Architecture-Dependent Software Optimization Metrics: Optimizing Kieker using Evolutionary Search

Natalie Kaufhold

2026

Kiel Software Engineering Research

KSER, Volume 13

Edited by

Prof. Dr. Wilhelm Hasselbring (hasselbring@email.uni-kiel.de)

DOI: 10.38071/2026-04022-1

This work is published under the Creative Commons Attribution 4.0 International license (https://creativecommons.org/licenses/by/4.0/deed.en).

The electronic open access version of this work is permanently available on MACAU (https://macau.uni-kiel.de), Open Access Repository of Kiel University

eISSN: 3053-4186

Abstract

For software re-modularization the cohesion and coupling metrics are the most common ones [Mka+15]. We will answer the question, whether *pipe-and-filter*-specific metrics complement cohesion and coupling metrics in providing satisfying re-modularization suggestions for the *pipe-and-filter*-architecture using Evolutionary Search. Therefore, we will use *Kieker* as the software under test. With *Kieker* using a *pipe-and-filter*-architecture [HH20; HWH12], we will focus on a subset of *pipe-and-filter*-specific metrics and provide multiple sets of metrics as fitness functions.

Contents

1 Introduction **1**
1.1 Motivation, Goals and Research Questions 1
1.2 Document Structure 2

2 Foundations **3**
2.1 NSGA-II 3
2.2 The *pipe-and-filter*-Architecture 5
2.3 Kieker and TeeTime 7
2.4 Related Works and the Significance of this Thesis 8

3 Metrics Discussion **10**
3.1 Metrics Introduction 10
3.1.1 Coupling and Cohesion 11
3.1.2 Filter-based Fan-in and Fan-out Coupling 16
3.1.3 Pipe-based PullThroughputMetric 18
3.1.4 #PaFs 19
3.2 Suggested Sets of Metrics 20
3.3 Inverting maximization Problems 21
3.3.1 Lack of Cohesion 21
3.3.2 Lack of Fan-in Coupling 22
3.3.3 Lack of PullThroughputMetric 22

4 Technical Necessities **23**
4.1 Data type & Operations 23
4.1.1 Operations on Filters 25
4.1.2 Operations on Pipes 26
4.1.3 Parsing to Tuple Specification 32
4.2 Analyzing the Kieker Analysis Part 33
4.3 Throughput Approximation for Evolutionary Search 36
4.4 Filter Technique 37

5 Evaluation **39**
5.1 Experiments 39
5.2 Limitations and Problems 40
5.3 Results of both Approaches 41
5.3.1 Package Re-Modularization 41

5.3.2 P&F Optimization 43
5.3.3 *Kieker*-Developer Feedback 50

6 Conclusion **57**

7 Prospects and Future Work **58**

A Used Tools **60**

B Run Result Files **67**
B.1 Package Structure Re-Modularization Runs 67
B.2 P&F Optimization Runs 73

C Personal Correspondence **82**

D Thrown Exceptions **84**

Bibliography **91**

List of Figures

2.1 The genetic architecture optimization re-drawn version of Figure 2 by [SHJ25b] 4
2.2 Own illustration of a package hierarchy with a root package and n child packages on $n-1$ layers, adapted from [Kau25]; node labels modified and annotations added in this work 5
2.3 Own illustration of a *pipe-and-filter*-architecture as a DAG, based on [Ado16, Fig. 2.1]; edge and node labels adapted in this work 6
2.4 Own illustration of a *pipe-and-filter*-architecture where f_2 forwards data both to f_3 and f_4 as either shown as sub-graph in [Dob03, Fig. 1] or [WCB01, Fig. 5]; edge and node labels adapted in this work 6
2.5 Own illustration of a *pipe-and-filter*-architecture where f_1 and f_2 connect to f_3, which forwards data to f_4 as shown as sub-graph in [Dob03, Fig. 1]; edge and node labels adapted in this work 6
2.6 High-level Architecture of Kieker taken of [Kie] 8

3.1 Before either operation 14
3.2 Before either operation with bidirectional edges 14
3.3 Before either operation with two isolated classes 15
3.4 Before either operation worst case 15
3.5 One edge between packages and more packages in general 15
3.6 After merging unit A and B 16
3.7 Small packages with maximal cohesion and coupling 16
3.8 Large packages with maximal cohesion and coupling 16

4.1 Re-drawn class-diagram of `MultiLevelModularization`-class from [Kau25] . . 24
4.2 Re-drawn class-diagram of Node-class from [Kau25] 25
4.3 A pipe between an AbstractProducerStage and an AbstarctConsumerStage using the presented graph representation from Section 2.2 26
4.4 Port-class inheriting the Node-class from [Kau25] 28
4.5 Composition: Each ProgramUnit instance of [Kau25] has one instance of Port 29
4.6 Own Illustration based on source-code [Wal+26]: Example class diagram: AbstractProducerStage as parent of ModelRepositoryStage and ModelSource (and other stages). 31
4.7 Call Graph for Tool Allen Upper Limit using CallGraph plugin 33
4.8 Directed Pipe between two Ports with Throughput n before and after splitting operation using the presented graph representation in Section 2.2 36

List of Figures

5.1 Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 0 and 1 Table B.2, Table B.3 44
5.2 Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 2 and 3 Table B.4, Table B.5 45
5.3 Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 4 and 5 Table B.2, Table B.7 46
5.4 Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 6 and 7 Table B.2, Table B.3 47
5.5 Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layer8 Table B.10 48
5.6 Values for an excerpt of 9 of 1731 optimizations and the original state for the Metrics-set (Lack of Fan-in Coupling, Fan-out Coupling) of run 2026-07-16-14-12-44 Table B.14, Table B.15 51
5.7 Values for an excerpt of 9 of 1731 optimizations for the Metrics-set (Lack of Fan-in Coupling, Lack of PullThroughputMetric, #PaFs) of run 2026-07-16-14-12-44 Table B.16, Table B.17 52
5.8 Values for an excerpt of 9 of 1427 optimizations and the original state for the Metrics-set (Lack of Fan-in Coupling, Lack of PullThroughputMetric) of run 2026-07-16-14-12-44 Table B.18, Table B.19 53
5.9 Values for an excerpt of 9 of 1137 optimizations and the original state for the Metrics-set (Lack of Fan-in Coupling, #PaFs) of run 2026-07-16-14-12-44 Table B.20, Table B.21 54
5.10 Values for an excerpt of 9 of 1009 optimizations and the original state for the Metrics-set (Lack of PullThroughputMetric, #PaFs) of run 2026-07-21-18-37-02 Table B.22, Table B.23 55

List of Tables

4.1 Approaches to implement *pipe-and-filter*-architecture into `MultiLevelModularization` 24

5.1 Mapping of P&F-Jobs to used Sets of Metrics 40

A.1 Used Tools 61

B.1 The average number of changed units per optimization in run 2026-07-14-14-32-26 73
B.2 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 0 73
B.3 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 1 74
B.4 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 2 74
B.5 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 3 74
B.6 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 4 75
B.7 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 5 75
B.8 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 6 75
B.9 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 7 76
B.10 Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 8 76
B.11 Added Port Statistics of both July P&F Runs 76
B.12 Added Port Statistics of 2026-08-12-19-52-33 P&F Run 76
B.13 Port Type Statistics of both July P&F Runs 77
B.14 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 01 Layer 0 77
B.15 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 01 Layer 1 77
B.16 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 02 Layer 0 78
B.17 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 02 Layer 1 78
B.18 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 03 Layer 0 78
B.19 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 03 Layer 1 79
B.20 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 04 Layer 0 79
B.21 Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 04 Layer 1 79
B.22 Excerpt of Metrics.csv of run 2026-07-21-18-37-02 Job 05 Layer 0 80
B.23 Excerpt of Metrics.csv of run 2026-07-21-18-37-02 Job 05 Layer 1 80

List of Code

A.1 Example Prompts 60
B.1 Excerpt of the optimizations.txt: Original Modularization LayerGraphMap and layer 0 67
B.2 Excerpt of the optimizations.txt: Optimization-0 Layers 67
B.3 Excerpt of the optimizations.txt: Optimization 1 LayerGraphMap, Layer 0 and an excerpt of Layer 1 72
B.4 Excerpt of the optimization.txt of 2026-07-16-14-12-44 Job 02: Optimization-0 73
D.1 ClassCastException when starting Kieker collector 84
D.2 NullPointerException: CSV header is null 87
D.3 Example exception: Missing xmi-file 88
D.4 IllegalArgumentException: mergePort: Different processedData Types 89
D.5 IllegalArgumentException: mergePort: Different ProgramUnitType and not adjacent 90

List of Algorithms

1 Merge Random Ports of same Type 30
2 Merge Adjacent Ports of different Types 30
3 Split Random Port 31

Abbreviations

API: Application programming interface
COF: Coupling Factor
DAG: Directed acyclic graph
DNN: Deep Neuronal Network
MAFF: Metrics as Fitness Functions
NSGA: Nondominated sorting genetic algorithm
PaFs: Pipe-and-filter performance antipatterns
SMC: Software Module Clustering

Chapter 1

Introduction

1.1 Motivation, Goals and Research Questions

The main goal of this thesis is to decide, whether it is recommendable to invest time and resources in architecture-specific metrics, or whether the general well-known metrics cohesion and coupling are sufficient.

Kieker is an observability framework which shall be improved regarding sustainability, re-usability and high-quality [Sus; Yan+25]. The industrial impact of the *Kieker* observability framework and its predecessor is noteworthy [HH20; HWH12]. Therefore, it is of interest to suggest optimizations of *Kieker* itself. During writing this thesis *Kieker* uses a *pipe-and-filter*-architecture for analyzing the monitoring records provided by the configured monitoring probes [HH20; HWH12]. Hence, in the best case scenario the provided optimization suggestions are considering the *pipe-and-filter*-architecture to be as accurate as possible. Therefore, we will decide for and discuss a small set of appropriate metrics. For generating the optimizations suggestions a multilevel NSGA-II-framework provided by the AG Software Engineering is used to re-modularize *Kieker* from which an older version has already been used to suggest optimizations of research software [SHJ25b]. In this context multilevel means that the NSGA-II-framework can work with an arbitrary large hierarchical package depth and layers are defined as the set of nested nodes on the same level in the package hierarchy [Kau25]. This multilevel NSGA-II-framework currently uses structural lack of cohesion and coupling as software optimization metrics [SHJ25b].

Resulting in the following main research question, we will focus on:

RQ: Do *pipe-and-filter*-specific metrics complement cohesion and coupling metrics in providing satisfying re-modularization suggestions for the *pipe-and-filter*-architecture using Evolutionary Search?

In order to answer the main research question, we will discuss the following sub-research questions to a certain extent:

SQ1: Are cohesion and coupling sufficient to provide satisfying re-modularization suggestions for a software using the *pipe-and-filter*-architecture using Evolutionary Search?

SQ2: Does a subset of metrics out of the suggested set of metrics exist which provides satisfying re-modularization suggestions for the *pipe-and-filter*-architecture using Evolutionary Search?

SQ3: If multiple sets of metrics are existing, which are satisfying SQ2, then will there be one which is preferable and why?

SQ4: Does a set of mutation- and recombination-operations exist to provide optimization suggestions for a *pipe-and-filter*-architecture?

SQ5: How to adjust the given multilevel NSGA-II-implementation in order to consider a *pipe-and-filter*-architecture?

We will use two approaches and compare their results: One will optimize *Kieker* for package re-modularization, the other will optimize *Kieker* for *pipe-and-filter*-specific re-modularization.

1.2 Document Structure

This thesis is structured as following: Firstly, are the motivation, goals and research-questions discussed in Section 1.1. Then the necessary foundations and state-of-the-art with related works are in Chapter 2. The fitness of both cohesion and coupling and *pipe-and-filter*-specific metrics to be used as fitness functions for an evolutionary search is discussed in Chapter 3. Also, a filter technique to reduce the number of provided and stored optimization suggestions is presented. Secondly, the technical necessities are discussed in Chapter 4 to be able to both re-modularize *Kieker* with cohesion- and coupling-metrics as well as altering *Kieker*'s *pipe-and-filter*-structure to optimize *pipe-and-filter*-specific metrics. These include a data type and genetic operations. Afterward is a narrow set of results evaluated in Chapter 5. Finally, Chapter 6 contains a conclusion to the provided research-questions and in Chapter 7 are prospects and future works presented.

Chapter 2

Foundations

2.1 NSGA-II

The problem of optimizing software quality can be reduced to a multi- or many-objective optimization problem, because software quality can be measured using well-defined metrics [Arc+18]. Thus, the multi-objective genetic algorithm NSGA-II is used to re-modularize *Kieker* with evolutionary search. NSGA is the abbreviation for non-dominated sorting genetic algorithm [Deb+02]. The term re-modularization describes the solving of the software modularization problem as a version of the graph partitioning problem [Can+16].

The Pareto-front is defined as the set of vectors of the trade-off optimal solutions, the Pareto-optimal solutions, [NZE05] and the values of the used fitness functions for each solution in this set are representing the vectors [Can+16]. The Pareto-Dominance compares two candidates with the same number of objectives s.t., one solution Pareto-dominates another, if and only if, it is at least as good in all objectives and better in at least one objective. Furthermore, a Pareto-optimal solution is non-dominated by every other solution [NZE05].

After Deb et al., the NSGA-II computes a set of Pareto-optimal solutions and has a runtime of $\mathcal{O}(MN^2)$ with M being the number of objectives and N being the population size. To be precise, NSGA-II uses a set of metrics as fitness functions and generates multiple fronts each population with the goal to converge to the true Pareto-optimal front. In order to generate a new population, operations like binary tournament selection, recombination and mutation are performed [Deb+02]. "Each solution is assigned a fitness (or rank) equal to its non-domination level [...]" [Deb+02, p. 185] and the non-domination level depends on the objectives. Hence, the fitness of a solution depends directly on the objectives [Deb+02]. With the domination count equalling zero for each member of the first non-dominated front [Deb+02], the first front only consists of Pareto-optimal solutions. When we mention the non-dominated front, we will mean the first front, because after Deb et al. all the members of every other front are at least dominated by the members of the first front [Deb+02]. Even using one metric for each layer with $n \in \mathbb{N}$ layers will lead to n objectives to optimize and that becomes a problem the moment there are either $n > 3$ layers or $n \geqslant 2$ and $M \in \{2, 3\}$ objectives each layer.

Figure 2.1 is a re-drawn version of Figure 2 by Schnoor, Hasselbring, and Jung and visualizes how the process to generate Pareto-solutions of the given NSGA-II-framework

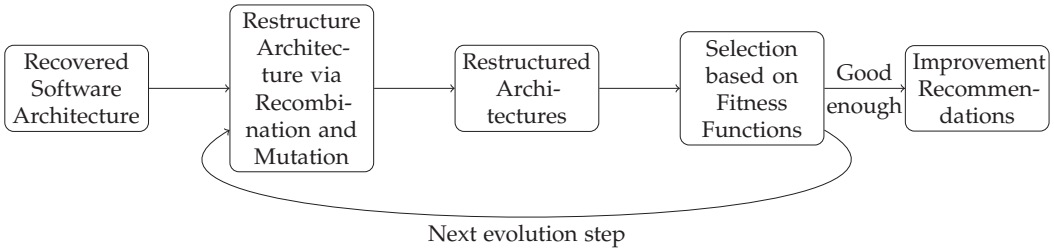

Figure 2.1. The genetic architecture optimization re-drawn version of Figure 2 by [SHJ25b]

works.

We will use a multilevel NSGA-II-framework further described in [Kau25]. In this context multilevel means that the NSGA-II-framework can consider package hierarchies with an arbitrary large tree-depth as shown in Figure 2.2 [Kau25]. Considering an arbitrary large tree-depth is relevant, because most programming languages, e.g., Java, allow as many package layers as decided by the developer [AB24]. The terms level and layer will be used interchangeable henceforth.

Haastert implements in [Haa23], among targeted and advanced operations, randomized merging and splitting operations as mutations and one recombination operation for our NSGA-II-framework. The current multilevel mutation and recombination operations implemented by Kaufhold in [Kau25] are based on the random splitting and merging operations by Haastert These are further used for the re-modularization of *Kieker*'s package-hierarchy. The split, merge and recombination operations used for the *pipe-and-filter*-architecture are discussed in Section 4.1.

With a fitness function being "[...] the characterisation of what is considered to be a good solution [...]" [HJ01, p. 834] a fitness function can be a single metric [HC04] or an aggregation of multiple metrics, because aggregating multiple metrics still fulfils the mentioned criterion. Due to the direct relation between metrics and fitness functions, architecture-dependent metrics can be used for NSGA-II as fitness functions to suggest optimizations for different software architectures as long as they satisfy the four properties *Large search space, low computational complexity, approximate continuity* and *absence of known optimal solutions* mentioned by Harman and Clark in [HC04] and further discussed in Section 3.1.

Nonetheless, NSGA-II has got the following constraint: "When $M > 3$, only the first rank may be assigned to every solution as almost all population individuals become non-dominated with each other [Khare 2003; Wang et al. 2010]." [Mka+15, as cited on p. 6] To work around this problem we will consider only sets of metrics with a size of at most three.

As prospect, it can be of interest to use a many-objective evolutionary algorithm, e.g., NSGA-III, which can work with more than three objectives [Mka+15], for the re-modularization. Alternatively, global aggregated sets of metrics could be discussed instead of the local layer-wise sets.

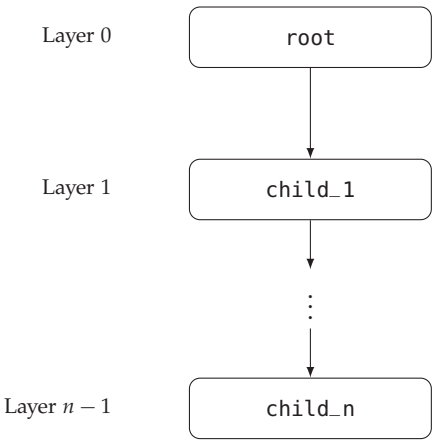

Figure 2.2. Own illustration of a package hierarchy with a root package and n child packages on $n-1$ layers, adapted from [Kau25]; node labels modified and annotations added in this work

2.2 The *pipe-and-filter*-Architecture

According to [Dob03] the *pipe-and-filter*-architecture consists of two main parts: Pipes and filters, where the filters are connected to each other through pipes. Data is sent through the pipes from one filter to another and processed in the filters [Ado16; Dob03]. Wulf, Hasselbring, and Ohlemacher mention in [WHO17] two additional components: Ports and configurations, where ports connect a filter with a pipe and can be used to ensure type-safety. Wulf describes in [Wul19] a configuration as a connected set of filters and pipes based on the introduction of a configuration by Abowd et al. 1993. We further discuss ports in Section 2.3 and define all components as a class of modules. A common example for a *pipe-and-filter*-structure are UNIX pipes [Ado16; Dob03].

Based on the assumption of Doberkat in [Dob03] the *pipe-and-filter*-architecture can be represented as a directed acyclic graph (DAG), as shown in Figure 2.3, where the filters are the nodes and the pipes are the edges.

Additionally, Hirzel et al. define in [Hir+14] the stream graph as a representation of a streaming application and mention pipelines and operators, which filter data. Leading to the assumption that an application using a *pipe-and-filter*-architecture is a streaming application. This assumption is further backed by Adolf in [Ado16] citing both Monroe et al. and Taylor, Medvidović, and Dashofy (Adolf cites an older version of [TMD10] from 2009 than we have access to), who mention a stream of data, e.g., characters [TMD10], being processed in a filter. As well as Doberkat also mentions "[...] that computations are performed in a stream like fashion [...]" [Dob03, p. 37] in a *pipe-and-filter*-architecture. Hirzel et al. further state that a stream graph can be cyclic, which can be seen as a generalization of the DAG representation by Doberkat.

Figure 2.3 is based on [Ado16, Fig. 2.1] but with edge labels like the graphs in [Dob03].

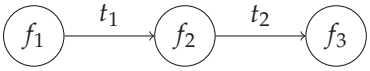

Figure 2.3. Own illustration of a *pipe-and-filter*-architecture as a DAG, based on [Ado16, Fig. 2.1]; edge and node labels adapted in this work

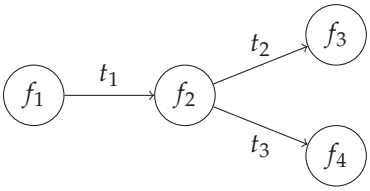

Figure 2.4. Own illustration of a *pipe-and-filter*-architecture where f_2 forwards data both to f_3 and f_4 as either shown as sub-graph in [Dob03, Fig. 1] or [WCB01, Fig. 5]; edge and node labels adapted in this work

However, instead of labeling the edges with the pre-image and image, like Doberkat does, in this thesis the edges will be weighted with the throughput t_i with $i \in \{1, 2, \ldots, n\}$ and n being the number of pipes, henceforth. This way the weight of the edges of the graph can be used to calculate metrics corresponding to the throughput.

Furthermore, it is possible to design a *pipe-and-filter*-architecture s.t., one filter sends data through multiple pipes to multiple other filters in parallel as shown in Figure 2.4 or s.t., one filter receives data from multiple pipes as shown in Figure 2.5 [WHO17].

The here presented graph representation can be seen as a directed coupling graph and thus this representation will be used to re-use the layer graphs and the caller-callee graph of the given NSGA-II-framework and to visualize pipes throughout this thesis. A layer graph is a coupling graph, where the set of nodes is equal to the units of one specific layer, and calculated layer graphs are stored in the `layerGraphMap` of the `MultiLevelModularization`-class, see Figure 4.1. Furthermore, the set of layer graphs is used to calculate the metrics for each layer [Kau25].

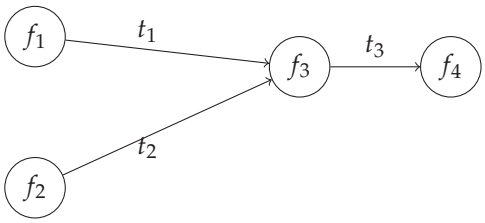

Figure 2.5. Own illustration of a *pipe-and-filter*-architecture where f_1 and f_2 connect to f_3, which forwards data to f_4 as shown as sub-graph in [Dob03, Fig. 1]; edge and node labels adapted in this work

2.3 Kieker and TeeTime

The current version of the *Kieker* framework is "[...] the Kieker Observability Framework Version 2, the successor of the Kieker Monitoring Framework." [Yan+25, p. 11] and *Kieker* uses a *pipe-and-filter*-architecture for analyzing the monitoring records provided by the configured monitoring probes [HH20; HWH12]. This *pipe-and-filter*-architecture is provided by the *TeeTime* Pipe-and-Filter Framework [WHO17; Yan+25]. The provided records are not only analyzed, but also visualized during the analysis part. One of the visualization approaches uses *ExplorViz*[1] [Wal14]. The high-level architecture of *Kieker* is shown in Figure 2.6. It shows that *Kieker* consists of two main parts: The monitoring and the analyzes part [Kie]. Thus, optimizing *Kieker*'s analyzes part provides an improvement of one major part of *Kieker*, leading to an improvement of *Kieker* overall.

The *TeeTime* Pipe-and-Filter Framework as presented by
Wulf, Hasselbring, and Ohlemacher in [WHO17] models, executes and allocates threads to filters, called stages in *TeeTime*, for any arbitrary *pipe-and-filter*-architecture [Ado16; Wul19; WHO17; Wul+19].

According to Wulf, Ehmke, and Hasselbring *TeeTime* is based on the Tee-and-Join-Pipeline design pattern by Buschmann et. al. 1996. Thus, it supports multiple input- and output-ports for each filter, or stage. Allowing multiple incoming and outgoing pipes for each filter [WEH14]. We are concluding from this that multiple pipes between two filters are also allowed.

A port connects a filter and a pipe using a queue: A filter expects data coming in through its input-port through a pipe and sends the processed data back through its output-port through another pipe. The ports themselves are typed to ensure that only ports of the same type are connected, making the resulting pipe type-safe, and which ports of which filters are connected is configured in *TeeTime*'s configuration [WEH14].

Filters can also consist of multiple other filters and this type of filters is called a *composite stage* in *TeeTime* [WHO17] or a *composite filter* after Shaw 1996 [Wul19]. Consequently, it should be possible for a *composite stage* to consist of multiple other *composite stages* making a multi-level data type relevant. However, Wulf, Hasselbring, and Ohlemacher mention in [WHO17] that a *composite stage* is "[...] flattened to the P&F configuration represented by its child stages." [WHO17, p. 292], which leads us to the assumption that a *composite stage* theoretically consisting of multiple other stages of any type is practically just one large configuration containing all child stages. This assumption leads to at most one layer containing which filter belongs to which tool.

There are also existing multiple other kinds of filters beside the *composite stage*. Two of them are the *distributor* and the *merger* Both are the exact opposites of each other: The *distributor* takes input through one input-port and distributes it over arbitrary many output-ports. Meanwhile, the *merger* takes input over multiple input-ports and merges at least some of the incoming data and sends it over one output-port [WEH14]. Concluding from

[1] https://explorviz.dev/

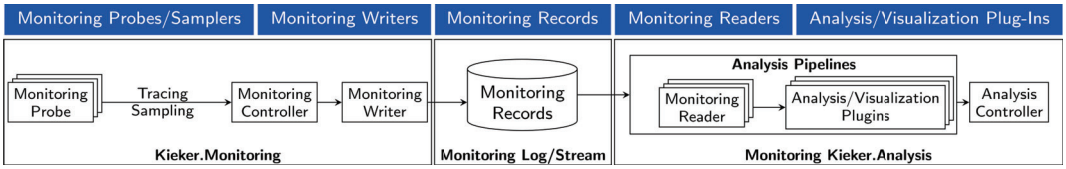

Figure 2.6. High-level Architecture of Kieker taken of [Kie]

this that one port having multiple adjacent ports is not allowed. Otherwise, there would not be any need for either the *distributor* nor the *merger*.

2.4 Related Works and the Significance of this Thesis

Arcelli et al. present EASIER to "[...] optimiz[e] architecture refactoring based on performance and on the intensity of change [...]" [Arc+18, p.1] with also considering performance antipatterns, but without a focus on evaluating a specific architecture.

Sellami et al. focus in [Sel+22] on the re-modularization of legacy software architecture into microservices with adding microservice granularity as metric to the set of cohesion and coupling instead of *pipe-and-filter*-specific metrics.

Adolf lists and implements in [Ado16] two metrics to monitor the stages of a *pipe-and-filter*-architecture, but does not use them in an evolutionary search to produce optimization suggestions.

Antunes and Barros compare in [AB24] the results of both a fitness function considering hierarchy and one which does not. Their focus lies on the Software Module Clustering (SMC) problem and does not explore the relation between SMC and the used software architecture.

Briand, Daly, and Wüst define in [BDW98] a unified framework for cohesion measurement. They compare different approaches of cohesion measurement and focus on cohesion on the class level, i.e., which method invokes which other method and needs which attributes. Their presented approaches are more detailed than the approach used during this thesis, because we only count method invocations as relations between classes, while some of the presented approaches in [BDW98] are also taking inheritance relations into account.

Also, Sarkar, Kak, and Rama focus in [SKR08] on metrics for large object-oriented software modularization with considering inheritance, polymorphism and being API-based. Their work is centred around coupling between modules and the different types of edges. They further discuss the cohesion of service APIs and do not specifically examine whether their proposed metrics are creating an advantage for the *pipe-and-filter*-architecture.

Abreu e Brito and Carapuça propose in [AC94] metrics in consideration of, among other criteria, inheritance and polymorphism, too. During this thesis we will further use their definition of the Coupling Factor (COF), which neither counts edges made by inheritance

nor polymorphism.

Harman and Clark present in [HC04] the approach 'Metrics as Fitness Functions' (MAFF), which enables the use of metrics as fitness functions in evolutionary search under specific conditions. We will use MAFF to explain why a metric or a set of metrics is suitable as fitness function.

Candela et al. ask in [Can+16] the question, whether cohesion and coupling are sufficient as fitness functions for software re-modularization. On the contrary to this thesis they do not discuss the *pipe-and-filter*-architecture and the effect of the used fitness functions on that type of architecture.

