



Semantic Analysis of Domain Model Evolution in Model-Driven Software Development

Constantin Buschhaus
 RWTH Aachen University
 Aachen, Germany
 buschhaus@se-rwth.de

Lukas Netz
 RWTH Aachen University
 Aachen, Germany
 netz@se.rwth-aachen.de

Bernhard Rumpe
 RWTH Aachen University
 Aachen, Germany
 rumpe@se.rwth-aachen.de

Max Stachon
 RWTH Aachen University
 Aachen, Germany
 stachon@se-rwth.de

Sebastian Stüber
 RWTH Aachen University
 Aachen, Germany
 stueber@se-rwth.de

Abstract—Model-Driven Engineering (MDE) aims to improve the development of software systems by utilizing models as primary source artifacts and leveraging automation, such as model-to-code translation. Since models serve as primary source artifacts, their evolution throughout the development process is a critical concern. To better understand the changes made to models, difference analyses can be employed. However, merely highlighting syntactic differences between subsequent model versions may be insufficient for comprehensively analyzing the history of a domain model in a model-driven development process. We propose an approach that analyzes the differences between subsequent model versions at the abstract syntax level and assesses their implications on the model’s semantics. This approach is integrated into a toolchain that aggregates and visualizes changes across the entire model development history. In this paper, we apply this integrated analysis to the domain model of MaCoCo, a business information system developed using model-driven methods. We utilize the visualization of the aggregated difference analysis to identify anomalies in the evolution of the model and discuss their implications for the development history of MaCoCo.

Index Terms—class diagrams, difference analysis, model evolution, model-driven development, domain model

I. INTRODUCTION

In software development, referencing prior projects can help guide developers; however, without the original developers, past mistakes may recur. To mitigate this risk, consulting the original developers is ideal, but if that is not feasible, analyzing the project’s development history, including the evolution of its domain model, can provide valuable insights.

Model-driven software engineering enhances development by using models as primary artifacts [Dobing and Parsons(2006)], [Stahl and Völter(2013)] and facilitating code generation. These models evolve from abstract drafts to fully-specified versions, influenced by changing requirements and development challenges. To improve change management, we have developed a tool that visualizes the evolution of the domain model during the development process. This tool

utilizes a custom model difference analysis to compare versions of a Class Diagram (CD), detecting and classifying changes in the model’s abstract syntax. It identifies matching elements through specified strategies, compares their syntax, interprets differences, and can be extended for semantic differencing [Maoz et al.(2011)], [Ringert et al.(2023)].

Our toolchain employs an artifact-analysis framework to collect model versions from a Git project’s commit history, computing their differences using the aforementioned analysis. These differences are aggregated and visualized through interactive diagrams, enabling an in-depth inspection of the model’s development history. We evaluated this toolchain on the MaCoCo model-driven development project [Gerasimov et al.(2024)], [Buschhaus et al.(2024)]. The main contributions of this paper are (1) the development of an integrated toolchain for analyzing the evolution of a domain model throughout the software development lifecycle in model-driven engineering projects, and (2) an evaluation of this toolchain within a successful business management software project.

The remainder of this paper is structured as follows: Related work is discussed in section II. The tool for aggregating and visualizing computed differences is introduced in section III. In section IV, we outline the model difference analysis utilized by the toolchain. A case study on the model-driven development of MaCoCo is presented in section V. Finally, we conclude in section VI, providing an outlook on future work.

II. RELATED WORK

Nowadays, syntactic differencing operators for models are widely established and often able to identify a variety of language-specific editing operations [Alanen and Porres(2003)], [Ohst et al.(2003)], [Kehrer et al.(2011)], [Kehrer et al.(2013)], [Küster et al.(2008)], [Küster et al.(2009)], [Taentzer et al.(2014)], [Thüm et al.(2009)], [Addazi et al.(2016)]. For class diagram differencing, we employ a custom matching of model elements and classification of syntactic differences similar to these approaches and integrate it into a toolchain for artifact analysis, thus enabling aggregation

