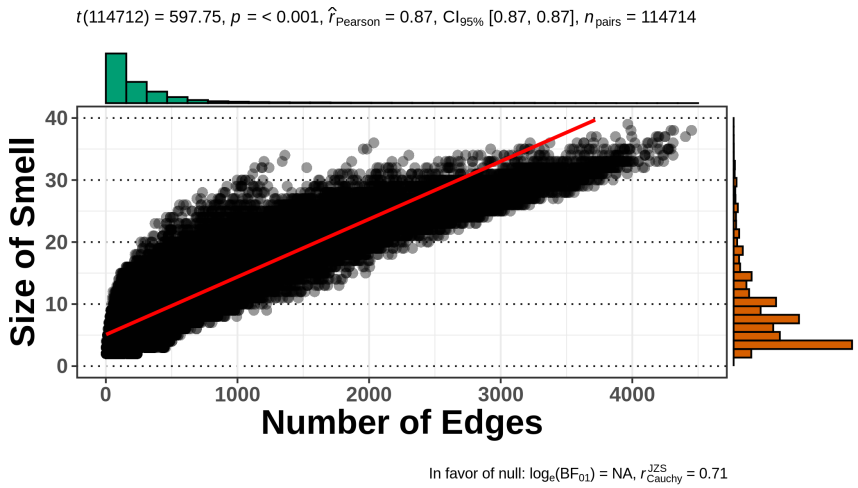


FIGURE 4.1: Correlation between Size of Smell and Number of Edges (Pearson Method)

The outliers of our first result about uncorrelated metrics are the projects Struts, Pgjdbc, and Tika, which have an inversely proportional relationship between Parent Centrality and one of Size of Smell or Number of Edges. However, some characteristics may be responsible for these exceptions. First

of all, the average number of smells in every project is around 5700. The mentioned projects, though, have far less smells, ($n_{struts} = 1047$, $n_{pgjdbc} = 388$ and $n_{tika} = 500$). Also, the mean and the median value of the Size of Smells is less than 5, while the overall mean, in the data-set, value is ≈ 8 and median ≈ 7. Also, the Number of Edges metric's mean and median values for the same projects are between 20 and 40. Again, the overall values for these metrics are way higher ($mean \approx 400$ and $median \approx 200$). Therefore, we conjecture that the combined effects of lower values of Size and Number of Edges may be accountable for the inversely proportional relationship. However, Chukwa and Mina projects exhibit similar results, and no correlation appears. The variable that differs is the shape distribution of *unclassified* shape smells. Struts, Pgjdbc and Tika projects have really low percentage of *unclassified* shapes, namely $unclassified_{Struts} = 62\%$, $unclassified_{Pgjdbc} = 44\%$ and $unclassified_{Tika} = 44\%$, while the average is around 73%. Even with this information, we cannot be sure about the exact reasons, and further investigation is required.

By extension to the previous observation, in all the correlation analyses that we did, when Size, Number of Edges, or Page Rank have a high value, there are no smells with low Parent Centrality. There is no clue to understand this phenomenon as Parent Centrality does not seem to have any correlation with any of the other metrics. The only assumption is that up until a certain value of Size of Smell, Number of Edges, or Page Rank, the Parent Centrality has equal chances of small or large value, but only big values, otherwise.

Chapter 5

Conclusion

In this thesis, we implemented the Parent Centrality metric into an existing tool called ASTracker[1] and validated the implementation. Next, we tried to answer the question "*Are all Cyclic Dependencies crucial, and how common are the non-critical Cyclic Dependencies in a program?*" by analysing a set of open-source Java systems.

We implemented the metric according to the mathematical definition provided by H. A. Al-Mutawa et al., who suggest that "For a given fixed tangle (Cyclic Dependency) (V, E), [...] we define two sets of edges. Let E_p be the set of edges [...] pointing from a child to a parent package. Formally, let E_p^+ be the set of edges pointing from a child to parent package, and where the parent package has a higher Betweenness Centrality than the child package. We then call the ratio $E_p = E_p^+ / E_p$ the Parent Centrality of a tangle (Cyclic Dependency)." (see Section 2.5). If Parent Centrality is high, the Cyclic Dependency has at least one child-to-parent connection and the parent package forms a hub among its children. This smell is suggested as non-critical. The basic steps of the implementation include: (i) the extraction of the sub-graph, (ii) the calculation of Betweenness Centrality, and (iii) the calculation of the appropriate cardinalities, namely E_p and E_p^+. Moreover, we conducted tests to ensure the correctness of the implementation.

From our analysis we found that our results match the results of H. A. Al-Mutawa et al. [3], who place Parent Centrality at around 0.8 and most common shape be the tiny one (see Section 3.4.1). We showed, eventually, that most of the Cyclic Dependencies of a project may not be critical. Also, we tried to correlate Parent Centrality with other existing metrics that ASTracker measures, namely Size of Smell (see Section 3.4.3), Number of Edges (see Section 3.4.4) and Page Rank (see Section 3.4.5). However, we discovered the absence of correlation, using a well-known correlation analysis method, the Pearson Method[2], meaning that they measure different aspects of the smell.

However, we found that the rho value of the correlation between Parent Centrality and Size of Smell is more or less equivalent to the correlation between the Parent Centrality and the Number of Edges. This may be the result of the

[1] Visit `https://github.com/darius-sas/astracker` to access the tool.

[2] Visit `https://libguides.library.kent.edu/SPSS/PearsonCorr` for more information about the Pearson Method.

direct correlation among Size of Smell and Number of Edges. However, a combination of a minimal number of smells, small Size of Smells, low Number of Edges, or/and below the normal *unclassified* shapes may result in an inversely proportional relationship between Parent Centrality and one of Size of Smell or Number of Edges.

