Architecture Recovery using Partitioning Clustering Technique

BY

ALESSANDRO CHETTA
B.S, Politecnico di Milano, Milan, Italy, 2013

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:

Ugo Buy, Chair and Advisor
Mark Grechanik
Pier Luca Lanzi, Politecnico di Milano

ACKNOWLEDGMENTS

Thank you to my advisor Prof. Ugo Buy. He demonstrated true friendship to me. He was always ready to understand my difficulties and to help me with wise suggestions. Thanks to him I was able to accomplish this research.

Thank you to Prof. Pier Luca Lanzi that gave me the data mining knowledge that I applied in this work. Moreover, at the beginning of my adventure at UIC, Prof. Pier Luca Lanzi suggested me to follow the course CS 474 with Prof. Ugo Buy.

Thank you to Lynn Ann Thomas that guided me in my path at UIC.

I dedicate this work to my family. My father Aldo Claudio, my mother Maria Grazia, my brother Marco, his fiancé Maria Rosaria, and their coming baby perhaps Alessandro or Azzurra Maria.

They supported me in everything I did in my life. Some days before the submission of this thesis Marco announced me that Maria Rosaria was waiting for a baby. This is the most important event of my life up to now. Her or his birth will bring a lot of joy to my family.

Thank you to my friends Doldo, Cervelli, and Marco Chirizzi that supported me with love from 4,500 miles away. Thank you to my roommates Alessandro and Enzo for all the great time we spent together in our little apartment.

Thank you to all the Italian crew from Politecnico di Milano: Alessandro Oddone, Lorenzo Di Tucci, Andrea Piscitello, Giorgio Conte, Massimo De Marchi, Dario Casula, Matteo Pal-

varini, Gianluca Venturini, Luca Buratti, Ettore Trainiti, Davide Pigiamino, Francesco Paduano, Paolo Bruzzo.

AC

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GUI	Graphical User Interface.
HTML	HyperText Markup Language.
JSON	JavaScript Object Notation.
MCV	Model View Control.
OOP	Object Oriented Programming.
PDF	Portable Document Format.
SLOC	Source lines of code.
UML	Unified Modeling Language.

SUMMARY

Software development evolved quickly in the last decades to satisfy market needs. The software market poses some constraints such as strict deadlines, high software reliability and performance, and limited budget. Hence, developer teams face more challenging problems and implement larger projects to realize a competitive software that satisfies the constraints. It implies team sizes are becoming larger. A large developer team contains some subteams such as a design and implementation team, a debugging team and testing team. Thus, information sharing among the team members is crucial in satisfying market demand.

Knowledge of the software architecture is a way to describe a software system at a high level of abstraction. The software architecture is an effective information sharing means beside natural language documentation. In addition software architecture are decrypting, not language dependent and often use graphical representations. Nonetheless, the software architecture is not always available or up to date. In this case, architecture recovery seeks to recover a software architecture based on reverse engineering techniques. The architecture recovery process extracts a software architecture taking as input the software row source code.

The state of the art in software architecture recovery does not provide a reliable solution yet. This motivated additional research on the application of clustering algorithms. My research contribution is a novel technique able to recover the architecture of a large software project from its source code. I named this tool K-recovery after the clustering algorithm K-means.

I have also assessed empirically the effectiveness of my method. A challenge in these empirical evaluations is the lack of a ground truth, that is, knowledge of the actual software architecture that my method is recovering.

To evaluate the recovered architectures, I recovered manually the analyzed projects architectures, then I compared them. K-recovery recovered succesfully the MVC components of a software video game. Moreover, it recovered the architectures of two software projects with 75% average accuracy. The software projects analyzed have more than 10K SLOC and the largest contains more than 80 classes.

Furthermore, I implemented some cluster cohesion and coupling metrics that provide an evaluation about the clusters quality.

CHAPTER 1

INTRODUCTION

Software stakeholders imposed new constraints to the programmers in order to meet the software market needs. The software development process changed significantly in the last decades to satisfy its stakeholder constraints. For instance, mobile and web applications request increased significantly. These applications must guarantee high reliability, low cost, and amazing user experience to be competitive. Thus, programmers had to deal with a difficult trade-off among these characteristics. Consequently, development teams became larger and they adopted new technologies such as software control version, formal bug report and formal testing methodologies. Furthermore, the computer science community agreed on the application of a new programming paradigm for the development of large software. OOP paradigm replaced the procedural paradigm for the development of large software. The new paradigm is based on the concept of object. An object contains both procedures and data representation of a given abstract entity. As stated by D. L. Parnas [1] OOP introduces several advantages: "The benefits expected of modular programming are: (1) managerial - development time should be shortened because separate groups would work on each module with little need for communication: (2) product flexibility - it should be possible to make drastic changes to one module without a need to change others; (3) comprehensibility - it should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood." Furthermore, D. L. Parnas introduces the concept of software understanding. In

fact, the larger a developer team is, the more difficult it is to share information among its members or subteams. The development team can share information about a large software by producing its documentation. The software architecture is an effective way to describe an large software besides the traditional natural language documentation. The software architecture [2] is a conceptual model that represents the components of a software system and the relations among them at an arbitrary granularity level. The software architecture describes the system at a high level of abstraction and brings several benefits to the development process. D. E. Perry [3] states "Some of the benefits we expect to gain from the emergence of software architecture as a major discipline are: 1) architecture as the framework for satisfying requirements; 2) architecture as the technical basis for design and as the managerial basis for cost estimation and process management; 3) architecture as an effective basis for reuse; and 4) architecture as the basis for dependency and consistency analysis." Nonetheless, the architecture is often misunderstood. Especially in legacy systems, the software architecture does not reflect the actual implementation or it is not available at all. In these cases architecture recovery aims to retrieve the software architecture. Architecture recovery is the process of recovering a candidate software architecture of a system based on the application of reverse engineering techniques on the raw source code. Researchers evaluates a recovered architecture by comparing it with the actual architecture. To it so the software architecture must be available. For this reason the evaluation of an architecture recovering technique is challenging. J Garcia et al [4] recognize this issue: "Many automated techniques of varying accuracy have been developed to help recover the architecture of a software system from its implementation. However, rigorously assessing

these techniques has been hampered by the lack of architectural 'ground truths'." Architecture recovery is the research field of this work. In this thesis I report in detail the development of K-recovery, a tool that recovers the architecture of a general software project written in a particular OOP language. The main goal of this work is to design techniques and tools for advancing the state of the art in the field of architecture recovery. The motivations of this work are at the low reliability of the architecture recovery techniques currently available, and the increasing cost, in term of money and time, of the software maintenance according to the inaccuracy of its documentation. Also I implemented some metrics to evaluate the recovered architectures to sidestep the lack of ground truths.

K-recovery performs architecture recovery in two independent phases a source code feature extraction phase, and clustering phase. In the first phase the tool parses the raw source code of the system under analysis. Then, the tool extracts classes, the interactions between them, and their hierarchical organization. A standardized lexical analyzer performs the source code parsing in support of this stage. K-recovery performs a static analysis of the system. This means that the code compilation is not needed and its issues are avoided. Furthermore, due to the static analysis approach external libraries and dependencies are not taken into consideration during system analysis. Finally, K-recovery represents internally the information gathered in a graph-like data structure. I call this data structure dependencies graph. This first phase is programming language dependent. In fact, the parser recognizes the patterns of a given programming language.

The clustering phase is programming language independent. It performs the clustering based only upon the dependencies graph created in the previous phase. It implies that the two phases are decoupled and replaceable. For instance, the parser can be replaced in order to parse another programming language. At this stage K-recovery analyzes C++ projects.

In the second phase, the tool applies a customized version of the K-means clustering algorithm that identifies the components within the system. K-recovery allows an expert user to set some algorithm parameters in order to adjust the clustering result.

The software architecture of the analyzed systems is not always available. I recovered the software architecture manually to evaluate the tool results. The tool is able to recognize the MVC components of a software video game called freeablo [5]. Moreover, it recovered with high accuracy the architecture of two complex software systems. these systems are Wkhtmltopdf [6] and mxnet [7]. The analyzed systems contain respectively 44, 46, and 84 classes and their size is more than 10K SLOC. I compared their recovered architecture with the one I recovered manually applying a metric that Garcia et al. [4] used for the same purpose. K-recovery scored an average of 75% accuracy on the architectures recovered. Garcia et al. [4] found that the best architecture recovery tool among the ones evaluated scored an average accuracy of 56%. About the comparison of these results, it must be considered that Garcia et al. recovered architecture of very large software projects, and they compared the results with the available actual architecture recovery. Garcia et al. analyzed projects with an average size of 10M SLOC.

In the following sections I will describe in detail the algorithms and data structure applied. Moreover, I will provide a formal definition of software component and dependencies graph for the purposes of this work.

The rest of the thesis is organized as follows. In chapter 2 I report some previous work in the field of architecture recovery. In chapter 3 I explain the logic behind K-recovery also with the analysis of a toy example. In chapter 4 I show the most important implementation details of K-recovery. In chapter 5 I show the results explaining in detail the analysis process. Finally in chapter 6 I discuss a conclusion and some future works.

CHAPTER 2

PREVIOUS WORK

In this section I discuss some previous work related to the field of architecture recovery. G Rasool et al. [8] describe a list of architecture recovery approaches. It can help to classify and compare the techniques mentioned later in this section. Here are explained some of these approaches.

- Data Flow based approaches.
  - These approaches recover the architecture of a software system based on the data flows among its entities such as classes. Based on these data flows these approaches outline the main components of a software system and their coupling.
- Knowledge based approaches.
  - These approaches focus on the representation of imperfect knowledge to be improved by reverse engineering process. An example of these approaches is K-means clustering algorithm. In fact, K-means first chooses randomly the cluster centers, and then it improves their positions through reverse engineering process. By contrast hierarchical clustering is based on perfect knowledge. It groups the closest observations since the first step of its algorithm in a final way.
- Design pattern based approaches.

  - These approaches recover the architecture of a software based on common implementation practices and patterns such as "(1) implementation-level entities that reside in the same source file, (2) entities in the same directory, (3) entities from the associated body and header files (e.g., .h and .c files in C), (4) entities that are leaves in a systems graph, (5) entities that are accessed by the majority of subsystems, (6) entities that depend on a large number of other resources, and (7) entities that belong to a subgraph obtained through dominance analysis." Garcia, Joshua [9]. These patterns are defined by domain experts. One application of these pattern could be the following. The classes that are defined in the same file belong to the same component. Thus, the identification of these patterns within the analyzed code contributes to recovery the architecture of a given software system.

- Formal method based approaches.

  - G. C. Gannod and Chang [10] use formal method techniques to re-implement a legacy procedural program to an OOP program. This work represent an example of formal method based approach for architecture recovery. They transforms sequences of code into formal specifications. For instance

```
for i := 0 to n
    if a[i] <= a[i+1]
        then
            m := a[i+1]
```

[10].

It is transformed into: $(\forall i : 0 \leq i \leq n : m \geq a[i])$ [10]. This last form is used write the software in another language, perhaps OOP, keeping its semantic. This is an effective methodology to retrieve the software architecture at a high level of abstraction. Then from its new representation it can be reimplemented adopting a different programming paradigm, perhaps OOP.

- Domain based approaches.
  - These approaches apply a reverse engineering process upon some known features of the analyzed software to perform its architecture recovery. Domain based approaches for architecture recovery take into consideration the domain of the analyzed software besides its structure.
- Clustering based approaches.
  - These approaches apply the unsupervised learning technique known as clustering. it groups the software entities in sets to recover the architecture of a system. This approach will be largely explained in the following chapter. These are existing approaches. Most of architecture recovery tools are clustering based.
- Metrics Based approaches.
  - These approaches apply operations research algorithms to solve the architecture recovery problem. Metrics Based approaches maximize or minimize some metrics

to partition the software entities. *Bunch* applies a metrics based approach [11] "The objective function used in Bunch is called Modularization Quality (MQ) and is defined as $MQ = \sum_{i=1}^{k} CF_i$. k is a partitions number of clusters. CF_i is the 'cluster factor' of cluster i, representing its coupling and cohesion, and is defined as

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{2\mu_i}{2\mu_i + \sum_{i=1, j \neq i}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & otherwise \end{cases}$$

μ_i is the number of edges within the cluster, which measures cohesion is the number of edges from cluster i to cluster j, which measures coupling." [4].