Schnoor, Hasselbring, and Jung apply in [SHJ25b] an older version of our NSGA-II-implementation [SHJ25a] on Earth System Models. Focusing on the impact of different reverse engineering techniques instead of the impact of different fitness functions. They use structural coupling and lack of structural cohesion as metrics.

Schnoor and Hasselbring examine in [SH20] the effects of weighted dynamic coupling with the result that weighted dynamic coupling complements both static and unweighted dynamic coupling. Their weighted coupling metrics are computed on a weighted coupling graph, while for a dynamic coupling metric the weights are measured with dynamic analysis. Therefore, they use 18 different measurement approaches composed of three orthogonal choices for coupling to compare the results. However, they neither used their proposed metrics as fitness functions for evolutionary search, nor have they discussed the impact on specific software-architecture, e.g., *pipe-and-filter*.

Zhao et al. present in [Zha+21] vPIPE, a software to accelerate the training of a DNN, which is trained using pipeline parallelism, by dynamically performing both layer partitioning and memory management. This paper is one example for the significance of *pipe-and-filter* in the machine learning domain.

Ghantasala et al. use in [Gha+26] NSGA-II as offline Pareto optimization to produce candidates for the objectives latency and energy-cost. These candidates are the base for their online genetic-algorithm offloader. They use a probabilistic approach to approximate throughput for their performance modelling. Whereas, these performance models are used as initial population for NSGA-II. Following their example and using a probabilistic approach instead of the naive approach discussed in Figure 4.3 could provide an impact on our results and should be further discussed and examined in future works.

Chapter 3

Metrics Discussion

3.1 Metrics Introduction

Mkaouer et al. mentions that "[...] cohesion and coupling are the main metrics used to improve the quality of existing packages (e.g., modules) [...]" [Mka+15, p.2] for software re-modularization. However, both cohesion and coupling are not architecture-specific and refer to modularity of an Object-Oriented software [Bav+10]. This leads to the question, whether a set of metrics exist which provides better re-modularization suggestions for a software with a specific architecture than cohesion and coupling. Considering *Kieker*, a set of metrics for the *pipe-and-filter*-architecture is of special interest and leads to the second research question SQ2.

To optimize the *pipe-and-filter*-architecture this thesis concentrates on optimizing the throughput and the modularity, because the reason why the *pipe-and-filter*-architecture is attractive for developers is its potential for high modularity and high throughput [Wul19; WHO17]. Based on the definition of Pareto-optimal solutions, a large set of Pareto-optimal solutions is desirable [Deb+02]. To ensure a large variety of Pareto-optimal solutions the set of metrics shall reward each mutation and recombination operation to a certain degree. Otherwise, if for example all members of a proposed set only reward the merging of units, then the optimal solution would be known as one large unit, resulting in one known optimal solution.

During this thesis sets of the following metrics are evaluated and compared regarding their fitness to improve *Kieker*. To do so, the set containing cohesion and coupling will be compared to different sets containing the *pipe-and-filter*-specific metrics. The specifics regarding the different sets of metrics will be further discussed in Section 3.2.

- Cohesion and Coupling
- Fan-in and Fan-out Coupling
- PullThroughputMetric
- #PaFs

More probably suitable metrics exist, e.g., the PushPullDifferenceMetric by [Ado16], but this thesis concentrates on one subset of metrics to be able to provide results for each metric. Otherwise, the limited time would be too short.

According to Harman and Clark in [HC04] choosing a metric as a single fitness function will be useful, if the following four properties: *Large search space, low computational complexity, approximate continuity* and *absence of known optimal solutions* are met. However, neither the maximization of cohesion nor the minimization of coupling are continuous. Although Harman and Clark mention in the same paper [HC04] that both cohesion and coupling are being used as fitness functions in evolutionary search, and they also state that a fitness function does not have to be continuous just that the search with a continuous fitness function is less error-prone. Because of that, and for simplicity the property *approximate continuity* will be further ignored during this thesis. Schnoor, Hasselbring, and Jung discover in [SHJ25b] that a restricted search space can also provide good or even better optimization suggestions, if the restriction correlates with the preferences of domain experts. Thus, the criterion *large search space* is subjective. The important point is that the search space can not be exhausted [HC04].

Furthermore, a multi-objective optimization problem with multiple fitness functions can be converted to a single-objective optimization problem with only one fitness function [BBG11] and both Harman and Clark discuss the use of only a single fitness function. Therefore and because the NSGA-II optimizes multiple fitness functions instead of one to obtain a set of Pareto-optimal solutions [Deb+02] it is sufficient when the entirety of the set of metrics satisfies all four properties without the need of every single metric needing to fulfil all four properties.

3.1.1 Coupling and Cohesion

Coupling is defined as the number of connections between modules with a connection being "[...] a reference to some label or address defined (or also defined) elsewhere" [SMC74, p. 232], which can be, e.g., method-calls, inheritance- or implementation relations between classes. Stevens, Myers, and Constantine further emphasize in [SMC74] on the importance of the weight of each connection between two modules. However, for simplicity we further ignore edge-weights and parallel-edges. Hence, a pre-condition is that the coupling graph does not allow parallel-edges. Otherwise, it is important to consider the existence of parallel-edges in the normalization, which is only possible, if an upper bound of parallel-edges per node is defined. Because without an upper bound there can be arbitrary many parallel-edges between two nodes.

To ensure that two systems with different sizes are comparable we further calculate coupling using the Coupling Factor (COF).The COF is defined as the coupling of the classes of a software [EHB20] normalized over the maximal number of connections between classes in a directed graph without self-loops [AC94]. Abreu e Brito and Carapuça define the connections between classes over directed client-supplier relations. Additionally, they define their number of edges explicitly over two distinct client-classes in their *is_client*(C_c, C_s) function. Thus, their used maximal number of connections between classes has also to be directed and without self-loops. Briand, Daly, and Wust mention in [BDW02] that the normalization of COF between 0 and 1 in general considers the size of a software system,

when comparing multiple COF values. Meaning that without the normalization software systems with many classes could have got the same coupling value as software systems with few classes, even though the coupling is, regard to the number of classes, better. By normalizing with the maximal number of connections between classes this will be taken into account. For example, comparing Figure 3.1 and Figure 3.5: Both figures have got the same number of edges between parent nodes. However, in Figure 3.5 has one class more than Figure 3.1, which could be connected to every other class. Considering that, the coupling of Figure 3.5 could be much worse. Thus, intuitively the coupling value of Figure 3.5 should be better than the coupling value of Figure 3.1, but without the normalization both figures would have got the same coupling value.

A package consists of classes, interfaces or sub-packages and the relations between the elements inside this package. Additionally, a relation between two packages A and B is defined s.t., it exists, if and only if there is at least one relation between $a \in A$ and $b \in B$. Furthermore, the coupling between two elements of a package is defined over the relations between both [GC12]. Gupta and Chhabra define these package-package relations specifically for sub-packages inside the same package. However, if we always add an artificial root-node, aka parent package, over the whole forest of package-hierarchies, then the definition of a package-package relation can be applied to all packages on every layer with each package being at least a sub-package of the artificial root-node. This makes it possible to define the coupling over all package-layers. Because of Gupta and Chhabras' definition of package-package relations the maximal number of package-package relations equals the maximal number of edges between classes and interfaces.

Let $G_p := (V_p, E_p)$ be the graph containing all edges and nodes on one layer and let $G_c := (V_c, E_c)$ be the graph of the child layer of G_p s.t., there exists a set of edges E between the nodes in V_p and V_c. Let further $P_i \in V_p$ be a parent node or package of the software. Then Equation 3.1 describes the coupling value of a multi-layered software based on the equation of the COF shown in [AC94]. In the definition of the COF in [AC94] we can not find a definition for the special case when there are less than two child nodes in the system. However, Briand, Daly, and Wüst in [BDW98] uses a re-defined version described in [AGE95] and states that the system consists of $c_1, c_2, \ldots, c_{TC}$ with TC being the total number of all classes which could be interpreted that a system needs to consist of at least two classes. Hence, the case with less than two classes is undefined even though this case can happen, either intended by the user as input or as a random product of the performed (randomized) operations of the NSGA-II, which are presented in Section 2.1. Thus, we have decided to use the neutral element for the division as lower bound for the normalization in the denominator. Correctly resulting in zero coupling, because one child node can only be connected to itself.

$$Coupling(G_p, G_c) = \frac{\sum_{P_i \in V_p} \sum_{(v_k \in P_i \wedge v_j \in P_j \wedge k \neq j \wedge (v_k, v_j) \in E \vee (v_j, v_k) \in E)} 1}{\max\{1, |V_c|^2 - |V_c|\}} \tag{3.1}$$

A package can only have as many edges, as its corresponding classes, because packages can only call each other by using classes. Hence, for the maximal value of the nominator is

between 0 and $|V_c|^2 - |V_c|$ and thus $Coupling(G_p, G_c) \in [0, 1]$.

High coupling leads to a high complexity of a software [EHB20; SMC74]. Hence, a low coupling is desirable [Bav+10; SMC74] and optimising coupling is a minimization problem. Coupling rewards the merging of units, because fewer units to reference to or to be referenced by result in fewer coupling. For example, Figure 3.1 has got a higher coupling value than Figure 3.6, because with only one existing package, there is no possibility for coupling.

Cohesion is defined as the number of connections of methods and attributes inside a class [DK18; LYC07; Sil+23]. Gupta and Chhabra define in [GC12] the cohesion of package A as the connections between elements of A based on [PCB92]. However, the packages mentioned in [PCB92] are Ada packages, which resemble more a class in modern object-oriented programming languages like Java [Ada]. Nevertheless, Stevens, Myers, and Constantine call cohesion in [SMC74] cohesiveness and define it as the relation of every element inside a module, emphasizing on "any form of a 'piece' of the module [...]" [SMC74, p. 237]. Hence, cohesion is defined as the number of edges between units inside another unit, i.e., the number of connections of methods and attributes inside a class but also the number of connections of classes or sub-packages inside a package in a broader sense. To make the cohesion value of a package independent of the size of a package, it has to be normalized over the maximal number of connections inside that package [GC12]. Hence, to calculate the cohesion value of each package we will use the $PCoh(p^i)$ function presented in [GC12], which is shown in Equation 3.2 and adopted s.t., it is consistent with Equation 3.1 with $V_{ci} \subseteq V_c$ being the subset of classes in the package P_i and $|V_{ci}|^2 - |V_{ci}|$ being the maximal possible number of edges in P_i in a directed graph without self-loops.

$$PCoh(P_i, G_c) = \begin{cases} 0 & \text{if } |P_i| = 0 \\ \frac{\sum_{(v_k, v_j \in P_i \wedge k \neq j \wedge ((v_k, v_j) \in E_c \vee (v_j, v_k) \in E_c))} 1}{|V_{ci}|^2 - |V_{ci}|} & \text{if } |P_i| > 1 \\ 1 & \text{if } |P_i| = 1. \end{cases} \tag{3.2}$$

For each arbitrary package it holds that $PCoh(P_i, G_c) \in [0, 1]$ [GC12]. Afterward, to acquire the cohesion for the whole layer we sum up all cohesion values for each package on one and further normalize the sum of one layer over the number of packages on that layer. This results in the average cohesion value of a package making the result of each layer better comparable regarding the size of a layer. Equation 3.3 describes the value of the cohesion of one layer using Equation 3.2 by Gupta and Chhabra. If there is no package, the denominator needs to be the neutral element of division to be defined and resulting in $Cohesion(G_p, G_c) = 0$. Thus and with $PCoh(P_i) \in [0, 1]$ it also holds that $Cohesion(G_p, G_c) \in [0, 1]$.

$$Cohesion(G_p, G_c) = \frac{\sum_{P_i \in V_p} PCoh(P_i, G_c)}{\max\{1, |V_p|\}} \tag{3.3}$$

Maximizing the cohesion of packages can reduce the coupling of the software [SMC74]. In addition, Bavota et al. states in [Bav+10] that a high cohesion is optimal. Thus and with

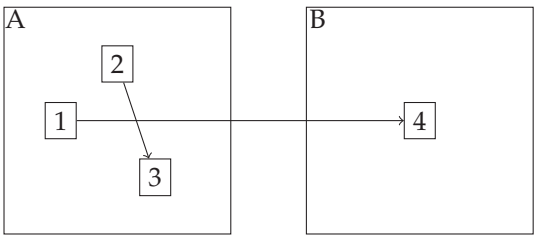

Figure 3.1. Before either operation

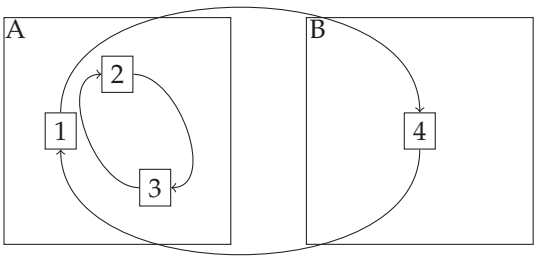

Figure 3.2. Before either operation with bidirectional edges

the optimization of coupling being a minimization problem, the optimization of cohesion results in a maximization problem.

Cohesion rewards the splitting operation, because there are two possible cases: Either (a) a connection between two child nodes of a package already exist, or (b) no such connection exist. In case (a) splitting the package would decrease the cohesion by one and in case (b) the cohesion in the package would increase, if the unconnected classes were moved to different packages, preferably to packages where those classes have got connections to or are the only class. This is also shown in the following example: Figure 3.3 shows two units A and B with an overall cohesion value of $\frac{2}{3}$. After splitting unit A into unit A and C in Figure 3.7 the cohesion value also increases to 1.

Furthermore, Cohesion also rewards merging operation, because if all classes of package A only have got connections to classes in package V, then the cohesion will increase by merging both packages, e.g., Figure 3.4 has a cohesion value of 0, which increases to $\frac{1}{6}$ by merging package A and B as shown in Figure 3.6.

As already mentioned both cohesion and coupling are used as fitness functions [HC04]. Because of that, their fitness regarding the remaining three properties does not need to be further discussed.

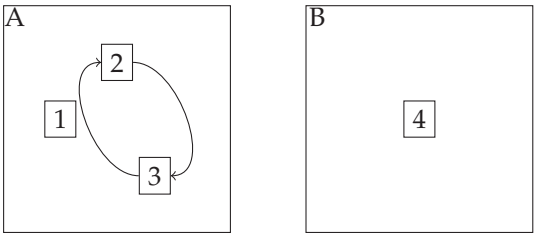

Figure 3.3. Before either operation with two isolated classes

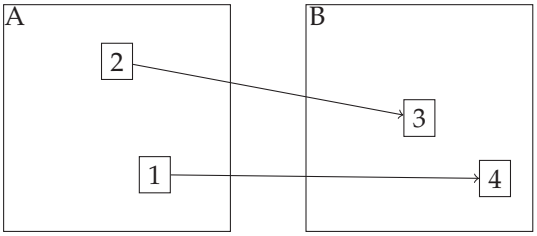

Figure 3.4. Before either operation worst case

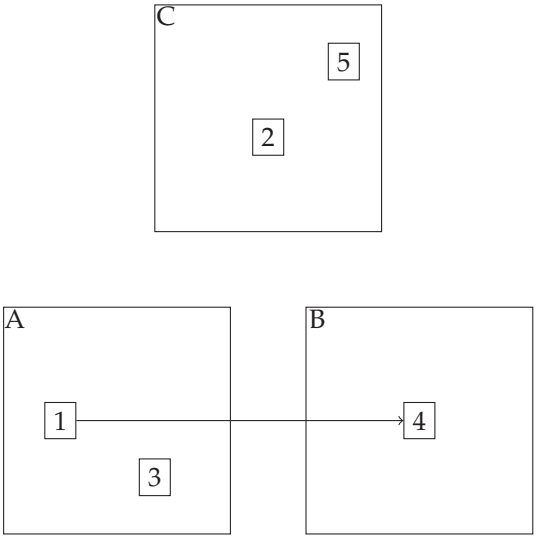

Figure 3.5. One edge between packages and more packages in general

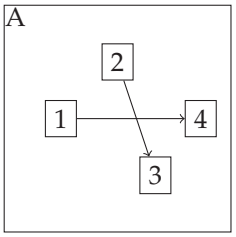

Figure 3.6. After merging unit A and B

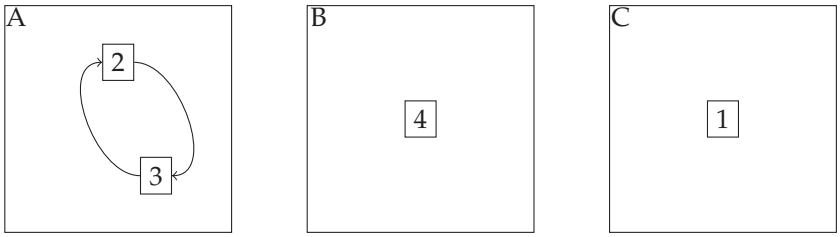

Figure 3.7. Small packages with maximal cohesion and coupling

3.1.2 Filter-based Fan-in and Fan-out Coupling

Fan-in coupling is defined as the number of incoming edges for all nodes combined and Fan-out coupling is defined as the number of outgoing edges for all nodes combined [LYC07; Mur+12]. The authors of the mentioned sources define fan-in and fan-out according to references made by classes, which will be equivalent to nodes and edges, if the set of classes is a node and the set of references are directed edges. Because coupling sums up both the in- and the out-degree of a node, as discussed in Section 3.1.1, either optimizing fan-in or fan-out coupling is a subproblem of the coupling optimization problem.

Lee, Yang, and Chang states in [LYC07] that a high fan-in and a low fan-out can imply a high re-usability, if viewed for classes inside a package. Since filters are communicating with each other through pipes [WEH14] does a filter with a high fan-in receive data through

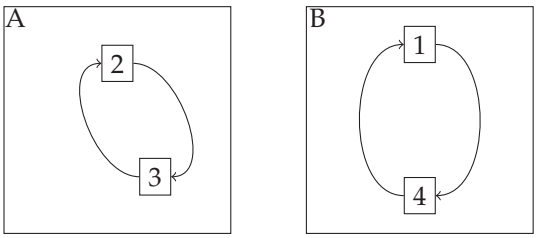

Figure 3.8. Large packages with maximal cohesion and coupling

many incoming pipes from multiple other filters, which is equivalent to a class which is re-used by multiple other classes. Additionally, a filter with a high fan-out sends data to many other filters through outgoing pipes, making it dependable on a high number of filters, while a filter with a low fan-out is less dependable on other filters making a filter with a low fan-out better re-usable [LYC07]. "Another main advantage of the P&F style is its potential for a high modularity." [Wul19, p. 61] Wulf also implies that Because of that, modularity there exists a potential for a high level of re-usability of the filters. Furthermore, the exact wording implies that a high modularity and consequentially a high re-usability are desirable. Hence, we have chosen these two metrics to optimize the re-usability of filters. Optimizing fan-in coupling is a maximization problem, which rewards the splitting of filters to increase the number of incoming edges in total. Whereas, optimizing fan-out coupling is a minimization problem, which rewards the merging of filters to decrease the number of outgoing edges in total.

With fan-in and fan-out being defined over filters and *TeeTime* connecting each pipe with a filter through a port [WEH14], the number of maximal possible edges between all filters is the number of the maximal number of connections between all ports. Let $G := (V, E)$ the directed graph containing all ports and filters as nodes and both edges between ports or between filters. Under the condition of the filters being on the parent layer of the ports, $V_c \subseteq V$ is the set of all ports. Hence, the normalization of both functions Equation 3.4 and Equation 3.5, which are based on the coupling function Equation 3.1, is equal to the normalization of the coupling factor. Furthermore, in a directed graph an incoming edge of one incident node is always an outgoing edge of another incident node. Consequently, the maximal number of either in- or outgoing edges is equal to the maximal number of possible edges and thus $FanIn(G), FanOut(G) \in [0, 1]$ holds.

$$FanIn(G) = \frac{\sum_{x_i \in V \wedge (x_j, x_i) \in E} 1}{\max\{1, |V_c|^2 - |V_c|\}} \tag{3.4}$$

$$FanOut(G) = \frac{\sum_{x_i \in V \wedge (x_i, x_j) \in E} 1}{\max\{1, |V_c|^2 - |V_c|\}} \tag{3.5}$$

Since coupling is often used as fitness function for evolutionary search-based algorithms as the NSGA-II [Mka+15], both fan-in and fan-out coupling satisfy the constraints to be used as a fitness function. However, because the optimal solution for fan-in coupling is one unit whereas many units as possible reference to, to ensure that there is no known set of optimal solutions, fan-in coupling needs to be paired with a metric which prioritizes the merging of pipes, e.g., coupling. Whereas, because the optimal solution of fan-out is a unit which references to as many units as possible, to ensure that there is no known set of optimal solutions, fan-out coupling needs to be paired with a metric which prioritizes the splitting of pipes, e.g., fan-in coupling.

3.1.3 Pipe-based PullThroughputMetric

Wulf, Hasselbring, and Ohlemacher in [WHO17] state that the *pipe-and-filter*-architecture is often used in industry and research because, among other things, it provides a high throughput. Adolf also mentions in [Ado16] that many algorithms to dynamically adapt *pipe-and-filter*-architectures are for instance aiming to increase the throughput. By implication a high throughput is desirable.

Adolf defines the PullThroughputMetric as the incoming throughput into one filter and "[i]f there is more than one pipe, the mean of all values is computed." [Ado16, p. 54] Furthermore, the PullThroughputMetric is a measurement for how fast a filter can process data [Ado16]. Let $G := (V, E)$ be the directed graph s.t., ports are the nodes and pipes are the edges. Equation 3.6 shows the function $f : V \mapsto N$ based on that definition.

Since data is sent from one filter to another through pipes [Ado16; Dob03] and an incoming pipe is connected to a filter through an input-port [WEH14], to measure the incoming throughput, the throughput received by the input-ports needs to be measured.

$$f(x_i) = \begin{cases} \frac{\sum_{t_i \in \{(x_i,x_j) \in E | x_j \in V\}} t_i}{|\{(x_i,x_j) \in E | x_j \in V\}|}, & \text{if } |\{(x_i, x_j) \in E | x_j \in V\}| \geqslant 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

$$\text{PullThroughputMetric}(G) = \frac{\sum_{x_i \in V \wedge (x_j,x_i) \in E} f(x_i)}{\max_{throughput}} \tag{3.7}$$

Therefore, we have chosen the PullThroughputMetric as a metric for the objective to optimize the throughput. As a high throughput is desirable, in order to optimize the PullThroughputMetric value of each filter the sum of all values shall be maximized. Furthermore, to make software systems with different capacities of throughput comparable the sum of all throughput values will be normalized over the maximal throughput of the whole system. This capacity can either be defined by the user, either for the whole system or per pipe, or if it is not defined it will be approximated by taking the sum of the throughput of all pipes. Resulting in a maximization problem. Because of each pipe connecting two ports [WEH14] for the sum of the throughput of all incoming pipes $\sum_{x_i \in V \wedge (x_j,x_i) \in E} f(x_i) \leqslant \max_{throughput}$ holds. Which leads in all cases to $\text{PullThroughputMetric}(G) \in [0, 1]$.

Optimizing the sum of all values instead of every single value yields the advantage of only the need to compare one value per candidate instead of comparing as many values as pipes. Especially if one candidate has got fewer pipes than another. In that case it would not be possible to consider all values and thus there is a need for an aggregation like this. The computational complexity is $\mathcal{O}(|V||E|)$ with the need of iterating over all edges for each node. Assuming each edge is unique, the computational complexity is still linear.

Maximizing the PullThroughputMetric rewards the merging of pipes, because the mean value of the throughput of multiple pipes is less than the summed up throughput, see Figure 4.3 for more details on throughput approximation. Only considering the PullThroughputMetric on its own results in one optimal solution where all pipes are merged, which are allowed to be merged by the software constraints. However, if this met-

ric is combined with a metric which is prioritizing splitting of pipes, e.g., fan-in coupling, then there is no known set of optimal solutions anymore, because a metric rewarding the splitting of pipes contradicts the PullThroughputMetric. This combination also leads to a large search space.

3.1.4 #PaFs

One of the objectives of EASIER presented by Arcelli et al. in [Arc+18] is the #*PAs*-metric, which counts the number of performance antipatterns and considers the *pipe-and-filter* performance antipattern. This metric can be used to especially recognize the occurrence of the *pipe-and-filter* performance antipattern (PaF) in a software with the *pipe-and-filter*-architecture [Arc+18]. Thus, we have decided to use an altered version of this metric to optimize the performance by decreasing the occurrences of performance antipatterns.

Arcelli et al. count a PaF only when the slowest filter is beneath a set threshold. To count the number of all PaFs (#PaFs) of the whole system we introduce a quantitative metric #PaFs based on Arcelli et al.s' #*PAs*-metric. According to this #PaFs is further defined as the number of outgoing edges in the coupling graph where the weight is beneath a set threshold. If a filter has got more than one outgoing pipes the throughput will be summed up to indicate slow filters.

With TH being the set threshold Equation 3.9 shows the calculation of #PaFs normalized over the current number of pipes between filters in the software to achieve the mean value corresponding to the size of the software system. In case there are no pipes at all, the number of pipes under a given threshold is 0, but for this case to be properly defined, we have decided to use the neutral element of a division for the denominator comparable to the other presented metrics.

If we iterate over ports instead of pipes, with two ports connecting one pipe in *TeeTime* [WEH14], then we will either have to divide the throughput by two, multiply the threshold by two or normalize over the current number of ports instead of pipes.

Let $G := (V, E)$ be a directed graph s.t., the filters or ports are the nodes and the pipes are the edges. Then Equation 3.9 shows the value for G. In the worst case all pipes are beneath TH and hence $\#\text{PaFs}(G) \in [0, 1]$ holds in all cases.

$$f(x_i) = \begin{cases} \sum_{t_i \in \{(x_i, x_j) \in E | x_j \in V\}} t_i, & \text{if } |\{(x_i, x_j) \in E | x_j \in V\}| > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

$$\#\text{PaFs}(G) = \frac{\sum_{v_j \in V \wedge (v_j, v_i) \in E \wedge f(v_j) < TH} 1}{\max\{1, |E|\}} \tag{3.9}$$

The set of counted edges is a subset of the set of counted edges for coupling. Thus, the computational complexity for each edge is as low as the computational complexity of coupling.

Optimizing the objective #*PAs* metric is a minimization problem [Arc+18]. Thus, the subproblem to optimize #PaFs is a minimization problem, too. This rewards the merging

of pipes to increase the throughput of the resulting pipes in order to reduce the number of pipes with a low throughput. In order to merge pipes some filters need to be merged. If a pipe with a throughput beneath the defined threshold is removed during a mutation operation, the number of PaFs will also be reduced. Because of that, either removing a pipe as well as merging two pipes is desirable to optimize #PaFs.

There are multiple different optimal solutions, e.g., both two nodes connected with one edge, where the throughput is higher than the threshold and two isolated nodes are optimal regarding #PaFs. Resulting in both an absence of a known set of optimal solutions and a large search space.

3.2 Suggested Sets of Metrics

After introducing the metrics and discussing why the chosen ones are of interest, we can now discuss possible sets of metrics and whether they fulfil the following three out of four properties by [HC04]: *Large search space, low computational complexity* and *absence of known optimal solutions*. As discussed in Section 3.1 we do not consider the fourth property *approximate continuity*. At least one metric of a set needs to satisfy the property *Large search space* for the whole set to fulfil this property, because then at least this objective can provide a large set of solutions. This extends to the *absence of known optimal solutions*, because if at least one member M_1 has got no known optimal solutions, then there is also no known optimal solution for optimizing the many-objective optimization problem containing M_1.

To satisfy the *low computational complexity* property all members of a set need to fulfil that property on their own. If at least one member has got a high computational complexity, then the whole set results in having a high computational complexity.

In case of a set containing only one metric, a single objective problem with one optimal solution is sufficient. Hence, for a multi-objective problem to be interesting, the sets should contain at least two metrics which contradict each other to provide a set of solutions as search space instead of just one optimal solution [NZE05]. Additionally, to stay in the in Section 2.1 discussed constraints of NSGA-II and to reduce the number of objectives to consider, all sets consist of at most three metrics.