An area that requires further research is the absence of the observations having low Parent Centrality in big values of Size of Smell, Number of Edges, and Page Rank metrics, since there is no finding that can justify it. Moreover, an investigation in unique smells will reveal whether the aged smells affect the aforementioned results or not.

Appendix A

Appendix Plots

Extra plots:

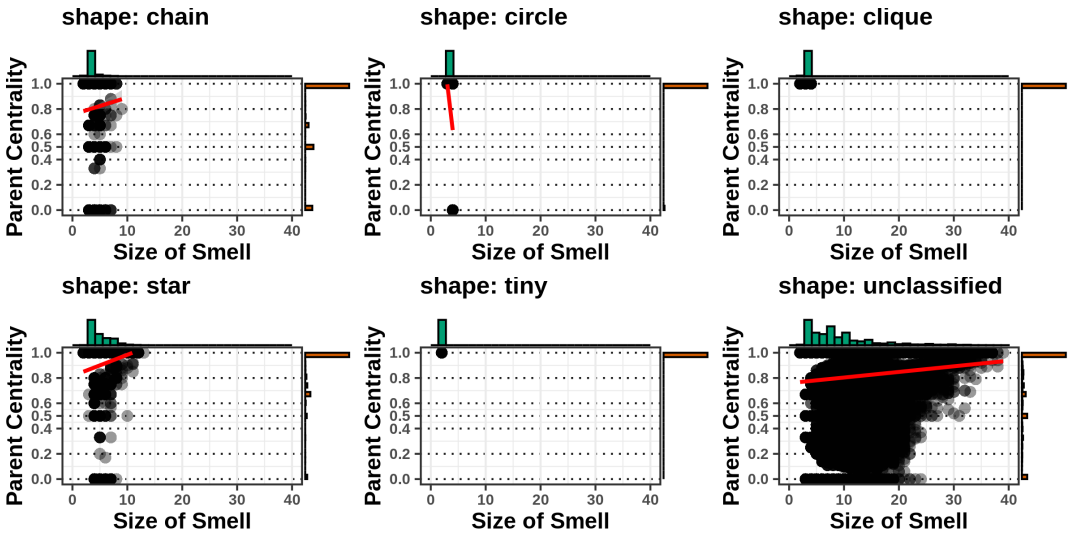


FIGURE A.1: Correlation between Parent Centrality and Size of Smell grouped by Shape (Pearson Method)

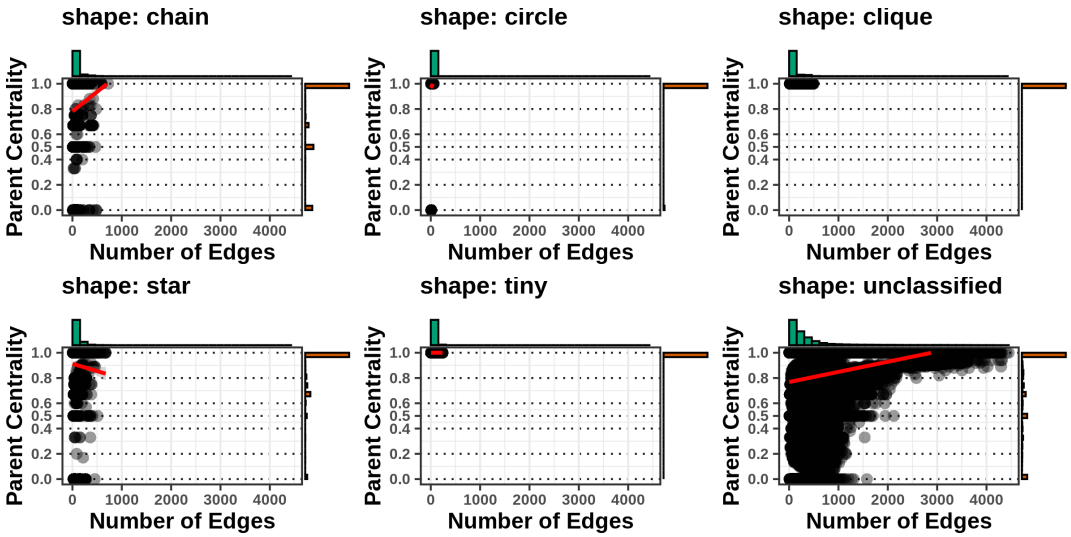


FIGURE A.2: Correlation between Parent Centrality and Number of Edges of Smell grouped by Shape (Pearson Method)