First of all, the number of different approaches to the problem tells us how large is the scope of the research in this field. I show later in this section that the current research status does not provide a reliable solution to the architecture recovery problem. The rest of the thesis discusses my contribution in the architecture recovery field.

G Rasool et al. [8] state that "The best features of domain based, program comprehension based, design pattern based and clustering based recovery approaches are used to recover the architecture of software systems under study." Nevertheless, I combined the clustering based approaches, domain based approaches, knowledge based approaches, and design pattern based approaches to perform architecture recovery.

J. Garcia et al. [4] selected and compared six architecture recovery techniques. I list the selected techniques grouped by approach type.

- Design pattern based + clustering based approach

  - Algorithm for Comprehension-Driven Clustering (ACDC), V Tzerpos et al. [12].

- Metrics Based approach

  - Bunch, S Mancoridis et al. [11].

- Clustering based approach

  - Weighted Combined Algorithm (WCA), O Maqbool et al. [13].
  - Zone-Based Recovery (ZBR), A. Corazza et al. [14], [15].
  - scaLable InforMation BOttleneck (LIMBO), P Andritsos et al. [16].
  - Architecture Recovery using Concerns (ARC), J. Garcia et al. [17].

J. Garcia et al. recovered the architecture of some open source software applying each of these architecture recovery techniques. The systems analyzed were: Arch Studio, Bash, Hadoop, Linux, Mozilla, and OODT. Finally, they compared the recovered architecture with the actual architecture using some cluster comparing metrics to measure the accuracy of their results. J. Garcia et al. [4] found that "relying upon even the top-performing techniques alone is insufficient to reliably perform an architectures recovery in general. This unpredictability of existing techniques, in concert with their overall unreliability, suggest that effective architecture recovery is likely to require extensive manual intervention - the very thing automated techniques have aimed to eliminate."

The architecture recovery techniques taken into consideration by J. Garcia et al. [4] are design pattern based, metrics based and clustering based. In average the clustering based techniques performed better. The analyzed clustering based techniques implement a hierarchical

clustering algorithm. “Hierarchical clustering algorithms produce a sequence of nested partitions. [...] At each stage, the algorithm either merges two clusters (agglomerative methods) or splits a cluster in two (divisive methods). The result of the clustering can be displayed in a tree-like structure, called a dendrogram” [18]. Therefore, Hierarchical clustering appears to be the most appropriate clustering technique for architecture recovery. Its advantages includes that the number of clusters is not defined in advance because the dendrogram can provide several granularity levels of clustering. A coupling metric drives the hierarchical clustering algorithm. However, J. Garcia et al. [4] obtained only up to 56% accuracy for architectures recovered with a hierarchical clustering. Consequently I explored a different clustering algorithm, focusing on the partitioning clustering. “The partitional or non-hierarchical document clustering approaches attempt a flat partitioning of a collection of documents into a predefined number of disjoint clusters.” [19]. In particular I implemented a customized version of the K-means algorithm. “K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters. To perform K-means clustering, we must first specify the desired number of clusters K; then the K-means algorithm will assign each observation to exactly one of the K clusters.” [20]. The following sections illustrates how I customized the algorithm exploiting the software engineering domain knowledge. With this approach K.recovery scored 75% accuracy in average on the analysis of two complex projects.

CHAPTER 3

LOGICAL DESCRIPTION

K-recovery recovers the architecture of a C++ project. The architecture recovery process is divided in two phases. The first performs the static parsing of the project source code to extract its dependencies graph. The second phase runs a customized version of K-means algorithm to detect the main components within the analyzed project. In the next two sections I provide a definition of software component and dependencies graph. These definitions are due since the clustering algorithm aim is to group the classes in components and it bases its computation upon a dependencies graph. Then I explain in detail the two phases from a logical point of view.

3.1 Definition of component

The component is a module of a software project. A software project contains different modules, the modules contain classes in the case of OOP projects. A component performs a specific task within the software project. The classes that are linked together to accomplish the same task can be grouped in the same component. For instance, the classes that manage the database interactions are pat of the DataBaseManagment component. In this work cluster is synonymous of component.

3.2 Definition of dependencies graph

The dependencies graph is a directed weighted graph that can be either connected or not based on the architecture recovered. A graph is defined by the set of its nodes N and the set of its edges or arcs E. In this case, N is the set of recovered classes, E contains the class-to-class dependencies and hierarchical relations. Hence, there are two kind of arcs within the dependencies graph. These arcs types are explained in the following two subsections.

3.2.1 Definition of class-to-class dependency

The class-to-class dependency represents the instantiation of a class and it is reflected as a weighted arc within the dependencies graph.

For instance, if class A instantiates class B once, then class B depends from A. In this case the direction of the arc goes from A to B. Furthermore, the weight of this arc depends on the number of instantiations. In this case the weight is one.

3.2.2 Definition of hierarchical relation

The hierarchical relation is represented as an arc of weight one within the dependencies graph. Hierarchical dependencies reflect the project hierarchical structure. A hierarchical relation represents the extension of a superclass by a base class. A base class is a class that inherits one or more entities from another class called superclass. In the dependencies graph a base class can inherit from more than one superclass.

The arc direction goes from the base class to its superclasses. Figure 3 gives an example of class hierarchy.

3.2.3 Definition of reachability

A class A reaches a class B within the dependencies graph if and only if there is at least one directed path that goes from class A to class B. The path is a set of sequential class-to-class directed dependency edges. Hierarchical relation edges are not taken into account for the purpose of reachability. The cost of a path is the sum of the weights of its arcs. A class A could reach a class B with more than one path.

3.2.3.1 Reachability cost

The reachability cost from a class A to a class B is the sum of the cost of all the possible paths from class A to class B.

3.3 First phase: parsing, static analysis

The static analysis phase analyzes the source code without compiling it. The parser uses the Flex [21] framework for lexical analysis of C++ source code to scan the source code.

As first step, the parser performs a recursive file system search for files with the following extension:

- cpp
- h
- cc
- hh

3.3.1 Flex Syntax Scanner

Flex is an open source standardized implementation of a syntax scanner, it scans the raw source code and produces a set of tokens related to the instructions found in the code. Flex produces a simplification of a context-free language. Flex works with Bison to form the Flex and Bison pattern. Bison analyzes the tokens to interpret the source code semantic. Bison applies a grammar to check whether the syntax is correct and to compile the source into object calls. "Flex and Bison are aging unix utilities that help you write very fast parsers for almost arbitrary file formats. Formally, they implement Look-Ahead-Left-Right (as opposed to 'recursive descent') parsing of non-ambiguous context-free (as opposed to 'natural language') grammars." [21]. For the purposes of this research Bison is not needed.

Flex performs a translation from source code instructions to tokens. This process is called tokenization. The purpose of the tokenization is to simplify the raw source code in order to better perform the semantic parsing. For instance, the token ID_B represents a generic identifier present in the code, STRING represents a generic string found in the code.

A set of regular expressions defines the code-token association. A token is generated whenever a regular expression matches the syntax encountered by Flex within the source code. At the end of this step all the source files are translated to their tokenized version. Appendix A and B reports the list of the regular expression for the code-token association and the list of the tokens used.

3.3.2 Parsing

The parser performs a recursive file system search for tokenized files after the Flex execution. The parser scans the tokens to detect: class definition and class instantiations. The parser builds the dependencies graph.

The parser is able to recognize class scopes. A class scope is the section of code that belongs to a class. There are two main assumptions applied on this phase. The parser analyzes only the instructions that are in a class scope. This means that class instantiations out of a class scope are ignored. The motivation of this assumption is that K-recovery bases the architecture recovery process on the analysis of relations among classes. Furthermore, it ignores *external* classes. This means that it ignores all the classes whose definition is not found within the project.

3.3.2.1 Class definition detection

The parser is able to detect the definition of a C++ class based on some C++ keywords. The most discriminant keyword for the definition of a class in C++ is *class*. The parser performs a set of steps to check whether the syntax under analysis meets one of the possible class definition patterns. The parser creates an internal representation of the recovered class if a class definition pattern is recognized. The internal representation collects the information recovered at this step such as class name, and its hierarchical relations.

An example of possible class definition pattern is the following: TOK_CLASS TOK_ID TOK_LBRACE. These tokens match the following snippet of code: “class ClassTest {”. Token

TOK_CLASS is associated to the keyword class, TOK_ID represents a generic identifier and TOK_LBRACE represents the symbol "{".

3.3.2.2 Class instantiation detection

Class instantiation detection is analogous to the class definition detection. It is syntax based. A set of patten based rules recognizes the syntax under analysis. The parser recovers the name of the class instantiated and the class scope in which it is instantiated. Then it creates an internal representation of the instantiated class. Then the parser creates a class-to-class dependency relation between the two classes within the dependencies graph. Figure 1 gives an example of class-to-class dependencies. The parser considers this class *external* if its definition is not found within the analyzed project.

An example of possible class instantiation pattern is the following: TOK_ID TOK_ID TOK_SEMI. These tokens match the following snippet of code: "ClassA a;". The token TOK_ID represents a generic identifier, TOK_SEMI represents the symbol ";".

The previous sequence of tokens match also other C++ patterns that are not a class instantiation. False positives are avoided thanks to the *external* tag. A false positive class will be considered external since its definition will not be included within the analyzed project source files.

3.4 Graph renderer

A web application renders the dependencies graph. The application implements a Force-Directed Graph [22]. The graph renderer aims to check the architecture recovery result. Figure 1, Figure 2, and Figure 3 show examples of rendering. Classes are represented by shapes

containing their names. A class-to-class dependency arc is represented by a gray line that goes from the top of the instantiated class and ends on the bottom of the class that instantiated it. The arc weight is not represented. Superclasses are highlighted in green. Hierarchical relations are represented by a green line and the base classes are bordered in green.

Graph renderer also visualizes the clusters. Each cluster has a different color. The cluster center class has a thicker white border. Figure 8 is an example of cluster rendering.

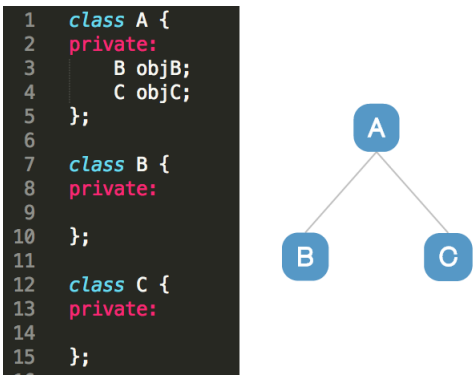

Figure 1: This figure shows the dependencies graph.

3.5 Second phase: clustering algorithm

In this phase K-recovery assigns each class to a component. In order to do it K-recovery applies a customized version of K-means clustering algorithm on the dependencies graph.

This section gives a definition of the K-means clustering algorithm. Furthermore, the K-means clustering algorithm customization is explained in detail including the assumptions and

```
1   class A {
2   private:
3       B b;
4       C c;
5       D d;
6   };
7
8   class B{
9   private:
10      C c;
11      D d;
12  };
13
14  class C{
15  private:
16      D d;
17  };
18
19  class D{
20
21  };
```

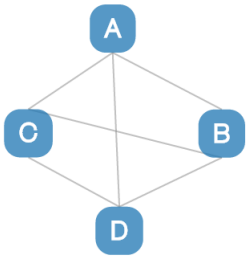

Figure 2: This figure shows a more complex snippet.

metrics adopted. Finally I discuss the coverage issue and a method to solve it. This K-covering limitation causes that some of the analyzed classes can not be assigned to any cluster due to some particular topology conditions.