To have contradictory metrics each set contains at least one metric rewarding splitting and one metric rewarding the merging of units. As discussed in Section 3.1 this also ensures a large set of Pareto-optimal solutions.

With discussing both the fitness of each metric as fitness function and their tendencies against either merging or splitting operations in Section 3.1, the following sets of metrics result:

- (Cohesion, Coupling)
- (Fan-in Coupling, Fan-out Coupling)
- (Fan-in Coupling, PullThroughputMetric, #PaFs)

▷ (Fan-in Coupling, PullThroughputMetric)

▷ (Fan-in Coupling, #PaFs)

However, during our experiments we have encountered unexpected behavior of at least one set of metrics, and therefore we have added the following control set of metrics for further investigation. This set does not need to provide the discussed characteristics, i.e., (PullThroughputMetric, #PaFs) only contains metrics rewarding the merge-mutations, see Chapter 5 for further information. This way we can also verify our assumptions regarding the discussed characteristics.

▷ (PullThroughputMetric, #PaFs)

3.3 Inverting maximization Problems

Because the Pareto-domination is defined for a minimization problem (or a maximization problem, if inverted) as well as that a multi-objective optimization problem is defined over the optimization of one matrix [NZE05], to simplify the domination comparison of the NSGA-II all members in each of the in Section 3.2 introduced sets of metrics will be inverted into either a minimization or a maximization problem s.t., all members of a set belong to the same type of optimization problem. Otherwise, it would be necessary to flag each objective with the according optimization problem type or to provide two matrices s.t., one contains all objectives to be minimized and the other contains all objectives to be maximized, to use the corresponding domination function.

We have decided to invert all metrics defined for maximization problems to minimization problems. By inverting a metric it keeps the characteristics discussed in Chapter 3.

3.3.1 Lack of Cohesion

The inverted problem of maximizing cohesion (Equation 3.3) is minimizing the lack of cohesion, which also rewards both merging and splitting of packages. Let $G_p := (V_p, E_p)$ and $G_c := (V_c, E_c)$ be defined as in Section 3.1.1. With $Cohesion(G_p, G_c) \in [0,1]$ and the optimization of cohesion being a maximization problem, as explained in Section 3.1.1, 1 is the optimal value. Hence, we subtract the actual cohesion value from the optimum to receive the difference. Equation 3.10 shows the calculation of lack of cohesion. Furthermore, leads $Cohesion(G_p, G_c) \in [0,1]$ to $LackOfCohesion(G_p, G_c) \in [0,1]$.

$$LackOfCohesion(G_p, G_c) = 1 - Cohesion(G_p, G_c) \tag{3.10}$$

The splitting of a package can result in unconnected sets of classes being moved to different packages, which results in a higher rate of connections in each of the resulting packages than before. For example, Figure 3.3 has got a lack of cohesion value of $\frac{5}{12}$, while Figure 3.7 has got an decreased lack of cohesion value of 0 with no missing cohesion.

Whereas the merging of packages can put multiple unconnected classes into one package. Resulting in a decreased rate of connections per package. Also, if multiple strongly connected classes are moved to one package together the mean lack of cohesion value per package can improve by decreasing. For example, merging class 1 from package A in Figure 3.2 into package B also improves the lack of cohesion from $\frac{1}{3}$ to 0, as shown in Figure 3.8.

3.3.2 Lack of Fan-in Coupling

The inverted problem of maximizing the fan-in coupling (Equation 3.4) is minimizing the lack of fan-in coupling. Since, fan-in coupling is defined over the number of incoming edges per component [LYC07; Mur+12] and normalized over the number of maximal possible edges, as discussed in Section 3.1.2, to invert it we can subtract the actual value from the number of maximal possible edges normalized over the maximal number of edges, too. This equals subtracting the fan-in value from 1 as shown in Equation 3.11. Let $G := (V, E)$ the directed graph containing all ports and filters as nodes and both edges between ports or between filters. With $FanIn(G) \in [0, 1]$, as explained in Section 3.1.2, it follows $LackOfFanIn(G) \in [0, 1]$.

$$LackOfFanIn(G) = 1 - FanIn(G) \tag{3.11}$$

Minimizing the lack of fan-in coupling rewards splitting ports equally to maximizing the fan-in coupling does, because by splitting existing pipes the fan-in value increases which minimizes the distance to the optimal value.

3.3.3 Lack of PullThroughputMetric

The inverted problem of maximizing the PullThroughputMetric (Equation 3.7) introduced by [Ado16] is minimizing the lack of PullThroughputMetric.

Let $G := (V, E)$ the directed graph containing all ports and filters as nodes and both edges between ports or between filters. As discussed in Section 3.1.3 the PullThroughputMetric is normalized s.t., its values are in $[0, 1]$. With the optimal value of the original maximization problem being equal to the upper bound 1, to invert the metric, we subtract the existing PullThroughputMetric after normalization from the optimum and measure the distance between the actual value and the optimum, resulting in Equation 3.12 with $LackOfPullThroughputMetric(G) \in [0, 1]$.

$$LackOfPullThroughputMetric(G) = 1 - PullThroughputMetric(G) \tag{3.12}$$

The lack of PullThroughputMetric still rewards the merging of pipes, as the PullThroughputMetric is increased by the merge operation, further discussed in Section 3.1.3. Hence, the distance between the optimum and the actual value decreases and thus lack of PullThroughputMetric is minimized.

Chapter 4

Technical Necessities

4.1 Data type & Operations

There are the following two options to implement the *pipe-and-filter*-architecture in the provided `MultiLevelModularization` NSGA-II-framework:
(1) Either the `MultiLevelModularization`-class as presented in [Kau25] and shown in Figure 4.1 will be adjusted and re-used for the *pipe-and-filter*-architecture, or (2) a new data type will be designed and implemented.

One major advantage of re-using the `MultiLevelModularization`-class would be that both not-*pipe-and-filter*-specific metrics and *pipe-and-filter*-specific metrics could be in the same set of metrics. However, for approach (1) to work, adjustments are necessary s.t., the *pipe-and-filter*-specific information are stored. If in the long run different architectures shall be optimized, then the `MultiLevelModularization`-class can expand unnecessary. Otherwise, if a new data type is used for each architecture type, then each data type only contains the necessary information, which speaks for option (2).

On the other hand for the *pipe-and-filter*-architecture the adjustment of `MultiLevelModularization` is small, because only two additional pieces of information are necessary:

- Which classes belong to which filter (type)
- The throughput of each pipe

Both pieces of information can be mapped on the already existing `MultiLevelModularization`-class and `Node`-class from [Kau25], which is shown in Figure 4.2, like explained in the *Simple Approach* in Table 4.1. If a filter consists of exactly one class, which is often the case in the current *Kieker*-version, a better approach will be the *Enhanced Approach* in Table 4.1 with a focus on the ports. Therefore, we will implement and further discuss the *Enhanced Approach*.

While a `ProgramUnit` can be of type `ProgramUnitType.PORT` the corresponding `Port` can either be of the `PortType.INPUTPORT` or `PortType.OUTPUTPORT` type. We do not use the `ProgramUnitType`, mentioned in Figure 4.2, because the instances of `Port` have different types as other units of other `ProgramUnitTypes`.

All approaches are in need of an adjusted parser, which can correctly map the *pipe-and-filter*-architecture to either the `MultiLevelModularization`-class or a new class.

Table 4.1. Approaches to implement *pipe-and-filter*-architecture into `MultiLevelModularization`

Simple Approach	**Enhanced Approach**
Either throughput as edge-weight.	Either throughput as edge-weight.
Adding `ProgramUnitType.FILTER`	Adding `ProgramUnitType.FILTER`, `ProgramUnitType.PORT`, `PortType.INPUTPORT` and `PortType.OUTPUTPORT`
Classes, which belong to a filter are the keys of their layer in the `layerList` and their corresponding filter is the value	Ports, which belong to a filter are the keys of the child layer of their corresponding filters and filters are the corresponding values.
∅	Filters are the keys of their layer (with an offset of one) and, in the case of nested filters, their parent filters are the corresponding values. Otherwise, filters, which belong to a specific sub-program (e.g., *Kieker*-tool) have got the corresponding sub-program as value.

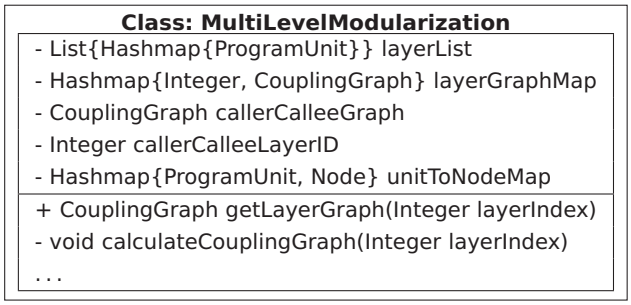

Figure 4.1. Re-drawn class-diagram of `MultiLevelModularization`-class from [Kau25]

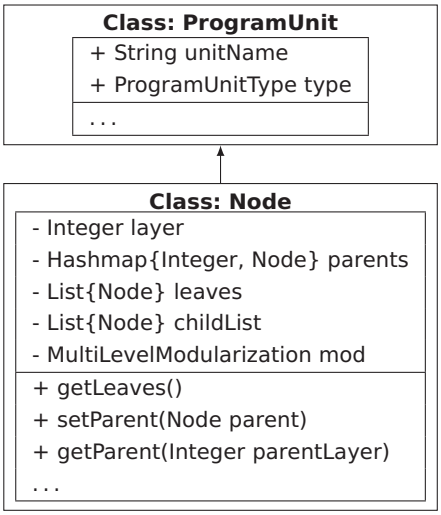

Figure 4.2. Re-drawn class-diagram of Node-class from [Kau25]

As mentioned in Section 2.1 we will perform merge- and splitting operations. Those can be performed on both filters and pipes. We will further discuss both types of operations in Section 4.1.1 and Section 4.1.2.

4.1.1 Operations on Filters

As seen in the source-code of *Kieker* [Wal+26] a filter inherits from a *TeeTime*-stage, like `AbstractConsumerStage, AbstractProducerStage, CompositeStage, AbstractStage` and `AbstractFilter`. We will consider these *TeeTime*-stages as filter types henceforth. An example excerpt of a class-diagram is shown in Figure 4.6. A filter will be of a specific filter type, if and only if it inherits directly or transitively that specific type. If a filter inherits multiple different filter types, it will belong to either of the filter types or to completely different one. Furthermore, if filter type A inherits another filter type B, A will be a subtype of B.

A merge-operation performed on filters can suggest for a filter to inherit another filter type, if both filters are of different filter types, e.g., we want to merge a filter F_c of type `AbstractConsumerStage` with a filter F_p of type `AbstractProducerStage`, as shown in Figure 4.3, then we have to decide whether the resulting filter will be of type `AbstractConsumerStage` or `AbstractProducerStage` or whether even of a completely different type. This decision can either be made randomly or following a heuristic. Another version of the merge-operation is that the new filter inherits both filter types. However, the developer then needs to decide how to realize the multiple inheritance. Therefore, customizing, whether multiple inheritance is allowed to be suggested or not, can be of interest for the developer.

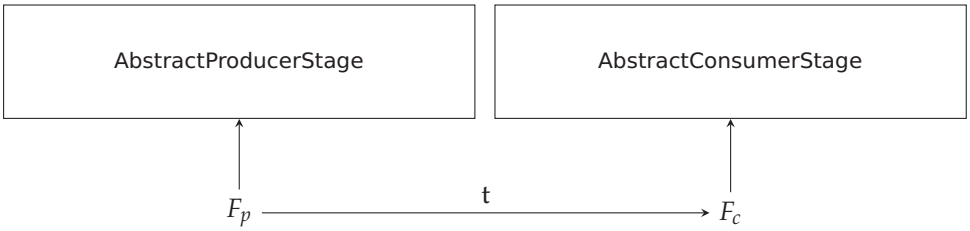

Figure 4.3. A pipe between an AbstractProducerStage and an AbstarctConsumerStage using the presented graph representation from Section 2.2

Meanwhile, a split-operation performed on a filter can suggest creating multiple subtypes. Hirzel et al. present in [Hir+14], among others, the *fusion*- and the *fission*-operation. The presented *fusion*-operation merges two filters, which reduces the number of pipes by removing all pipes between the merged filters. Whereas the presented *fission*-operation increases the number of pipes by splitting a filter to $n \in \mathbb{N}_{>1}$ filters. Both operations need to move methods to different classes, which is currently a prospect for the unforeseen future. However, Hirzel et al. also define *operator reordering*-operation. This operation aims for an improvement of the produced overhead by reducing the number of processed data. At the current state the given NSGA-II-framework has already implemented a `RandomMoveNode`-operation, which moves a node randomly to a different position inside the package-tree.[1] This `RandomMoveNode`-operation can be altered and used to implement the *operator reordering*-operation by [Hir+14] with randomly moving one filter upstream or downstream. To realize this move-operation the functionality of at least one filter has to change, because if we want to move filter1 up the stream, then filter1 needs to receive data from a different filter or send data to different filter than before. Because of the complexity the move-operation is currently not implemented.

4.1.2 Operations on Pipes

A merge-operation on pipes can suggest reducing the number of ports and use one port for more cases. Whereas, a split-operation for a pipe can suggest increasing the number of ports of the same data type. To suggest the merging and splitting of ports these ports must be known.

Performing the operations on pipes and accordingly on ports has got the advantage that there is no need to perform any operations on the method layer. However, it also has the disadvantage that there are not as many operations available as there are on the method layer and thus for altering the filters, because, i.e., moving a port to another filter would alter the functionality of at least two filters, e.g., one filter loses a port and therefore

[1] `https://cau-git.rz.uni-kiel.de/ifi-ag-se/oceandsl/genetic-restructuring/-/blob/main/src/main/java/cau/agse/hs/genopt/nestedcoupling/genetic/operators/mutation/MultiLevelRandomMoveNode.java?ref_type=heads`

loses at least one pipe. Thus, this filter can neither receive the data of the lost pipe nor send data through it. Our provided optimization suggestions shall not introduce such alterations in order to minimize the mental obstacles for a developer to actually implement one suggestion.

For operations on ports one invariant need to be fulfilled for all involved ports to ensure type-safety of a pipe and thus to ensure a type-safe environment, which is intended by *TeeTime* [WEH14]:

1. All involved ports transport the same data type

To store the data type of the processed data the `Node`-class shown in Figure 4.2 from [Kau25] can be altered in at least the following discussed ways. After Kaufhold in [Kau25] the `Node` inherits the `ProgramUnit` to reduce the re-factoring effort for the developers used to the implementation of the TwoLevelModularization by Haastert in [Haa23]. However, because Haastert never introduces the `ProgramUnit` it is possible that that class has already existed before as well as the mentioned `ProgramUnitType`-class.

1. Re-writing `Node` s.t., it is a generic class given the data type of the processed data
2. Storing the type of the processed data as an instance of `Object`
3. Implementing a generic child-class `Port`, which inherits from `Node`
4. Use composition s.t., `ProgramUnit` contains an instance of `Node` and an instance of `Port`, instead of `Node` inheriting from `ProgramUnit`

The first approach of re-writing the `Node`-class into a generic class has the disadvantages of both the need to re-write all already existing instances in the framework and every `Node`-instance to include a processed data type. With storing the processed data type of each node as an instance of `Object`, the mentioned disadvantages are avoided, but this solution requires every `Node`-instance to have a processed data type attribute even without needing one. The third approach moves all the processed data type-related information and methods to an isolated class with the advantage of none of the already existing `Node`-instances have to be touched. As well as, the advantage of the type-safety that comes with the inheritance: A `Port`-instance is a `Node`-instance with additional functionality. Because of the `Port` being generic it can have any type of processed data type. However, because of type erasure the generic type of one instance is not known during run-time, which makes the equality checking of two `Port` instances more complex than usual [jav26a]. Also, the processed data types are often not known by the NSGA-II framework, because they are part of the to be optimized software. Hence, it makes more sense to store the class names as Strings.

To define throughput of a pipe we adjust the definition of Zhao et al. in [Zha+21] s.t., the throughput of a pipe is the amount of send data per time unit through that specific pipe. Because of, two connected input- and output-ports building one pipe in *TeeTime*

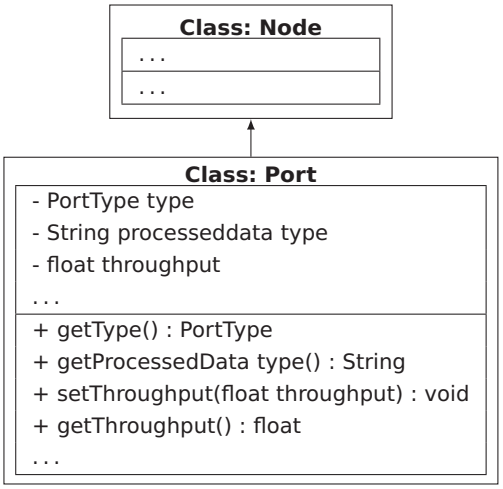

Figure 4.4. Port-class inheriting the Node-class from [Kau25]

[WEH14], we can store the throughput of a pipe either as the incoming amount of data of an input-port or as the produced or outgoing amount of data of an outgoing-port. Alternatively or additionally we can store the throughput as edge-weight of a pipe in a coupling graph. On the one hand to be able to perform all operations on all kinds of ports without being dependent on another port, we will store the throughput of a pipe in both incident ports. If e.g., an output-port is split in two ports, only the outgoing data will be distributed over both new ports, the incoming data of the input-port will not be affected by this operation. How the throughput will be approximated after each operation is further discussed in Figure 4.3. Figure 4.4 shows such a `Port`-class.

Using composition instead of inheritance has the advantage that we are able to re-use all existing functions, which are performing on instances of `ProgramUnit`. We then can also perform all additional operations necessary for ports or nodes afterward in e.g., isolated methods. On the other hand when using inheritance we need to either convert each `ProgramUnit` into a `Port` or re-factor or re-write the necessary functions for instances of `Port`. Also, currently the NSGA-II-framework has a bidirectional association between the `MultiLevelModularization`-class and the `Node`-class as shown in Figure 4.1 and Figure 4.2 [Kau25]. This bidirectional association can become a uni-directional association, with moving all methods and attributes of `Node`, which are needing an instance of the `MultiLevelModularization`-class or are specifically for one `MultiLevelModularization`, into the `MultiLevelModularization`-class. As well as the `MultiLevelModularization`-class only needs to store a list of specific instances of the `ProgramUnit`-class, which have a corresponding instance of `Node`. This way a `Node` can belong to multiple instances of `MultiLevelModularization` and it would reduce or even remove the need of converting instances of either `Node` or `ProgramUnit`. It is even discussable whether the `Node`-class is still

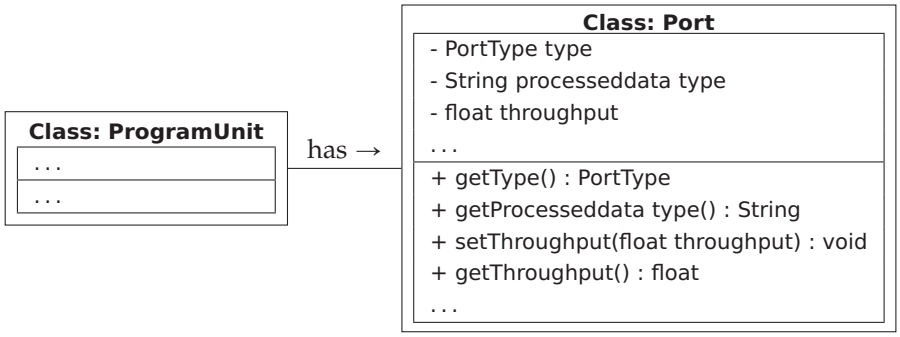

Figure 4.5. Composition: Each ProgramUnit instance of [Kau25] has one instance of Port

necessary, but to keep the in [Kau25] discussed lazy implementation it is important to store some `MultiLevelModularization`-specific information regarding an instance of `Node`. However, this approach requires a lot of refactorings, and thus we have to see how much is possible in the given time interval of this thesis.

An attenuated variant would be to keep the inheritance of `Node` from `ProgramUnit`, but use composition for `Port` as following: Each `Node` is a `ProgramUnit` and each `ProgramUnit` has an instance of the `Port`-class. Transitively, each `Node` has an instance of the `Port`-class. This way the `Node` does not need to be touched, but we still can re-use all functions for a `ProgramUnit` and for an instance of `Port` as described above reducing the re-writing expense. However, one major disadvantage of both composition approaches is that even `ProgramUnit` instances, which do not need a `Port` instance, are having an attribute of type `Port`. This weakened composition approach is shown in Figure 4.5.

The in [Kau25] presented mutation-operations for the `MultiLevelModularization`-class are altered, s.t., each operation can be performed on a given set of `ProgramUnitTypes`. This way it can be ensured that, e.g., only filters are either merged or split.

The mutation operations, both merge random ports (Algorithm 1) and split random ports (Algorithm 3), are inspired by the mutation operations presented in [Kau25] in such a way that the number of new code-lines is minimized. Both mutations can be implemented s.t., stale ports, which are not connected to at least one pipe, are identified or even removed. This can help the developer to reduce the number of clean code and thus reduce the complexity of their software.

A variation of the merging random ports operation is randomly choosing a set of ports and merge them with their adjacent ports - ports which are connected through a pipe - in order to merge at most n filters (Algorithm 2). This operation resembles the *fusion*-operation introduced by Hirzel et al. in [Hir+14]. However, this mutation can suggest merging filters with different functionalities. The already mentioned and currently already implemented operation `RandomMoveNode` can also be used to move port units randomly to a different filter, which would also result in at least partially merging the involved filters, for the moved

port to still function correctly. Moving ports to a different filter can result in re-ordering the filters, like in the *Operator Reordering* operation presented in [Hir+14]. To keep the functionality of a tool, we do not move ports to filters from different tools.

A variation of the splitting random ports can be to create copies of the corresponding filter, instead of adding the new ports to the original filter, similar to the *fission*-operation in [Hir+14]. We can perform the *load balancing*-operation on the resulting split pipes [Hir+14]. However, because of the limited time for this thesis, the throughput is distributed evenly, as further discussed in Figure 4.3, which leaves experimenting with *load balancing* as a prospect.

Algorithm 1 Merge Random Ports of same Type

Require: Original modularization M
Require: Number of ports to merge $n \in \mathbb{N}_{>1}$
1: **function** MERGERANDOMPORTS(M, n)
2: **while** $p_1, p_2, \ldots, p_n$ not chosen or do not have the same weight type **do**
3: Choose n random input- xor output-ports: $p_1, p_2, \ldots, p_n$
4: **end while**
5: Create new port and re-direct all incident edges to this port
6: **end function**

Algorithm 2 Merge Adjacent Ports of different Types

Require: Original modularization M
Require: Number of ports to merge $n \in \mathbb{N}_{>1}$
1: **function** MERGERANDOMPORTSDIFFERENTTYPES(M, n)
2: **while** $p_1, p_2, \ldots, p_n$ not chosen or do not have the same weight type **do**
3: Choose $\frac{n}{2}$ random input- or output-ports: $p_1, p_2, \ldots, p_n$
4: Find the adjacent ports.
5: **end while**
6: Merge adjacent ports to merge the filter
7: **end function**

When re-using the `MultiLevelModularization`-class, it is possible to re-use the recombination operation presented in [Kau25] to perform recombination on multiple candidates. However, a modularization containing ports does not need to fulfil the defined correctness property, which states that "[a] modularization M' is correct, if and only if all leaves of the parent modularization M are existing in M'" [Kau25, p. 12, Definition 4.1], because the ports are the leaves and we want to change the set of ports. Hence, it is possible that the two candidates to combine are having distinguished sets of ports. Thus, when we only take the corresponding filters and tools of one of the sets of ports, as proposed in the recombination operation by Kaufhold, then the resulting modularization will be equal to one of the original modularizations. Consequently, it is necessary to alter the recombination

Algorithm 3 Split Random Port

Require: Original modularization M
Require: Number of resulting ports $n \in \mathbb{N}_{>1}$
1: **function** SPLITRANDOMPORTS(M, n)
2: $i := 0$
3: Choose a random port p_{orig}
4: **while** $i \neq n$ **do**
5: Create new port p_i
6: $i = i + 1$
7: **end while**
8: Randomly re-direct edges from p_{orig} to $p_1, p_2, \ldots p_n$
9: Create new port and re-direct all edges to this port
10: **end function**

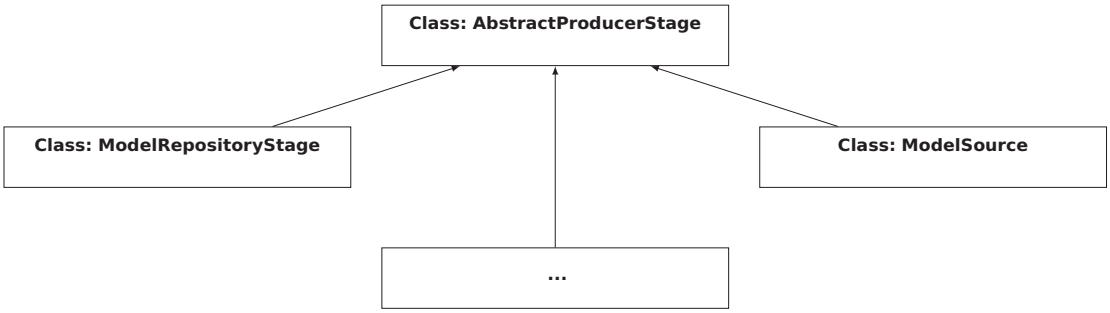

Figure 4.6. Own Illustration based on source-code [Wal+26]: Example class diagram: AbstractProducerStage as parent of ModelRepositoryStage and ModelSource (and other stages).

operation s.t., there are leaves randomly chosen from all modularizations to re-combine, instead of only taking the whole set of leaves of one modularization. Additionally, to keep the functionality of each tool it has to be ensured, that each filter stays in their original tool, which is verified by the presented recombination in [Kau25] with only choosing the leaves randomly and keeping the corresponding path to the root. This ensures that there is a possibility of receiving a new candidate with different ports, even if no ports were moved to a different filter. The rest of the recombination operation of [Kau25] can be re-used. However, the current implementation of [Kau25] is still not working for all cases and Because of that, and for simplicity we will concentrate on mutation-operation and not implement the adapted recombination version in the course of this thesis.

In the end we have decided to implement the weakened composite approach shown in Figure 4.5 with the introduced mutation-operations on pipes in Algorithm 1, Algorithm 2 and Algorithm 3.

4.1.3 Parsing to Tuple Specification

Our strategy to parse the necessary pieces of information for performing operations on pipes to the NSGA-II-framework uses an altered version of the *tuple specification* as shown in [KZ24, p. 4, Listing 3] as following:

▷ (output-port, input-port)

▷ (port, filter)

▷ (filter, tool)

▷ (port, throughput, processed data, type)

The last entry contains all the information about a port. Further information about the produced files and the naming-convention of ports can be found in Section 4.2. With isolating all the new attributes to a new layer instead of writing them in the first entry like (output-port, input-port, throughput, processed data) enables us to re-use an already existing parser `AlgorithmSetupMultiLevelCouplingTupleSpecification.readFromFile(..)`[2], which is processing instances of the `ProgramUnit`-class, with only the need of converting the created units to ports afterward. Otherwise, we would need to distinguish between a `MultiLevelModularization`-modularization containing ports and one which does not in each entry of the tuple format. Indicating the port information entry, e.g., with `---Port Info` helps with simplifying the parser. Furthermore, we implement a new class `AlgorithmSetupPipeFilterTuple`, which inherits from `AlgorithmSetupMultiLevelCouplingTupleSpecification`. In this subclass a subset of the methods is adjusted methods with the goal to maximize the re-use of already existing methods of `AlgorithmSetupPipeFilterTuple`.

Because of the definition of the *pipe-and-filter*-specific mutation-operations in Section 4.1.2 the caller-callee graph can be different in each optimization suggestion, which is not the case with neither the mutation- nor the recombination-operations discussed in [Kau25]. Hence, we need a different output-format for our produced *pipe-and-filter*-specific optimization, than for the package-hierarchy related optimizations, to provide the corresponding caller-callee graph of each optimization. Therefore, we also implement a class `ResultPrinterPipeFilter`, which inherits from `ResultPrinterMultiLevelModularization` and overrides `printOptimization(..)`[3] s.t., the corresponding caller-callee graph is printed to the same `PrintStream`.