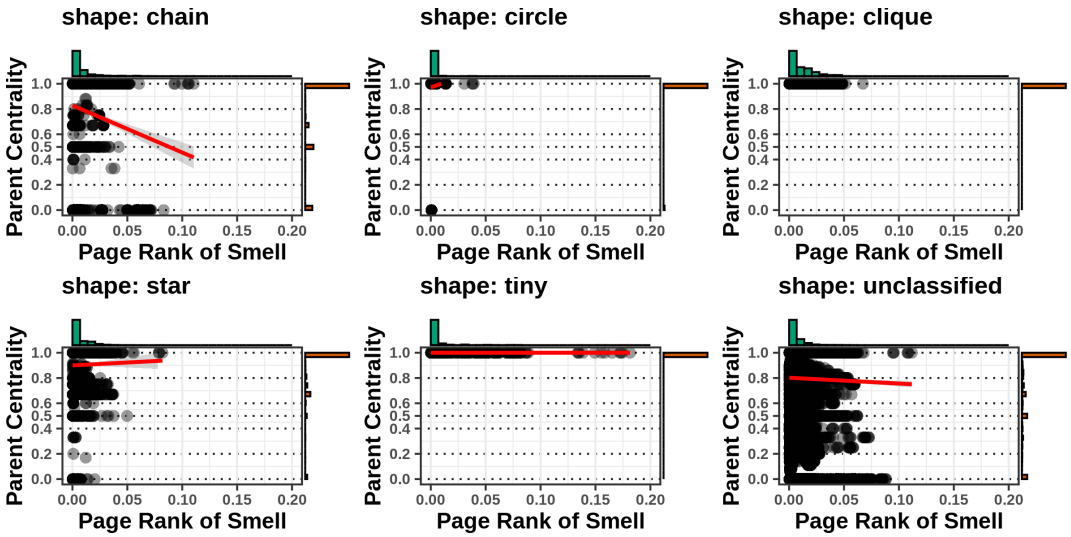


FIGURE A.3: Correlation between Parent Centrality and Page Rank of Smell grouped by Shape (Pearson Method)