3.5.1 K-means clustering algorithm

The K-means clustering algorithm seeks to partition n observation into k clusters. The number of clusters is decided in advance. As first step K-means selects randomly k centers among the observations. Each center represent a cluster. Then K-means assigns each observation to the cluster with the nearest mean at each iteration. Then it adjusts the centers position to the center of its observations at each iteration. K-means uses a proximity metric to evaluate the distances between two observations. K-means evaluates a convergence function at each iteration. Table I shows the K-means algorithm steps.

```
1   // Subclass example
2
3   class A {
4   private:
5
6   };
7
8   class B : public A{
9   private:
10
11  };
```

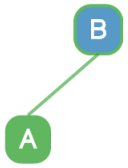

Figure 3: This figure shows the rendering of a dependencies graph.

TABLE I: K-MEANS CLUSTERING ALGORITHM

k-means clustering algorithm (*number_of_clusters*): pick *number_of_clusters* observations as centers do until convergence assign every observation to its nearest cluster center move each cluster center to the center of its assigned items evaluate convergence endfor
return *clusters_data_structure*

3.5.2 K-means customization and assumptions

K-means is usually applied to mono dimensional observations that can lay on a Cartesian coordinate system. In fact, K-means evaluates distances between observations. The domain of this research is far from the concept of Cartesian coordinate system. The observations are the recovered classes, and the number of centers is the number of components. Nonetheless,

the dependencies graph can provide information such us distance between nodes. The idea is to exploit the dependencies graph topology and weights in order to measure the proximity between nodes.

The initial centers selection is crucial for the clustering result. A wrong initial center selection can bring some issues such as bad clustering or the algorithm could not converge. K-means chooses the centers randomly because the observations are mono dimensional data, it makes difficult the application of a heuristic for an optimal initial centers selection.

Data recovered from the analyzed project are multi dimensional. It implies that a pre-processing phase could analyze the observation features and select the optimal initial centers. Furthermore, there is no need to adjust the cluster centers during the clustering since they are considered optimal. This is the strongest assumption of this work. In fact, the customized K-means algorithm applies a heuristic that selects k centers at the first step.

After the center selection, the algorithm evaluates the proximity of each class to each center. It assigns each class to the closest cluster center. The algorithm converge when the proximity of all the classes has been evaluated. Proximity is evaluated based on the reachability cost defined earlier in this chapter.

The algorithm does not take into account the hierarchical relation edges during the clustering. These edges will be considered in the post-processing phase which is explained later in this chapter. In the rest of the thesis cluster center and center are synonymous.

3.5.3 Center heuristics

Center heuristic is based on the assumption that the most essential classes in an OOP software project are the ones that uses the greatest number of other classes. This section presents two heuristics: the outgoing arcs based heuristic and the reachability based heuristic. The latter center heuristic takes into consideration the number of classes reachable by a given class to maximize the coverage.

These heuristics are exclusive, only one of them is applied at each algorithm execution. The expert user decides which center heuristic to use based on the analyzed project characteristics.

3.5.4 Center detection: outgoing arcs based

The outgoing arcs based heuristic selects as centers the classes that instantiate the greatest number of other classes. First the heuristic sorts all the classes based on the number of classes instantiated, then it selects the first k classes as centers.

The outgoing arcs based heuristic has two versions.

3.5.4.1 Instance fan-out

This version of the outgoing arcs based heuristic takes into consideration the number of class instantiations contained in a class. For example: if class A instantiates class B twice and class C once, Instance_fan-out(A) = 3

3.5.4.2 Class fan-out

This version of the outgoing arcs based heuristic takes into consideration the number of different classes instantiated by a class. For example: if class A instantiates class B twice and class C once, Class_fan-out(A) = 2

3.5.5 Center detection: reachability based

This heuristic is based on the reachability definition stated at the beginning of this chapter. This heuristic chooses as centers the classes from which it is possible to reach the greatest number of other classes. The set of classes reached by two classes could be overlapped. For this reason the reachability based heuristic has two versions. The first does not take into account the overlap, the second avoid the overlap.

3.5.5.1 Reachability with overlap

This version first sorts all the classes based on the number of other classes reached. Then it selects the first k class of the sorted list as centers.

3.5.5.2 Reachability no overlap

This version sorts all the classes based on the number of other classes reached. Then it selects the first class in the sorted list as center, then selects the next *k-1* classes in the sorted list that reach different classes from the ones reached by the classes already chosen as centers. Finally the sets of classes reached by each center class are disjoined.

3.5.6 Proximity metric

The customized K-means applies a proximity metric based on the reachability cost to measure distances between classes. The reachability cost indicates how strong two classes are bonded.

The first version of this metric was only based on the reachability cost. Some first experiments showed that a center class with high weighted outgoing arcs could easily include all the classes in its cluster. It produced very unbalanced clusters in term of number of classes

included. The second and final version introduces a decreasing factor based on the number of arcs that separates the two classes under proximity evaluation. This version reflects the idea that if a center class is linked to a target class through many other classes, then the target class may belong to another cluster. The second version partitions more equally the classes within the clusters.

However, the expert user can choose which of these two version to apply. These two metrics are exclusive, only one of them can be applied at each algorithm execution.

3.5.6.1 First version: weight based

This version calculates the proximity as the sum of the weights of all the possible weighted and directed paths between two classes. Cycles are not taken into account.

3.5.6.2 Second version: weight and step based

This version of the proximity metric introduces a decreasing factor to the previous version. It is so defined:

$$weight\_step\_based\_proximity(A, B) = weight\_based\_proximity(A, B) \times \frac{1}{e^{\lambda}} \qquad (3.1)$$

Where λ is the number of edges of the shortest path between the two classes. The shortest path is calculated as the directed path that links two classes with the smallest number of arcs among all the possible paths.

3.6 Architecture recovery of a toy example

This section shows in detail the execution of K-recovery. It shows how K-recovery recovers the architecture of a toy software example. The idea behind this software example is to simulate an architecture in which three group of classes cooperate sequentially. I assumed that a class A calculates some value using a set of classes (U, W, Z, Y). This is the first component. Then A calculates another value using a class B which uses another set of classes to perform the task (G, H, J). This represent the second component. Furthermore, B also uses class C in support of its task. Class C operates with a group of classes to produce the requested result (K, I, T, W). This represent the third component. Moreover, I assumed that the classes whitin each compoenet cooperate among them. I expect K.recovery to recover three clusters of classes: (A, U, W, Z, Y), (B, G, H, J), and (C, K, I, T, W).

3.6.1 Software example implementation and analysis

The example has 14 classes organized in three logical components. The source code is given as input to K-recovery. Appendix C contains the toy example source code. Flex converts the source files into their tokenized version. Appendix D contains a partial tokenization class A.

3.6.2 Parsing

The parser scans all the tokenized files produced by Flex and generates the dependencies graph. Here is the parser output for class A.

```
A size: 9

    files:

    ../source/componentBasedExample/A.cpp.xml at 1
```

```
    is superclass

    instances:

 B type: basic - at line 5 in ../source/componentBasedExample/A.cpp.xml

 U type: cpy_constr - at line 6 in ../source/componentBasedExample/A.cpp.xml

 W type: pointer - at line 7 in ../source/componentBasedExample/A.cpp.xml

 Z type: basic - at line 8 in ../source/componentBasedExample/A.cpp.xml

 Y type: constr - at line 9 in ../source/componentBasedExample/A.cpp.xml
```

Where *size* is the class size in terms of SLOC, *files* is the list of tokenized source files and line numbers in which the class has been detected. Then the parser tagged class A as superclass because class W and U extend class A. Finally there is the list of all the classes instantiated by A. For each instantiation the parser is able to recognize the instantiation type, the line and the file in which the instantiation is located within the analyzed project. These relations and dependencies stored in the parser internal representation of the analyzed project classes constitute the dependencies graph.

A multiple instantiation of the same class would be listed within the instances list. The class-to-class dependency arc weights are calculated based on the occurrence of the instantiated class in this list. For this example the weights of each class-to-class dependency arc is one.

3.6.3 Clustering algorithm

I executed the clustering algorithm with the following configuration:

- Center detection heuristic: outgoing arcs based, Instance fan-out

TABLE II: CLASS LIST FOR CENTER DETECTION

Class	Instantiated classes
A	B, U, W, Z, Y
B	C, G, H, J
C	K, I, T, V
U	W, Z, Y
K	I, T, V
I	K, T, V
T	I, K, V
V	I, T, K
Z	W, U, Y
W	Z, Y
Y	W, U
G	H, J
H	G, J
J	G, H

  - For this example instance fan-out and class fan-out produce the same clustering results because all the arcs are weighted one within the dependencies graph.

- Proximity metric: weight and step based
- Number of clusters: 3

3.6.3.1 Center detection

The algorithm applies the center heuristic selected by the user. Thus, the algorithm sorts the classes based on the number of classes they instantiate. Then, it selects the first k classes, in this case the first three are: classes A, B, and C. Table II represents the sorted classes list for this example.

3.6.3.2 Proximity calculation

After the centers selection, the algorithm calculates the proximity from each cluster center to each class. The algorithm assigns each class to the cluster of the closest cluster center.

Weight and step based proximity depends from reachability cost between two classes and the number of steps of the shortest path that links the two classes (Equation 3.1). A proximity equal to zero indicates that there is no connection between the two classes under evaluation.

Table III represents the proximity values for this example.

3.6.3.3 Clustering result

Figure 8 shows the clustering result for this example. As expected, K-recovery recovered the three components presented at the beginning of this section.

3.7 Clustering coverage issue

The cluster algorithm could not assign some classes to any cluster in some cases. It happened because these classes could not be reached by any center. In most of the cases, the not assigned classes were *abstract* classes. An abstract class in OOP is a class that defines a class structure but is never instantiated. Abstract classes are extended by base classes to inherit their structure such as methods and attributes.

In the first experiments the clustering coverage was 70% in the best cases, it means that 70% of classes were assigned to a cluster, the rest were not assigned. A detailed definition of coverage is reported later.

TABLE III: CLASS LIST FOR CENTER DETECTION

Center	Class	Reachability cost	steps	proximity
A	U	26	1	9.56
B	U	0	0	0.0
C	U	0	0	0.0
A	W	38	1	13.98
B	W	0	0	0.0
C	W	0	0	0.0
A	Z	26	1	9.56
B	Z	0	0	0.0
C	Z	0	0	0.0
A	Y	38	1	13.98
B	Y	0	0	0.0
C	Y	0	0	0.0
A	G	16	2	2.17
B	G	11	1	4.05
C	G	0	0	0.0
A	H	16	2	2.17
B	H	11	1	4.05
C	H	0	0	0.0
A	J	16	2	2.17
B	J	11	1	4.05
C	J	0	0	0.0
A	K	81	3	4.03
B	K	65	2	8.8
C	K	49	1	18.03
A	I	81	3	4.03
B	I	65	2	8.8
C	I	49	1	18.03
A	T	81	3	4.03
B	T	65	2	8.8
C	T	49	1	18.03
A	V	81	3	4.03
B	V	65	2	8.8
C	V	49	1	18.03

3.7.1 Post-processing: Enhancing the clustering coverage

A post-processing function integrates the clustering algorithm that performs a set of heuristics to increase the clustering coverage. The post-processing function performs a set of steps that are part of an heuristic. The rest of the section shows, with an example, that when the clustering algorithm performs the post-processing, the coverage raises up to 90% or 100%. The clustering coverage enhancement is due to the application of the following steps:

1. A superclass that is not included in any cluster is assigned to the cluster to which belongs the class that extends the superclass.
2. A base class that is not assigned to any cluster is assigned to the cluster to which its superclass belongs.
3. The classes that are not of the previous types and are not included in any cluster are assigned to their nearest cluster.
4. The rest of the not covered classes are assigned to the cluster to which belong the other classes in the same directory.

3.7.2 Post-processing application

In the rest of the thesis the cluster center name will also be the cluster name. This subsection shows the application of the coverage heuristics.