[2] https://cau-git.rz.uni-kiel.de/ifi-ag-se/oceandsl/genetic-restructuring/-/blob/93ad89a3a70cabe8bf05c1c35c4ce094268cfdf3/src/main/java/cau/agse/hs/genopt/nestedcoupling/jobs/AlgorithmSetupMultiLevelCouplingTupleSpecification.java#L118

[3] https://cau-git.rz.uni-kiel.de/ifi-ag-se/oceandsl/genetic-restructuring/-/blob/93ad89a3a70cabe8bf05c1c35c4ce094268cfdf3/src/main/java/cau/agse/hs/genopt/nestedcoupling/jobs/ResultPrinterMultiLevelModularization.java#L84

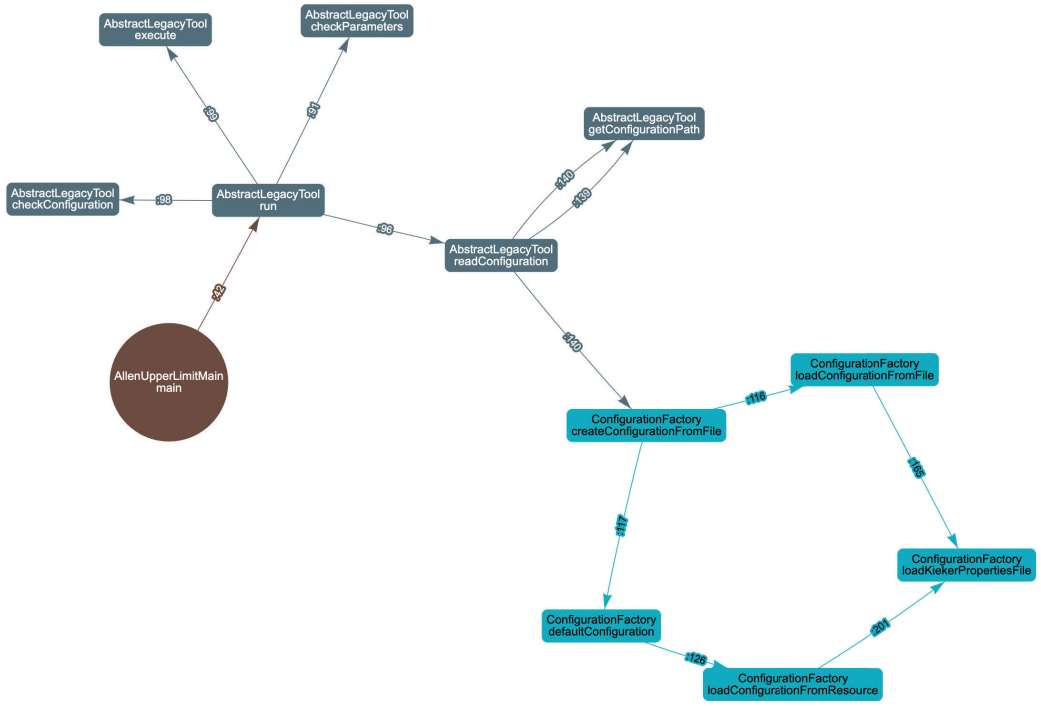

Figure 4.7. Call Graph for Tool Allen Upper Limit using CallGraph plugin

4.2 Analyzing the Kieker Analysis Part

Our goal is to monitor and analyze the whole analysis part of *Kieker*. However, *Kieker* consists of multiple analysis tools, also called plugins as show in Figure 2.6 and also mentioned in [HH20]. To do so we tried the following approaches to either monitor or reverse engineer *Kieker*'s analyze package.

IntelliJ IDEA from Jetbrains[4] supports the creation of a call graph by using static analysis [jav26b]. Furthermore, by using a plugin like *CallGraph* by Yunus Emre GUL[5] a call graph can be visualized, as shown in Figure 4.7 for one of *Kieker*'s analysis tools. Based on the source-code, the edge labels shown in Figure 4.7 are the line-numbers of the corresponding call and not the frequency of calls.[6] However, only a static call graph is not sufficient, because to gain information about the throughput of a pipe it is necessary to at least get the frequency of method calls if not the throughput directly.

[4] https://www.jetbrains.com/idea/

[5] https://plugins.jetbrains.com/plugin/27227-callgraph

[6] https://github.com/yunusemregul/callgraph/blob/a1037a8ad1867c2317863fd220ce4ef34629b9cb/src/main/java/callgraph/callgraph/CallGraphGenerator.java#L309

For dynamic analysis, e.g., runtime analysis, IntelliJ provides different types of profiling: *CPU and memory live charts, CPU and allocation profiling, Memory snapshots* and *Thread dumps* [Jet26]. But none of these types provide information about the method call frequency out of the box.

Another approach for both dynamic and static analysis is to use *Kieker* to monitor and analyze itself, while analyzing another software system. Waller has already performed the monitoring of *Kieker* 1.10 monitoring a software system by re-naming *Kieker* in Kicker to avoid an endless monitoring loop [Wal14]. Based on that idea we want to monitor and analyze the analysis part of *Kieker*.

The AG Software Engineering provides a replication package[7] to analyze earth system models as part of the SustainKieker project using *Kieker* with OceanDSL tools. OceanDSL is a project of the AG Software Engineering, which, among others, offers domain-specific languages for ocean models [Jun+21; WJ23]. When comparing the set of analysis tools of *Kieker*[8] and OceanDSL it can be seen that the OceanDSL tools[9] are based on the analysis tools of *Kieker*. Thus, by monitoring this replication package we could monitor a run of *Kieker 4 C* monitoring and analyzing example earth system models, which provides information over some *Kieker* analysis tools.[10] However, a problem occurred with both the collector of the latest *Kieker* binaries and version 1.15.4, which is mentioned as the version to use in the README-file of the project. Based on the thrown exception in Listing D.1, the main cause of the issues was a problem with the logging dependencies, i.e., the logback-dependency. This issue has been fixed from the AG Software Engineering later on by updating all logback-dependencies to the latest version. However, other problems kept occurring, like empty or broken csv-files (Listing D.2) and missing model files (Listing D.3). Bringing this attempt to a hold. The thrown exceptions are at least proof that the OceanDSL tools used in the replication package are using *TeeTime* and thus by monitoring and analyzing this package information about the used *pipe-and-filter*-architecture could be retrieved.

The `MultiLevelModularization`-class, as shown in Figure 4.1, of the provided NSGA-II framework needs a coupling graph as `callerCalleeGraph` and a package-tree hierarchy as `layerList` [Kau25]. Currently the BCEL library[11] is used to gain both a coupling graph between classes and a package-tree hierarchy.[12]

With the BCEL library it is possible to gain an inheritance-tree for each class.[13] Thus and with all filters inheriting from or implementing the *TeeTime*-stages, as discussed in

[7] https://git.se.informatik.uni-kiel.de/SustainKieker/esm-architecture-analysis-replication-package/

[8] https://github.com/kieker-monitoring/kieker/tree/main/tools

[9] https://git.se.informatik.uni-kiel.de/oceandsl/oceandsl-tools/-/tree/ed81a0d7bc9ddf9cb3ae696427e6da39925a88e5/tools

[10] https://git.se.informatik.uni-kiel.de/SustainKieker/esm-architecture-analysis-replication-package/-/blob/f194c23b2ad42a7725ae53b921f22b2e75368b35/README.md

[11] https://commons.apache.org/proper/commons-bcel/

[12] https://cau-git.rz.uni-kiel.de/ifi-ag-se/oceandsl/genetic-restructuring/-/blob/045cf6f29429a1735bf0e01bb95cca3871820d29/src/main/java/cau/agse/hs/genopt/flatcoupling/jobs/AlgorithmSetupCouplingBCELParser.java

[13] e.g., https://commons.apache.org/proper/commons-bcel/apidocs/org/apache/bcel/Repository.html#getSuperClasses(org.apache.bcel.classfile.JavaClass)

Section 4.1 and shown in Figure 4.6, it is possible to find all filters of *Kieker* by parsing the BCEL log-files. Furthermore, both the input- and output-ports are realised as instances of `teetime.framework.OutputPort` or `teetime.framework.InputPort`, e.g., the `inputPort` variable in `AbstractConsumerStage.java`[14]. Because of that, it is also possible to count the number of input- and output-ports for each filter. However, the exact names of each port are unknown in the logs, and thus we use a naming pattern like `filterA.inputporti` for the ith input-port of filterA. This way the BCEL logs can be used to create a partial tuple specification as discussed in Section 4.1 containing the sections (port, filter) and (filter, tool). Therefore, we have created a bash-script *teetimeFiltersToTupleSpecification.sh*, which gets a BCEL log-file and returns a file named *originalModBCEL.mod* in the defined output-directory.

We have also created another bash-script *createOriginalModFromSource.sh* to be independent of the BCEL library, which generates a file named *originalMod.mod* generated by parsing the *Kieker*-analysis source-code for methods like `get{Input,Output}Port()`, which are functions of abstract classes like `AbstractTransformation` and `AbstractConsumerStage`, which are in turn super-classes of most stage-classes [Jun+24]. However, some classes use different methods to get a port, e.g., `kieker.analysis.generic.time.TimestampFilter`[15], and thus the resulting file is only an approximation with an error rate. Also different classes are creating different instances of the same filter with the same name, e.g., the `timestampFilter`-instance of `kieker.tools.log.replayer.TeetimeConfiguration`[16] and the `timestampFilter`-instance of `kieker.tools.trace.analysis.TraceAnalysisConfiguration`[17]. Because these filters can belong to different tools, as seen in the example above, we have decided to use the class and package of the corresponding tool instead of the actual class of the instance, i.e., `declaringClass.variableType.variable.portType<PortAppearance>` with the `declaringClass` containing of their corresponding tool and package. This way the filter-instances are mapped to the correct scopes and the variable of one non-static instance is unique per defining class.

Equally, the *couplingmodel* file is generated by a bash-script *createCouplingModelPortsFilters.sh* also parsing *Kieker*'s source-code for both the generation and the connection of ports. To connect an output-port with an input-port the *TeeTime*-function `connectPorts()` can be used and thus by parsing for this function-call we can receive information about which output-port connects to which input-port and which ports are existing [Wul+19; WHO17]. Actually, the script *createWholeModelFromSource.sh* calls *createCouplingModelPortsFilters.sh* to receive the necessary information about the ports. This way both the resulting *originalMod.mod* and *couplingmodel* files are consistent, too. For simplicity, at first we assign each

[14] https://github.com/teetime-framework/TeeTime/blob/60a8a42688a0da5e2e25103374f041fd3b4f22b3/src/main/java/teetime/framework/AbstractConsumerStage.java#L25

[15] https://github.com/kieker-monitoring/kieker/blob/0231b0185278eccfd80d15ea98d7f9aca166ffa4/analysis/src/kieker/analysis/generic/time/TimestampFilter.java#L111

[16] https://github.com/kieker-monitoring/kieker/blob/0348d4e5d9c6d4e9e24bee37583f626931b87b91/tools/log-replayer/src/kieker/tools/log/replayer/TeetimeConfiguration.java#L58

[17] https://github.com/kieker-monitoring/kieker/blob/0348d4e5d9c6d4e9e24bee37583f626931b87b91/tools/trace-analysis/src/kieker/tools/trace/analysis/TraceAnalysisConfiguration.java#L138

(a) Before splitting **(b)** After splitting

Figure 4.8. Directed Pipe between two Ports with Throughput n before and after splitting operation using the presented graph representation in Section 2.2

port the same throughput and a generic processed data type. If wished, a *Kieker*-developer can assign more accurate information to the ports.

4.3 Throughput Approximation for Evolutionary Search

To minimize the run-time of a run of the NSGA-II framework it is necessary to reduce the number of measurements. Thus, instead of measuring the actual throughput for each candidate, we will approximate it.

Our naive approach is to adjust the throughput of a port percentage-wise to the new number of ports, i.e., if the output-port P_1 is split into two output-ports $P_{1.1}$ and $P_{1.2}$, then the original produced throughput of $n \in \mathbb{R}_{\geqslant 0}$ will be equally distributed over both new ports. This way the consumed throughput of input-port P_2 stays the same as before the split operation. This example is shown in Figure 4.8a and Figure 4.8b. Whereas, the merging operation sums up all the throughput over all merged ports. Equation 4.1 defines the explained approximation recursively.

If two adjacent ports, i.e., ports connected through a pipe [WEH14], are merged, the incident pipe will be removed which also removes the throughput of the involved ports.

$$throughput(P_i) = \begin{cases} \frac{throughput(P)}{\#\text{new ports}}, & \text{if } P \text{ is split in \#new ports} \\ \sum_{P_j \in \text{ports to merge}} throughput(P_j), & \text{if } P_i = \text{merged port of multiple ports.} \end{cases} \tag{4.1}$$

Due to the time limitation and focus of this thesis we have not yet researched this topic properly. As prospect, it would be interesting to discuss and research existing heuristics and approaches, e.g., probabilistic models like the ones used in [Gha+26], to approximate the throughput in order to minimize the error.

4.4 Filter Technique

In case there is no single optimal solution, the Pareto-front consists of multiple trade-off optimal solutions [NZE05]. Hence, it is desirable to filter the Pareto-front s.t., the number of candidates to evaluate by the user are minimized. Since all solutions of the Pareto-front are by definition of the Pareto-front non-dominated, "[representing] different [...] trade-offs between the objectives" [NZE05, p. 85], the decision which solution is preferable is subjective and that makes filtering non-trivial. However, Bechikh, Ben Said, and Ghédira state in [BBG11] that so called knee-solutions inside the Pareto-front are of high interest to the user, because of their characteristic regarding trade-off. To be precise, Rachmawati and Srinivasan states in [RS09] that such a knee-solution x_i correlates with the local maximal minimum trade-off when substituting any other Pareto-solution with x_i. To calculate this minimum trade-off for each solution $x_i \in S$ [RS09] proposes the metric shown in Equation 4.2 with S being the set of non-dominated solutions. In the original equation by [RS09] is an additional parameter j, which is not further explained and vanishes when [BBG11] re-states that equation. Thus, we have decided to remove it like [BBG11] does.

$$\mu(x_i, S) = Min_{x_j \in S, x_i \neq x_j, x_j \neq x_i} T(x_i, x_j) \tag{4.2}$$

The domination-relation $\prec$ is defined over the set of all solutions and if one has $x_i \prec x_j$ with $x_i \in x_j$, then x_i dominates x_j in at least one objective and is at least as good as x_j in every other objective [RS09].

Instead of using the trade-off metric proposed by [RS09], we choose to use the adapted version by [BBG11] shown in Equation 4.3, which is based on the trade-off metric by [RS09], because this adopted version contains the normalization over the maximal deterioration, while the original metric by [RS09] presumes that the normalization is done beforehand. In Equation 4.3 the function T is defined for a minimization problem, because otherwise the highest trade-off would be a negative value, which would be ignored both by the maximization and the lower-bound of 0. For a maximization problem either the numerator needs to be negated or the term needs to be minimized with an upper-bound of 0 instead of the lower bound. In the second option a negative trade-off is achieved, and thus the first option needs less adjustment steps.

$$T(x_i, x_j) = \frac{\sum_{m-1}^{M} max[0, \frac{f_m(x_j) - f_m(x_i)}{f_m^{max} - f_m^{min}}]}{\sum_{m-1}^{M} max[0, \frac{f_m(x_i) - f_m(x_j)}{f_m^{max} - f_m^{min}}]} \tag{4.3}$$

M is the number of all objectives, while $0 < m \leqslant M$ is the number of one single objective. The value of objective m for solution x_i is defined as the value of $f_m(x_i)$ with f_m^{max} and f_m^{min} being the maximal and minimal values over the whole population [BBG11].

Bechikh, Ben Said, and Ghédira propose in [BBG11] the Trade-off based Knee-based Reference point-based NSGA-II (TKR-NSGA-II) by using mobile reference points to find the knee regions, which is an enhanced version of the NSGA-II. These mobile reference

points are vectors of the preferences of a user and are "[...] automatically updated in each generation" [BBG11, p. 1812]. To find the new set of mobile reference points, the solutions of the Pareto-Front of one generation are descendant sorted according to μ and only the $k \in \mathbb{N}$ first solutions are selected [BBG11].

The NSGA-III also uses reference points to be able to propose only a small subset of solutions on preferences of the user, if provided, by just providing a part of the Pareto-front instead of the whole front [DJ13]. Therefore, the NSGA-III provides a filtering of the solutions on its own.

Because of time limitations we do not implement this technique.

Chapter 5

Evaluation

5.1 Experiments

Some excerpts of our experiment-results can be found in Appendix B, as well as links to the whole files.

Our NSGA-II-framework produces, among other files, an `optimizations.txt` in the `ResultPrinterMultiLevelModularization.printResults(...)`[1] containing all optimization suggestions in the Tuple Specification introduced by Kaufhold and Zwanger in [KZ24]. To make the suggested optimizations better readable we suggest a similar approach as Kaufhold and Zwanger using the UNIX `diff`-command to show the differences between the original modularization and one optimization suggestion. Therefore, each single solution needs to be moved to its own file using a script or be written to its own file in the first place by modifying the `ResultPrinterMultiLevelModularization.printResults(...)` of the NSGA-II-framework. Otherwise, most lines of the file count as different, because of the definition of the `diff`-command [Ker26]. If the user prefers one large file containing all optimization suggestions and differences for each solution all files can be merged into one file again or the user does not need to alter the original `optimizations.txt` at all, if they do not want to have got the differences. The advantage of using a script instead of altering the mentioned method is that no redundant output is produced in case the user prefers the original `optimizations.txt` and thus the single optimization-files can be generated on demand. Because of this advantage we have decided for using a script called `splitOptimizations.sh` instead of altering `ResultPrinterMultiLevelModularization.printResults(...)`, even though this approach introduces an extra step for the user. The script produces $n + 1$ files with $n \in \mathbb{N}$ optimization suggestions. The extra file contains the original modularization, which can be used as input for the UNIX `diff`-command, i.e., `diff optimizations.txt_1 optimizations.txt_n+1` for showing the differences between the first optimization suggestion and the original modularization.

For the discussed experiments in this section n job-files in the `genetic-restructuring`-project are used: Each job-file uses one of the mentioned sets of metrics in Section 3.2 and equals one optimization run, which results in n optimization runs in total. The job-files, which use the *pipe-and-filter*-specific metrics as fitness functions and the discussed *pipe-and-*

[1] `https://cau-git.rz.uni-kiel.de/ifi-ag-se/oceandsl/genetic-restructuring/-/blob/3b46b776d714451e992c750dd46eb761dd65d67d/src/main/java/cau/agse/hs/genopt/nestedcoupling/jobs/ResultPrinterMultiLevelModularization.java#L75`

Table 5.1. Mapping of P&F-Jobs to used Sets of Metrics

Job	Set of Metrics
Job01	(Fan-in Coupling, Fan-out Coupling)
Job02	(Fan-in Coupling, PullThroughputMetric, #PaFs)
Job03	(Fan-in Coupling, PullThroughputMetric)
Job04	(PullThroughputMetric, #PaFs)
Job05	(PullThroughputMetric, #PaFs)

filter-specific mutation-operations in Section 4.1.2 can be found in `resources/jobs/kieker-pipe-filter`. The mapping which job uses which set of metrics as fitness functions can be found in Table 5.1. While the job-file, which uses the coupling-metrics discussed in Section 3.1.1, uses the mutation- and recombination-operations for a `MultiLevelModularization` introduced in [Kau25] and the file can be found in `resources/jobs/multilevel-kieker`. Since not all optimization-jobs use the same operations and are receiving different types of the tuple specification introduced by [KZ24] as further discussed in Section 4.1.3, as inputs, they are not actually comparable. All jobs have got a population size of 1000, mutation and recombination rates of 0.125 and run for either at most two hours or 1000 generations as in [SHJ25b]. Additionally, all pipes have got a default throughput of 10, artificial processed data types in the original state and the threshold for the #PaFs-metric is set to a comparably high value of 500 to ensure the optimization to perform operations in order to optimize this metric.

The following figures show a tendency of the optimization of objectives in different metrics sets (Section 3.2) in randomized optimization runs as discussed in Section 5.1. The original state is marked with a square with a dot inside. All used excerpts of values to plot the following figures can be found in Appendix B.

5.2 Limitations and Problems

We further distinguish between the re-modularization of the package-structure of *Kieker* and the re-modularization of *Kieker*'s *pipe-and-filter*-structure. Both types of experiments can not be compared directly, because of the different types of edges and operations used, which are further described in Section 4.1.

Since each run performs the NSGA-II as described by Deb et al. in [Deb+02] on a random parent population and with different fitness functions, we have at least as many distinguished experiments as sets of metrics. The NSGA-II sorts the members of the next population descending by their degree of fitness, using both the non-dominated sorting into different fronts and the crowding distance sorting [Deb+02]. Because of that, the resulting optimization suggestions of our experiments are also sorted descending according to their

fitness quality and thus, it is sufficient to only plot a few suggestions with both ends of the solution sets being presented to show the range. However, in order to improve readability, we are negligent of possible exceptional optimizations in the middle of the spectrum.

As discussed in Section 3.3 the optimal case will be, if all metrics are minimized and thus, a small value is better than a large value. However, when considering an inverted metric, we will talk about the original metric value, i.e., lack of cohesion has got a low value, and we will mention the corresponding high value of cohesion. Furthermore, two optimizations can have got the same values, but be different regarding the proposed structure, i.e., classes can be in packages with different names in both optimizations, but the two optimizations are topologically equal. However, this assumption needs to be further examined.

It also needs to be taken into account that with optimizing each objective for each of the $n \in \mathbb{N}_{>1}$ layers, the NSGA-II has got at least n times the number of objectives, which can result into the problem with only non-dominated solutions [Mka+15], further discussed in Section 2.1.

If we discuss the minimal average re-factoring effort it needs to be taken into account that it is possible that a experiment, which runs for a longer time than ours, will result in a different recommendation. Also, the actual amount of necessary changes to the corresponding source-code can be larger than expected by the re-modularization suggestion [Hal+14].

Beyond all mentioned above, currently are still the exceptions shown in Listing D.4 and Listing D.5 thrown during the *pipe-and-filter*-runs. Both exceptions hint to a problem with `mergeAdjacentPortsRandomly(...)`, which is implementing Algorithm 2. As discussed in Section 4.1.2 to keep the type-safe environment of *TeeTime* in order to merge a set of ports do need all ports the same type of processed data to guarantee the type-safety of the resulting pipe [WEH14]. Furthermore, we currently only allow to either merge adjacent ports, i.e., ports connected through a pipe [WEH14], in Algorithm 2 or ports of the same type, e.g., input-port, in Algorithm 1. These exceptions are caused by an unexpected behavior, which can lead to a distortion of the results of the *pipe-and-filter*-experiments. Because of that, we have performed an additional experiment, which does not perform Algorithm 2: Run 2026-08-12-19-52-33. The corresponding logs do not contain a single exception, which could prove our assumption. However, it is also possible that this result is just a coincidence.

5.3 Results of both Approaches

5.3.1 Package Re-Modularization

Firstly, our NSGA-II-implementation suggests optimizations, which worsen the metrics of one layer in order improve another as intended by the NSGA-II after [Deb+02], e.g., *Kieker*'s original state has got a better coupling factor value and a worse cohesion value for layer one than optimization-2 (Table B.3) and thus both solutions do not dominate each other.

However, there are the following additional observations:

Beginning with layer four (Figure 5.3) the original-state for *Kieker*'s CouplingFactor (Section 3.1.1) is already optimal and NSGA-II provides many metrics-wise optimal optimizations: Optimizations, where both objectives are equal to zero. Especially layer eight has got only optimal CouplingFactor values, at least when considering the excerpt in Table B.10. Furthermore, layer five to eight even start with optimal values regarding cohesion. It needs to be taken into account that some optimizations have got less than eight layers, e.g., the original modularization with only six layers: Layer zero to five. As we can see in Table B.2 until Table B.10, metrics were calculated or set respectively for eight layers of the original state and, furthermore, to calculate metrics for one layer a layer graph is necessary [Kau25]. Thus, the fact that only six layer graphs exist in Listing B.1 indicates that the original state only has got six actual layers. This results in the non-existent higher layers being optimal by default as indicated by the values of the original-state in layer six (Table B.8) to eight (Table B.10), because if at least one candidate has got $n + 1$ layers, then every optimization needs $n + 1$ layers to be comparable regarding the Pareto-Dominance after Ngatchou, Zarei, and El-Sharkawi. Therefore, we fill up the missing objectives with placeholder values, which are currently 0.0. It is noticeable that the higher optimizations regarding the sorting of solutions tend to have more layers than the lower ones, e.g., opt-999 has eight layers which is proven by the fact that in Table B.10 the lack of cohesion value is > 0. If we consider the definition of Pareto-Dominance [NZE05] and that the fitness of a solution is defined over it [Deb+02], then this behavior is expectable: If we have two solutions opt-x and opt-y and both are having the same values for the first n layers, but opt-y has got a value for layer $n + 1$ which is not equal zero and opt-x only has got n layers with a value equal to zero for the non-existent layer $n + 1$, then opt-x Pareto-dominates the solution opt-y. If this behavior shall be prevented, then the non-existing layers can be punished by assigning the highest possible value of 1.0.

Layer one (Figure 5.1) and two (Figure 5.2) also start with an optimal CouplingFactor value, but provide mostly optimizations with worse CouplingFactor values than the original-state. Regarding the Pareto-dominance [NZE05] they can still be elements in the same front, because they provide an improvement for cohesion and thus are not dominated by the original-state. Solutions, which are dominated, are elements in one of the higher fronts and NSGA-II fills up the population with dominated solutions, if the size of the non-dominated front is smaller than the configured size of the population [Deb+02]. Both layer zero (Figure 5.1) and three (Figure 5.2) provide some metrics-wise optimal solutions, but the ratio between the number of optimal and non-optimal optimizations is more balanced than in layer four to eight.

An interesting optimization is optimization-1, because this optimization proposes a small isolated complete tree growing over layer zero until one instead of placing the root on the highest layer, layer 4:
`kieker.analysis.IProjectContext` → `263layer1237layer1kieker` → `none` (Listing B.3). The string `none` is a substitute for a null-value and indicates that `263layer1237layer1kieker` is

one root node in this context, because none-Nodes are added to a `MultiLevelModularization`, when there would be a null-value otherwise. A proof for this assumption is the case when a null-value would be added as value for a root node in the corresponding `layerList`-hashmap, but a none-node is added instead[2]. This means that `263layer1237layer1kieker` is not a member of any package and with optimization-1 having five layers indicated by five layer-graphs (Listing B.3) this suggestion is not implementable as it is: The developer needs to decide how to implement an isolated tree in the middle of the hierarchy. Two options are either `263layer1237layer1kieker` will be to moved to the highest layer with `kieker.analysis.IProjectContext` following it to layer three or `263layer1237layer1kieker` is moved to a different package.

In run 2026-07-14-14-32-26 with 1000 optimizations are 107.82 units changed per average (Table B.1). This number only contains the moved and removed units, because if a new unit is created by either splitting or merging units, then other units are either removed or moved in the process [Kau25]. Also, it symbolizes the minimal average number of necessary refactorings to implement the suggested optimizations. In direct comparison to the suggested optimizations of *Kieker*'s *pipe-and-filter*-structure the only job, which produces less amount of minimal mean re-factoring effort is job 3, which uses the metrics set (Fan-in coupling, PullThroughputMetric), with less than 44.23 added ports on average (Table B.11).

5.3.2 P&F Optimization

The lack of filter fan-in coupling *LFilterFanInCoup* (Section 3.3.2) is a member of each *pipe-and-filter*-specific set of metrics mentioned in Section 3.2, except for the control sets, because it is the only proposed *pipe-and-filter*-metric which rewards the splitting of pipes, see Chapter 3 for more details. Furthermore, a fresh port is a port, which is the result when multiple ports of the same type are merged during the mutation-operation Algorithm 1. Thus, the number of fresh ports indicates the average number of performed merge-operations per optimization.