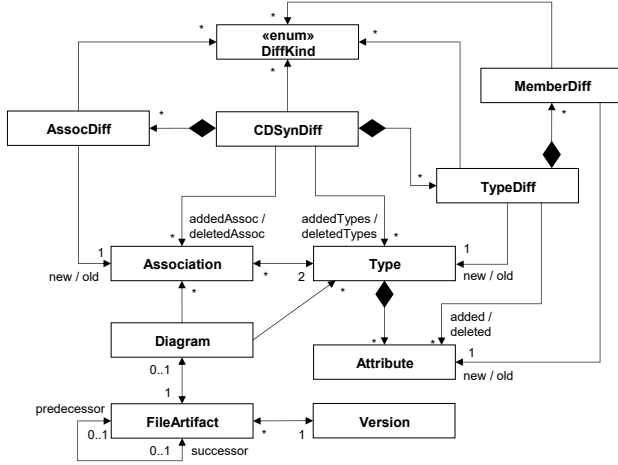


Fig. 1. CD defining the data-structure for identifying and categorizing differences between two class diagrams.

of analysis results and visualization of domain model evolution over an entire model-driven development process.

The result of this analysis can also be used to check for semantic refinement and detect semantic differences in the form of legal instances permitted by the first model but not the second [Harel and Rumpe(2004)], [Maoz et al.(2010)]. Unlike previous semantic differencing operators for CDs [Maoz et al.(2011)], [Ringert et al.(2023)], it traces back these semantic differences to the specific syntactic differences that cause them. This approach of combined syntactic and semantic reasoning is used by another differencing operator [Fahrenberg et al.(2014)] to produce a delta CD instead of diff-witnesses.

There are several tools that visualize software evolution, *e.g.*, Seesoft [Eick et al.(1992)], GEVOL [Collberg et al.(2003)] or EPOSEE [Burch et al.(2005)]. Compared to them our approach is focused on the domain model evolution.

III. ARTIFACT ANALYSIS FRAMEWORK

Domain models in large software projects frequently change, *e.g.*, due to the implementation of new features, refactorings, or an improved understanding of the domain. For our purposes, we consider domain models in the form of UML/P CDs, a variant of Unified Modeling Language (UML) class diagrams for model-driven development of object-oriented software systems [Rumpe(2016)]. More specifically, we focus on models in the textual notation defined by the `CD4Analysis`¹ grammar.

A CD may contain three types of declarations: classes, interfaces, and enumerations, along with associations linking precisely two types. Classes can extend other classes and implement interfaces, while interfaces can extend other interfaces. Both can include attribute declarations with a name and specified type. Enumerations define a set of constants.

¹<https://github.com/MontiCore/cd4analysis>

Associations may have a name, along with role names and cardinalities for each side. They can be unidirectional, bidirectional, or underspecified in direction. The denotational semantics of a CD is the set of object structures it permits [Maoz et al.(2011)].

As these models are textual in syntax, we rely on file-based version control systems like Git, where changes are captured in commits. The evolution of a domain model refers to all changes made during its development, *i.e.*, its entire Git history. The model difference analysis to detect changes between two versions will be detailed in section IV. However, inspecting individual changes is insufficient when considering the bigger picture; we need data aggregations and visualizations to highlight relevant information.

Our domain model evolution analysis tool integrates model difference analysis into an artifact analysis framework that manages the extraction of a repository’s artifact history. The tool is developed using MontiGem [Buschhaus et al.(2024)], a model-driven code generator for web applications. MontiGem establishes the architecture of our tool by generating code from the data model shown in Figure 1 as well as declarative GUI models that are partially implemented with libraries [Gerasimov et al.(2025)]. All charts from section V are part of such libraries. The analysis results are visualized in an interactive web-based dashboard, with a technical stack consisting of an Angular/TypeScript client communicating with a Java/Spring backend server, which in turn interacts with a Neo4j graph database.