Figure 4 shows the clusters when the post-processing is off. D and F are the cluster centers, since they do not instantiate any class the clustering algorithm can not reach other classes to

be included in the clusters. The initial coverage is 18%. The post-processing assign the rest of the classes to a cluster by performing the following steps:

1. First the post-processing calculates the distance from the not assigned classes to the clusters. In particular, the nearest cluster to the class A and B is the cluster F. The cluster D is nearest cluster to the class E and C. This step takes into consideration only the dependencies edges (gray ones) and it ignores the hierarchical edges (green ones). Figure 5 shows the result of this step. After this step the coverage goes from 18% up to 55%. Application of heuristic 3.
2. This step assigns the superclasses to the cluster which most of their extended classes belong to. In the Figure 5 SuperClass2 is extended by four classes, but only three are part of a cluster. In particular classes E and C are part of the cluster D, class B is part of the cluster F. Hence, the right class for SuperClass2 is the cluster D. Application of heuristic 1. Then this step includes SuperClass1 to the cluster F and finally SuperClass3 to the cluster D. Figure 6 shows the result of this step. After this step the coverage goes from 55% up to 81%.
3. This step applies the heuristic 2. Hence, the step assigns the class G to the cluster D. After this step the coverage goes from 81% up to 91%.
4. Finally, this step assign the disconnected class H to the cluster D. In this case only one directory contains all the source code, so all the classes belong to the same directory. The class H is assigned to the cluster D because in the directory most of

the classes belong to the cluster D. Application of heuristic 4. This step raises the coverage up to 100%. The Figure 7 shows the final result.

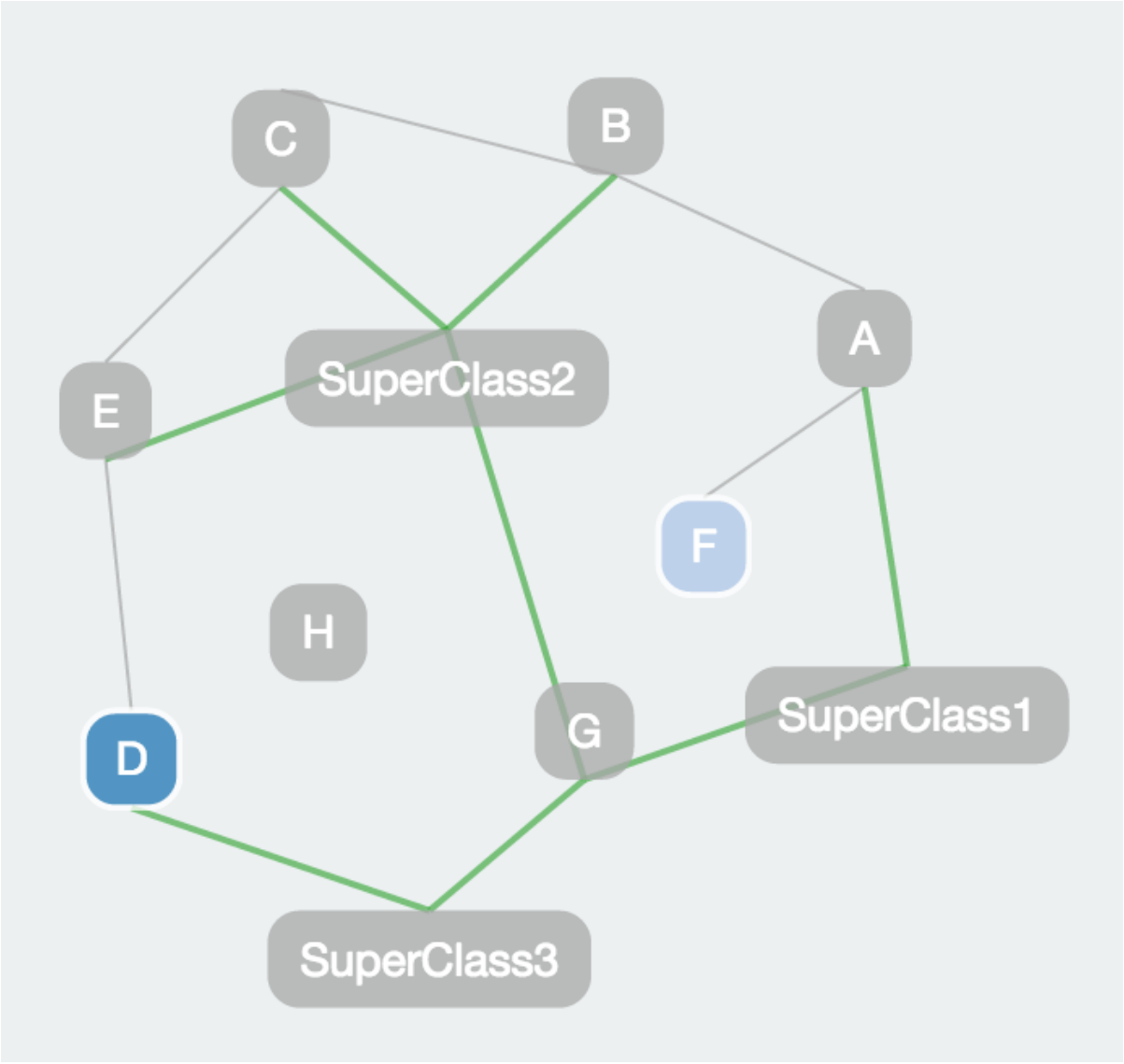

Figure 4: Coverage toy example: Post-processing off.

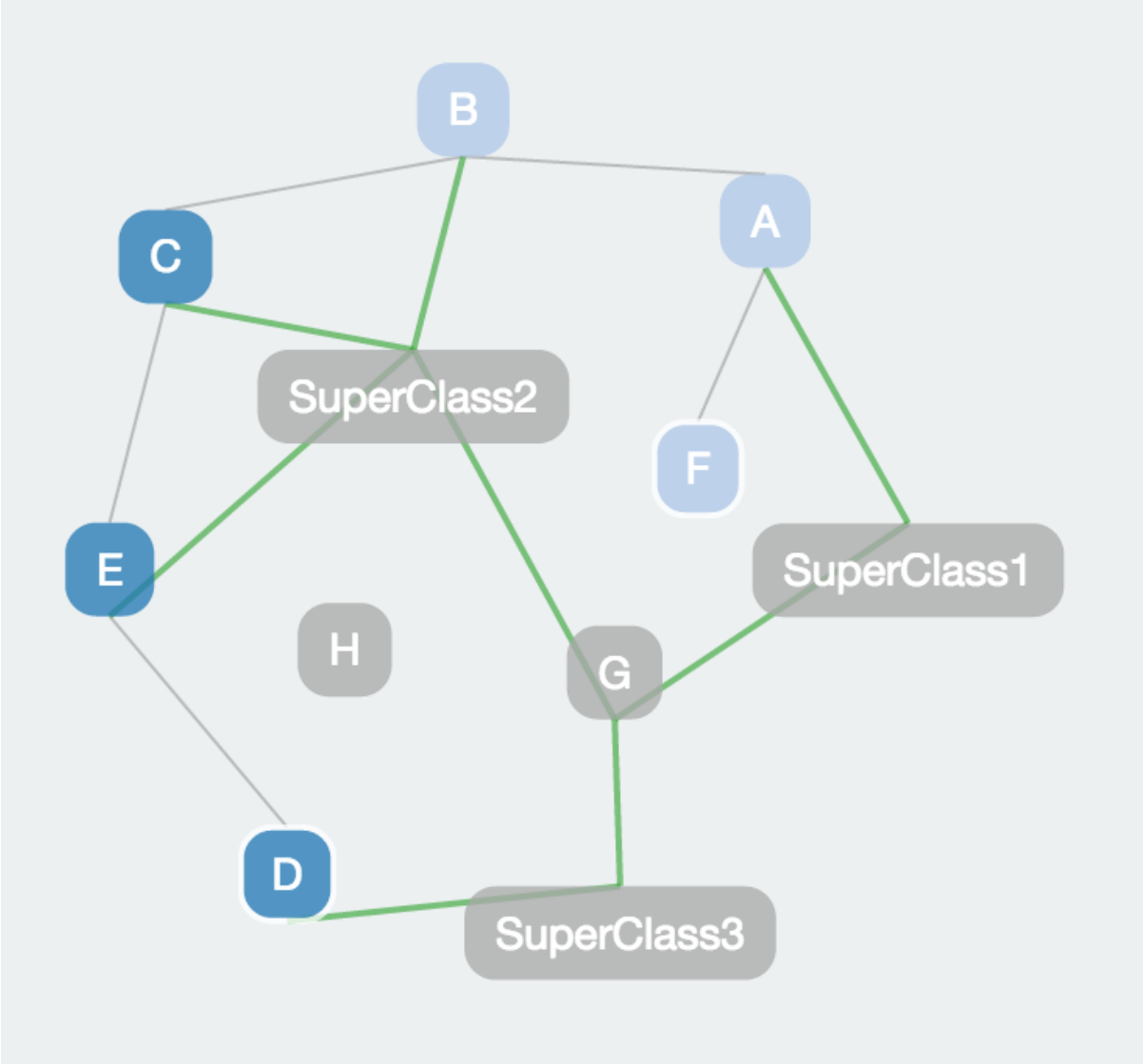

Figure 5: Coverage toy example: After the application of heuristic 3.

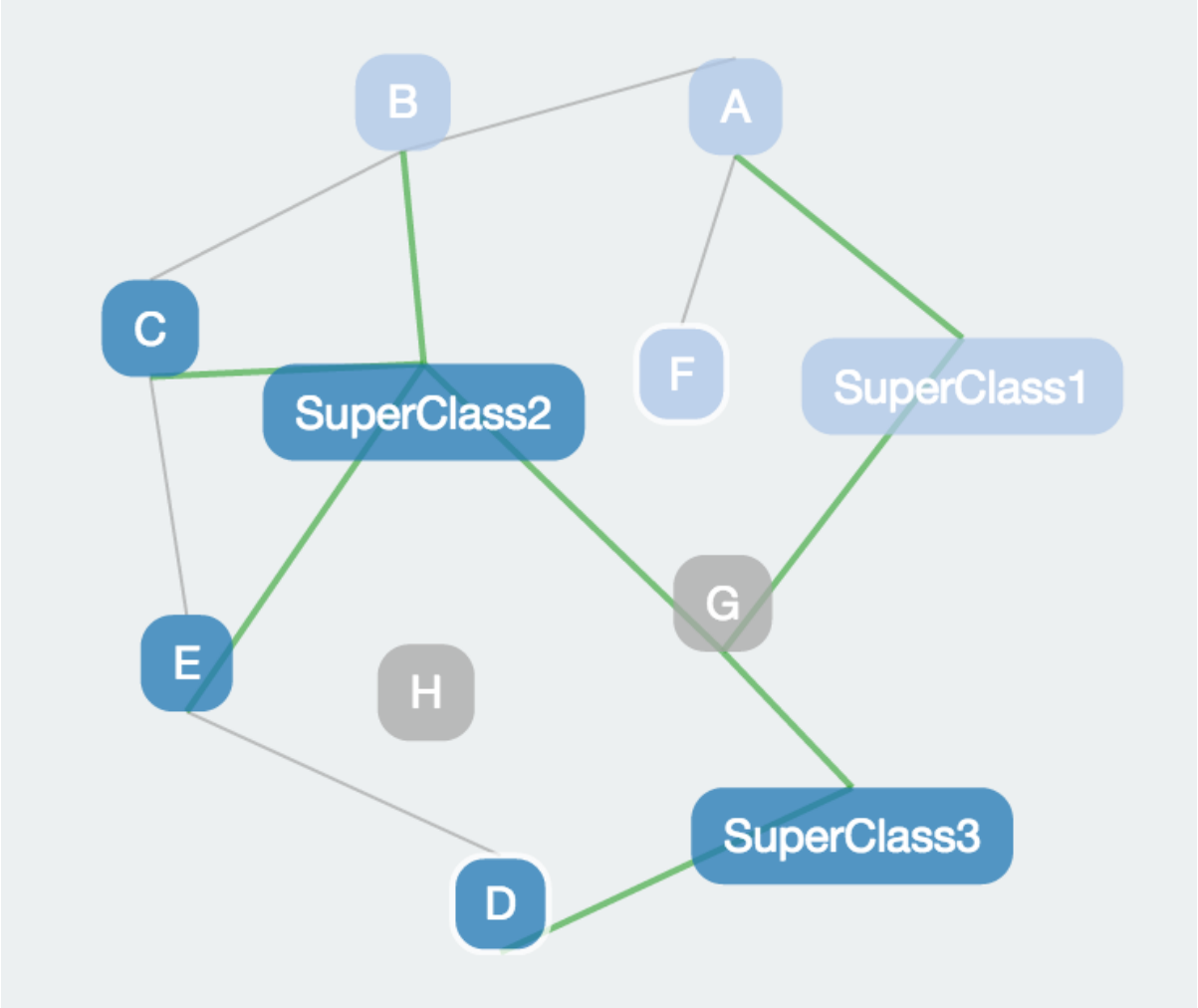

Figure 6: Coverage toy example: After the application of heuristic 1.