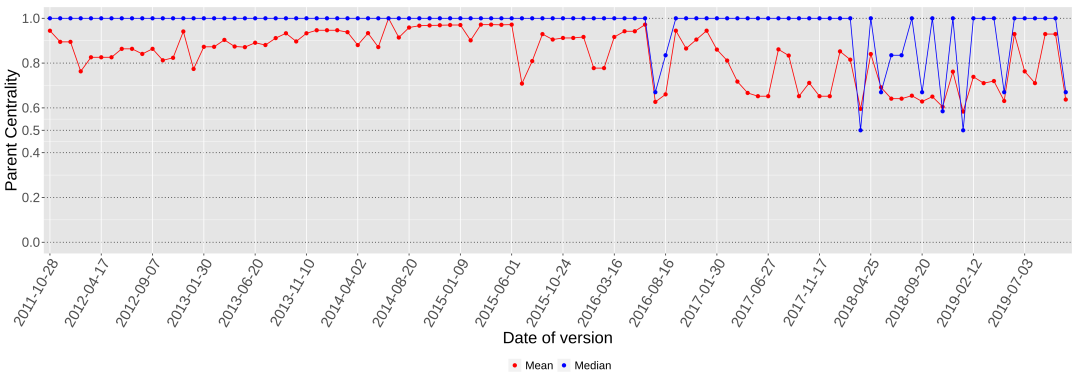


FIGURE A.4: Parent Centrality of accumulo's project versions

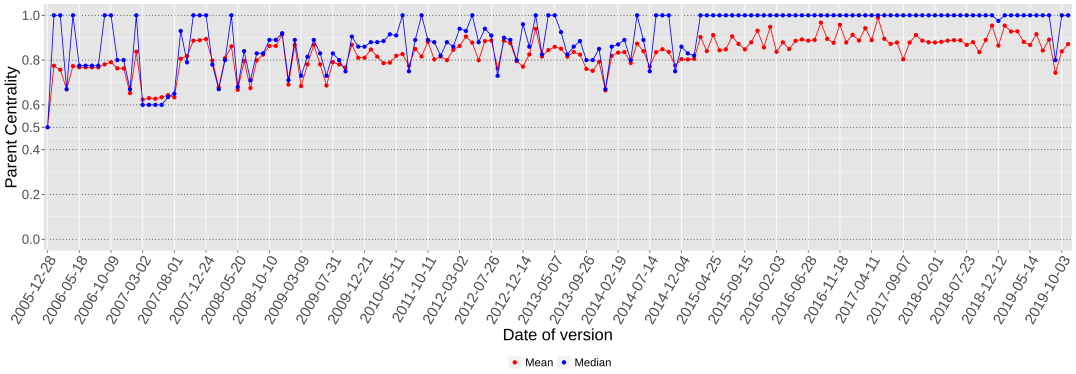


FIGURE A.5: Parent Centrality of activemq's project versions

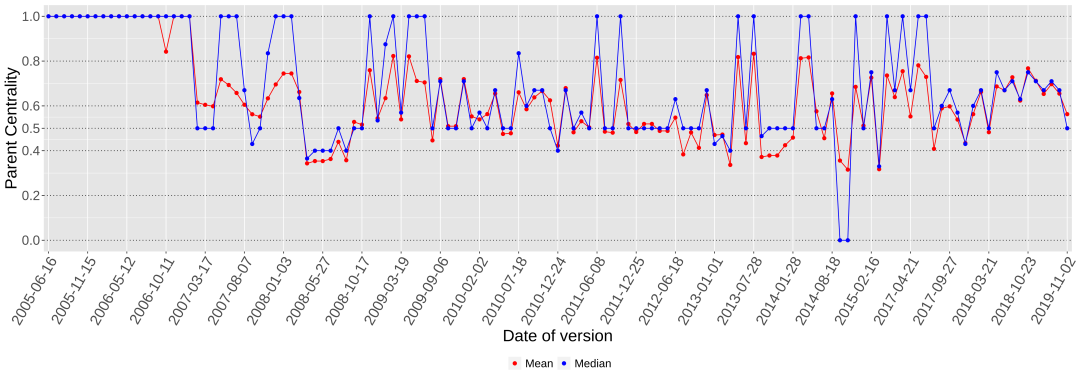


FIGURE A.6: Parent Centrality of ant-ivy's project versions

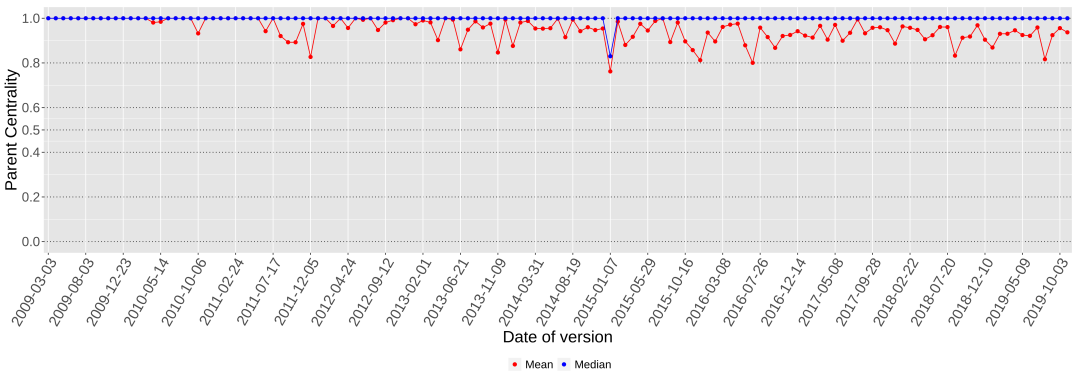


FIGURE A.7: Parent Centrality of cassandra's project versions

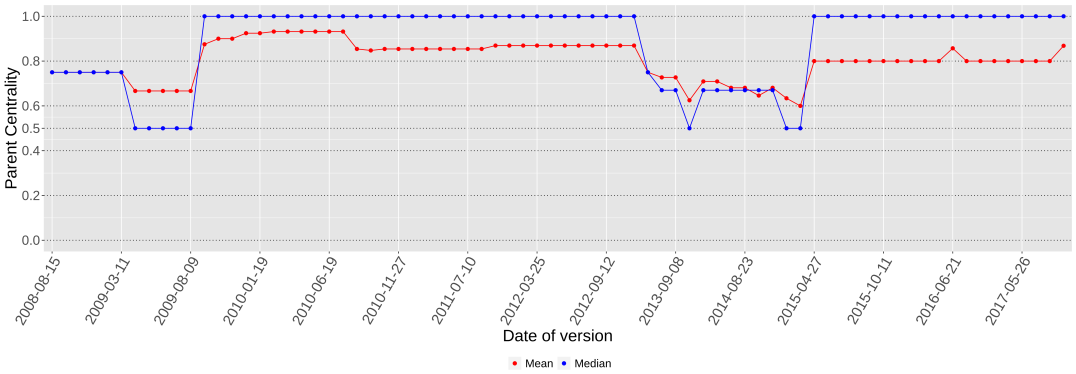


FIGURE A.8: Parent Centrality of chukwa's project versions

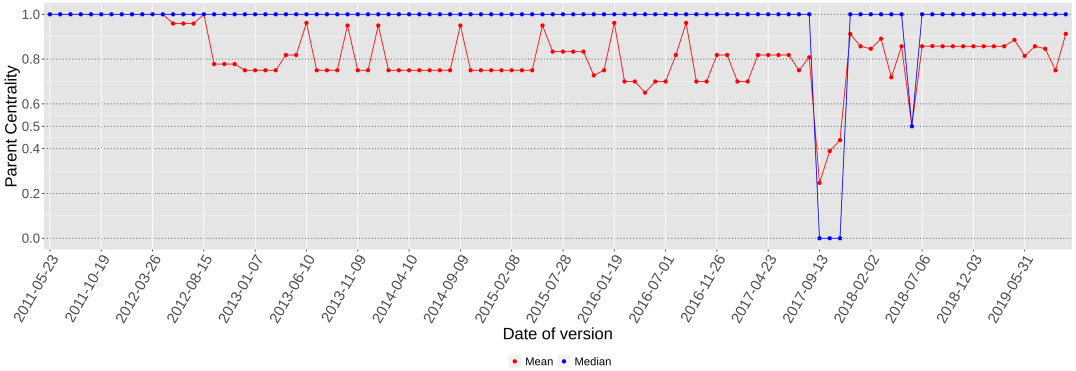


FIGURE A.9: Parent Centrality of druid's project versions

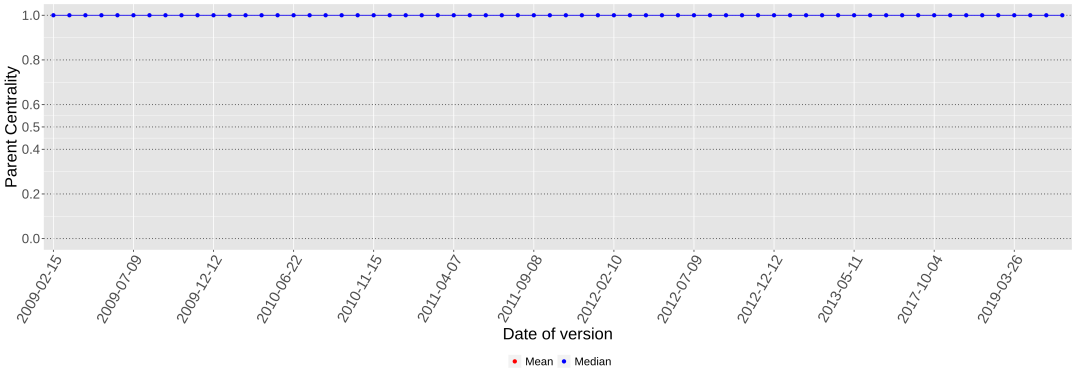


FIGURE A.10: Parent Centrality of httpcomponents-client's project versions

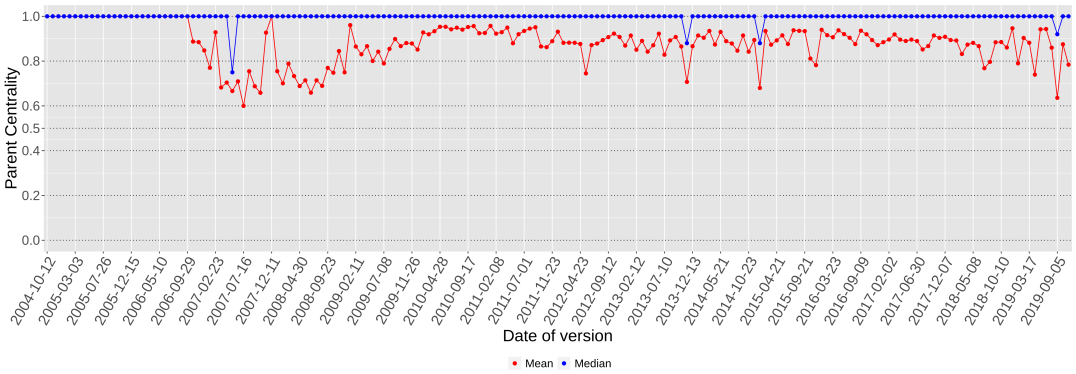


FIGURE A.11: Parent Centrality of jackrabbit's project versions

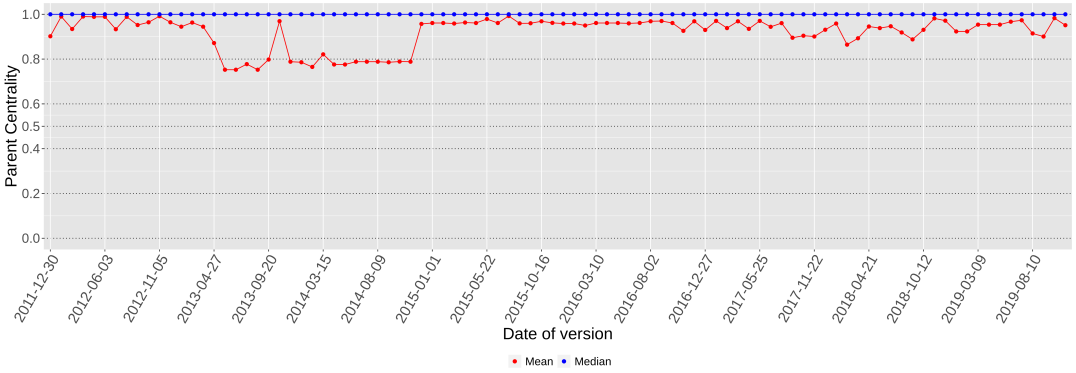


FIGURE A.12: Parent Centrality of jackson-databind's project versions

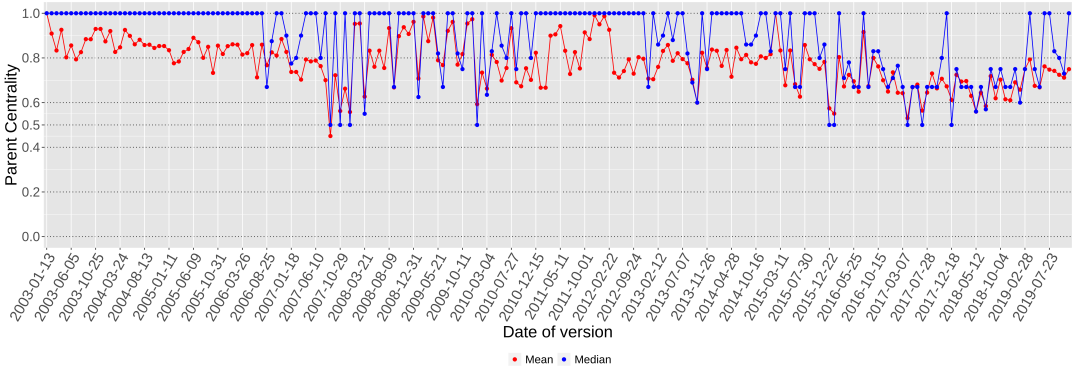


FIGURE A.13: Parent Centrality of jena's project versions

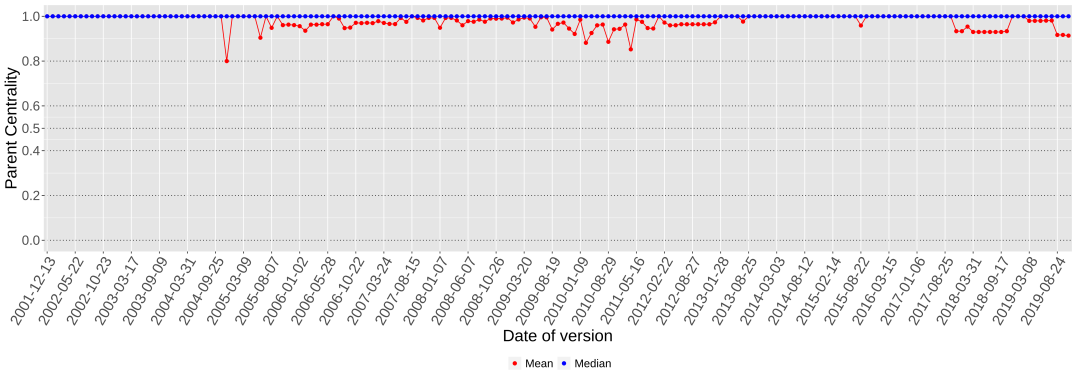