The central class of the artifact analysis framework’s data model is the `FileArtifact` class (bottom left in Figure 1). Each `FileArtifact` belongs to exactly one `Version` (snapshot) and can have a `successor` and a `predecessor`. The contents of a `FileArtifact` (*e.g.*, a `Diagram`) belong to exactly one `FileArtifact`. The tool accepts a software repository containing the domain model as input. As the most prevalent version control system for text-based files, Git is used internally for data extraction. (1) First, the Git history of the domain model artifact is extracted. (2) Then, the content of each artifact is parsed, and its abstract syntax tree is traversed to identify the CD types, type members, associations, and their relationships. (3) Lastly, the domain model evolution analysis tool performs model difference analysis on each artifact and its direct predecessor, if it exists. Once this extraction process is complete, data aggregations and visualizations are computed through parameterized database queries or directly in code.

IV. MODEL DIFFERENCE ANALYSIS

We developed a custom model difference operator for UML/P CDs to enable language-specific classification of differences, such as promoting an attribute to a superclass. This approach enables further subsequent analyses, including semantic differencing [Maoz et al.(2010)].

Our analysis takes two CDs as input: the first represents the newer version, while the second is the predecessor. It computes a matching of types, associations, and attributes across the

two versions, ensuring that each element from one model is matched with at most one element from the other. Unmatched elements in the first model are identified as additions, and those in the second as deletions. This matching utilizes custom strategies, implemented via the strategy pattern as described in [Gamma et al.(1997)].

In the next step, matched elements are compared based on their abstract syntax. Identified differences are recorded in data objects: `CDTypeDiffs` for types, `CDAssocDiff` for associations, and `CDMemberDiff` for attributes. We classify 32 types of differences using the `DiffKind` enumeration, which also provides insights into the semantics of model changes, e.g., whether an attribute was moved to a super-type.

The `Diff` objects form a comprehensive data structure, as shown in Figure 1. Each `CDMemberDiff` links to its corresponding `CDTypeDiff`, which includes lists of added and deleted attributes. The `CDSynDiff` aggregates all `CDTypeDiffs` and `CDAssocDiffs`, along with lists for added and deleted types and associations. Each of these `Diff` objects contain a set of `DiffKinds`. The semantic information stored in the `DiffKinds` can be leveraged for further analyses, such as semantic differencing [Maoz et al.(2010)], using iterative exploration of object structure configurations to identify diff-witnesses—instances of the new model not permitted by the old model. Compared to previous semantic differencing operators for CDs [Maoz et al.(2011)], [Ringert et al.(2023)], our approach computes minimal witnesses that trace back to specific differences in the abstract syntax.

V. EXAMPLE CASE: MACOCO, A MODEL-DRIVEN BUSINESS APPLICATION

In this section, we demonstrate our tool’s application through a real-world example: the MaCoCo application, which stands for Management Cockpit for Controlling. Developed at RWTH Aachen University, this web-based enterprise information system was launched in 2016 as a joint project between the *Chair of Controlling* and the *Chair of Software Engineering*. Its goal is to assist university chairs and institutes in managing internal administrative processes, particularly in finance, human resources, and project planning.

MaCoCo functions as a multi-user platform that enables decentralized management while ensuring compliance with central university regulations. Its development is largely model-driven [Adam et al.(2018)], as much of the frontend and backend is automatically generated from domain-specific models using the MontiCore language workbench [Hölldobler et al.(2021)], which promotes maintainability, consistency, and efficient development across the system [Gerasimov et al.(2024)]. Currently, MaCoCo is utilized by a fifth of the chairs and institutes at RWTH Aachen, boasting hundreds of daily users. Notably, three-quarters of the application’s source code is generated, making models essential artifacts that define the software.

The development of MaCoCo and its domain model began in 2016 and continues eight years later. There are a total of 539 model versions. The first 14 versions served only for

documentation, while all subsequent versions were utilized for code generation. The code generator for MaCoCo does not differentiate between associations and compositions, and no model version has included interfaces to date. Consequently, compositions and interfaces will not be addressed in the following analyses.