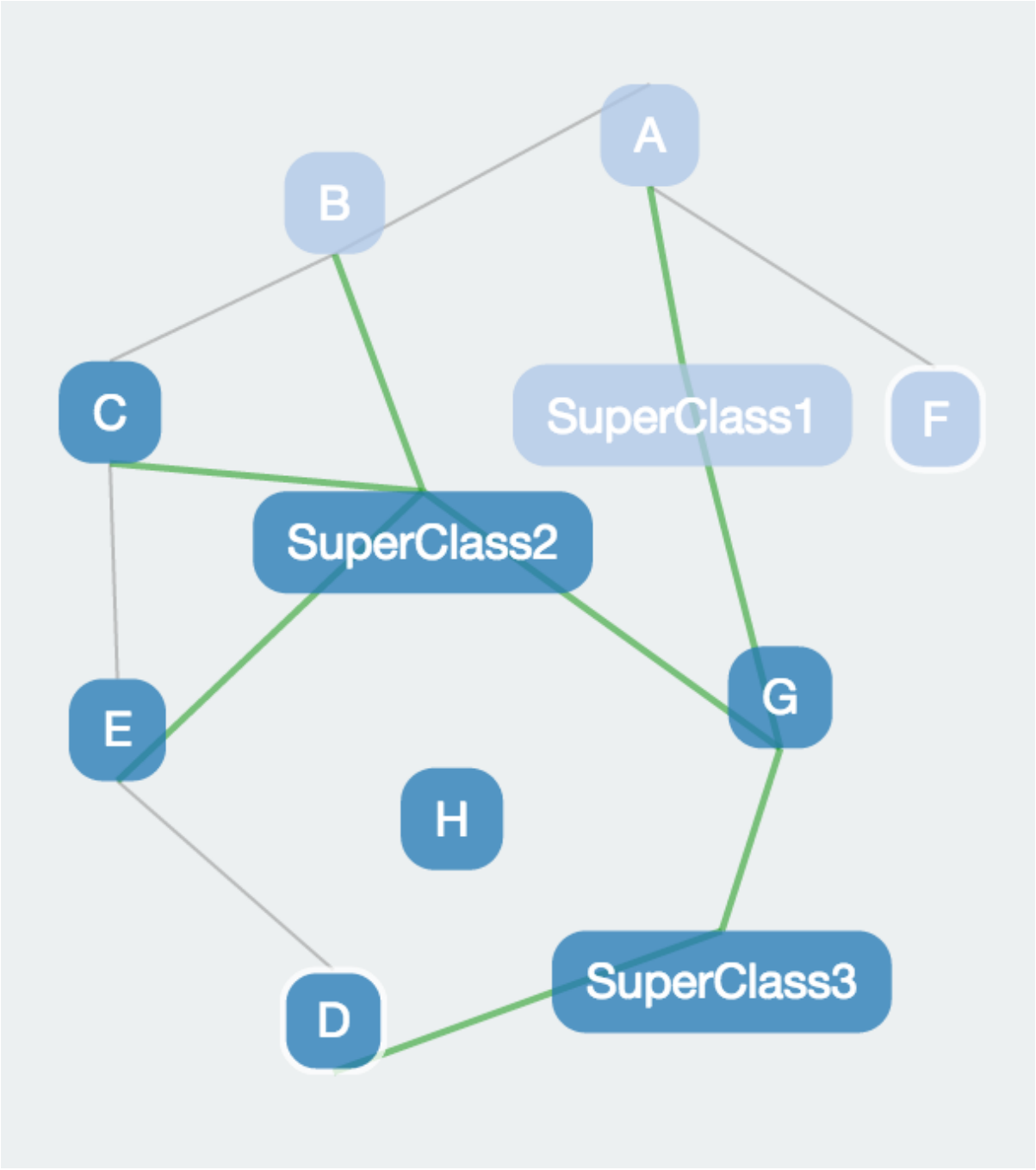

Figure 7: Coverage toy example: After the application of heuristic 2 and 4.

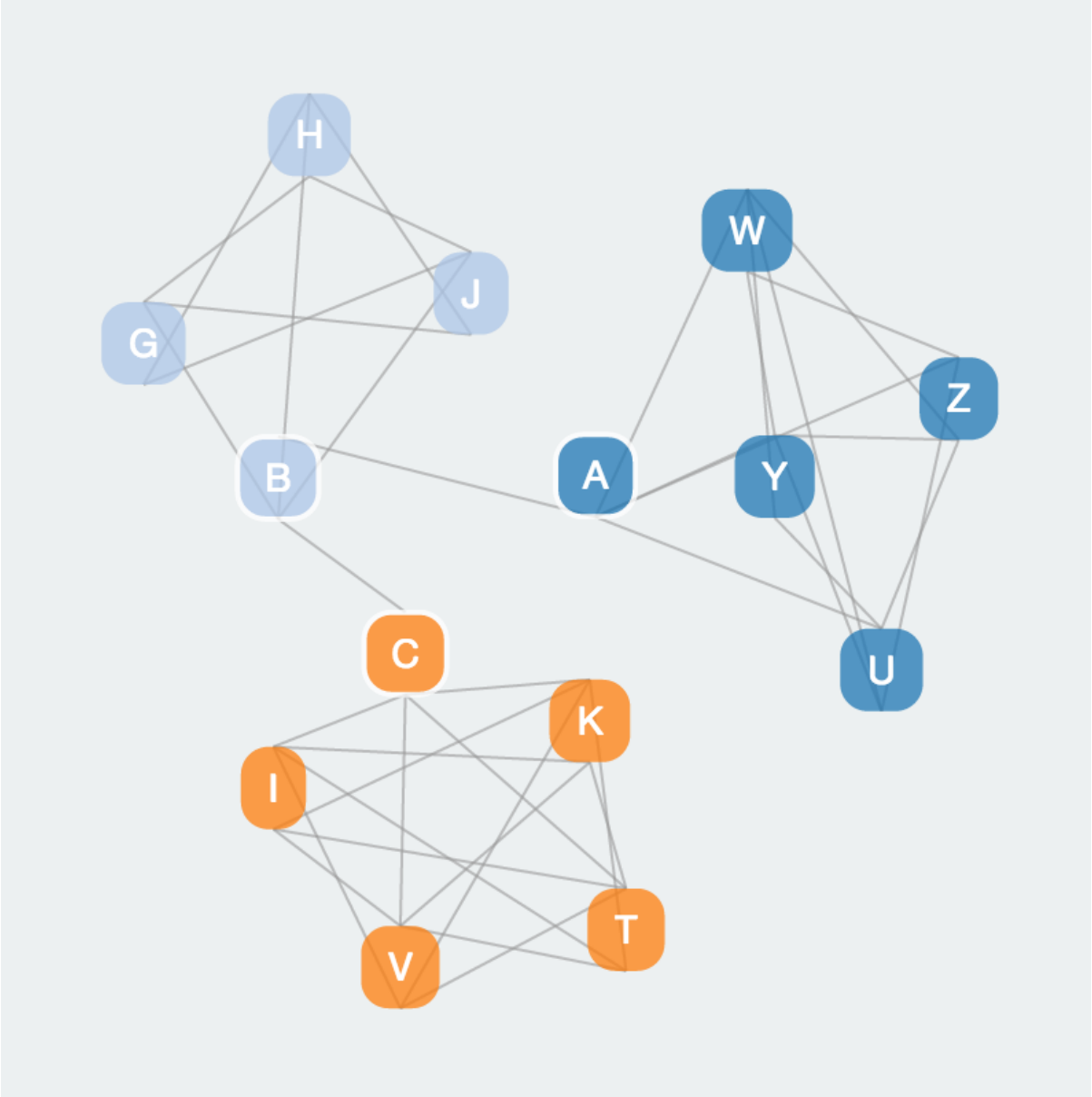

Figure 8: This figure shows the clustering of a toy example.

CHAPTER 4

IMPLEMENTATION DETAILS

This chapter presents the K-recovery modules and data structures. In particular in the first section I explain the benefit of the modularization and the adopted technologies. Then I present the data structures used to keep an internal representation of the analyzed classes and their relations.

4.1 K-recovery modules

The main scripts parser.py and graph.py are written in Python. Furthermore, Flex is compiled in C and the graph renderer is developed in JavaScript. K.recovery uses JSON and XML for data representation.

The previous chapter explains that K-recovery preforms architecture recovery in two phases. These two phases involve two different software modules. K-recovery has two independent modules that operate sequentially. The first module is programming language dependent and performs the static analysis. It parses the C++ project raw source code. The second module runs a clustering algorithm to detect the project components. The first module produces the dependencies graph file as output as a JSON file with a predefined structure. The second module takes the dependencies graph file as input. This organization allows the replacement of the first module as long as its output meets the JSON predefined structure. This is convenient since the first module is programming language dependent. Moreover, the syntax Flex analysis

is important because it speeds up the parsing process and it simplify the source code. In fact, Flex act like a filter, it translate into tokens only the keywords and features that are relevant for parsing process. Appendix A and B reports the list of regular expressions and tokens used.

Furthermore, a JavaScript application, called the renderer, performs a graphical visualization of the result.

Figure 9 illustrates the main modules and sub-modules explained previously in this section. Moreover, the diagram represents the input and output interactions.

4.2 Data structures

This section presents the K-recovery data structures.

4.2.1 StaticClass

K-recovery represents internally the classes recovered from the analyzed project. The Python class StaticClass represents the C++ class within K-recovery. The parser creates a StaticClass object for each class recovered. Here are listed the most significant StaticClass fields:

- name
  - name of the class. This field is used as primary key for research purposes.
- file
  - list of paths to the files that contain all the implementations of the class.
- external

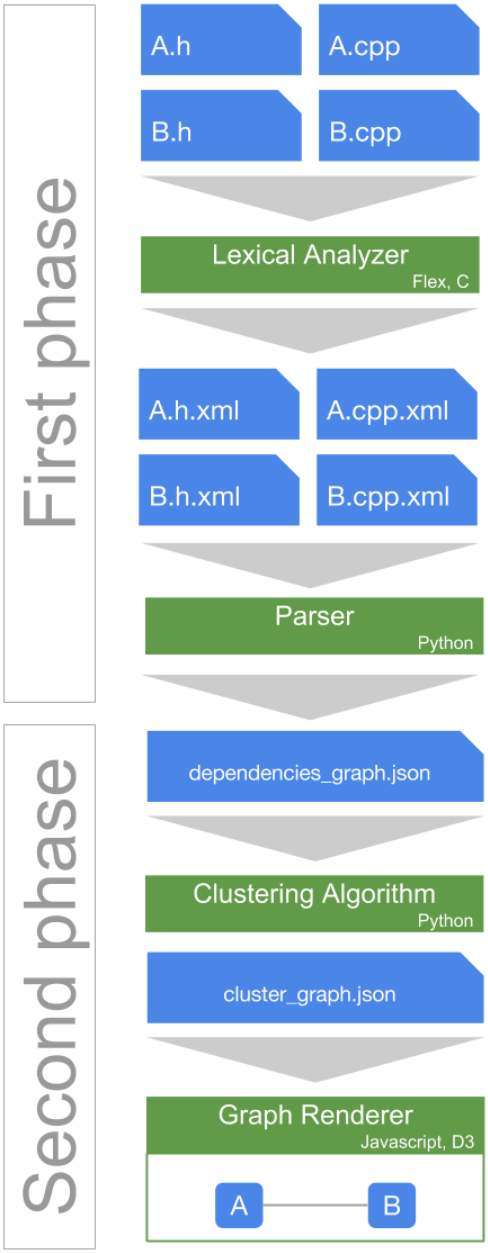

Figure 9: This diagram shows the K-recovery information flow.