FIGURE A.14: Parent Centrality of jspwiki's project versions

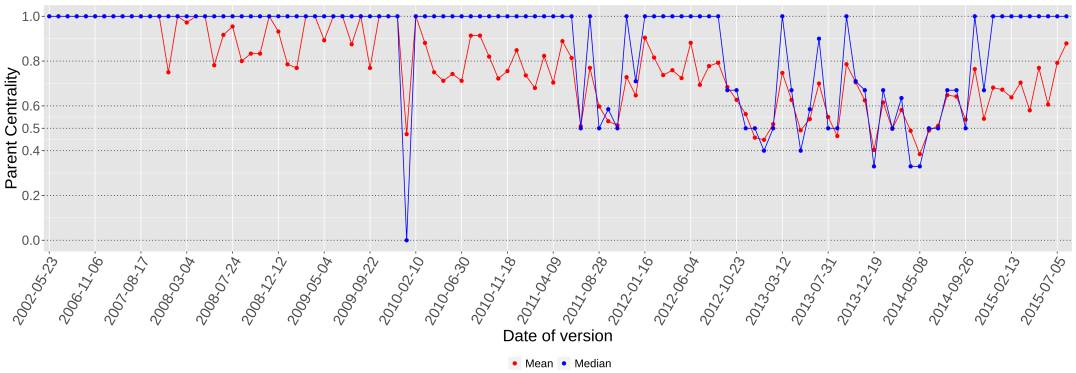


FIGURE A.15: Parent Centrality of lucene-solr's project versions

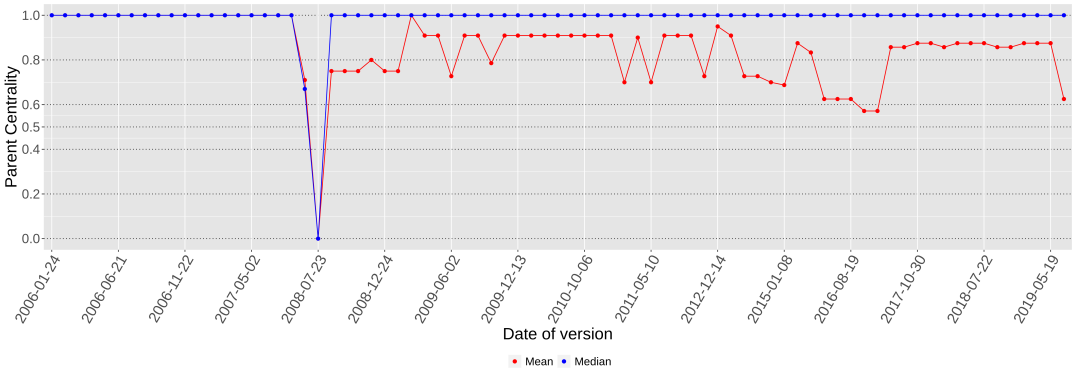


FIGURE A.16: Parent Centrality of mina's project versions

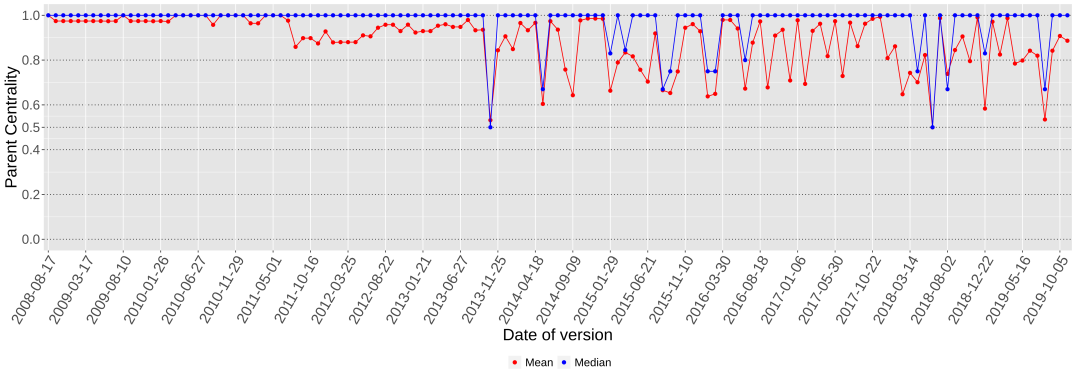


FIGURE A.17: Parent Centrality of pdfbox's project versions

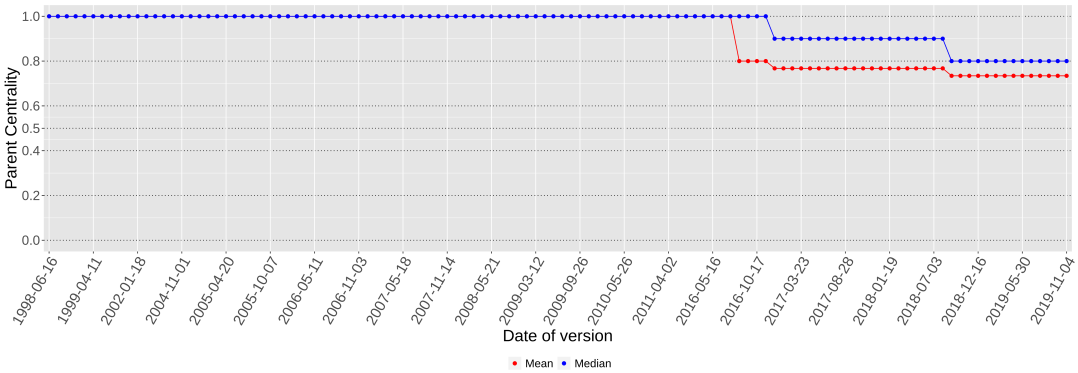


FIGURE A.18: Parent Centrality of pgjdbc's project versions

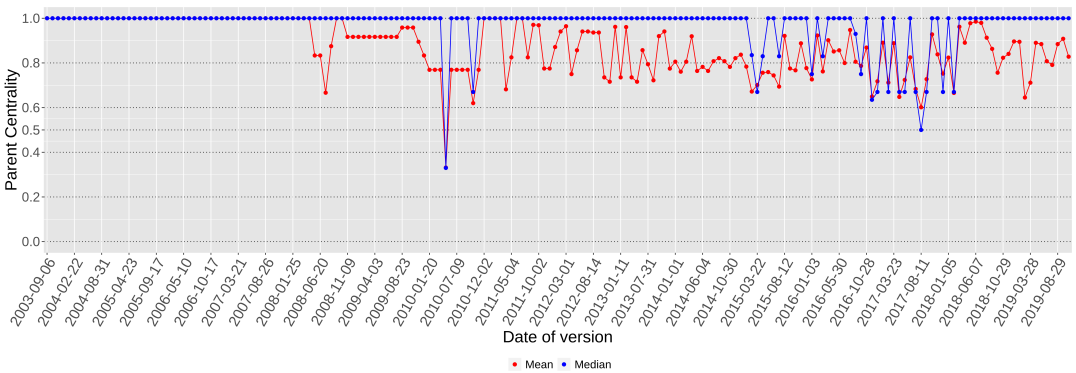


FIGURE A.19: Parent Centrality of poi's project versions

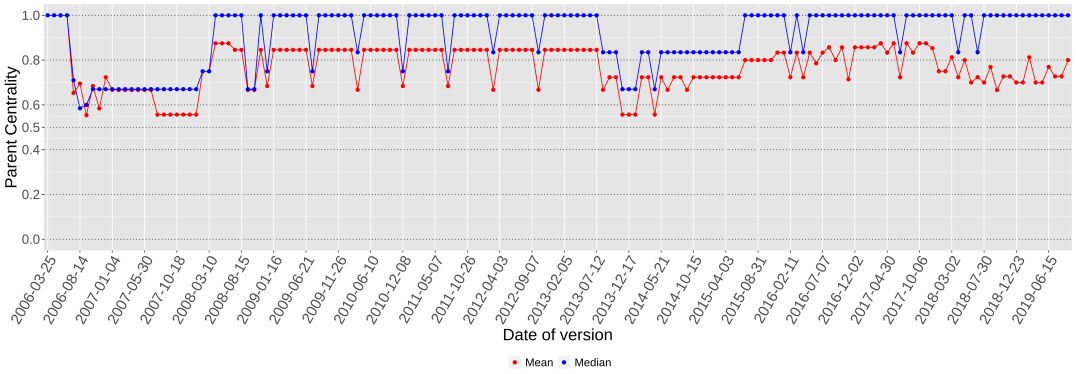


FIGURE A.20: Parent Centrality of struts's project versions

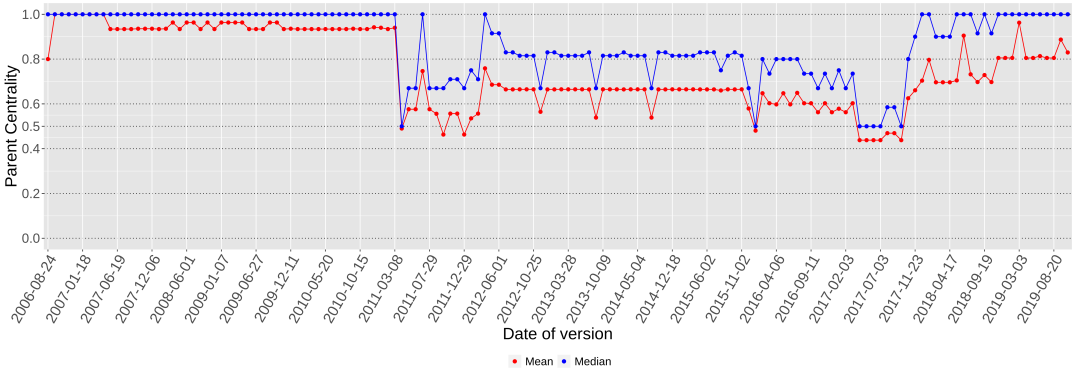


FIGURE A.21: Parent Centrality of testng's project versions

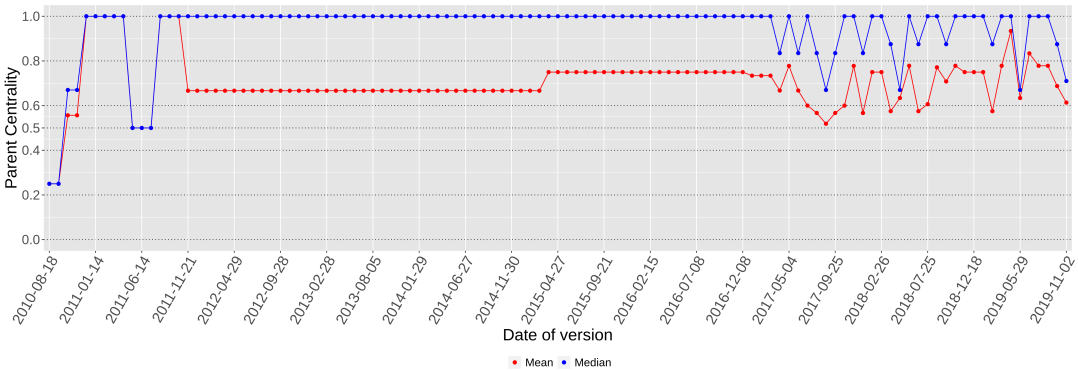


FIGURE A.22: Parent Centrality of tika's project versions

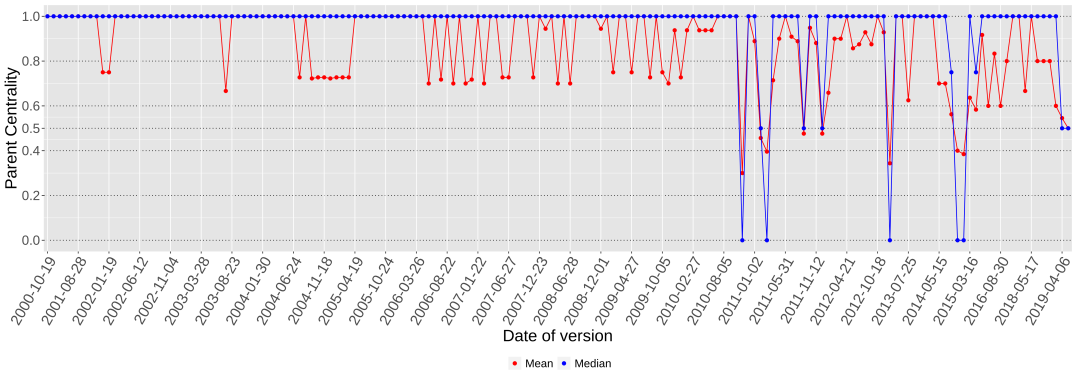


FIGURE A.23: Parent Centrality of xerces2-j's project versions

Bibliography

[1] Robert C Martin. "Design principles and design patterns". In: *Object Mentor* 1.34 (2000), p. 597. URL: http://staff.cs.utu.fi/staff/jouni.smed/doos_06/material/DesignPrinciplesAndPatterns.pdf.