It is noteworthy that the objective of minimizing *LFilterFanInCoup* was only noticeable improved when it was used as fitness function in combination with the objective of minimizing the lack of PullThroughputMetric *LPortPullThroughput* (Section 3.3.3). This is best shown in Figure 5.8 for layer 0. The moment the objective of minimizing #PaFs (= *PipeFilterAntipatterns*) (Section 3.1.4) was added as fitness function, *LFilterFanInCoup* was not noticeably improved any more as shown in Figure 5.7. But even when *LPortPullThroughput* was substituted with another metric, which rewards the merging-operations, e.g., fan-out coupling *FilterFanOutCoup* (Section 3.1.2), *LFilterFanInCoup* was not noticeably improved anymore, as shown in Figure 5.6. The splitting operations produce new pipes and if each filter has got more outgoing than ingoing ports, then the value for lack of fan-in coupling worsens. Thus, this leads us to the assumption that more input-ports exist than output-

[2] `https://cau-git.rz.uni-kiel.de/ifi-ag-se/oceandsl/genetic-restructuring/-/blob/a685a68618e821b1cf0918c7d8880738a51ed540/src/main/java/cau/agse/hs/genopt/nestedcoupling/tools/ParseMultiLevelModels.java#L367`

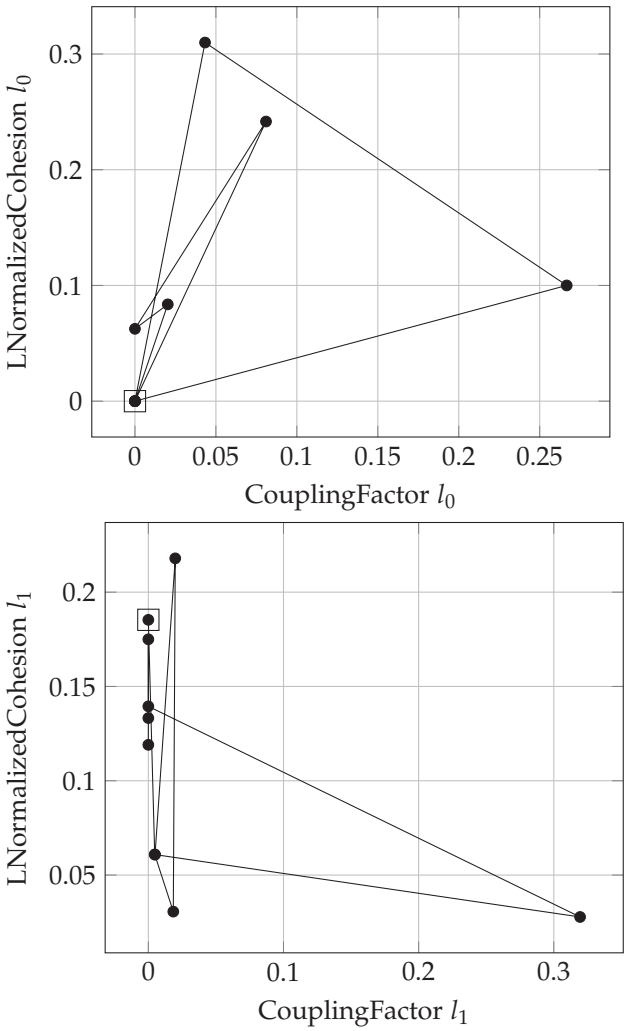

Figure 5.1. Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 0 and 1 Table B.2, Table B.3

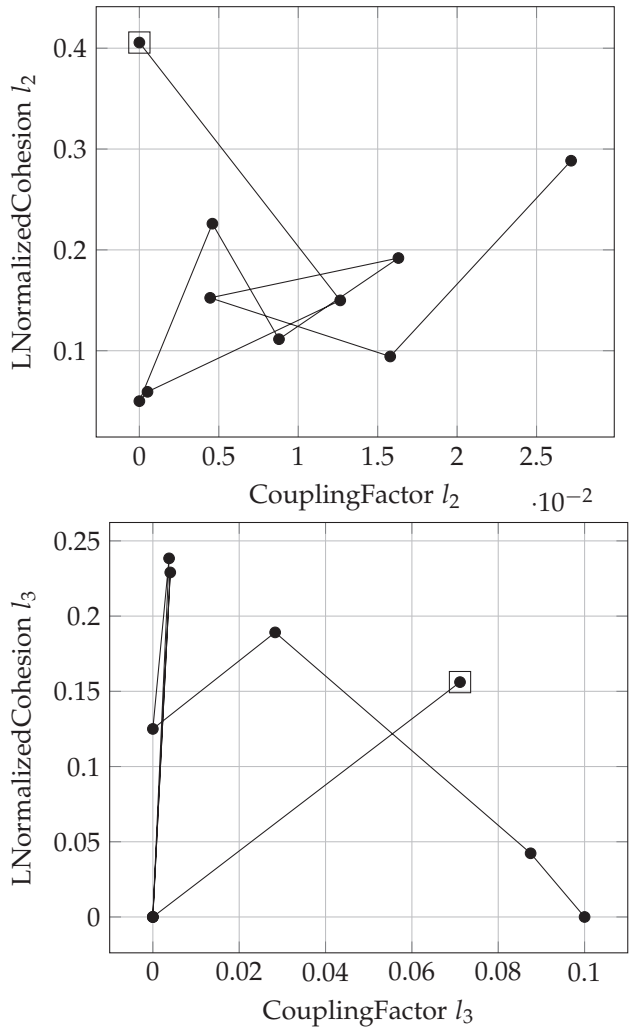

Figure 5.2. Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 2 and 3 Table B.4, Table B.5

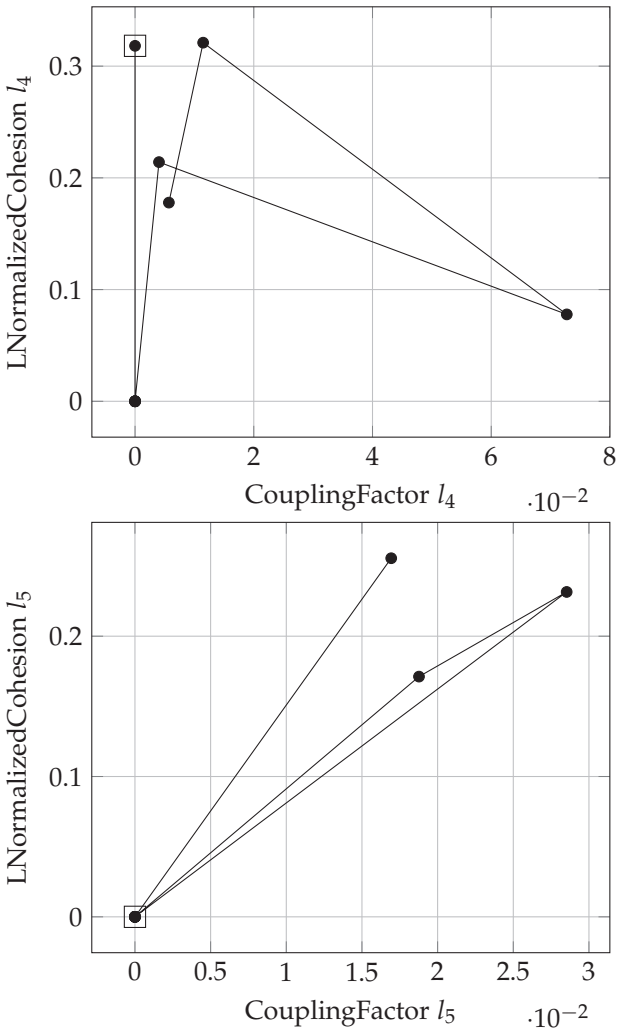

Figure 5.3. Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 4 and 5 Table B.2, Table B.7

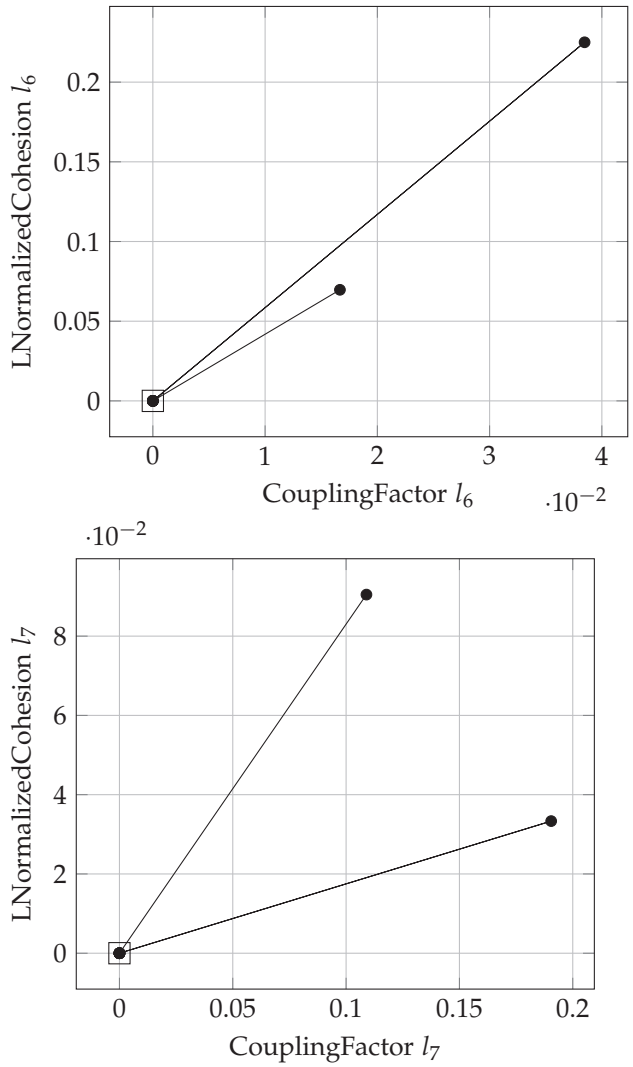

Figure 5.4. Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layers 6 and 7 Table B.2, Table B.3

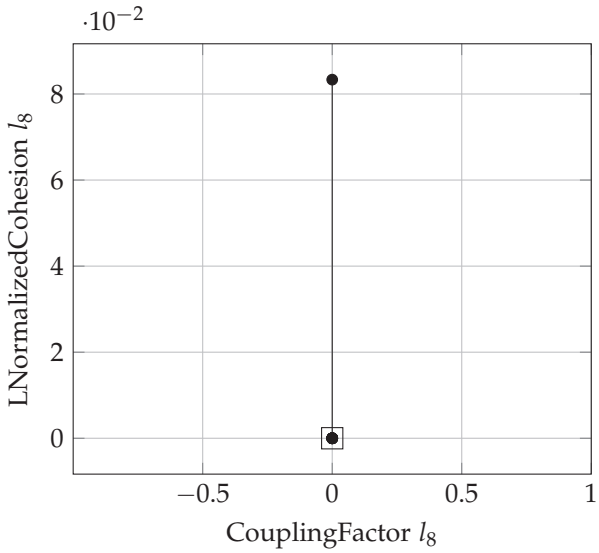

Figure 5.5. Values for an excerpt of 9 of 1000 optimizations and the original state for the Metrics-set (CouplingFactor, LNormalizedCohesion) of run 2026-07-14-14-32-26 Layer8 Table B.10

ports. This is proven by the ratio of the average number of output ports to the average number of input ports per optimization shown in Table B.13. Whereas, the average number of ports per optimization is less than the number of ports of the original state, which is also shown in Table B.13. This indicates that some merge-operations were performed.

Figure 5.8 shows for layer 0 that a high PullThroughputMetric value, respectively a low lack of PullThroughputMetric value, is independent of the fan-in coupling value. This behavior is unexpected, because after the definition of the PullThroughputMetric by Adolf less pipes should result in a higher value or else more pipes in a lower value, except of the case when the mean throughput is equal to the summed up throughput of all involved pipes [Ado16]. Leading to the conclusion that this case is met at least once. Equivalently behavior is observable with the #PaFs values in combination with the *LFilterFanInCoup*. Furthermore, a low lack of PullThroughputMetric value is paired with a high #PaFs value, see Figure 5.7. Since both metrics are rewarding the merging of pipes, we would have expected a low #PaFs value instead. Hence, we have added a set of metrics only consisting of PullThroughputMetric and #PaFs to verify this behavior and the role of *LFilterFanInCoup* in it. It produces, among others, high #PaFs values with low PullThroughputMetric values in Figure 5.7, which answers our expectations, because when the filters of an optimization have got a low incoming throughput, then the high throughput threshold is not met. The #PaFs values are expectedly high, because the threshold has been set quite steep with 500 in comparison to the default throughput values of 10. It is to be expected that a lower threshold or higher throughput values will lead to better #PaFs values. Figure 5.10 also

shows that the metrics set of (PullThroughputMetric, #PaFs) behaves like the larger set of (Fan-in Coupling, PullThroughputMetric, #PaFs): Both low and high PullThroughputMetric values are paired with low #PaFs values. Remember pairing high PullThroughputMetric values with low #PaFs would have been the expected behavior. Thus, optimizing fan-in coupling does not have much impact on the optimization of PullThroughputMetric and #PaFs.

Table B.11 shows that job 5, which uses the metrics set (PullThroughputMetric, #PaFs) introduces the most average fresh ports, with 47 fresh ports per optimization, which is expectable, because both metrics are rewarding the merge-operations. However, job 2 using the metrics set (Fan-in Coupling, PullThroughputMetric, #PaFs) introduces the most new ports on average, including fresh ports. While job 3 using the metrics set (Fan-in Coupling, PullThroughputMetric) introduces the least amount of new ports on average. Hence, job 2 has got the most necessary changes made to implement this optimization and thus the highest minimal re-factoring effort, while job 3 has got the lowest minimal re-factoring effort of all *pipe-and-filter*-jobs. If the user prefers a small re-factoring effort, the metrics set (Fan-in Coupling, PullThroughputMetric) of job 3 is recommendable. However, our NSGA-II-framework can suggest hyperedges, which connect one port with multiple other ports. As discussed in Section 2.3, *TeeTime* does not directly support hyperedges and thus this suggestion can introduce additional re-factoring overhead, because of the need of implementing either a *distributor* or *merger* as introduced by Wulf, Ehmke, and Hasselbring in [WEH14].

Looking at the ratio of fresh ports and overall added ports summed up over all optimizations in Table B.11 we see that more ports were introduced by splitting than by merging. This contradicts our assumption that the current NSGA-II-implementation prefers the merging-operations.

The fan-out coupling *FilterFanOutCoup* (Section 3.1.2) has got noticeably good values. As shown in Figure 5.6 the original *pipe-and-filter*-structure of *Kieker* has already low values and thus our implementation of NSGA-II has not notably worsen this objective.

Layer l_0 contains the ports and layer l_1 contains the filters. Except for *LFilterFanInCoup* all metrics are worse for l_1 than for l_0, because they have already been worse in the original-state, i.e., comparing the values for *FilterFanOutCoup* of the original state both l_0 with $\leqslant 8.65 \times 10^{-4}$ (Table B.14) and l_1 with $\leqslant 28.17 \times 10^{-4}$ (Table B.15). However, the values of the optimizations are for most cases an improvement in comparison to the values of the original state, e.g., the values of both metrics of job 04 layer 0 (Table B.20). One exception are the values of the PullThroughputMetric in job 03 layer 1 (Table B.19): The first four optimizations are an improvement and even optimal, but the last five optimizations are all worse metrics-wise than the original state.

In the package re-modularization there are both better and worse values for the used metrics in the higher layers than in the lower layers, e.g., optimization-997 of run 026-07-14-14-32-26 Job 01 has an optimal value for the CouplingFactor in layer 1 (Table B.3) but not in layer 0 (Table B.2). One possible cause is that, if the original state is already optimal,

the performed operations only either worsen or keep the original values, e.g., in layer 1 (Table B.3) the values of the CouplingFactor. Whereas, in the case that the values of the original state are not optimal, then the operations can still improve them. This is the case for the values of the LNormalizedCohesion in layer 1 (Table B.3).

Run 2026-08-12-19-52-33 does not perform Algorithm 2. Table B.12 shows that the number of optimizations is significantly lower than the number of optimizations of both runs performing all merge-operations (Table B.11) This leads us to the assumption that additionally performing Algorithm 2 provides a potential for producing a higher variety of solutions then without performing Algorithm 2, because each population is filled up by solutions of higher non-dominated fronts until the set size of a population is reached if possible and thus if a population is smaller than another, then there are less different solutions to choose from [Deb+02]. A special case is job03 optimizing fan-in coupling and PullThroughputMetric, because the number of optimizations dropped to 2, which is a difference of 1487 optimizations in comparison to the runs performing all merge-operations and hints to a smaller search space. Whereas, both the number of averagely added ports and maximal added ports are higher for all jobs than in the runs performing all merge-operations.

5.3.3 *Kieker*-Developer Feedback

We have asked a developer of *Kieker* for his feedback to our optimization suggestions and he has looked at one job for the package and *pipe-and-filter* re-modularization each. The original correspondence can be found in Appendix C. For better understanding of the results he has generated a shell-script to convert our tuple specification, as discussed in Section 4.1, into dot-graphs [Yan26].

As a feedback to optimization-0 of 2026-07-14-12-35-59 Job 01, which uses Coupling (Equation 3.1) and lack of cohesion (Equation 3.3) as fitness functions, Yang has mentioned that the package `616layer1kieker.analysis.analysisComponent` contains most classes of the packages `kieker.analysis.plugin` and `kieker.analysis.model.analysisMetaModel`. However, the classes `kieker.analysis.plugin.filter.select.TimestampFilter`, `kieker.analysis.plugin.filter.select.TypeFilter` and `kieker.analysis.model.analysisMetaModel.util.AnalysisMetaModelSwitch` are not members of this package [Yan26], which can be verified when looking at Listing B.4. This observation leads him to re-think the compositional choices regarding the `kieker.analysis.plugin.filter.select` package [Yan26].

According to Yang in optimization-0 of run 2026-07-16-14-12-44 Job 02, which uses Fan-in Coupling, PullThroughputMetric and #PaFs as fitness functions (Table 5.1), one pipe was split especially often [Yan26], see Listing B.4 for further details regarding that pipe. With a pipe connecting two ports [WEH14], this can be seen by the fact that the same two filters have many connected ports between each other. Yang further mentioned that this could hint to a "[...] substantial amount of parallelism through this pipe [...]" [Yan26]. However, it needs to be considered that, for these optimizations only generic throughput

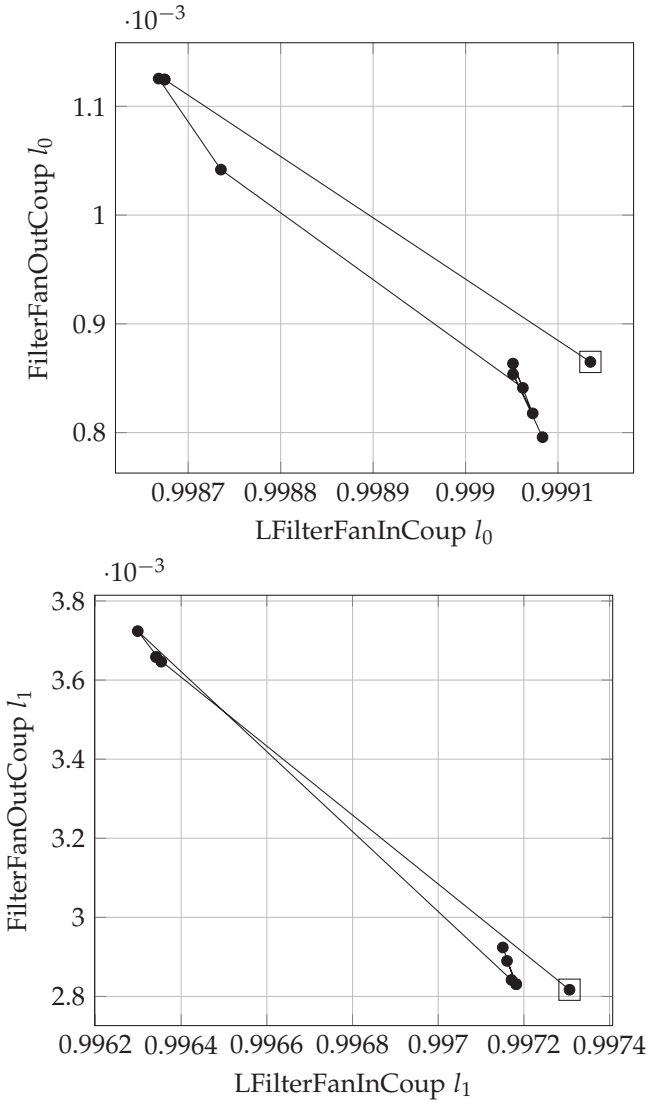

Figure 5.6. Values for an excerpt of 9 of 1731 optimizations and the original state for the Metrics-set (Lack of Fan-in Coupling, Fan-out Coupling) of run 2026-07-16-14-12-44 Table B.14, Table B.15

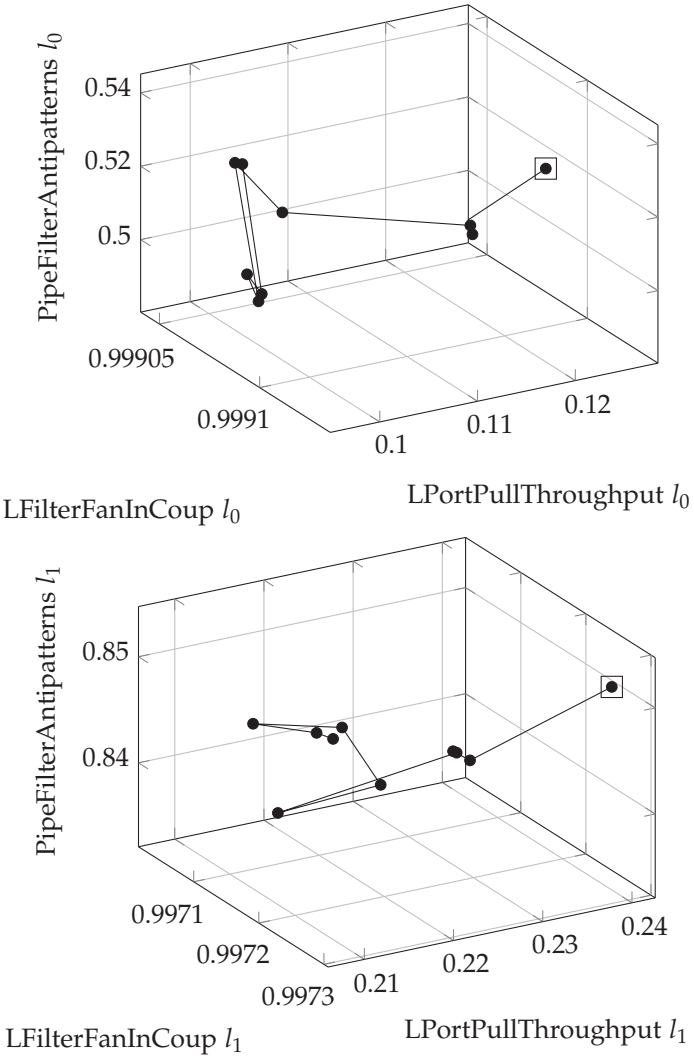

Figure 5.7. Values for an excerpt of 9 of 1731 optimizations for the Metrics-set (Lack of Fan-in Coupling, Lack of PullThroughputMetric, #PaFs) of run 2026-07-16-14-12-44 Table B.16, Table B.17

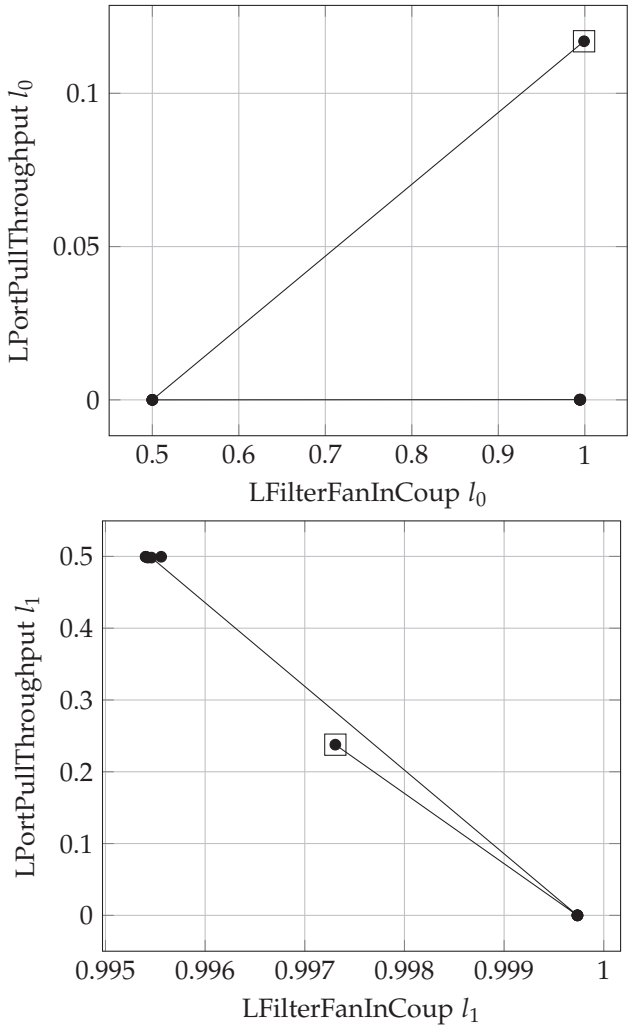

Figure 5.8. Values for an excerpt of 9 of 1427 optimizations and the original state for the Metrics-set (Lack of Fan-in Coupling, Lack of PullThroughputMetric) of run 2026-07-16-14-12-44 Table B.18, Table B.19

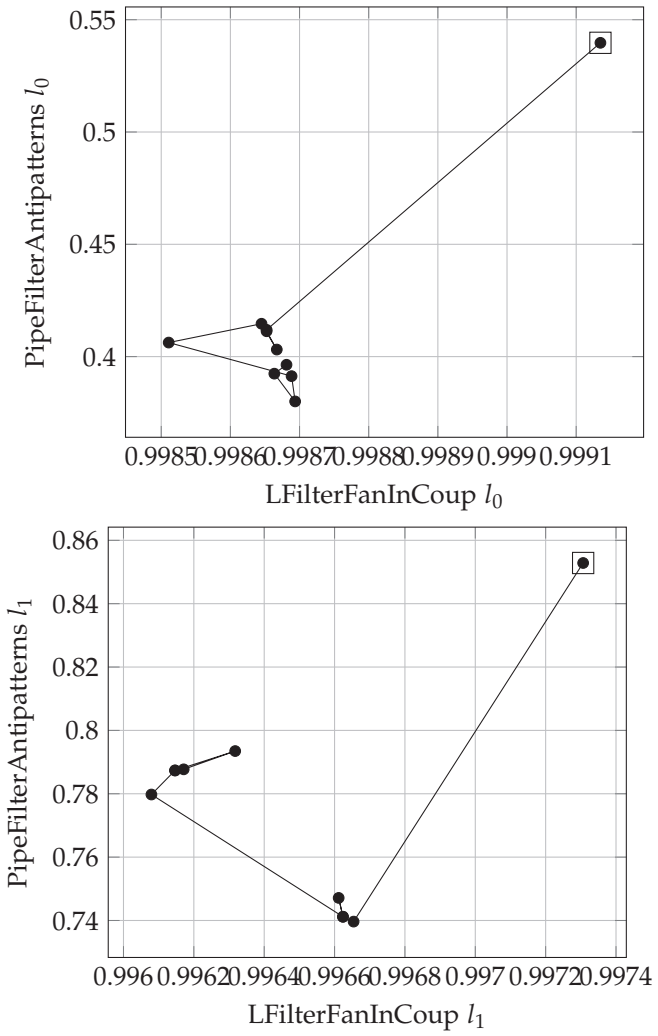

Figure 5.9. Values for an excerpt of 9 of 1137 optimizations and the original state for the Metrics-set (Lack of Fan-in Coupling, #PaFs) of run 2026-07-16-14-12-44 Table B.20, Table B.21

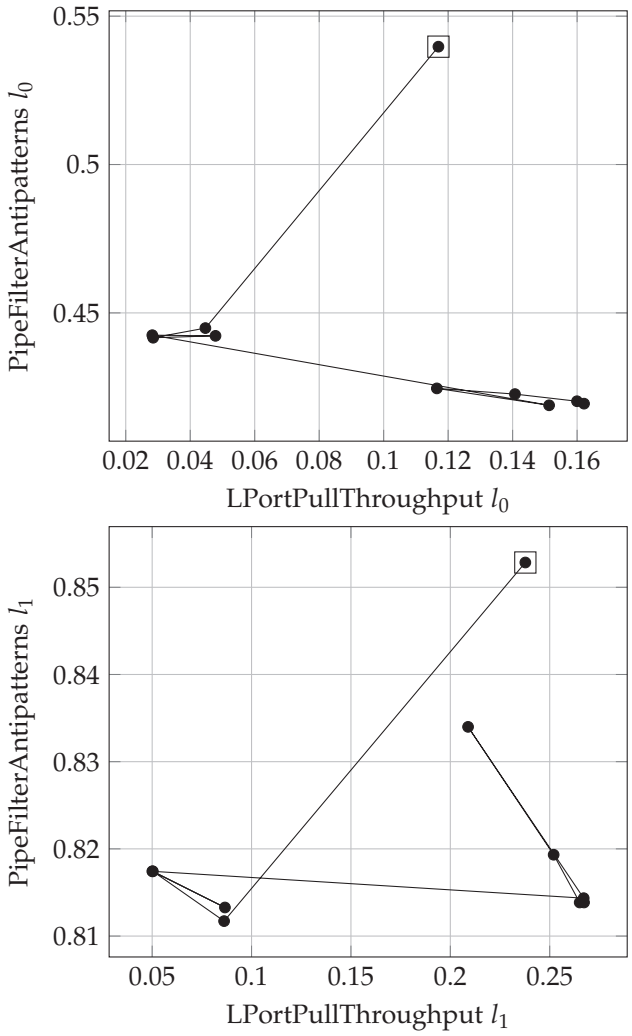

Figure 5.10. Values for an excerpt of 9 of 1009 optimizations and the original state for the Metrics-set (Lack of PullThroughputMetric, #PaFs) of run 2026-07-21-18-37-02 Table B.22, Table B.23

values have been used, which is both mentioned in Section 5.2 and the approximation of the throughput values for each pipe is further discussed in Figure 4.3. Hence, this result does not provide information about the actual amount of throughput of that specific pipe.