The graphs presented will visualize the evolution of the domain model, reflecting its changes throughout the development life cycle. The number of commits or versions required to address any issue (bug, feature, refactoring, etc.) varies significantly based on the development style. Therefore, we avoid using versions on the time axes of the graphs, as this could hinder readability and skew interpretations. Instead, the time axes of all charts are consistently divided into quarterly periods.

Figure 3 presents a line chart depicting the domain model’s growth throughout its life cycle. Four lines represent the number of classes, enums, associations, and attributes in the domain model. The x-axis reflects time in quarters, while the y-axis shows the absolute counts. Since multiple versions of the domain model may exist per quarter, the average for each quarter is used. For MaCoCo, the x-axis ranges from the fourth quarter (Q4) of 2016 to the first quarter (Q1) of 2025. Overall, the lines indicate a steady growth of the domain model.

An outlier appears in 2018, where the number of attributes drops sharply before rising again. Comparing two specific versions from these quarters reveals the cause: deprecated classes were removed from the domain model. Additionally, these versions show that the deprecated classes lacked corresponding replacements, indicating feature removal from the application.

Figure 2 displays a zoomed-in excerpt from a larger heatmap of the domain model’s full history. The heatmap is a matrix where each row (y-axis) corresponds to a type (class or enum) that existed at some point in the MaCoCo domain model, and each column (x-axis) represents a quarter in the development history. Each cell is colored based on the number of changes made to a type or its associations during that quarter; darker colors indicate more changes. Quarters in which a type was added or deleted are marked with an `A` for added and a `D` for deleted. The types are sorted by age, from the oldest at the top to the newest at the bottom.

The class `Person` was added to the domain model in Q1 of 2018, early in development, and has remained present since. This quarter also marks the period of the aforementioned refactoring, evident from the number of additions and deletions in that column. The `Person` class has undergone frequent changes (as seen by the dark cells in Q2 2022) and is regularly updated, indicating its centrality throughout development. Additionally, the heatmap suggests an evolutionary coupling [Zimmermann et al.(2003)] between `Person` and `Vertrag` (German for `Contract`), with changes in the `Vertrag` class occurring in 8 out of 10 quarters that also show changes in the `Person` class.

Figure 4 presents two bar charts. The left chart displays the sum of added and deleted lines as two stacked bars per quarter, while the right chart shows the total number of type

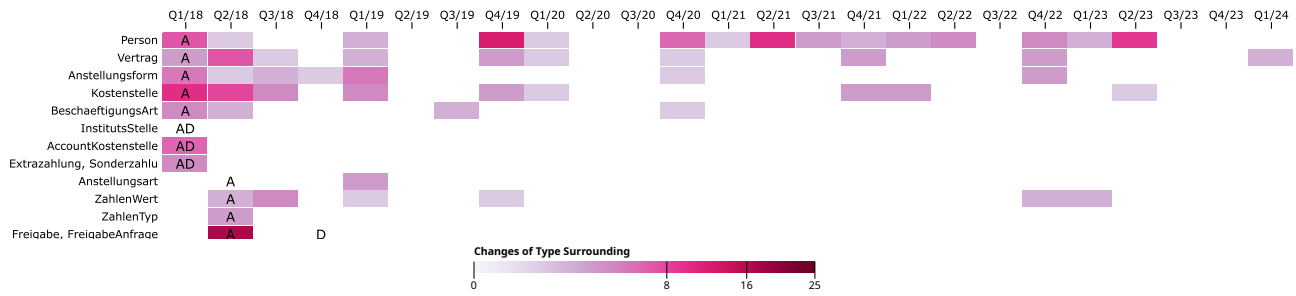


Fig. 2. Excerpt of a larger heatmap visualizing the amount of changes of each type or one of its associations of the MaCoCo domain-model per quarter.