  - true if the class is used within the project but is part of an external library, false otherwise.

- superclasses
  - list of the super classes if any, it contains other StaticClass object.
- instances
  - list of all the instantiated instances, it contains other StaticClass object.
- is_superclass
  - true if the class is a superclass in the C++ project, false otherwise.

4.2.2 Recovered classes list

The recovered classes list contains the classes recovered. The parser populates it dynamically during the parsing phase. Once a class is recovered the parser stores its information in a new StaticClass object and appends it to the recovered classes list.

4.2.3 Tokenized file

K-recovery analyzes the tokenized version of the project source files sequentially. The Python class Tokenized_file manages the operations on the tokenized file. It provides a cursor that returns the current token within the tokenized file. The cursor points to the first token within the tokenized file initially. The parser moves the cursor next during the parsing.

4.2.3.1 Token attributes

The parser scans the tokens to parse the code, for each token it can access to these information:

- Line number
  - It represents the number of line within the source file where the token has been generated.
- Text
  - Portion of source code represented by the token.
- Token name
  - It is the token name such as TOK_ID, TOK_SEMI etc.

4.2.4 Dependencies graph

StaticClass are the graph nodes and StaticClass pointers to other StaticClass objects are the graph edges. Each StaticClass is linked to other StaticClass objects by its lists *superclasses* or *instances*. These lists are attributes of StaticClass as explained previously in this section.

CHAPTER 5

RESULTS

This chapter presents the architecture recovered with K-recovery and their evaluations. Here follows the three analyzed systems description. The systems are open source projects available on GitHub [23]. GitHub provides free public repository hosting. This chapter discusses the MVC recovery of Freeablo [5], which is a video game. Moreover, I manually recovered the architecture of Wkhtmltopdf [6], which is a software that converts HTML pages into PDF or images. Then I compared it with the Wkhtmltopdf architecture recovered by K-recovery. I applied Cluster-to-Cluster Comparison to evaluate the architecture recovered. The recovered architecture for Wkhtmltopdf scored a Cluster-to-Cluster Comparison of 57%. Different K-recovery settings configurations produced a set of possible architectures for Wkhtmltopdf. The result reported is related to the architecture that performed better in the comparison. Furthermore, the last system analyzed is a deep learning framework called mxnet [7]. I recovered its architecture manually and I compared it with the one recovered by K-recovery scoring 93% accuracy.

Garcia et al. [4] in their research reported that the architecture recovery tools recovered six architectures with an accuracy of 33% in average. Moreover the architecture recovery tool that performed better scored 55% accuracy as average accuracy among the six architectures recovered. Garcia et al. [4] recovered architecture of large software projects. In average their analyzed software have 10M SLOC. Freeablo, Wkhtmltopdf, and mxnet average dimension

is 10K SLOC. K-recovery failed on the analysis of large software projects. It may be due to its scripting implementation that is slower and more memory consuming than a complied implementation. Nonetheless its results are significant because Cluster-to-Cluster Comparison value does not depend of the analyzed software size.

I based the manual architecture recovery on meaningful class names, comments, C++ namespaces, and folders. Furthermore, I took into consideration the source file organization within the file system. After this analysis I decided a possible number of components and ran K-recovery. In Freeablo I found an MVC organization, thus I recovered three logical components. In Wkhtmltopdf and mxnet I found some functional components that I will discuss in the next sections. Furthermore, in the rest of the chapter, the K-recovery components have a 'K-' prefix to be differentiated from the manually recovered components.

5.1 Cluster-to-Cluster Comparison

Cluster-to-Cluster Comparison ($c2c_{meas}$) is an accuracy metric adopted by Garcia et al. [4] to evaluate the architecture recovered accuracy. $C2c_{meas}$ is so defined:

$$c2c_{meas}(A, B) = \frac{|A \cap B|}{|A|} \times 100\% \tag{5.1}$$

Where A is the set of classes grouped in a K-recovery cluster. B is the set of classes grouped in a cluster of the ground truth architecture.

5.2 Freeablo: MVC recovery

This software is an action role-playing video game implemented in C++. It organizes its classes in an MVC paradigm.

Scalable and flexible software projects that involve a GUI organize their classes in three logical components. These components are Model View Control. This organization allows the developers to work independently on each component. The model component holds the data structures. While the control component manages the interactions between the other two components. Furthermore the view component is often not strongly coupled to the other two components. This means that the view component could be easily replaced by a different view component to manage the GUI on multiple type of devices. By contrast, control and model components are often more coupled.

The next two sessions show comparison between the manual architecture recovered and the K-recovery architecture recovered. The manual architecture recovered from Freeablo is partial because of the lack of discriminant features in the project. However the classes recovered are enough to outline the MVC components.

5.2.1 Manual architecture recovery

In this software I found an MVC organization of its classes. I grouped the most significant classes as follow:

- View component
  - MainWindow

  - GraphicsPage
  - Settings
  - Page
  - PlayPage

- Model component
  - World
  - Actor
  - Position
  - Monster
  - Player

- Control component
  - Renderer
  - ThreadManager
  - Level
  - LevelObjects
  - TileSet

5.2.2 K-recovery architecture recovery

K-recovery recovered the following components:

- K-View component (center: MainWindow)

  - MainWindow
  - MainWindow
  - GraphicsPage
  - Settings
  - PlayPage
  - ProcessInvoker
  - Page

- K-Model component (center: World)
  - World
  - Position
  - Actor
  - Monster
  - World
  - Player
  - DiabloExe
  - FAFile
  - Npc
  - FAIOFileInterface
  - RocketSDL2Renderer
  - RocketSDL2SystemInterface

- K-Control component (center: Renderer)
  - Renderer
  - RenderState
  - TileSet
  - FASpriteGroup
  - Level
  - Tileset
  - SpriteCache
  - SpriteGroup
  - SpriteManager
  - CelFile
  - Pal
  - Dun
  - MinPillar
  - Min
  - Sol
  - LevelObjects
  - ThreadManager
  - SpriteCacheBase
  - Room

– CelFrame

K-recovery recovered the MVC successfully. It grouped the most significant classes manually detected in the right logical component. Figure 10 shows the architecture recovered

K-recovery recognizes a high coupling between the control and model components and a low coupling between view components and the other components. It reflects the idea that the view component is less bonded to the other components, while model and control strongly cooperate together.

K-recovery measures the coupling within the range [0-1]. This value represents how much two clusters are bonded to each other. 0 means that the two clusters are not linked at all, while 1 means that the two clusters are strongly bonded. Here are the coupling values for the MVC components recovered:

- Control - Model: 0.9
- Control - View: 0
- Model - View: 0

In this case the view component seems completely disconnected to the other two. It is due to the fact that the parser does not recognize any link among these components. It is plausible because the view component is often instantiated by the main C++ function which is not in the scope of any class.

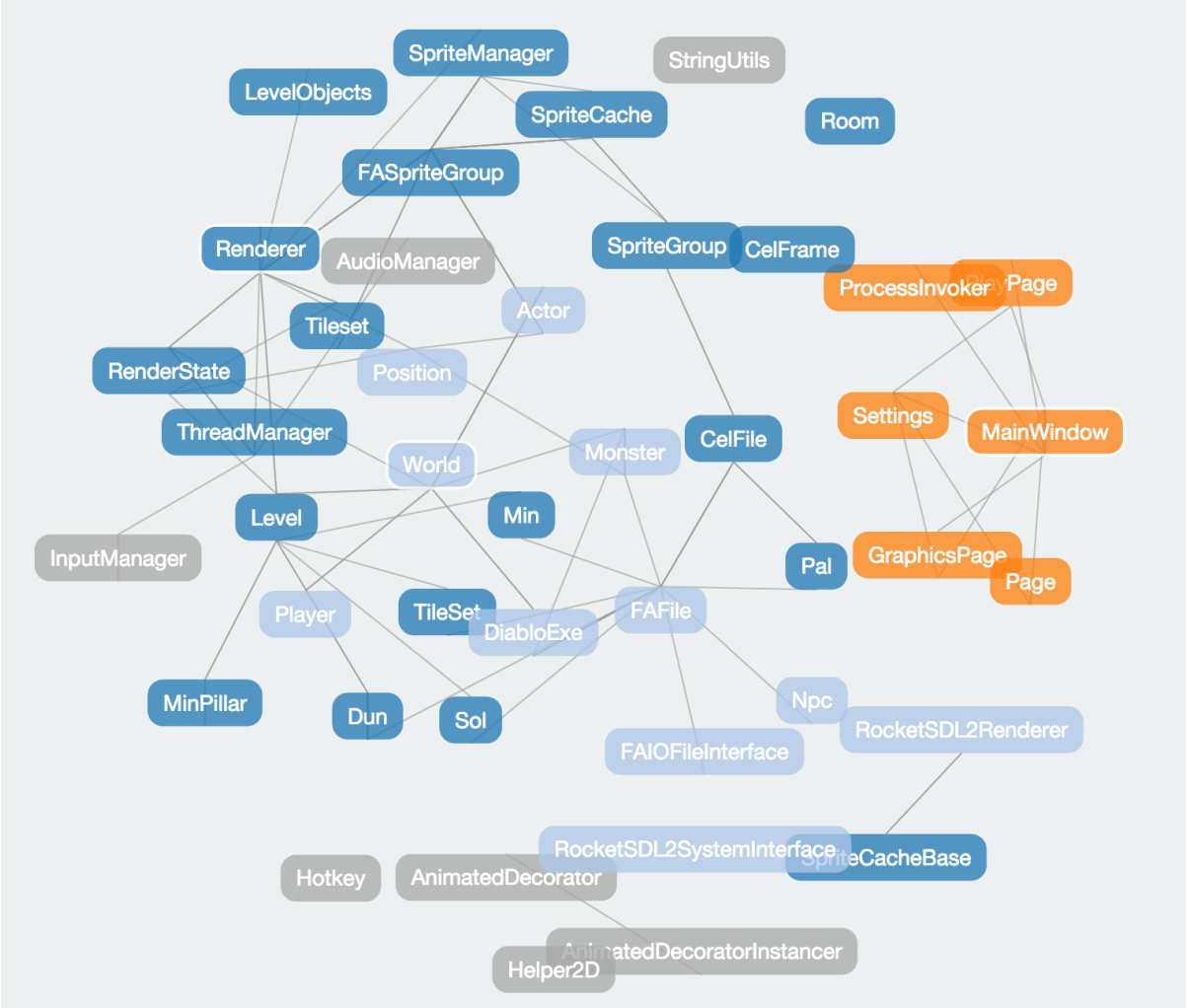

Figure 10: This figure shows the Freeablo recovered architecture.

5.3 Mxnet: architecture recovery

Mxnet is a deep learning framework. This is the largest project analyzed. It contains 84 classes and its size is 17K SLOC. I manually recovered 5 logical components. The documentation available for this project helped me to have a good understanding of the function of the main classes. Where the documentation was less detailed I relied to the source file organization and the class names.

5.3.1 Manual architecture recovery

I identified the following components: IO, Engine, Storage, Graph, Utilities.

IO is the component that manages the input output operations. The project organizes these classes in a folder called *io*. I assumed that the classes within this folder constitute the IO component. The documentation did not report any information about these classes. Furthermore, the classes within the folder *engine* are part of the Engine logical component and the documentation supported this assumption in this case. The engine component manages the parallel execution of threads. For this reason I assigned to this component also the classes whose name contained the word *thread*. Furthermore, the documentation reports a component that handles the storage across multiple devices and the main class is called *Storage*. I assumed that all the classes in *storage* folder may belong to the Storage component. This assumption is reinforced by the fact that all these classes names contain the word *storage*. Moreover the documentation reports a component that is used to represent a dynamically generated symbolic computation graph, its main class is called *Symbol*. I grouped all the classes within the folder *symbol* in the Graph component. Even in this case all the classes names contain the word

graph. Finally I considered the rest of the classes utility classes and assigned them to the Utility component. The motivation of this assumption is that the rest of the classes represent less significant data structures and operations on them. In fact, the folders containing these classes are *kvstore* (key-value store), *ndarray*, *operator*, and *common*. Table IV shows the manually recovered components and their classes.