Therefore, both the optimization of package re-modularization and *pipe-and-filter*-specific re-modularization provide useful results for the developer.

Chapter 6

Conclusion

We provide a NSGA-II-framework with a *pipe-and-filter*-specific environment as discussed in Section 4.1 to provide optimization suggestions by mutating pipes and filters.

The set of metrics containing only cohesion and coupling has a higher tendency towards metrics-wise optimal solutions than the proposed *pipe-and-filter*-specific sets of metrics. One possible cause is that the analysis part of *Kieker* has already got a better package structure regarding cohesion and coupling than *pipe-and-filter*-structure regarding the in Chapter 3 discussed metrics. Regarding the minimal mean re-factoring effort the set of metrics (Fan-in coupling, PullThroughputMetric) provides the best results. However, all in Chapter 5 mentioned limitations need to be considered.

The developer feedback, discussed in Section 5.3.3, proves that both approaches produce useful results for developers. Thus, there are *pipe-and-filter*-specific metrics, which do complement cohesion and coupling metrics in providing satisfying re-modularization suggestions for the *pipe-and-filter*-architecture using Evolutionary Search.

Chapter 7

Prospects and Future Work

In Section 3.1.1 we have ignored possible edge-weights as well as possible parallel-edges. Thus, one prospect is to discuss weighted coupling and cohesion metrics as well. Also, it could be of interest to discuss the actual theoretically fitness of both cohesion and coupling to be a fitness function, despise being already used as one, regarding the proposed properties by Harman and Clark. Another aspect to examine is the influence on the quality of the provided optimization suggestions of inheritance and polymorphism on coupling metrics.

Another prospect is to either implement the TKR-NSGA-II or the NSGA-III to ensure the filtering of the solutions in order to find solutions, which are of interest to the user, as discussed in Section 4.4. As well as, comparing different approaches of filtering.

Currently, we assign to each port a generic processed data type and the same generic throughput value, as discussed in Section 4.2. In the future, we can parse the actual processed data types and the actual throughput values and re-run our experiments to be able to compare, whether the actual values produce better optimization suggestions than the artificial ones. Therefore, it is of special interest to use *Kieker* to monitor *Kieker* analyzing a software as dynamic analysis. However, which software is analyzed probably has got a direct effect on the produced monitoring results and a transitive effect on our optimization results. This assumption also needs to be investigated. As well as researching different approaches to approximate the throughput for solutions could have an impact on the quality of the results.

Of interest could also be to use global aggregated metrics as fitness functions instead of local fitness functions for each layer and compare the corresponding results to check, which approach produces better results, especially regarding the in Section 2.1 discussed problems of NSGA-II with more than three objectives.

Additionally, to the currently used *pipe-and-filter*-specific mutation-operations, discussed in Section 4.1.2, can also recombination-operations, the move-operation of ports and move operations on methods be examined, whether those are useful and leading to a broader search space. This includes experimenting with the *load balancing* operation as described by Hirzel et al. in [Hir+14].

Yet another future work could be to research additionally metrics for either *pipe-and-filter* or different architectures, e.g., microservices.

We discussed in Section 5.3, which sets are producing few minimal mean re-factoring effort. One next step could be to measure the actual re-factoring effort and to further

discuss and research different metrics for the re-factoring effort, especially as fitness function. Including the research question whether the optimization goal is a maximum or minimum re-factoring effort.

During this thesis we only have the feedback of one *Kieker* developer, which is further evaluated in Section 5.3.3. It is interesting to us to have more feedback of either more *Kieker* developers or more feedback on different optimizations.

Each one of our experiments in Section 5.1 has the same parameters, like mutations, recombinations ratios and population size. It needs to be further examined, whether different ratios and population sizes are leading to different observations or not.

Also, if the mentioned exceptions in Section 5.2 are fixed in the future, then the corresponding results can be discussed.

Appendix A

Used Tools

Table A.1 contains all tools, which were used during writing this thesis and the corresponding code base, but the actual used version of a tool can be different during the time period of this thesis. All suggested results have been revised.

In ChatGPT a project for this thesis was used. Inside a project ChatGPT should be able to connect all conversations and files and give more differentiated answers. However, ChatGPT also connects conversation from another project to this project leading to answers focusing on another context and thus the projects are not isolated. This mix-up in conversations is sometimes misleading.

Example (start) prompts solving the mentioned use cases for the listed AI-tools in Table A.1 are listed in Listing A.1. Each line is either one single prompt or the concatenation of multiple succeeding prompts. They are either real prompts or generic ones. Additional prompts can differ accordingly to their use case. Most of the time these prompts have resulted in conversations with the AI-tool, including follow-up questions or the correction of answers.

Disclaimer: These are only example prompts and neither a transcript nor the whole set of used prompts, i.e., many in-line GitHub Copilot prompts are missing.

Listing A.1. Example Prompts

```
1 Non-domination in NSGA. What is a "rank" in this context? Explain the crowding
      distance of a solution. Why would many solution become equivalent according
      to the Pareto dominance (Mkaouer et. Al., 2015) if there are more than three
       objectives?
2 Are there software metrics specific for the pipe-and-filter architecture? Please
       provide your sources for the structural metrics.
3 I have the following LaTeX/Java/... problem. Why is that and how do I fix it?
      The problem still exist. I have already tried ...
4 Explain why knee regions in the Pareto-Front are characterised by the fact that
      small improvements in one direction lead to deterioration in another
      direction.
5 Doesn't Pareto-optimal mean that solution a is non-dominated and thus all other
      solutions are at best as good as a and worse in one objective? Why would
      improving one objective in a lead to a deterioration in another objective?
6 Are we interested in finding the solutions with the largest minimal trade-off
```

Table A.1. Used Tools

Name	Version	Description	Use Case
BCEL [a]	6.12.0	Byte Code Engineering Library by Apache Commons	Static analysis of java byte code
ChatGPT [b]	1.2026.184	AI Chatbot from OpenAI	Information retrieval, brainstorming and discussion/explanation of topics, summary and explanation of literature, aid with best practice of citations
Gantt Chart Maker [c]	Updated on 2026, March 24	Free Online Gantt Chart Maker	Creating Gantt Charts
GitHub Copilot Chat [d]	v0.62.0	Microsoft's AI pair programmer extension for Visual Studio Code	Java, LaTeX and Shell/Bash/AWK code generation, debugging and explanation
Docker Desktop [e]	4.76.0	Docker's containerization software	Using containers to run software
IntelliJ IDEA [f]	2026.2.1 Ultimate	JetBrains IDE for developing Java and Kotlin	analyzing of code, e.g., creating a callgraph or profiling
Kieker Observability Framework [g]	2.0.3	An observability framework provided by the, among others, the AG Software Engineering	Dynamic and static analysis of software
Visual Studio Code [h]	1.134.0	Text editor and IDE with extensions	Java code and TeX writing, Spell- and Grammar-checking
Zotero [i]	9.0.6	Open Source Research Assistant	Organize, store, search and annotate papers, generating BibTeX file

[a] https://commons.apache.org/proper/commons-bcel/
[b] https://chatgpt.com/
[c] https://ganttchartmaker.com/
[d] https://github.com/features/copilot
[e] https://www.docker.com/products/docker-desktop/
[f] https://www.jetbrains.com/idea/
[g] https://kieker-monitoring.net/
[h] https://code.visualstudio.com/
[i] https://www.zotero.org/

```
    regarding the Pareto-optimal set?
7   What is a possible fitness function for cohesion and coupling?
8   Are max(Cohesion) and min(Coupling) continuos?
9   So for NSGA-2 two fitness functions instead of one are used, correct?
10  I have the following figure in source A which I want to adapt in the following
        way. How do I cite source A in the caption correctly?
11  I have two classes: A being an abstract class and B being a child class of A.
        Now A has got a method evaluate() and uses the method normalize(). In B I
        override normalize(). When calling B.evaluate() which version of the
        normalize-function will be used? I assume B.
12  Explain why we divide through n^2-n in the coupling factor with n being the set
        of classes.
13  But with n=2 and excluding self-loops we only have 1 possible edge, no?
        Meanwhile n^2-n results in 2 maximal edges.
14  Hence is n(n-1) the number of possible edges in a directed graph?
15  Is there a way to get the call graph of a whole package in IntelliJ also
        containing the number of calls or even the throughput produced through a
        method call?
16  I have an sh-script with the following line: rm sub-dir/* -rf. The corresponding
         directory exists, however I get: No such file or directory. Why is that so?
17  When to use a paraphrase and when to use a verbatim quotation? Best practice.
18  Please explain throughput and where is the difference between mbit/s and bps?
19  I get the following exception when performing X. Why is that and what are
        possible next steps?
20  Docker can't write to bind-mount even though the host directory has permissions
        777.
21  java.lang.NullPointerException: Cannot invoke "org.csveed.api.Header.size()"
        because "header" is null.
22  Merge both given tikzfigures into one.
23  I want to use make on Ubuntu 24 and I get:
24  make[4]: *** No rule to make target 'Makefile.am', needed by 'Makefile.in'.
        Stop. And how do I fix this: dpkg-checkbuilddeps: error: Unmet build
        dependencies: debhelper (>= 9). dpkg-genbuildinfo: error: binary build with
        no binary artifacts found; .buildinfo is meaningless. autoreconf: error: '
        configure.ac' is required. Many directories seem to be missing. I'm using
        debuild -us -uc.
25  Take a file-path as argument and store it in a variable.
26  I want to iterate over a fix list of Strings in the given sh-file.
27  Iterate over the resulting output in line 18, iterate over the lines and store
        the child. Every line looks like: [superclass chain] child -> parent1 -> ...
         -> parentn
```

```
28 Store the package in another variable, i.e. A.B.C.child then tool=A.B.C
29 Now I want the script to create a new file outputDir/originalMod.mod and write
       each pair in a new line like: (child, tool)
30 I want to search for each child, whether in the sourceCode is either
       createOutputPort or createInputPort called and if so I want to create an
       entry like (outputport, child) or (inputport, child). If there are multiple
       ports, I want to count them child-wise and store the results in the
       beginning of the output file.
31 Search in the input file for the both child -> teetime.framework.InputPort and
       child -> teetime.framework.OutputPort and count for each call the ports
       child-wise.
32 Make this more efficient: I want to grep both OutputPort and InputPort and every
        time we encounter either of them, I want to create a new entry with a
       corresponding counter value instead of counting the occurrences first.
       Adjust this \st entry is written for each port, i.e. for outputPort=2: (
       outputport1, child), (outputport2, child)
33 Ensure that there are no duplicates in the resulting file.
34 I'm using the BCEL library for Java, now I want to receive a trace of method
       calls like: A -> B -> C -> D. How do I do this in the best way?
35 I have an own sub-class of NotifyingMethodVisitor and there I use two queues:
       nextClasses, visitedClasses. How do I give these to the InstructionHandle
       instance?
36 Let visitMethod use the nextClasses queue and visit the nextClasses first,
       before the next element in the handle, iff the queue is not empty.
37 It is probably the best way to store the next instruction in i to process it
       correctly. Is it possible to receive the corresponding instruction to the
       next String in the queue and store that in i?
38 Now add that we don't want to re-visit already visited classes.
39 Filter all files for the String "connectPorts(A,B)" with A and B being specific
       ports. Then store both A and B in variables. Keep a counter for each class-
       instance regarding input and output-ports and if you encounter a port
       multiple times alter the port name like: A.outputPort0, A.outputPort1.
40 As port name use the class and package of the instance if possible and start the
        portCount with 0 \st every port has got a number.
41 Use the corresponding import as class name.
42 Like shown in this example: add for each port exactly one entry in the port-
       section: (name, default-throughput, type, direction)
43 All ports shall get the same default-throughput and connected ports shall get
       the same type, \eg type1.
44 For some ports the classes are incorrect and methods are used instead, e.g
       kieker.analysis.generic.sink.graph.graphml.getInputPort1().inputPort0 should
```

```
     be kieker.analysis.generic.sink.graph.graphml.GraphMLWriterStage.inputPort0
    , and strip generic classes of the <>.
45  Taking the two given scripts create a third bash script which generates the
    originalMod.mod, but by parsing the source code instead of the log-files.
46  Awk: Does this return a concatenated String: A "." B?
47  Explain the given line/code block/equation or text.
48  Expression to check, whether a string contains a either / or ; as substring.
49  Awk: Why is portSeen[portName] always empty, even if we encounter the same
    portName twice? <Code-block> portSeen is only used in one method. Does that
    mean a new instance is created every call?
50  What is wrong with the given source-code?
51  The array does not seem to grow after a n entries, even though there should be
    more.
52  How do I access the package belonging to an instance? \eg variableA is declared
    in class A. Generate corresponding functions, i.e. extract_class_name.
53  Adjust the following function to use the package and class where the instance
    has been declared instead of the actual type of the instance: <Code-block>
54  Regex for awk to check, whether a string contains "..". Is this correct? str ~
    /[\/;]/\.{2,}/
55  How do I specify the style of a node in tikz? How do I set the following options
    ? How do I draw an edge as following?
56  Generate a shell or bash script, which takes a file as input. This file contains
     n entries. The number of entries is written in the first line and each
    entry is introduced by a line containing "##" as sub-string. The output
    shall be n files, one for each entry in the same directory as the given
    input-file. The first line containing the number of entries looks like this:
     "x n y."
57  I have the following case, generate a shell script or awk snippet to solve it.
    The following problem occurs, solve it.
58  Assuming there is one optimal solution and I use evolutionary search on the
    problem. Would I converge to that one solution or always filling up the next
     population with dominated solutions?
59  LaTeX: lstlistOflistings and listofalgorithms are currently introducing an
    additional empty page. I do not want that.
60  I have a CSV file with 16 columns: 8 objectives and two columns per objective.
    Give me an awk command to generate 8 csv files instead, one for each
    objective. Objectives: CouplingFactor(flat -> i) and LNormalizedCohesion(
    flat -> i). The first column of my file contains the name of the
    optimization. This shall be in each file. Re-direct the resulting files in
    the same directory as the input file. I want to use a variable for the
    number of files. Make the number of columns customisable, too, \eg one file
```

```
    has 3 objectives for each file.
Take the following source-block and write it as lambda function.
/explain The following problem or line.
Fix why this does not compile because of casting from ProgramUnit to Node.
Please make it possible for liftFlatMetric to accept IFitness<Modularization<P>>
     with P extending ProgramUnit.
I want to have a liftFlatMetrics method both for ProgramUnit and Node, while
    Node extends ProgramUnit. Is there a safe solution? I would prefer to keep
    one liftFlatMetric() to reduce the need to check which method has to be
    called. It does not have to be completely type-safe. Re-write the
    liftFlatMetric as we talked about. I want to change the liftFlatMetric to
    expect a Node.
Will the other instance even created, if the value is present? Because I'm
    calling the constructor even though the Optional is not empty.
Generate a bash script which takes a file as input which contains:
------------ (0,1)
(x, y)
...
------------ (1,2)
Return the average number of fresh ports, the first element of the pairs
    contains "fresh" as substring, using the following equation: average = #
    numOfFreshPorts / (#numOfOptimizations times max{#numOfPorts}) with #
    numOfOptimizations=given in the first line of the file, #numOfPorts=the
    number of lines between --------- (0,1) and --------- (0,2). The first line
    contains characters and one integer. That integer is the number of
    optimizations. Also re-direct the results in an output-file as CSV-file by
    concatenating to the already existing file. Use a given name as indicator
    for the corresponding line.
The number of fresh ports is counted incorrectly: There are more freshports than
     maxPorts times optimizations. Fix the script accordingly. The
    separatorCounter needs to be reset each time a new optimization starts with
    a leading "##".
The optimization introduced by "## original" is the original state. Additionally
     to the average number of fresh ports, count the number of ports of this
    original state and calculate the average number of added ports in comparison
     to the original state per optimization.
This script does not seem to count the number of originalPorts correctly. I'm
    resulting in 0.
Given a file in the tuple specification like following containing multiple
    optimizations including the original modularization:
## Optimization n
```

```
78  ------- (x,y)
79  (unit_j, unit_k)
80  ...
81  Generate me a shell script which (1) calculates the average number of moved
        units in comparison to the original state and (2) removed units. Meaning
        that the second element differs, or the first element does not exist anymore
         in one optimization. Re-direct the results in a csv-file. The original
        state is the last entry in the file. Adjust the suggested script accordingly
        .
82  How is a graph called, which contains nodes inside nodes?
83  Given the following case, what is the best practice to cite the corresponding
        sources? Is something like ... fine or is there a better way to phrase it?
84  I want use a unique marking for each point which belongs to the row "original-
        state" in the CSV-Files of the plotted figures. Only that one point not all.
         No it highlights all. Still wrong.
85  Put all subfigures in rows underneath each other instead of next to each other
        and center them.
86  The x values of figure fig:metrics20260716141244job4 overlap but I need the
        precision. Raise the width of the grids without adding more x-labels.
```

Appendix B

Run Result Files

This section contains all referenced excerpts of the results.

B.1 Package Structure Re-Modularization Runs

The whole files for each run can be found on https://cau-git.rz.uni-kiel.de/ifi-ag-se/abschlussarbeiten/natalie-kaufhold-msc-2026-multilevel-metrics/-/tree/87156df29d456f8ebc31a747955ef763a37b89b6/optimization-runs/coupling-runs.

Listing B.1. Excerpt of the optimizations.txt: Original Modularization LayerGraphMap and layer 0

```
1  ## original multi-level modularization
2  ...
3  LayerGraphMap: {0=cau.agse.hs.genopt.flatcoupling.model.CouplingGraph@588e0fb6
       , 1=cau.agse.hs.genopt.flatcoupling.model.CouplingGraph@59abb689, 2=cau.
       agse.hs.genopt.flatcoupling.model.CouplingGraph@5982ac86, 3=cau.agse.hs.
       genopt.flatcoupling.model.CouplingGraph@46f0b849, 4=cau.agse.hs.genopt.
       flatcoupling.model.CouplingGraph@30727f43, 5=cau.agse.hs.genopt.
       flatcoupling.model.CouplingGraph@1dc116a5, 6=cau.agse.hs.genopt.
       flatcoupling.model.CouplingGraph@478dffd0}
4  ...
5  ------------ (0,1)
6  (kieker.analysis.plugin.filter.forward.util.KiekerHashMap, kieker.analysis.
       plugin.filter.forward.util)
7  ------------ (1,2)
8  ...
```

Listing B.2. Excerpt of the optimizations.txt: Optimization-0 Layers

```
1  ------------ (0,1)
2 (kieker.analysis.model.analysisMetaModel.util.AnalysisMetaModelAdapterFactory, 926
       layer4537layer3kieker.analysis.model.analysisMetaModel.util)
3 (kieker.analysis.plugin.filter.record.MonitoringThroughputFilter, 909layer1kieker.
       analysis.plugin.filter.record)
```

```
4  (kieker.analysis.plugin.reader.jms.JMSReader, 758layer4kieker.analysis.plugin.
       reader.jms)
5  (kieker.analysis.plugin.reader.jmx.JMXReader, 758layer2112layer1kieker.analysis.
       plugin.reader.jmx)
6  (kieker.analysis.plugin.reader.list.ListReader, 753layer4kieker.analysis.plugin.
       reader.list)
7  (kieker.analysis.plugin.reader.tcp.TCPReader, kieker.analysis.plugin.reader.tcp)
8  (kieker.analysis.plugin.reader.timer.TimeReader, 45layer5kieker.analysis.plugin.
       reader.timer)
9      ------------ (1,2)
10 (45layer5kieker.analysis.plugin.reader.timer, 616layer1kieker.analysis.
       analysisComponent)
11 (753layer4kieker.analysis.plugin.reader.list, 616layer1kieker.analysis.
       analysisComponent)
12 (758layer2112layer1kieker.analysis.plugin.reader.jmx, 616layer1kieker.analysis.
       analysisComponent)
13 (758layer4kieker.analysis.plugin.reader.jms, 616layer1kieker.analysis.
       analysisComponent)
14 (909layer1kieker.analysis.plugin.filter.record, 616layer1kieker.analysis.
       analysisComponent)
15 (926layer4537layer3kieker.analysis.model.analysisMetaModel.util, 616layer1kieker.
       analysis.analysisComponent)
16 (kieker.analysis.AnalysisController, 616layer1kieker.analysis.analysisComponent)
17 (kieker.analysis.IAnalysisController, 380layer321layer3kieker.analysis)
18 (kieker.analysis.IProjectContext, 616layer1kieker.analysis.analysisComponent)
19 (kieker.analysis.analysisComponent.AbstractAnalysisComponent, 616layer1kieker.
       analysis.analysisComponent)
20 (kieker.analysis.analysisComponent.IAnalysisComponent, 616layer1kieker.analysis.
       analysisComponent)
21 (kieker.analysis.configuration.GlobalConfigurationRegistry, 616layer1kieker.
       analysis.analysisComponent)
22 (kieker.analysis.display.AbstractDisplay, 616layer1kieker.analysis.
       analysisComponent)
23 (kieker.analysis.display.HtmlText, 616layer1kieker.analysis.analysisComponent)
24 (kieker.analysis.display.Image, 616layer1kieker.analysis.analysisComponent)
25 (kieker.analysis.display.TagCloud, 616layer1kieker.analysis.analysisComponent)
26 (kieker.analysis.exception.AnalysisConfigurationException, 616layer1kieker.
       analysis.analysisComponent)
27 (kieker.analysis.exception.InvalidProjectContextException, 111layer321layer3kieker
       .analysis)
28 (kieker.analysis.exception.PluginNotFoundException, 616layer1kieker.analysis.
```

```
    analysisComponent)
29 (kieker.analysis.model.MetaModelHandler, kieker.analysis.model)
30 (kieker.analysis.model.analysisMetaModel.MIAnalysisComponent, 616layer1kieker.
    analysis.analysisComponent)
31 (kieker.analysis.model.analysisMetaModel.MIAnalysisMetaModelFactory, 616
    layer1kieker.analysis.analysisComponent)
32 (kieker.analysis.model.analysisMetaModel.MIDependency, 616layer1kieker.analysis.
    analysisComponent)
33 (kieker.analysis.model.analysisMetaModel.MIDisplay, 616layer1kieker.analysis.
    analysisComponent)
34 (kieker.analysis.model.analysisMetaModel.MIFilter, 616layer1kieker.analysis.
    analysisComponent)
35 (kieker.analysis.model.analysisMetaModel.MIInputPort, 616layer1kieker.analysis.
    analysisComponent)
36 (kieker.analysis.model.analysisMetaModel.MIProject, 616layer1kieker.analysis.
    analysisComponent)
37 (kieker.analysis.model.analysisMetaModel.MIProperty, 616layer1kieker.analysis.
    analysisComponent)
38 (kieker.analysis.model.analysisMetaModel.MIRepositoryConnector, 616layer1kieker.
    analysis.analysisComponent)
39 (kieker.analysis.model.analysisMetaModel.impl.MAnalysisComponent, 616layer1kieker.
    analysis.analysisComponent)
40 (kieker.analysis.model.analysisMetaModel.impl.MAnalysisMetaModelFactory, 616
    layer1kieker.analysis.analysisComponent)
41 (kieker.analysis.model.analysisMetaModel.impl.MAnalysisMetaModelPackage, 616
    layer1kieker.analysis.analysisComponent)
42 (kieker.analysis.model.analysisMetaModel.impl.MDependency, 616layer1kieker.
    analysis.analysisComponent)
43 (kieker.analysis.model.analysisMetaModel.impl.MDisplay, 616layer1kieker.analysis.
    analysisComponent)
44 (kieker.analysis.model.analysisMetaModel.impl.MDisplayConnector, 616layer1kieker.
    analysis.analysisComponent)
45 (kieker.analysis.model.analysisMetaModel.impl.MFilter, 616layer1kieker.analysis.
    analysisComponent)
46 (kieker.analysis.model.analysisMetaModel.impl.MInputPort, 616layer1kieker.analysis
    .analysisComponent)
47 (kieker.analysis.model.analysisMetaModel.impl.MOutputPort, 616layer1kieker.
    analysis.analysisComponent)
48 (kieker.analysis.model.analysisMetaModel.impl.MPlugin, 616layer1kieker.analysis.
    analysisComponent)
49 (kieker.analysis.model.analysisMetaModel.impl.MPort, 616layer1kieker.analysis.
```

```
    analysisComponent)
50 (kieker.analysis.model.analysisMetaModel.impl.MProject, 616layer1kieker.analysis.
    analysisComponent)
51 (kieker.analysis.model.analysisMetaModel.impl.MProperty, 616layer1kieker.analysis.
    analysisComponent)
52 (kieker.analysis.model.analysisMetaModel.impl.MReader, 616layer1kieker.analysis.
    analysisComponent)
53 (kieker.analysis.model.analysisMetaModel.impl.MRepository, 616layer1kieker.
    analysis.analysisComponent)
54 (kieker.analysis.model.analysisMetaModel.impl.MRepositoryConnector, 616
    layer1kieker.analysis.analysisComponent)
55 (kieker.analysis.model.analysisMetaModel.impl.MView, 616layer1kieker.analysis.
    analysisComponent)
56 (kieker.analysis.model.analysisMetaModel.util.AnalysisMetaModelSwitch, kieker.
    analysis.model.analysisMetaModel.util)
57 (kieker.analysis.plugin.AbstractPlugin, 616layer1kieker.analysis.analysisComponent
    )
58 (kieker.analysis.plugin.AbstractUpdateableFilterPlugin, 616layer1kieker.analysis.
    analysisComponent)
59 (kieker.analysis.plugin.IPlugin, 616layer1kieker.analysis.analysisComponent)
60 (kieker.analysis.plugin.filter.IFilterPlugin, 616layer1kieker.analysis.
    analysisComponent)
61 (kieker.analysis.plugin.filter.flow.EventRecordTraceReconstructionFilter, 616
    layer1kieker.analysis.analysisComponent)
62 (kieker.analysis.plugin.filter.flow.TraceAggregationFilter, 616layer1kieker.
    analysis.analysisComponent)
63 (kieker.analysis.plugin.filter.flow.TraceEventRecords, 616layer1kieker.analysis.
    analysisComponent)
64 (kieker.analysis.plugin.filter.forward.AnalysisThroughputFilter, 616layer1kieker.
    analysis.analysisComponent)
65 (kieker.analysis.plugin.filter.forward.ListCollectionFilter, 616layer1kieker.
    analysis.analysisComponent)
66 (kieker.analysis.plugin.filter.forward.StringBufferFilter, 616layer1kieker.
    analysis.analysisComponent)
67 (kieker.analysis.plugin.filter.forward.TeeFilter, 616layer1kieker.analysis.
    analysisComponent)
68 (kieker.analysis.plugin.filter.forward.util.KiekerHashMap, 616layer1kieker.
    analysis.analysisComponent)
69 (kieker.analysis.plugin.filter.record.RealtimeRecordDelayFilter, 616layer1kieker.
    analysis.analysisComponent)
70 (kieker.analysis.plugin.filter.select.TimestampFilter, 590layer2kieker.analysis.
```

```
    plugin.filter.select)
71 (kieker.analysis.plugin.filter.select.TypeFilter, 149layer3kieker.analysis.plugin.
    filter.select)
72 (kieker.analysis.plugin.filter.sink.GCDisplayFilter, 616layer1kieker.analysis.
    analysisComponent)
73 (kieker.analysis.plugin.filter.visualization.AbstractWebVisualizationFilterPlugin,
     616layer1kieker.analysis.analysisComponent)
74 (kieker.analysis.plugin.filter.visualization.IWebVisualizationFilterPlugin, 616
    layer1kieker.analysis.analysisComponent)
75 (kieker.analysis.plugin.filter.visualization.PlainTextWebVisualization, 616
    layer1kieker.analysis.analysisComponent)
76 (kieker.analysis.plugin.reader.AbstractReaderPlugin, 616layer1kieker.analysis.
    analysisComponent)
77 (kieker.analysis.plugin.reader.IReaderPlugin, 616layer1kieker.analysis.
    analysisComponent)
78 (kieker.analysis.plugin.reader.amqp.AMQPReader, 616layer1kieker.analysis.
    analysisComponent)
79 (kieker.analysis.plugin.reader.amqp.RegistryRecordHandler, 616layer1kieker.
    analysis.analysisComponent)
80 (kieker.analysis.plugin.reader.amqp.RegularRecordHandler, 616layer1kieker.analysis
    .analysisComponent)
81 (kieker.analysis.plugin.reader.database.DbReader, 616layer1kieker.analysis.
    analysisComponent)
82 (kieker.analysis.plugin.reader.filesystem.FSReader, 616layer1kieker.analysis.
    analysisComponent)
83 (kieker.analysis.plugin.reader.filesystem.FSZipReader, 616layer1kieker.analysis.
    analysisComponent)
84 (kieker.analysis.plugin.reader.filesystem.IMonitoringRecordReceiver, 616
    layer1kieker.analysis.analysisComponent)
85 (kieker.analysis.plugin.reader.tcp, 616layer1kieker.analysis.analysisComponent)
86 (kieker.analysis.plugin.reader.tcp.TCPStringReader, 616layer1kieker.analysis.
    analysisComponent)
87 (kieker.analysis.repository.AbstractRepository, 438layer2kieker.analysis.
    repository)
88 (kieker.analysis.repository.IRepository, 438layer2kieker.analysis.repository)
89 ------------ (2,3)
90 (111layer321layer3kieker.analysis, kieker.analysis.plugin)
91 (149layer3kieker.analysis.plugin.filter.select, kieker.analysis.plugin)
92 (380layer321layer3kieker.analysis, kieker.analysis.plugin)
93 (438layer2kieker.analysis.repository, 414layer3kieker.analysis.plugin)
94 (590layer2kieker.analysis.plugin.filter.select, kieker.analysis.plugin)
```

```
95 (616layer1kieker.analysis.analysisComponent, kieker.analysis.plugin)
96 (kieker.analysis.AnalysisControllerThread, kieker.analysis.plugin)
97 (kieker.analysis.display.MeterGauge, kieker.analysis.plugin)
98 (kieker.analysis.display.PieChart, kieker.analysis.plugin)
99 (kieker.analysis.display.PlainText, kieker.analysis.plugin)
100 (kieker.analysis.display.XYPlot, kieker.analysis.plugin)
101 (kieker.analysis.model, 228layer2197layer2540layer5kieker.analysis)
102 (kieker.analysis.model.analysisMetaModel.MIAnalysisMetaModelPackage, kieker.
    analysis.plugin)
103 (kieker.analysis.model.analysisMetaModel.MIDisplayConnector, kieker.analysis.
    plugin)
104 (kieker.analysis.model.analysisMetaModel.MIOutputPort, kieker.analysis.plugin)
105 (kieker.analysis.model.analysisMetaModel.MIPlugin, kieker.analysis.plugin)
106 (kieker.analysis.model.analysisMetaModel.MIPort, kieker.analysis.plugin)
107 (kieker.analysis.model.analysisMetaModel.MIReader, kieker.analysis.plugin)
108 (kieker.analysis.model.analysisMetaModel.MIRepository, kieker.analysis.plugin)
109 (kieker.analysis.model.analysisMetaModel.MIView, kieker.analysis.plugin)
110 (kieker.analysis.model.analysisMetaModel.util, kieker.analysis.plugin)
111 (kieker.analysis.plugin.filter.AbstractFilterPlugin, kieker.analysis.plugin)
112 (kieker.analysis.plugin.filter.forward.CountingFilter, kieker.analysis.plugin)
113 (kieker.analysis.plugin.filter.select.TraceIdFilter, kieker.analysis.plugin)
114 (kieker.analysis.plugin.filter.sink.CPUUtilizationDisplayFilter, kieker.analysis.
    plugin)
115 (kieker.analysis.plugin.filter.sink.MemSwapUtilizationDisplayFilter, kieker.
    analysis.plugin)
116 (kieker.analysis.plugin.filter.sink.MethodAndComponentFlowDisplayFilter, kieker.
    analysis.plugin)
117 (kieker.analysis.plugin.filter.sink.ThreadsStatusDisplayFilter, kieker.analysis.
    plugin)
118 (kieker.analysis.plugin.reader.filesystem.FSDirectoryReader, kieker.analysis.
    plugin)
119 (kieker.analysis.plugin.reader.namedRecordPipe.PipeReader, kieker.analysis.plugin)
```