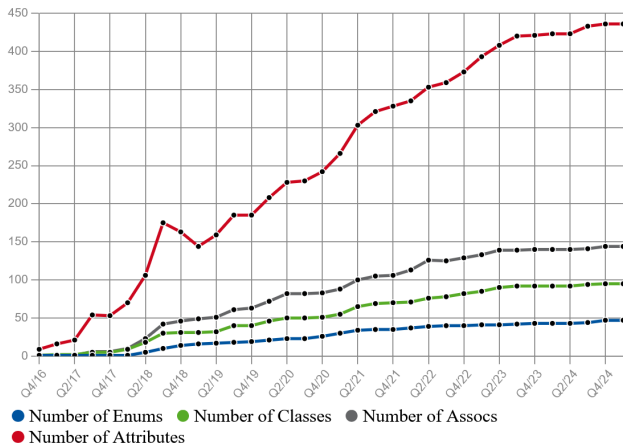


Fig. 3. Line chart of the number of the MaCoCo domain model elements (enums, classes, associations and attributes) over time.

additions, deletions, and changes as three separate stacked bars per quarter. Together, these charts illustrate the difference between simple line-based diffs and type diffs in the abstract syntax of the domain model. While the line-based diff suggests that Q4 of 2019 experienced few changes, the type diff reveals it as the quarter with the most type changes, despite no types being added or deleted.

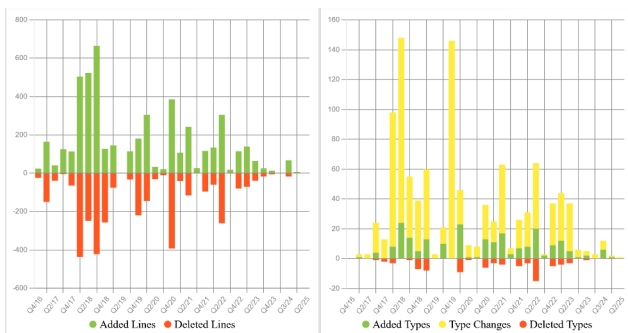


Fig. 4. Bar charts of MaCoCo domain model changes per quarter. The left chart shows the summed up amount added and deleted lines, while the right chart shows the summed up amount of added, deleted and changed types per quarter.

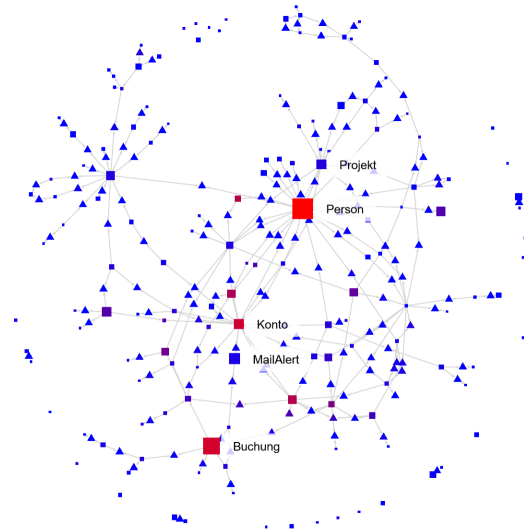


Fig. 5. MaCoCo domain model hot spot graph. Hot = red nodes indicate many changes were done to this element of the model.

Figure 5 features a network graph of the latest version of the domain model using a force-based layout. Types are represented as squares and associations as triangles. The graph conveys three types of information: a) the connections between model elements indicated by edges, b) the size of the types based on their members represented by node size, and c) the frequency of modifications to each domain model element indicated by color. The color scale ranges from blue (cold = few changes) to red (hot = many changes), with the latter referred to as hotspots.

In the hotspot graph, the type *Person* appears as the most frequently changed. Unlike the heatmap in Figure 2, this graph also shows that *Person* is the largest type with the most members. Generally, larger types are modified more often than smaller ones; however, there are notable exceptions. The types *MailAlert* and *Projekt* experienced significantly fewer changes than *Konto* (German for Account), despite being roughly the same size. This indicates that the *Konto* type was less stable during development.

VI. CONCLUSION AND FUTURE WORK

We have developed a toolchain that aggregates and visualizes changes in the domain CD of model-driven development projects. By integrating our dashboard for visualizing model evolution into development methodologies as project retrospectives, we aim to enhance understanding of project development dynamics and facilitate long-term improvements.