5.3.2 K-recovery architecture recovery

K-recovery recovered successfully most of the manually recovered components. The cluster-to-cluster accuracy is 93%.

However K-recovery wrongly considered *GraphExecutor* class as the center class of K-Utility component. *GraphExecutor* is part of Graph component instead. Figure 11 shows the recovered architecture.

Table V represent the K-recovery recovered architecture. In each K-component the symbol '×' indicates that the class is not present in the related manually recovered component. Furthermore $c2c_{meas}$ is calculated for each component.

5.4 Wkhtmltopdf: architecture recovery

Wkhtmltopdf is a command line tool that converts an HTML page into a PDF file or into an image. It contains 46 classes and its size is 12K SLOC. The documentation for this project does not give any information about the project composition in term of logical components. Nonetheless I was able to recover five logical components. I based the architecture recovery on the class names and the folder where they were located.

TABLE IV: MXNET MANUALLY RECOVERED COMPONENTS

Engine	IO	Graph
CallbackOnComplete	IIterator	GraphExecutor
Engine	ImageAugmenter	GraphStorageAllocator
final	ImageLabelMap	Node
MXAPIThreadLocalStore	ImageNormalizeIter	StaticGraph
StreamManager	ImageRecordIOParser	Symbol
ThreadedEngine	ImageRecordIter	
ThreadedEnginePerDevice	ImageRecParserParam	
ThreadedEnginePooled	InstVector	
ThreadedVar		
ThreadPool		
Var		
Utilities		Storage
ActivationOp	Executor	CPUDeviceStorage
ActivationProp	FlattenProp	GPUDeviceStorage
BackwardOpWrapper	FnProperty	KVStore
BatchLoader	FullyConnectedOp	KVStoreDevice
BatchNormOp	FullyConnectedProp	KVStoreLocal
BatchNormProp	LeakyReLUOp	PinnedMemoryStorage
ConcatOp	LeakyReLUProp	Storage
ConcatProp	LocalResponseNormOp	StorageImpl
Context	LocalResponseNormProp	StorageManager
ConvolutionOp	NDArray	
ConvolutionProp	NDArrayFunction	
CuDNNActivationOp	Operator	
CuDNNConvolutionOp	OperatorProperty	
CuDNNLocalResponseNormOp	Opr	
CuDNNPoolingOp	PoolingOp	
DropoutOp	PoolingProp	
DropoutProp	ReshapeOp	
ElementWiseBinaryOp	ReshapeProp	
ElementWiseBinaryOpProp	ResourceManager	
ElementWiseSumOp	ResourceManagerImpl	
ElementWiseSumProp	ResourceManagerImpl	
SliceChannelOp	SliceChannelProp	
SoftmaxOp	SoftmaxProp	

TABLE V: MXNET K-RECOVERY RECOVERED COMPONENTS.

	K-Engine		K-IO	K-Graph
	CallbackOnComplete	×	BatchLoader	Node
	Engine		IIterator	Symbol
	final		ImageAugmenter	
×	FnProperty		ImageLabelMap	
×	Opr		ImageNormalizeIter	
	StreamManager		ImageRecordIOParser	
	ThreadedEngine		ImageRecordIter	
	ThreadedEnginePerDevice		ImageRecParserParam	
	ThreadedEnginePooled		InstVector	
	ThreadedVar		MNISTIter	
	ThreadPool		MNISTParam	
	Var		PrefetcherIter	
			TensorVector	
	$c2c_{meas} = 10/12 \times 100 = 83\%$		$c2c_{meas} = 92\%$	$c2c_{meas} = 100\%$

	K-Utilities			K-Storage
	ActivationOp	×	GraphStorageAllocator	CPUDeviceStorage
	ActivationProp		LeakyReLUOp	GPUDeviceStorage
	BackwardOpWrapper		LeakyReLUProp	PinnedMemoryStorage
	BatchNormOp		LocalResponseNormOp	Storage
	BatchNormProp		LocalResponseNormProp	StorageImpl
	ConcatOp		MXAPIThreadLocalStore	StorageManager
	ConcatProp		NDArray	
	Context	×	KVStore	
	ConvolutionOp	×	KVStoreDevice	
	ConvolutionProp	×	KVStoreLocal	
	CuDNNActivationOp		Operator	
	CuDNNConvolutionOp		OperatorProperty	
	CuDNNLocalResponseNormOp		PoolingOp	
	CuDNNPoolingOp		PoolingProp	
	DropoutOp		ReshapeOp	
	DropoutProp		ReshapeProp	
	ElementWiseBinaryOp		ResourceManager	
	ElementWiseBinaryOpProp		ResourceManagerImpl	
	ElementWiseSumOp		SliceChannelOp	
	ElementWiseSumProp		SliceChannelProp	
	Executor		SoftmaxOp	
	FlattenProp		SoftmaxProp	
	FullyConnectedOp	×	StaticGraph	
	FullyConnectedProp		NDArrayFunction	
×	GraphExecutor			
	$c2c_{meas} = 89\%$			$c2c_{meas} = 100\%$

Figure 11: This figure shows the mxnet recovered architecture.

5.4.1 Manual architecture recovery

Wkhtmltopdf download an HTML page then it converts it into PDF or image. Thus I expected a Network component, a PDF Converter component, and an Image Converter component. Furthermore the tool should interact with the command line for input output purposes. Hence I expected a IO component. The source files organization confirmed my ainitial ssumptions. In fact, the project contains a *pdf* and an *image* folder. Furthermore under *lib* folder I found classes related to the Internet connection and under *lib* folder I found classes related to the command line input output operations. Moreover I found classes that perform the Outline of the web page for the PDF conversion. Finally I found five components: Image Converter, IO, Network, Outline, PDF Converter. Table VI represents the portion of classes that I was able to recover.

5.4.2 K-recovery architecture recovery

K-recovery recovered successfully most of the components. However K-recovery did not recognize the Outline component. Furthermore the clustering is unbalanced, K-PDF Converter is way larger than K-Image Converter. Nonetheless K-PDF Converter includes all the manually recovered classes involved in the PDF conversion. In this case K-recovery scored 57% accuracy. Table VII shows the K-component and their cluster-to-cluster accuracy value. Figure 12 shows the architecture recovered.

5.5 Cluster evaluation metrics

K-recovery calculates some metrics at the end of the architecture recovery process. These metrics evaluates the quality of the clusters obtained. They take into account different aspects

Figure 12: This figure shows the Wkhtmltopdf recovered architecture.

TABLE VI: WKHTMLTOPDF MANUALLY RECOVERED COMPONENTS

Image Converter	IO	Network
ImageConverter	ArgHandler	MultiPageLoaderPrivate
ImageConverterPrivate	CommandLineParserBase	MyCookieJar
ImageGlobal	HtmlOutputter	MyNetworkAccessManager
MyImageConverter	ImageCommandLineParser	MyQWebPage
	ManOutputter	Web
	Outputter	LoaderObject
	PdfCommandLineParser	
	TextOutputter	
Outline	PDF Converter	
Outline	MyPdfConverter	
OutlineItem	PdfConverter	
OutlinePrivate	PdfConverterPrivate	
	PdfGlobal	
	PdfObject	
	TableOfContent	

such as percentage of classes clusterized, source file organization, cluster cohesion, and cluster coupling. In the following sections I give a definition of these evaluation metrics. The system calculates the value of *fitness*, *basic cohesion*, and *advanced cohesion* for each cluster. By contrast, *coupling values* is a matrix, thus it refers to all the clusters. Furthermore, a good clustering result should have a high cohesion and a low overall coupling. It means that the classes within the clusters are strongly bonded together while the clusters are slightly bonded.

These metrics do not affect in any way the architecture recovery process. They are only provided to the K-recovery user to better understand the clustering result.

TABLE VII: WKHTMLTOPDF K-RECOVERY RECOVERED COMPONENTS.

	K-Image Converter		K-IO	K-Network
	ImageConverter		ArgHandler	MultiPageLoaderPrivate
	ImageConverterPrivate		CommandLineParserBase	MyCookieJar
	ImageGlobal		ConverterPrivate	MyNetworkAccessManager
×	LoaderObject		HtmlOutputter	MyQWebPage
	MyImageConverter		ImageCommandLineParser	
			ManOutputter	
			Outputter	
			PdfCommandLineParser	
			TextOutputter	
		×	Web	
	$c2c_{meas} = 80\%$		$c2c_{meas} = 80\%$	$c2c_{meas} = 100\%$

	K-Outline		K-PDF Converter	
	ResourceObject		CropSettings	TempFile
			HeaderFooter	PdfConverterPrivate
			LoadGlobal	PdfGlobal
			LoadPage	PdfObject
			Margin	ProgressFeedback
			MultiPageLoader	Proxy
			MyLooksStyle	Reflect
			MyPdfConverter	ReflectClass
			Outline	ReflectImp
		×	OutlineItem	ReflectSimple
		×	OutlinePrivate	Size
		×	PageObject	TableOfContent
			PdfConverter	
	$c2c_{meas} = 0\%$		$c2c_{meas} = 24\%$	

5.5.1 Coverage

It indicates the percentage of classes assigned to a cluster. At the end of the clustering process the algorithm checks whether some classes are not assigned to any cluster yet. The target is to obtain coverage equal to one. The coverage is so defined:

$$coverage = \frac{number\_of\_classes\_assigned}{total\_number\_of\_classes} \tag{5.2}$$

5.5.2 Fitness value

This metric is based on the assumption that the classes of each component should be in the same folder. This is an assumption that is not always true. Many developers do not organize the classes of a component in the same folder. However in Freeablo classes of the same component were mostly organized in the same folder. This metric provides a value within the range [0-1], where 1 means that all the classes in a cluster are located in the same folder. By contrast, when this value is closer to 0 it means that the classes of the cluster are located in different folders.

5.5.3 Basic cohesion value

This metric measures how much the classes of a cluster are bonded together. To calculate the basic cohesion the algorithm counts all the outgoing arcs of each class of the cluster and divide it by the number of classes in the cluster.

$$basic\_cohesion(cluster) = \frac{\sum_{class \in cluster} class.number\_outgoing\_arcs}{|cluster|} \tag{5.3}$$

5.5.4 Advanced cohesion value

This metric is a variant of the basic cohesion value. It takes into account only the outgoing arcs directed to an internal class. An internal class is a class that is part of the cluster under analysis. This cohesion value is more significant than the basic one. In average this value is lower than the basic one.

$$advanced\_cohesion(cluster) = \frac{\sum_{class \in cluster} class.number\_outgoing\_internal\_arcs}{|cluster|} \tag{5.4}$$

5.5.5 coupling values

This metric produces a triangular matrix. The matrix represent the coupling values of every couple of clusters within the clustering result. The coupling between two classes is evaluated as the ratio between the arcs between the two clusters and all the cluster outgoing arcs. For example, cluster A is connected to cluster B with one arc, cluster A is also connected to C with another arc. Then the coupling is calculates as follows:

$$coupling(A, B) = \frac{arcs\_between\_A\_and\_B}{arcs_A + arcs_B} = \frac{1}{2 + 0} = 0.5 \tag{5.5}$$

CHAPTER 6

CONCLUSION

Architecture recovery field is a large research domain that needs to be explored. This work proposes K-recovery, a tool for architecture recovery that implements a partitioning clustering algorithm while most of the architecture recovery tools apply a hierarchical clustering technique. It means that K-recovery could produce completely novel results compared with the ones produced by the existing tools. In this thesis I compared K-recovery performance with the existing tools performance. The result of this comparison must be well analyzed. K-recovery scored in average 75% accuracy. Furthermore K-recovery was able to recover an MVC from a software video game. Garcia et al. [4] achieved 56% accuracy in the best case with the best tool under analysis. I applyed the same metric Garcia et al. used to evaluate the architecture recovery. This metric is called cluster-to-cluster comparison and it does not depend on the analyzed project size. However the first point to clarify is that Garcia et al. compared the recovered architecture with the actual architecture as ground truth. I recovered the architecture manually to compare it with the one recovered by K-recovery. Furthermore Garcia et al. analyzed project of 10M SLOC, while the biggest project I analyzed had almost 20K SLOC. Garcia et al. analyzed projects with a higher level of complexity such as Mozilla. K-recovery failed in the analysis of Mozilla. It showed that K-recovery is not so scalable. This failure may be due to the implementation choice. In fact, is preferable a compiled software than an interpreted one to perform such a heavy computation. As future work it could be considered

to re-implement K-recovery in C++. Moreover it could be implemented a method to decide automatically the number of component. It may be possible by running K-means several times with different settings configurations and increasing number of components. Then the process could stop when the cluster cohesion is maximum and the coupling is minimum.