Listing B.3. Excerpt of the optimizations.txt: Optimization 1 LayerGraphMap, Layer 0 and an excerpt of Layer 1

```
1    ## optimization result 1
2    ...
3    LayerGraphMap: {0=cau.agse.hs.genopt.flatcoupling.model.CouplingGraph@6cd28a38
         , 1=cau.agse.hs.genopt.flatcoupling.model.CouplingGraph@51a0147e, 2=cau.
         agse.hs.genopt.flatcoupling.model.CouplingGraph@5ed6b1ce, 3=cau.agse.hs.
         genopt.flatcoupling.model.CouplingGraph@1ddac70a, 4=cau.agse.hs.genopt.
```

Table B.1. The average number of changed units per optimization in run 2026-07-14-14-32-26

optimizations	averageMovedUnits	averageRemovedUnits	averageChangedUnits
1000	87.80	20.02	107.82

Table B.2. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 0

optimization	CouplingFactor(flat -> 0)	LNormalizedCohesion(flat -> 0)
original-state	0.0	0.0
optimization-0	0.0	0.0
optimization-1	0.0	0.0
optimization-2	0.02027027027027027	0.08366013071895426
optimization-3	0.0	0.0625
optimization-995	0.08095238095238096	0.24166666666666667
optimization-996	0.0	0.0
optimization-997	0.26666666666666666	0.1
optimization-998	0.04329004329004329	0.30992063492063493
optimization-999	0.0	0.0

```
        flatcoupling.model.CouplingGraph@6885561f}
4   ...
5   ------------ (0,1)
6   (kieker.analysis.IProjectContext, 263layer1237layer1kieker)
7   (kieker.analysis.exception.PluginNotFoundException, 804layer1784layer1kieker.
        analysis)
8   ------------ (1,2)
9   (263layer1237layer1kieker, none)
10  ...
```

B.2 P&F Optimization Runs

The whole files for each run can be found on https://cau-git.rz.uni-kiel.de/ifi-ag-se/abschlussarbeiten/natalie-kaufhold-msc-2026-multilevel-metrics/-/tree/4a7c4f33c61aba4c0da69697a4bfc90b80a28f30/optimization-runs/pipe-filter-runs.

Listing B.4. Excerpt of the optimization.txt of 2026-07-16-14-12-44 Job 02: Optimization-0

```
1   ------------ (CALLER,CALLEE)
2   ...
3   kieker.tools.trace.analysis.TraceAnalysisConfiguration.
        TraceEventRecords2ExecutionAndMessageTraceStage.
```

Table B.3. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 1

optimization	CouplingFactor(flat -> 1)	LNormalizedCohesion(flat -> 1)
original-state	0.0	0.18532295271049593
optimization-0	0.004706264199935086	0.06091549295774648
optimization-1	0.01847290640394089	0.030612244897959183
optimization-2	0.01990740740740741	0.21791392232568704
optimization-3	0.004955570745044429	0.06084825234441603
optimization-995	0.3194444444444444	0.027777777777777776
optimization-996	0.0	0.13945578231292516
optimization-997	0.0	0.175
optimization-998	0.0	0.13324979114452798
optimization-999	0.0	0.11904761904761904

Table B.4. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 2

optimization	CouplingFactor(flat -> 2)	LNormalizedCohesion(flat -> 2)
original-state	0.0	0.40561314903420165
optimization-0	0.01264367816091954	0.14991181657848324
optimization-1	5.120327700972862E-4	0.05925133689839572
optimization-2	0.0	0.05
optimization-3	0.004597701149425287	0.22598522167487683
optimization-995	0.008788159111933395	0.11144752018454442
optimization-996	0.016304347826086956	0.1919191919191919
optimization-997	0.004456327985739751	0.15244200244200246
optimization-998	0.015789473684210527	0.09428571428571428
optimization-999	0.02717391304347826	0.2884126984126984

Table B.5. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 3

optimization	CouplingFactor(flat -> 3)	LNormalizedCohesion(flat -> 3)
original-state	0.07114624505928854	0.15612244897959182
optimization-0	0.0	0.0
optimization-1	0.004032258064516129	0.22903225806451613
optimization-2	0.0	0.0
optimization-3	0.0	0.0
optimization-995	0.0037593984962406013	0.2383838383838384
optimization-996	0.0	0.125
optimization-997	0.02832512315270936	0.18916666666666665
optimization-998	0.0875	0.042328042328042326
optimization-999	0.1	0.0

Table B.6. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 4

optimization	CouplingFactor(flat -> 4)	LNormalizedCohesion(flat -> 4)
original-state	0.0	0.3181818181818182
optimization-0	0.0	0.0
optimization-1	0.0	0.0
optimization-2	0.0	0.0
optimization-3	0.0	0.0
optimization-995	0.0	0.0
optimization-996	0.004032258064516129	0.2141025641025641
optimization-997	0.07272727272727272	0.07777777777777778
optimization-998	0.011470985155195682	0.32100840336134456
optimization-999	0.005698005698005698	0.17783882783882782

Table B.7. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 5

optimization	CouplingFactor(flat -> 5)	LNormalizedCohesion(flat -> 5)
original-state	0.0	0.0
optimization-0	0.0	0.0
optimization-1	0.0	0.0
optimization-2	0.0	0.0
optimization-3	0.0	0.0
optimization-995	0.0	0.0
optimization-996	0.01876876876876877	0.1711996336996337
optimization-997	0.028528528528528527	0.23152412280701753
optimization-998	0.0	0.0
optimization-999	0.01693404634581105	0.25555555555555554

Table B.8. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 6

optimization	CouplingFactor(flat -> 6)	LNormalizedCohesion(flat -> 6)
original-state	0.0	0.0
optimization-0	0.0	0.0
optimization-1	0.0	0.0
optimization-2	0.0	0.0
optimization-3	0.0	0.0
optimization-995	0.0	0.0
optimization-996	0.038461538461538464	0.225
optimization-997	0.0	0.0
optimization-998	0.0	0.0
optimization-999	0.016666666666666666	0.06969696969696969

Table B.9. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 7

optimization	CouplingFactor(flat -> 7)	LNormalizedCohesion(flat -> 7)
original-state	0.0	0.0
optimization-0	0.0	0.0
optimization-1	0.0	0.0
optimization-2	0.0	0.0
optimization-3	0.0	0.0
optimization-995	0.0	0.0
optimization-996	0.19047619047619047	0.03333333333333333
optimization-997	0.0	0.0
optimization-998	0.0	0.0
optimization-999	0.10897435897435898	0.09047619047619047

Table B.10. Excerpt of Metrics.csv of run 2026-07-14-14-32-26 Job 01 Layer 8

optimization	CouplingFactor(flat -> 8)	LNormalizedCohesion(flat -> 8)
original-state	0.0	0.0
optimization-0	0.0	0.0
optimization-1	0.0	0.0
optimization-2	0.0	0.0
optimization-3	0.0	0.0
optimization-995	0.0	0.0
optimization-996	0.0	0.0
optimization-997	0.0	0.0
optimization-998	0.0	0.0
optimization-999	0.0	0.08333333333333333

Table B.11. Added Port Statistics of both July P&F Runs

Name	FreshPorts	AddedPorts	Optimizations	AveragePorts	AverageFreshPorts	AverageAddedPorts	MaxAddedPorts
job1	48459	777496	1732	449.22	27.98	448.90	561
job2	41662	775175	1428	543.23	29.18	542.84	594
job3	39020	65294	1489	44.22	26.21	43.85	134
job4	40835	430212	1138	378.53	35.88	378.04	475
job5	47580	518977	1010	514.39	47.11	513.84	590

Table B.12. Added Port Statistics of 2026-08-12-19-52-33 P&F Run

Name	FreshPorts	AddedPorts	Optimizations	AveragePorts	AverageFreshPorts	AverageAddedPorts	MaxAddedPorts
job01	26030	290077	429	677.46	60.68	676.17	706
job02	65671	593792	971	612.10	67.63	611.53	670
job03	502	1082	2	818.00	251.00	541.00	541
job04	55251	665702	1081	616.33	51.11	615.82	683
job05	15126	99760	113	887.73	133.86	882.83	946

Table B.13. Port Type Statistics of both July P&F Runs

Name	InputPorts	OutputPorts	Optimizations	AveragePorts	AverageInputPorts	AverageOutputPorts	OriginalInputPorts	OriginalOutputPorts
job01	337793	441194	1732	449.22	195.03	254.73	554	554
job02	350907	425907	1428	543.23	245.73	298.25	554	554
job03	30248	35621	1489	44.22	20.31	23.92	554	554
job04	180880	250471	1138	378.53	158.95	220.10	554	554
job05	222864	297346	1010	514.39	220.66	294.40	554	554

Table B.14. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 01 Layer 0

optimization	LFilterFanInCoup(flat -> 0)	FilterFanOutCoup(flat -> 0)
original-state	0.9991350102166717	8.649897833282196E-4
optimization-0	0.9986745391010965	0.0011246334899787122
optimization-1	0.9986745391010965	0.0011246334899787122
optimization-2	0.9986680518926982	0.00112549615066997
optimization-3	0.9987353663824252	0.0010418172182878065
optimization-1727	0.99906197863665	8.411040269153289E-4
optimization-1728	0.9990512738514018	8.634473936680524E-4
optimization-1729	0.9990726337202346	8.176777950620239E-4
optimization-1730	0.9990513734005102	8.537639395408647E-4
optimization-1731	0.999083269671505	7.95772854596384E-4

Table B.15. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 01 Layer 1

optimization	LFilterFanInCoup(flat -> 1)	FilterFanOutCoup(flat -> 1)
original-state	0.9973064577675024	0.00281699625148706
optimization-0	0.9963416350077389	0.003658364992261151
optimization-1	0.9963416350077389	0.003658364992261151
optimization-2	0.996353586027499	0.003646413972500929
optimization-3	0.9962988038817423	0.003723764387271496
optimization-1727	0.9971710189452125	0.0028417818740399388
optimization-1728	0.9971820377632912	0.002830948237799652
optimization-1729	0.9971605966532899	0.0028898816284293684
optimization-1730	0.9971820377632912	0.002830948237799652
optimization-1731	0.9971507490170706	0.002923903847110934

Table B.16. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 02 Layer 0

optimization	LFilterFanInCoup(flat -> 0)	LPortPullThroughput(flat -> 0)	PipeFilterAntipatterns(flat -> 0)
original-state	0.9991350102166717	0.11698113207547167	0.5397111913357401
optimization-0	0.9990403501041799	0.12845781492494446	0.4864864864864865
optimization-1	0.9990476399786086	0.12740054895848407	0.48565965583174
optimization-2	0.9990440195642508	0.12792699750789926	0.48653846153846153
optimization-3	0.9990440797387867	0.10867410166999225	0.500945179584121
optimization-1423	0.9990404154436667	0.10457516352335616	0.5155038759689923
optimization-1424	0.9990992191117793	0.0949275362318841	0.50355871886121
optimization-1425	0.9990859413810234	0.09647495361781078	0.5054545454545455
optimization-1426	0.9990991566572962	0.09527272727272729	0.5053571428571428
optimization-1427	0.99904412756961	0.10457516352335616	0.5164410058027079

Table B.17. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 02 Layer 1

optimization	LFilterFanInCoup(flat -> 1)	LPortPullThroughput(flat -> 1)	PipeFilterAntipatterns(flat -> 1)
original-state	0.9973064577675024	0.23773584905660372	0.8528428093645485
optimization-0	0.997021938882404	0.2426052166713224	0.833976833976834
optimization-1	0.9970232832301797	0.24060846591682594	0.8352490421455939
optimization-2	0.997015147015147	0.24160271577598635	0.8346153846153846
optimization-3	0.9970792106881323	0.21688794487027874	0.835820895522388
optimization-1423	0.9971026376930436	0.2267040151708266	0.8376383763837638
optimization-1424	0.9972150656361183	0.21417069227799124	0.8496503496503497
optimization-1425	0.9971911540416879	0.20593692022263455	0.8505338078291815
optimization-1426	0.9972150656361183	0.21131313115900208	0.8496503496503497
optimization-1427	0.9970832428912525	0.22278244654337565	0.8419117647058824

Table B.18. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 03 Layer 0

optimization	LFilterFanInCoup(flat -> 0)	LPortPullThroughput(flat -> 0)
original-state	0.9991350102166717	0.11698113207547167
optimization-0	0.5	0.0
optimization-1	0.5	0.0
optimization-2	0.5	0.0
optimization-3	0.5	0.0
optimization-1484	0.9946349673890175	7.602481449942022E-5
optimization-1485	0.9942571785268415	4.572369173083679E-5
optimization-1486	0.9942571785268415	4.572369173083679E-5
optimization-1487	0.9941222570532915	4.572473708275293E-5
optimization-1488	0.9948553678897302	7.602481449942022E-5

Table B.19. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 03 Layer 1

optimization	LFilterFanInCoup(flat -> 1)	LPortPullThroughput(flat -> 1)
original-state	0.9973064577675024	0.23773584905660372
optimization-0	0.9997355896351137	0.0
optimization-1	0.9997355896351137	0.0
optimization-2	0.9997355896351137	0.0
optimization-3	0.9997355896351137	0.0
optimization-1484	0.995461060678452	0.4983274540810121
optimization-1485	0.9954034729315628	0.4994856084680277
optimization-1486	0.9954034729315628	0.4994856084680277
optimization-1487	0.995559038662487	0.4994970278920896
optimization-1488	0.9954243880118966	0.4983274540810121

Table B.20. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 04 Layer 0

optimization	LFilterFanInCoup(flat -> 0)	PipeFilterAntipatterns(flat -> 0)
original-state	0.9991350102166717	0.5397111913357401
optimization-0	0.9986523506539413	0.41192411924119243
optimization-1	0.9986522911051213	0.4112903225806452
optimization-2	0.9986670514353245	0.4031413612565445
optimization-3	0.9986449864498645	0.4146341463414634
optimization-1133	0.9985107485107485	0.40625
optimization-1134	0.9986884385861368	0.391304347826087
optimization-1135	0.9986935866983373	0.38004750593824227
optimization-1136	0.9986634967551243	0.3924050632911392
optimization-1137	0.9986811116344795	0.3963730569948187

Table B.21. Excerpt of Metrics.csv of run 2026-07-16-14-12-44 Job 04 Layer 1

optimization	LFilterFanInCoup(flat -> 1)	PipeFilterAntipatterns(flat -> 1)
original-state	0.9973064577675024	0.8528428093645485
optimization-0	0.9966539870386024	0.7396449704142012
optimization-1	0.9966237382526976	0.7411764705882353
optimization-2	0.9966115208291808	0.7471264367816092
optimization-3	0.9966237382526976	0.7411764705882353
optimization-1133	0.9960792700313659	0.7797619047619048
optimization-1134	0.9961464354527938	0.7873563218390804
optimization-1135	0.9963174150629603	0.7934782608695652
optimization-1136	0.9961709873830896	0.7877094972067039
optimization-1137	0.9961464354527938	0.7873563218390804

Table B.22. Excerpt of Metrics.csv of run 2026-07-21-18-37-02 Job 05 Layer 0

optimization	LPortPullThroughput(flat -> 0)	PipeFilterAntipatterns(flat -> 0)
original-state	0.11698113207547167	0.5397111913357401
optimization-0	0.04470301419496536	0.4448818897637795
optimization-1	0.02850126116838836	0.44163424124513617
optimization-2	0.04783971349691851	0.44227005870841485
optimization-3	0.028235762682344623	0.442495126705653
optimization-1005	0.15131578989196237	0.4188679245283019
optimization-1006	0.11649168870580473	0.4245614035087719
optimization-1007	0.14072463802669355	0.4226415094339623
optimization-1008	0.1599652780095736	0.42023346303501946
optimization-1009	0.16214953290834355	0.41941747572815535

Table B.23. Excerpt of Metrics.csv of run 2026-07-21-18-37-02 Job 05 Layer 1

optimization	LPortPullThroughput(flat -> 1)	PipeFilterAntipatterns(flat -> 1)
original-state	0.23773584905660372	0.8528428093645485
optimization-0	0.0861849881867145	0.8117154811715481
optimization-1	0.050334085652857685	0.8174273858921162
optimization-2	0.08660547728503287	0.8132780082987552
optimization-3	0.05006858716681395	0.8174273858921162
optimization-1005	0.26709861818112823	0.8143459915611815
optimization-1006	0.20892388463958977	0.83399209486166
optimization-1007	0.2519828723824542	0.819327731092437
optimization-1008	0.2650810184081396	0.8138528138528138
optimization-1009	0.26726664348059603	0.8138528138528138

```
        traceEventRecords2ExecutionAndMessageTraceStage.outputport20 -> kieker.
        tools.trace.analysis.TraceAnalysisConfiguration.Merger.messageTraceMerger.
        inputport10
4 kieker.tools.trace.analysis.TraceAnalysisConfiguration.
    TraceEventRecords2ExecutionAndMessageTraceStage.
    traceEventRecords2ExecutionAndMessageTraceStage.outputport20 -> kieker.tools.
    trace.analysis.TraceAnalysisConfiguration.Merger.messageTraceMerger.
    inputport11
5 ...
6 kieker.tools.trace.analysis.TraceAnalysisConfiguration.
    TraceEventRecords2ExecutionAndMessageTraceStage.
    traceEventRecords2ExecutionAndMessageTraceStage.outputport210 -> kieker.tools.
    trace.analysis.TraceAnalysisConfiguration.Merger.messageTraceMerger.
    inputport10
```

```
7 kieker.tools.trace.analysis.TraceAnalysisConfiguration.
      TraceEventRecords2ExecutionAndMessageTraceStage.
      traceEventRecords2ExecutionAndMessageTraceStage.outputport210 -> kieker.tools.
      trace.analysis.TraceAnalysisConfiguration.Merger.messageTraceMerger.
      inputport11
8 kieker.tools.trace.analysis.TraceAnalysisConfiguration.
      TraceEventRecords2ExecutionAndMessageTraceStage.
      traceEventRecords2ExecutionAndMessageTraceStage.outputport211 -> kieker.tools.
      trace.analysis.TraceAnalysisConfiguration.Merger.messageTraceMerger.
      inputport10
9 kieker.tools.trace.analysis.TraceAnalysisConfiguration.
      TraceEventRecords2ExecutionAndMessageTraceStage.
      traceEventRecords2ExecutionAndMessageTraceStage.outputport211 -> kieker.tools.
      trace.analysis.TraceAnalysisConfiguration.Merger.messageTraceMerger.
      inputport11
```

Appendix C

Personal Correspondence

Appendix C is a personal communication with the *Kieker* developer Yang from 2026-08-18. We left out the exact implementations of the mentioned scripts.

> Hi all,
>
> Regarding Section 5.3.1 Package Re-Modularization, I had a look at optimization-0 of run 2026-07-14-12-35-59 Job 01.
>
> I used the following shell script (thanks to Gemini) to effectively convert the layer description into a dot file:
>
> [...]
>
> I hoped that the resulting diagram will allow for more intuitive understanding of the resulting optimization, and indeed here, I found that: 616layer1kieker.analysis.analysisComponent
>
> plays an important role, which has most of kieker.analysis.plugin.**and kieker.analysis.model.analysisMetaModel.** classes as children, except for kieker.analysis.plugin.filter.select.TimestampFilter,TypeFilter, and kieker.analysis.model.analysisMetaModel.util.AnalysisMetaModelSwitch.
>
> Interestingly, TimestampFilter and TypeFilter belong to .select parent nodes, 590layer2kieker.analysis.plugin.filter.select and 149layer3kieker.analysis.plugin.filter.select. It may give hints to the Kieker developers to have another look at the architectural composition of *.select.* classes in the kieker.analysis package.
>
> Also, I had another look at optimization-1. It has additional two small isolated complete tree growing over layer one until three: (804layer1784layer1kieker.analysis, 281layer6kieker) (281layer6kieker, none)
>
> and, layer three until four: (kieker.analysis.display.PlainText, 277layer4kieker.analysis.plugin.reader.filesystem)
> vor 3 Tagen
>
> Regarding Section 5.3.2 P&F Optimization, I had a look at optimization-0 of run 2026-07-16-14-12-44 Job 02. For drawing layers, I used the same conv-graph.sh. For drawing caller-callee pairs, I changed the script to show caller -> callee correctly.
>
> [...]
>
> The layer graph showed a lot of sub graphs. The most interesting found was from caller-callee graph: traceEventRecords2ExecutionAndMessageTraceStage.outputport20

-> messageTraceMerger.inputport10
traceEventRecords2ExecutionAndMessageTraceStage.outputport20
-> messageTraceMerger.inputport11
traceEventRecords2ExecutionAndMessageTraceStage.outputport210
-> messageTraceMerger.inputport10
traceEventRecords2ExecutionAndMessageTraceStage.outputport210
-> messageTraceMerger.inputport11
traceEventRecords2ExecutionAndMessageTraceStage.outputport211
-> messageTraceMerger.inputport10
traceEventRecords2ExecutionAndMessageTraceStage.outputport211
-> messageTraceMerger.inputport11

This is from Line 228 of TraceAnalysisConfiguration.java in trace-analysis. As a Kieker developer, one could expect substantial amount of parallelism through this pipe, and consider this for further optimizations and applications.