[2] D. Sas, P. Avgeriou, and F. Arcelli Fontana. "Investigating Instability Architectural Smells Evolution: An Exploratory Case Study". In: (Sept. 2019), pp. 557–567. ISSN: 2576-3148. DOI: 10.1109/ICSME.2019.00090. URL: https://ieeexplore.ieee.org/abstract/document/8919109.

[3] H. A. Al-Mutawa et al. "On the Shape of Circular Dependencies in Java Programs". In: (Apr. 2014), pp. 48–57. ISSN: 2377-5408. DOI: 10.1109/ASWEC.2014.15. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6824106&isnumber=6824087.

[4] JOHN W. ESSAM and MICHAEL E. FISHER. "Some Basic Definitions in Graph Theory". In: *Rev. Mod. Phys.* 42 (2 July 1970), pp. 271–288. DOI: 10.1103/RevModPhys.42.271. URL: https://link.aps.org/doi/10.1103/RevModPhys.42.271.

[5] Neil Robertson and P.D Seymour. "Graph minors. X. Obstructions to tree-decomposition". In: *Journal of Combinatorial Theory, Series B* 52.2 (1991), pp. 153 –190. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(91)90061-N. URL: http://www.sciencedirect.com/science/article/pii/009589569190061N.

[6] Linton C. Freeman. "A Set of Measures of Centrality Based on Betweenness". In: *Sociometry* 40.1 (1977), pp. 35–41. ISSN: 00380431. URL: http://www.jstor.org/stable/3033543.

[7] Martin Lippert and Stephen Roock. *Refactoring in large software projects: performing complex restructurings successfully*. John Wiley & Sons, 2006.

[8] Martin Lippert and Stephen Roock. *Refactoring in large software projects: performing complex restructurings successfully*. John Wiley & Sons, 2006.

[9] D. L. Parnas. "Designing Software for Ease of Extension and Contraction". In: *IEEE Transactions on Software Engineering* SE-5.2 (1979), pp. 128–138. URL: https://ieeexplore.ieee.org/document/1702607.

[10] W. P. Stevens, G. J. Myers, and L. L. Constantine. "Structured design". In: *IBM Systems Journal* 13.2 (1974), pp. 115–139. URL: https://ieeexplore.ieee.org/abstract/document/5388187.

[11] T. D. Oyetoyan et al. "Circular dependencies and change-proneness: An empirical study". In: (2015), pp. 241–250. URL: https://ieeexplore.ieee.org/abstract/document/7081834.

[12] J. Garcia et al. "Identifying Architectural Bad Smells". In: (2009), pp. 255–258. URL: https://ieeexplore.ieee.org/abstract/document/4812762.