After the comparison between K-recovery results and Garcia et al. results, I would say that K-recovery could be a starting point to enhance the state of the art of architecture recovery.

APPENDICES

Appendix A

REGULAR EXPRESSIONS

KEYWORD “auto” | “*break*” | “*case*” | “*continue*” | “$default$” | “*do*” | “for” | “*goto*” | “*register*” | “*switch*” | “*byte*” | “*while*” | “*throw*” | “*alignas*” | “*alignof*” | “*asm*” | “*catch*” | “*concept*” | “*constexpr*” | “*decltype*” | “*enum*” | “*explicit*” | “*export*” | “*extern*” | “$false$” | “*true*” | “*inline*” | “*mutable*” | “*namespace*” | “*noexcept*” | “*nullptr*” | “$reinterpret_cast$” | “*requires*” | “*static*” | “*static_assert*” | “*static_cast*” | “*struct*” | “*template*” | “*this*” | “$thread_local$” | “*try*” | “*typedef*” | “*typeid*” | “*typename*” | “*usign*” | “*virtual*” | “*volatile*”

OTHER_OPS“$\sim$ ” | “?” | “++” | “$--$” | “!” | “|“ | “$\wedge$” | “ << ” | “ >> ” | “ >>> ” | “$+=$ ” | “$-=$ ” | “$*=$ ” | “$/=$ ” | “$\wedge=$ ” | “$\%=$ ” | “ <<= ” | “ >>= ” | “$\&=$ ” | “ $|=$ ” | “ >>>= ” | “*and*” | “*and_eq*” | “*bitand*” | “*bitor*” | “*compl*” | “*const_cast*” | “*delete*” | “*dynamic_cast*” | “*not*” | “*not_eq*” | “*or*” | “*or_eq*” | “*sizeof*” | “*union*” | “*xor*” | “*xor_eq*” | “\”

Appendix A (continued)

ACCESS	“public” | “*private*” | “*protected*”
DIGIT	[0-9]
NON_ZERO_DIGIT	[1-9]
LITERAL	[a-zA-Z]
ID_A	“_”| ”$”\|$LITERAL$
ID_B	ID_A| $DIGIT$
NUMBER	NON_ZERO_DIGITDIGIT*| ”0”
NON_NUMBER	“0”DIGIT+
SPACES	[\t\n\r\f\v]
MULTILINE_COMMENT	“/*”([∧] | [*] + [∧ * /]) * [*] + [/]
STRING	“””[∧”\r\n] * “”” | ”’”[∧”\r\n] * ”’”

Appendix B

TOKENS

“new” return TOK_NEW_OPERAND;

“operator” return TOK_OPERATOR;

“class” return TOK_CLASS;

“const” return TOK_CONST;

“else” return TOK_ELSE;

“if” return TOK_IF;

“int” return TOK_FUNDAMENTAL_TYPE;

“void” return TOK_FUNDAMENTAL_TYPE;

“bool” return TOK_FUNDAMENTAL_TYPE;

“char” return TOK_FUNDAMENTAL_TYPE;

“short” return TOK_FUNDAMENTAL_TYPE;

“long” return TOK_FUNDAMENTAL_TYPE;

“float” return TOK_FUNDAMENTAL_TYPE;

“double” return TOK_FUNDAMENTAL_TYPE;

“wchar_t” return TOK_FUNDAMENTAL_TYPE;

Appendix B (continued)

“char16_t”	return TOK_FUNDAMENTAL_TYPE;
“char32_t”	return TOK_FUNDAMENTAL_TYPE;
“String”	return TOK_FUNDAMENTAL_TYPE;
“string”	return TOK_FUNDAMENTAL_TYPE;
“signed”	return TOK_SIGN_UNSIGN;
“unsigned”	return TOK_SIGN_UNSIGN;
“return”	return TOK_RETURN;
STRING	return TOK_STRING;
ACCESS	return TOK_ACCESS;
OTHER_OPS	return TOK_OTHER_OPS;
KEYWORD	return TOK_KEYWORD;
NON_NUMBER	return TOK_DONT_CARE;
ID_AID_B*	return TOK_ID;
NUMBER	return TOK_NUM;
“(”	return TOK_LPAREN;

Appendix B (continued)

“)”	return TOK_RPAREN;
“[”	return TOK_LSQ;
“]”	return TOK_RSQ;
“{”	return TOK_LBRACE;
“}”	return TOK_RBRACE;
“=”	return TOK_ASSIGN;
“>”	return TOK_GT;
“<”	return TOK_LT;
“==”	return TOK_EQ;
“>=”	return TOK_GE;
“<=”	return TOK_LE;
“!=”	return TOK_NE;
“+”	return TOK_PLUS;
“-”	return TOK_MINUS;
“*”	return TOK_MULT;

Appendix B (continued)

“/”	return TOK_DIV;
“:”	return TOK_COLON;
“::”	return TOK_COLON_COLON;
“;”	return TOK_SEMI;
“,”	return TOK_COMMA;
“&”	return TOK_AMP;
“#”	return TOK_HASH;
MULTILINE_COMMENT	;
COMMENT	;
SPACES	;
.	return TOK_DONT_CARE;

Appendix C

COVERAGE TOY EXAMPLE SOURCE CODE

```
class A {

  public:

    A() {

            std::cout<<"I'm A"<<std::endl;

            B b;

            U a = b;

            W *d;

            Z z;

            Y a(b);

        }

};

class U: A{

public:

    W w;

    Z z;

    Y y;

};

class W: A{
```

Appendix C (continued)

```
public:
    Z z;
    Y y;
};
class Z: U{
public:
    W w;
    U z;
    Y y;
};
class Y: U{
public:
    W w;
    U z;
}
class B {
    public:
        B(){
            std::cout<<"I'm B"<<std::endl;
            C c;
            G g;
```

Appendix C (continued)

```
            H h;
            J j;
        }
};
class G{
public:
    H h;
    J j;
};
class H{
public:
    G h;
    J j;
};
class J{
public:
    H h;
    G j;
};
class C {
    public:
```

Appendix C (continued)

```
        C(){ std::cout<<"I'm C"<<std::endl; }

        K k;

        I i;

        T t;

        V v;

};

class K {

    public:

        I i;

        T t;

        V v;

};

class I {

    public:

        K i;

        T t;

        V v;

};

class T {

    public:

        I i;
```

Appendix C (continued)

```
        K t;

        V v;

};

class V {

    public:

        I i;

        T t;

        K v;

};
```

Appendix D

COVERAGE TOY EXAMPLE: PARTIAL TOKENIZATION OF CLASS A

```
<?xml version="1.0"?>
<file name="../source/componentBasedExample/A.cpp">
<line number="1" token="TOK_CLASS" text=""/>
<line number="1" token="TOK_ID" text="A"/>
<line number="1" token="TOK_LBRACE" text=""/>
<line number="2" token="TOK_ACCESS" text=""/>
<line number="2" token="TOK_COLON" text=""/>
<line number="3" token="TOK_ID" text="A"/>
<line number="3" token="TOK_LPAREN" text=""/>
<line number="3" token="TOK_RPAREN" text=""/>
<line number="3" token="TOK_LBRACE" text=""/>
<line number="4" token="TOK_ID" text="std"/>
<line number="4" token="TOK_COLON_COLON" text=""/>
<line number="4" token="TOK_ID" text="cout"/>
<line number="4" token="TOK_OTHER_OPS" text=""/>
<line number="4" token="TOK_STRING" text=""/>
<line number="4" token="TOK_OTHER_OPS" text=""/>
<line number="4" token="TOK_ID" text="std"/>
```

Appendix D (continued)

```
<line number="4" token="TOK_COLON_COLON" text=""/>

<line number="4" token="TOK_ID" text="endl"/>

<line number="4" token="TOK_SEMI" text=""/>

<line number="5" token="TOK_ID" text="B"/>

<line number="5" token="TOK_ID" text="b"/>

<line number="5" token="TOK_SEMI" text=""/>

...
```

CITED LITERATURE

1. Parnas, D. L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12):1053–1058, 1972.

2. Commission, I. E. et al.: Systems and software engineering: architecture description. ISO, 2011.

3. Perry, D. E. and Wolf, A. L.: Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, 17(4):40–52, 1992.

4. Garcia, J., Ivkovic, I., and Medvidovic, N.: A comparative analysis of software architecture recovery techniques. In Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, pages 486–496. IEEE, 2013.

5. freeablo. `https://github.com/wheybags/freeablo`. Accessed on August 5, 2015.

6. wkhtmltopdf. `https://github.com/wkhtmltopdf/wkhtmltopdf`. Accessed on August 5, 2015.

7. mxnet. `https://github.com/dmlc/mxnet`. Accessed on August 5, 2015.

8. Rasool, G. and Asif, N.: Software architecture recovery. International Journal of Computer, Information, and Systems Science, and Engineering, 1(3), 2007.

9. Garcia, J.: A Unified Framework for Studying Architectural Decay of Software Systems. University of Southern California, 2014.

10. Gannod, G. C. and Cheng, B. H.: A two-phase approach to reverse engineering using formal methods. In Formal Methods in Programming and Their Applications, pages 335–348. Springer, 1993.

11. Mancoridis, S., Mitchell, B. S., Chen, Y., and Gansner, E. R.: Bunch: A clustering tool for the recovery and maintenance of software system structures. In Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, pages 50–59. IEEE, 1999.

CITED LITERATURE (continued)

12. Tzerpos, V. and Holt, R. C.: Acdc: An algorithm for comprehension-driven clustering. In wcre, page 258. IEEE, 2000.

13. Maqbool, O. and Babri, H. A.: The weighted combined algorithm: A linkage algorithm for software clustering. In Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on, pages 15–24. IEEE, 2004.

14. Corazza, A., Di Martino, S., and Scanniello, G.: A probabilistic based approach towards software system clustering. In Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on, pages 88–96. IEEE, 2010.

15. Corazza, A., Di Martino, S., Maggio, V., and Scanniello, G.: Investigating the use of lexical information for software system clustering. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pages 35–44. IEEE, 2011.

16. Andritsos, P. and Tzerpos, V.: Information-theoretic software clustering. Software Engineering, IEEE Transactions on, 31(2):150–165, 2005.

17. Garcia, J., Popescu, D., Mattmann, C., Medvidovic, N., and Cai, Y.: Enhancing architectural recovery using concerns. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, pages 552–555. IEEE Computer Society, 2011.

18. Rokach, L. and Maimon, O.: Clustering methods. In Data mining and knowledge discovery handbook, pages 321–352. Springer, 2005.

19. Sathiyakumari, K., Manimekalai, G., Preamsudha, V., and Scholar, M. P.: A survey on various approaches in document clustering. International Journal of Computer Technology and Application (IJCTA), 2(5):1534–1539, 2011.

20. James, G., Witten, D., Hastie, T., and Tibshirani, R.: An introduction to statistical learning. Springer, 2013.

21. Flex. `http://aquamentus.com/flex_bison.html`. Accessed on November 16, 2015.

22. Force-Directed Graph. `http://bl.ocks.org/mbostock/4062045`. Accessed on November 16, 2015.

23. GitHub. `https://github.com/`. Accessed on November 16, 2015.

VITA

NAME	Alessandro Chetta
EDUCATION	
	Master of Science in Computer Science, University of Illinois at Chicago, 2016, USA
	Master of Science in Computer Science, Politecnico di Milano, 2016, Italy
	Bachelor's Degree in Computer Science, Politecnico di Milano, 2013, Italy