Appendix D

Thrown Exceptions

Listing D.1. ClassCastException when starting Kieker collector

```
1  Starting collector
2  [info] Wait for startup
3  Failed to instantiate [ch.qos.logback.classic.LoggerContext]
4  Reported exception:
5  java.lang.ClassCastException: class ch.qos.logback.classic.Level cannot be cast to
       class java.util.Iterator (ch.qos.logback.classic.Level is in unnamed module
      of loader 'app'; java.util.Iterator is in module java.base of loader '
      bootstrap')
6  at jdk.proxy1/jdk.proxy1.$Proxy17.iterator(Unknown Source)
7  at org.codehaus.groovy.runtime.DefaultGroovyMethods.toList(DefaultGroovyMethods.
       java:3381)
8  at org.codehaus.groovy.runtime.DefaultGroovyMethods.asCollection(
       DefaultGroovyMethods.java:11805)
9  at org.codehaus.groovy.runtime.DefaultGroovyMethods.minus(DefaultGroovyMethods.
       java:13967)
10 at org.codehaus.groovy.runtime.DefaultGroovyMethods.minus(DefaultGroovyMethods.
       java:13874)
11 at org.codehaus.groovy.runtime.dgm$538.doMethodInvoke(Unknown Source)
12 at org.codehaus.groovy.runtime.callsite.
       PojoMetaMethodSite$PojoMetaMethodSiteNoUnwrap.invoke(PojoMetaMethodSite.java
       :234)
13 at org.codehaus.groovy.runtime.callsite.PojoMetaMethodSite.call(PojoMetaMethodSite
       .java:57)
14 at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.
       java:45)
15 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
       java:125)
16 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
       java:139)
17 at ch.qos.logback.classic.gaffer.ComponentDelegate.analyzeArgs(ComponentDelegate.
       groovy:110)
```

```
18 at ch.qos.logback.classic.gaffer.ComponentDelegate$analyzeArgs.callCurrent(Unknown
       Source)
19 at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallCurrent(
       CallSiteArray.java:49)
20 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callCurrent(
       AbstractCallSite.java:171)
21 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callCurrent(
       AbstractCallSite.java:185)
22 at ch.qos.logback.classic.gaffer.ComponentDelegate.methodMissing(ComponentDelegate
       .groovy:48)
23 at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
24 at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(
       NativeMethodAccessorImpl.java:77)
25 at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(
       DelegatingMethodAccessorImpl.java:43)
26 at java.base/java.lang.reflect.Method.invoke(Method.java:569)
27 at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:343)
28 at groovy.lang.MetaClassImpl.invokeMissingMethod(MetaClassImpl.java:925)
29 at groovy.lang.MetaClassImpl.invokePropertyOrMissing(MetaClassImpl.java:1414)
30 at groovy.lang.MetaClassImpl.doInvokeMethod(MetaClassImpl.java:1336)
31 at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1089)
32 at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1008)
33 at org.codehaus.groovy.runtime.InvokerHelper.invokePogoMethod(InvokerHelper.java
       :645)
34 at org.codehaus.groovy.runtime.InvokerHelper.invokeMethod(InvokerHelper.java:628)
35 at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.
       invokeOnDelegationObjects(ClosureMetaClass.java:391)
36 at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(
       ClosureMetaClass.java:330)
37 at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1008)
38 at org.codehaus.groovy.vmplugin.v8.IndyInterface.fromCache(IndyInterface.java:321)
39 at Script1$_run_closure1.doCall(Script1.groovy:9)
40 at Script1$_run_closure1.doCall(Script1.groovy)
41 at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
42 at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(
       NativeMethodAccessorImpl.java:77)
43 at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(
       DelegatingMethodAccessorImpl.java:43)
44 at java.base/java.lang.reflect.Method.invoke(Method.java:569)
45 at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:343)
46 at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:328)
```

```
47 at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(
       ClosureMetaClass.java:279)
48 at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1008)
49 at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.call(PogoMetaClassSite.
       java:39)
50 at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.
       java:45)
51 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
       java:125)
52 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
       java:130)
53 at ch.qos.logback.classic.gaffer.ConfigurationDelegate.appender(
       ConfigurationDelegate.groovy:143)
54 at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
55 at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(
       NativeMethodAccessorImpl.java:77)
56 at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(
       DelegatingMethodAccessorImpl.java:43)
57 at java.base/java.lang.reflect.Method.invoke(Method.java:569)
58 at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:343)
59 at org.codehaus.groovy.runtime.metaclass.MixinInstanceMetaMethod.invoke(
       MixinInstanceMetaMethod.java:61)
60 at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:328)
61 at org.codehaus.groovy.vmplugin.v8.IndyInterface.fromCache(IndyInterface.java:321)
62 at Script1.run(Script1.groovy:8)
63 at Script1$run.call(Unknown Source)
64 at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.
       java:45)
65 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
       java:125)
66 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.
       java:130)
67 at ch.qos.logback.classic.gaffer.GafferConfigurator.run(GafferConfigurator.groovy
       :77)
68 at ch.qos.logback.classic.gaffer.GafferConfigurator$run.callCurrent(Unknown Source
       )
69 at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallCurrent(
       CallSiteArray.java:49)
70 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callCurrent(
       AbstractCallSite.java:171)
```

```
71 at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callCurrent(
       AbstractCallSite.java:185)
72 at ch.qos.logback.classic.gaffer.GafferConfigurator.run(GafferConfigurator.groovy
       :44)
73 at ch.qos.logback.classic.gaffer.GafferUtil.runGafferConfiguratorOn(GafferUtil.
       java:43)
74 at ch.qos.logback.classic.util.ContextInitializer.configureByResource(
       ContextInitializer.java:67)
75 at ch.qos.logback.classic.util.ContextInitializer.autoConfig(ContextInitializer.
       java:150)
76 at org.slf4j.impl.StaticLoggerBinder.init(StaticLoggerBinder.java:84)
77 at org.slf4j.impl.StaticLoggerBinder.(StaticLoggerBinder.java:55)
78 at org.slf4j.LoggerFactory.bind(LoggerFactory.java:150)
79 at org.slf4j.LoggerFactory.performInitialization(LoggerFactory.java:124)
80 at org.slf4j.LoggerFactory.getILoggerFactory(LoggerFactory.java:417)
81 at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:362)
82 at kieker.tools.common.AbstractLegacyTool.(AbstractLegacyTool.java:53)
83 at kieker.tools.common.AbstractService.(AbstractService.java:58)
84 at kieker.tools.collector.CollectorMain.(CollectorMain.java:47)
85 at kieker.tools.collector.CollectorMain.main(CollectorMain.java:58)
```

Listing D.2. NullPointerException: CSV header is null

```
 1     WARN - Exception occurred in CsvRowReaderProducerStage-0
 2 java.lang.NullPointerException: Cannot invoke "org.csveed.api.Header.size()"
       because "header" is null
 3    at org.csveed.bean.ColumnIndexMapper.checkKey(ColumnIndexMapper.java:25)
 4    at org.csveed.bean.AbstractMapper.verifyHeader(AbstractMapper.java:34)
 5    at org.csveed.bean.BeanReaderImpl.readBean(BeanReaderImpl.java:65)
 6    at org.csveed.api.CsvClientImpl.readBean(CsvClientImpl.java:123)
 7    at org.oceandsl.analysis.generic.source.CsvRowReaderProducerStage.execute(
          CsvRowReaderProducerStage.java:80)
 8    at teetime.framework.AbstractStage.executeWithCatchedExceptions(AbstractStage.
          java:220)
 9    at teetime.framework.AbstractStage.executeByFramework(AbstractStage.java:214)
10    at teetime.framework.StageFacade.runStage(StageFacade.java:76)
11    at teetime.framework.scheduling.pushpullmodel.AbstractRunnableStage.run(
          AbstractRunnableStage.java:56)
12    at java.base/java.lang.Thread.run(Thread.java:840)
13    at teetime.framework.scheduling.pushpullmodel.TeeTimeThread.run(TeeTimeThread.
          java:46)
14 Exception in thread "main" teetime.framework.ExecutionException: 3003 - {Thread[
```

```
    Thread for CsvRowReaderProducerStage-0,5,]=[java.lang.NullPointerException:
    Cannot invoke "org.csveed.api.Header.size()" because "header" is null]} error(
    s) occurred while execution. Check thrown exception(s).
15    at teetime.framework.Execution.waitForTermination(Execution.java:177)
16    at teetime.framework.Execution.executeBlocking(Execution.java:218)
17    at org.oceandsl.tools.sar.StaticArchitectureRecoveryMain.executeConfiguration(
          StaticArchitectureRecoveryMain.java:122)
18    at org.oceandsl.tools.sar.StaticArchitectureRecoveryMain.execute(
          StaticArchitectureRecoveryMain.java:95)
19    at org.oceandsl.tools.sar.StaticArchitectureRecoveryMain.run(
          StaticArchitectureRecoveryMain.java:232)
20    at org.oceandsl.tools.sar.StaticArchitectureRecoveryMain.main(
          StaticArchitectureRecoveryMain.java:75)
21 Exception in thread "Thread-1" java.lang.IllegalStateException: Illegal state
    change from TERMINATED to TERMINATING
22    at teetime.framework.AbstractStage.changeState(AbstractStage.java:398)
23    at teetime.framework.AbstractStage.terminateStageByFramework(AbstractStage.java
          :679)
24    at teetime.framework.AbstractStage.abort(AbstractStage.java:683)
25    at teetime.framework.StageFacade.abort(StageFacade.java:38)
26    at teetime.framework.scheduling.pushpullmodel.PushPullScheduling.abortStages(
          PushPullScheduling.java:196)
27    at teetime.framework.scheduling.pushpullmodel.PushPullScheduling.onTerminate(
          PushPullScheduling.java:190)
28    at teetime.framework.Execution.abortEventually(Execution.java:186)
29    at org.oceandsl.tools.sar.StaticArchitectureRecoveryMain$1.run(
          StaticArchitectureRecoveryMain.java:143)
30    at java.base/java.lang.Thread.run(Thread.java:840)
```

Listing D.3. Example exception: Missing xmi-file

```
1    Exception in thread "main" teetime.framework.ExecutionException: 3003 - {
         Thread[Thread for ModelRepositoryProducerStage-0,5,]=[kieker.common.
         exception.ConfigurationException: Error reading model file /replication/
         data/mitgcm/tutorial_barotropic_gyre/static-plain-call-file/type-model.xmi
         . File does not exist.]} error(s) occurred while execution. Check thrown
         exception(s).
2    at teetime.framework.Execution.waitForTermination(Execution.java:177)
3    at teetime.framework.Execution.executeBlocking(Execution.java:218)
4    at kieker.tools.common.AbstractService.execute(AbstractService.java:77)
5    at kieker.tools.common.AbstractLegacyTool.run(AbstractLegacyTool.java:99)
6    at org.oceandsl.tools.maa.ModelArchitectureAnalysisMain.main(
```

```
        ModelArchitectureAnalysisMain.java:38)
7 Exception in thread "Thread-0" java.lang.IllegalStateException: Illegal state
      change from TERMINATED to TERMINATING
8     at teetime.framework.AbstractStage.changeState(AbstractStage.java:398)
9     at teetime.framework.AbstractStage.terminateStageByFramework(AbstractStage.java
          :679)
10    at teetime.framework.AbstractStage.abort(AbstractStage.java:683)
11    at teetime.framework.StageFacade.abort(StageFacade.java:38)
12    at teetime.framework.scheduling.pushpullmodel.PushPullScheduling.abortStages(
          PushPullScheduling.java:196)
13    at teetime.framework.scheduling.pushpullmodel.PushPullScheduling.onTerminate(
          PushPullScheduling.java:190)
14    at teetime.framework.Execution.abortEventually(Execution.java:186)
15    at kieker.tools.common.AbstractService$1.run(AbstractService.java:103)
16    at java.base/java.lang.Thread.run(Thread.java:840)
```

The exceptions Listing D.4 and Listing D.5 were thrown in run 2026-07-16-14-12-44 and can be found in the latest log file: `latest.log`.

Listing D.4. IllegalArgumentException: mergePort: Different processedData Types

```
1     Caused by: java.lang.IllegalArgumentException: mergePort: All ports must
          process the same data type!
2     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.mutation.
          MultiLevelPortMutations.mergePorts(MultiLevelPortMutations.java:212)
3     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.mutation.
          MultiLevelPortMutations.mergeAdjacentPortsRandomly(MultiLevelPortMutations.
          java:141)
4     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.mutation.
          MultiLevelPortMutations.lambda$static$2(MultiLevelPortMutations.java:53)
5     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.
          AnnotatedOperations$MergeAdjacentPorts.mutate(AnnotatedOperations.java:99)
6     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.
          AnnotatedOperations$MergeAdjacentPorts.mutate(AnnotatedOperations.java:95)
7     at cau.agse.hs.genopt.genetic.jobs.general.handling.InitializeOptimization.
          lambda$parseMutationString$1(InitializeOptimization.java:151)
8     at cau.agse.hs.genopt.genetic.algorithms.common.BasicSteps.
          lambda$performMutationsByRatioPlusFixedCandidates$0(BasicSteps.java:240)
9     at cau.agse.hs.utils.concurrency.Threading.lambda$runParallelInChunks$0(
          Threading.java:119)
10    at cau.agse.hs.utils.concurrency.Threading.lambda$runParallel$1(Threading.java
          :98)
11    at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java
```

```
        :545)
12    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:328)
13    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(
        ThreadPoolExecutor.java:1090)
14    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(
        ThreadPoolExecutor.java:614)
15    at java.base/java.lang.Thread.run(Thread.java:1474
```

Listing D.5. IllegalArgumentException: mergePort: Different ProgramUnitType and not adjacent

```
1     Caused by: java.lang.IllegalArgumentException: mergePort: All ports must have
        the same ProgramUnitType or be at least adjacent!
2     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.mutation.
        MultiLevelPortMutations.mergePorts(MultiLevelPortMutations.java:202)
3     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.mutation.
        MultiLevelPortMutations.mergeAdjacentPortsRandomly(MultiLevelPortMutations.
        java:141)
4     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.mutation.
        MultiLevelPortMutations.lambda$static$2(MultiLevelPortMutations.java:53)
5     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.
        AnnotatedOperations$MergeAdjacentPorts.mutate(AnnotatedOperations.java:99)
6     at cau.agse.hs.genopt.nestedcoupling.genetic.operators.
        AnnotatedOperations$MergeAdjacentPorts.mutate(AnnotatedOperations.java:95)
7     at cau.agse.hs.genopt.genetic.jobs.general.handling.InitializeOptimization.
        lambda$parseMutationString$1(InitializeOptimization.java:151)
8     at cau.agse.hs.genopt.genetic.algorithms.common.BasicSteps.
        lambda$performMutationsByRatioPlusFixedCandidates$0(BasicSteps.java:240)
9     at cau.agse.hs.utils.concurrency.Threading.lambda$runParallelInChunks$0(
        Threading.java:119)
10    at cau.agse.hs.utils.concurrency.Threading.lambda$runParallel$1(Threading.java
        :98)
11    at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java
        :545)
12    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:328)
13    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(
        ThreadPoolExecutor.java:1090)
14    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(
        ThreadPoolExecutor.java:614)
15    at java.base/java.lang.Thread.run(Thread.java:1474)
```

Bibliography

[AB24] Léo CR Antunes and Márcio de O Barros. "A comparison between hierarchical and non-hierarchical software clustering". In: *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC. 2024, pages 69–79 (cited on pages 4, 8).

[AC94] Fernando Abreu e Brito and Rogério Carapuça. "Object-oriented software engineering: measuring and controlling the development process". In: *Proceedings of "4th Int. Conf. on Software Quality"* (1994). Revised Version (cited on pages 8, 11, 12).

[Ada] AdaCore. *Packages - learn.adacore.com*. https://learn.adacore.com/courses/Ada_For_The_CPP_Java_Developer/chapters/07_Packages.html. (Visited on 04/30/2026) (cited on page 13).

[Ado16] Marc Adolf. "Self-adapting execution of pipe-and-filter systems". Master's Thesis. Institut für Informatik, 2016 (cited on pages 5–8, 10, 18, 22, 48).

[AGE95] F Brito Abreu, Miguel Goulão, and Rita Esteves. "Toward the design quality evaluation of object-oriented software systems". In: *Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA*. 1995, pages 44–57 (cited on page 12).

[Arc+18] Davide Arcelli et al. "Easier: an evolutionary approach for multi-objective software architecture refactoring". In: *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2018, pages 105–10509 (cited on pages 3, 8, 19).

[Bav+10] Gabriele Bavota et al. "Software re-modularization based on structural and semantic metrics". In: *2010 17th Working Conference on Reverse Engineering*. IEEE. 2010, pages 195–204 (cited on pages 10, 13).

[BBG11] Slim Bechikh, Lamjed Ben Said, and Khaled Ghédira. "Searching for knee regions of the pareto front using mobile reference points". In: *Soft Computing* 15.9 (2011), pages 1807–1823 (cited on pages 11, 37, 38).

[BDW02] Lionel C. Briand, John W. Daly, and Jurgen K Wust. "A unified framework for coupling measurement in object-oriented systems". In: *IEEE Transactions on software Engineering* 25.1 (2002), pages 91–121 (cited on page 11).

[BDW98] Lionel C Briand, John W Daly, and Jürgen Wüst. "A unified framework for cohesion measurement in object-oriented systems". In: *Empirical Software Engineering* 3.1 (1998), pages 65–117 (cited on pages 8, 12).

[Can+16] Ivan Candela et al. “Using cohesion and coupling for software remodularization: is it enough?” In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25.3 (2016), pages 1–28 (cited on pages 3, 9).

[Deb+02] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: nsga-ii”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pages 182–197 (cited on pages 3, 10, 11, 40–42, 50).

[DJ13] Kalyanmoy Deb and Himanshu Jain. “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints”. In: *IEEE transactions on evolutionary computation* 18.4 (2013), pages 577–601 (cited on page 38).

[DK18] Udaya Dayanandan and Vivekanandan Kalimuthu. “Software architectural quality assessment model for security analysis using fuzzy analytical hierarchy process (fahp) method”. In: *3D Research* 9.3 (2018), page 31 (cited on page 13).

[Dob03] Ernst-Erich Doberkat. “Pipelines: modelling a software architecture through relations”. In: *Acta Informatica* 40.1 (2003), pages 37–79 (cited on pages 5, 6, 18).

[EHB20] Christoph Etzel, Florian Hofhammer, and Bernhard Bauer. “Towards metrics for analyzing system architectures modeled with east-adl”. In: (2020) (cited on pages 11, 13).

[GC12] Varun Gupta and Jitender Kumar Chhabra. “Package level cohesion measurement in object-oriented software”. In: *Journal of the Brazilian Computer Society* 18.3 (2012), pages 251–266 (cited on pages 12, 13).

[Gha+26] GS Pradeep Ghantasala et al. “A multi-layered framework for adaptive and optimized real-time data processing in edge-fog iot environments”. In: *IEEE Internet of Things Journal* (2026) (cited on pages 9, 36).

[Haa23] Jakob Haastert. “Multi level coupling optimization using genetic algorithms”. Bachelor’s Thesis. Institut für Informatik, 2023 (cited on pages 4, 27).

[Hal+14] Mathew Hall et al. “Establishing the source code disruption caused by automated remodularisation tools”. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE. 2014, pages 466–470 (cited on page 41).

[HC04] Mark Harman and John Clark. “Metrics are fitness functions too”. In: *10th International Symposium on Software Metrics, 2004. Proceedings.* Ieee. 2004, pages 58–69 (cited on pages 4, 9, 11, 14, 20, 58).

[HH20] Wilhelm Hasselbring and André van Hoorn. “Kieker: A monitoring framework for software engineering research”. In: *Software Impacts* 5 (June 2020) (cited on pages v, 1, 7, 33).

[Hir+14] Martin Hirzel et al. “A catalog of stream processing optimizations”. In: *ACM Computing Surveys (CSUR)* 46.4 (2014), pages 1–34 (cited on pages 5, 26, 29, 30, 58).

[HJ01] Mark Harman and Bryan F Jones. “Search-based software engineering”. In: *Information and software Technology* 43.14 (2001), pages 833–839 (cited on page 4).

[HWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. “Kieker: A framework for application performance monitoring and dynamic software analysis”. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. Boston, Massachusetts, USA, April 22-25, 2012: ACM, Apr. 2012, pages 247–248 (cited on pages v, 1, 7).

[jav26a] javathinking.com. *How to properly implement the 'equals' method for generic classes using 'instanceof' in java: a clean, error-free approach*. online. 2026. URL: https://www.javathinking.com/blog/how-to-implement-equals-method-for-generics-using-instanceof/ (visited on 06/11/2026) (cited on page 27).

[jav26b] javathinking.com. *Intellij java call graph: how to generate method call trees from a starting method without debugging*. online. 2026. URL: https://www.javathinking.com/blog/intellij-or-other-possible-to-see-a-graph-tree-of-all-calls-made-in-java/ (visited on 06/04/2026) (cited on page 33).

[Jet26] Jetbrains. *Introduction to profiling*. online. 2026. URL: https://www.jetbrains.com/help/idea/profiler-intro.html#available-tools (visited on 06/04/2026) (cited on page 34).

[Jun+21] Reiner Jung et al. “Developing domain-specific languages for ocean modeling”. In: *Software Engineering 2021 Satellite Events-Workshops and Tools & Demos*. Volume 2814. CEUR. 2021, pages 1–12 (cited on page 34).

[Jun+24] Reiner Jung et al. *Teetime*. online. 2024. URL: https://github.com/teetime-framework/TeeTime (visited on 07/02/2026) (cited on page 35).

[Kau25] Natalie Kaufhold. “Engineering of a multilevel data structure for a genetic framework”. Master’s Project. Institut für Informatik, 2025 (cited on pages 1, 4–6, 23–25, 27–32, 34, 40, 42, 43).

[Ker26] Michael Kerrisk. *Diff(1) - linux manual page*. online. HTML rendering created by author. 2026. URL: https://www.man7.org/linux/man-pages/man1/diff.1.html (visited on 07/13/2026) (cited on page 39).

[Kie] Kieker. *Kieker keeps an eye on your software - documentation*. https://kieker-monitoring.net/documentation/. Image received on 2026-05-13. (Visited on 05/13/2026) (cited on pages 7, 8).

[KZ24] Natalie Kaufhold and Esther Zwanger. “Optimizing a flutter-project with a genetic framework: tooling & analyzing”. Master’s Project. Institut für Informatik, 2024 (cited on pages 32, 39, 40).

[LYC07] Young Lee, Jeong Yang, and Kai H Chang. “Metrics and evolution in open source software”. In: *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE. 2007, pages 191–197 (cited on pages 13, 16, 17, 22).

[Mka+15] Wiem Mkaouer et al. “Many-objective software remodularization using nsga-iii”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24.3 (2015), pages 1–45 (cited on pages v, 4, 10, 17, 41).

[Mon+97] Robert T Monroe et al. “Architectural styles, design patterns, and objects”. In: *IEEE software* 14.1 (1997), pages 43–52 (cited on page 5).

[Mur+12] Alessandro Murgia et al. “Refactoring and its relationship with fan-in and fan-out: an empirical study”. In: *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE. 2012, pages 63–72 (cited on pages 16, 22).

[NZE05] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. “Pareto multi objective optimization”. In: *Proceedings of the 13th international conference on, intelligent systems application to power systems*. IEEE. 2005, pages 84–91 (cited on pages 3, 20, 21, 37, 42).

[PCB92] Sukesh Patel, William Chu, and Rich Baxter. “A measure for composite module cohesion”. In: *Proceedings of the 14th international conference on software engineering*. 1992, pages 38–48 (cited on page 13).

[RS09] Lily Rachmawati and Dipti Srinivasan. “Multiobjective evolutionary algorithm with controllable focus on the knees of the pareto front”. In: *IEEE Transactions on Evolutionary Computation* 13.4 (2009), pages 810–824 (cited on page 37).

[Sel+22] Khaled Sellami et al. “Improving microservices extraction using evolutionary search”. In: *Information and Software Technology* 151 (2022), page 106996 (cited on page 8).

[SH20] Henning Schnoor and Wilhelm Hasselbring. “Comparing static and dynamic weighted software coupling metrics”. In: *Computers* 9.2 (2020). URL: https://www.mdpi.com/2073-431X/9/2/24 (cited on page 9).

[Sha96] Mary Shaw. “Procedure calls are the assembly language of software interconnection: connectors deserve first-class status”. In: *Workshop on Studies of Software Design*. Springer. 1996, pages 17–32 (cited on page 7).

[SHJ25a] Henning Schnoor, Wilhelm Hasselbring, and Reiner Jung. *Replication package for: software architecture evaluation and restructuring of earth system models (restructuring part)*. Version 1.0.0.1. 2025. URL: https://doi.org/10.5281/zenodo.16628824 (cited on page 9).

[SHJ25b] Henning Schnoor, Wilhelm Hasselbring, and Reiner Jung. “Using genetic algorithms for research software structure optimization”. In: *Software* 4.4 (2025) (cited on pages 1, 3, 4, 9, 11, 40).

[Sil+23] Samira Silva et al. “Quality metrics in software architecture”. In: *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. IEEE. 2023, pages 58–69 (cited on page 13).

[SKR08] Santonu Sarkar, Avinash C Kak, and Girish Maskeri Rama. "Metrics for measuring the quality of modularization of large-scale object-oriented software". In: *IEEE Transactions on Software Engineering* 34.5 (2008), pages 700–720 (cited on page 8).

[SMC74] Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. "Structured design". In: *IBM systems journal* 13.2 (1974), pages 115–139 (cited on pages 11, 13).

[Sus] SustainKieker. *Research project for sustainable and reusable kieker - about*. https://sustainkieker.kieker-monitoring.net/About/. (Visited on 05/04/2026) (cited on page 1).

[TMD10] Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software architecture: foundations, theory, and practice*. Edited by Daniel Sayre. John Wiley & Sons, Inc., 2010 (cited on page 5).

[Wal+26] Jan Waller et al. *Kieker observability framework*. online. 2026. URL: https://github.com/kieker-monitoring/kieker (visited on 05/11/2026) (cited on pages 25, 31).

[Wal14] Jan Waller. *Performance benchmarking of application monitoring frameworks*. Kiel Computer Science Series 2014/5. Dissertation, Faculty of Engineering, Kiel University. Department of Computer Science, Kiel University, Dec. 2014 (cited on pages 7, 34).

[WCB01] Matt Welsh, David Culler, and Eric Brewer. "Seda: an architecture for well-conditioned, scalable internet services". In: *ACM SIGOPS operating systems review* 35.5 (2001), pages 230–243 (cited on page 6).

[WEH14] Christian Wulf, Nils Christian Ehmke, and Wilhelm Hasselbring. "Toward a generic and concurrency-aware pipes & filters framework". In: (2014) (cited on pages 7, 16–19, 27, 28, 36, 41, 49, 50).

[WHO17] Christian Wulf, Wilhelm Hasselbring, and Johannes Ohlemacher. "Parallel and generic pipe-and-filter architectures with teetime". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017, pages 290–293 (cited on pages 5–7, 10, 18, 35).

[WJ23] Hasselbring Wilhelm and Reiner Jung. *Oceandsl: domain-specific languages for ocean modeling and simulations*. online. 2023. URL: https://oceandsl.uni-kiel.de/oceandsl/ (visited on 06/11/2026) (cited on page 34).

[Wul+19] Christian Wulf et al. *Teetime: the next-generation pipe-and-filter framework*. online. 2019. URL: http://teetime-framework.github.io/ (visited on 07/02/2026) (cited on pages 7, 35).

[Wul19] Christian Wulf. "Efficient engineering and execution of pipe-and-filter architectures". PhD thesis. 2019 (cited on pages 5, 7, 10, 17).

[Yan+25] Shinhyung Yang et al. "The kieker observability framework version 2". In: *Companion of the 16th ACM/SPEC International Conference on Performance Engineering*. ICPE '25. Toronto ON, Canada: ACM, 2025, pages 11–15 (cited on pages 1, 7).

[Yan26] Shinhyung Yang. personal communication. Aug. 18, 2026 (cited on pages 50, 82).

[Zha+21] Shixiong Zhao et al. “V pipe: a virtualized acceleration system for achieving efficient and scalable pipeline parallel dnn training”. In: *IEEE Transactions on Parallel and Distributed Systems* 33.3 (2021), pages 489–506 (cited on pages 9, 27).