[13] Joshua Garcia et al. "Toward a Catalogue of Architectural Bad Smells". In: (2009). Ed. by Raffaela Mirandola, Ian Gorton, and Christine Hofmeister, pp. 146–162. URL: https://rdcu.be/b3gGM.

[14] F. A. Fontana et al. "Automatic Detection of Instability Architectural Smells". In: (2016), pp. 433–437. URL: https://ieeexplore.ieee.org/abstract/document/7816489.

[15] F. A. Fontana et al. "Arcan: A Tool for Architectural Smells Detection". In: (2017), pp. 282–285. URL: https://ieeexplore.ieee.org/abstract/document/7958506.

[16] J. Buckley et al. "JITTAC: A Just-in-Time tool for architectural consistency". In: (2013), pp. 1291–1294. URL: https://ieeexplore.ieee.org/abstract/document/6606700.

[17] N. Ali et al. "Improving Bug Location Using Binary Class Relationships". In: (2012), pp. 174–183. URL: https://ieeexplore.ieee.org/abstract/document/6392116.

[18] Jörg Lenhard et al. "Are Code Smell Detection Tools Suitable for Detecting Architecture Degradation?" In: ECSA '17 (2017), 138–144. DOI: 10.1145/3129790.3129808. URL: https://doi.org/10.1145/3129790.3129808.