Currently, the tool provides an overview of development history and highlights changes to the domain model, but it lacks automatic anomaly detection. Future work will focus on implementing detection mechanisms using heuristics or machine learning, which will require further evaluation of model-driven development processes and suitable training data. Additionally, we plan to enhance our dashboard by better integrating semantic change information from `DiffKinds` into our visualizations.

While our tool relies on a single artifact containing the domain model in UML/P CD syntax, future developments may include support for alternative CD syntaxes and direct extraction of domain models from code, despite the challenges this may present.

REFERENCES

- [Adam et al.(2018)] Kai Adam, Lukas Netz, Simon Varga, Judith Michael, Bernhard Rumpe, Patricia Heuser, and Peter Letmathe. 2018. Model-Based Generation of Enterprise Information Systems. In *Enterprise Modeling and Information Systems Architectures (EMISA '18)* (Rostock, Germany) (*CEUR Workshop Proceedings, Vol. 2097*), Michael Fellmann and Kurt Sandkuhl (Eds.). CEUR-WS.org.
- [Addazi et al.(2016)] Lorenzo Addazi, Antonio Cicchetti, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2016. Semantic-based Model Matching with EMFCompare.. In *Me@ models*. 40–49.
- [Alanen and Porres(2003)] Marcus Alanen and Ivan Porres. 2003. Difference and Union of Models. In «UML» 2003 - *The Unified Modeling Language. Modeling Languages and Applications*.
- [Burch et al.(2005)] Michael Burch, Stephan Diehl, and Peter Weißgerber. 2005. Visual data mining in software archives. In *Proceedings of the 2005 ACM Symposium on Software Visualization (SoftVis '05)*. Association for Computing Machinery.
- [Buschhaus et al.(2024)] Constantin Buschhaus, Arkadii Gerasimov, Jörg Christian Kirchhof, Judith Michael, Lukas Netz, Bernhard Rumpe, and Sebastian Stüber. 2024. Lessons Learned from Applying Model-Driven Engineering in 5 Domains: The Success Story of the MontiGem Generator Framework. *Science of Computer Programming* 232 (2024).
- [Collberg et al.(2003)] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. 2003. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization (San Diego, California) (SoftVis '03)*. Association for Computing Machinery, New York, NY, USA.
- [Dobing and Parsons(2006)] Brian Dobing and Jeffrey Parsons. 2006. How UML is used. *Commun. ACM* 49, 5 (2006).
- [Eick et al.(1992)] S.C. Eick, J.L. Steffen, and E.E. Sumner. 1992. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering* 18, 11 (1992).
- [Fahrenberg et al.(2014)] Uli Fahrenberg, Mathieu Acher, Axel Legay, and Andrzej Wąsowski. 2014. Sound Merging and Differencing for Class Diagrams. In *Fundamental Approaches to Software Engineering*.
- [Gamma et al.(1997)] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. 1997. *Design Patterns: Elements of Reusable Object-Oriented Software*. Prentice Hall.
- [Gerasimov et al.(2025)] Arkadii Gerasimov, Nico Jansen, Judith Michael, Bernhard Rumpe, and Sebastian Will. 2025. A Model-Driven Approach to Design, Generation, and Deployment of GUI Component Libraries. In *Proceedings of the 18th ACM SIGPLAN International Conference on Software Language Engineering (Koblenz, Germany) (SLE '25)*. Association for Computing Machinery, New York, NY, USA.
- [Gerasimov et al.(2024)] Arkadii Gerasimov, Peter Letmathe, Judith Michael, Lukas Netz, and Bernhard Rumpe. 2024. Modeling Financial, Project and Staff Management: A Case Report from the MaCoCo Project. *Enterprise Modelling and Information Systems Architectures - International Journal of Conceptual Modeling* 19 (2024).
- [Harel and Rumpe(2004)] David Harel and Bernhard Rumpe. 2004. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer Journal* 37, 10 (2004).
- [Hölldobler et al.(2021)] Katrin Hölldobler, Oliver Kautz, and Bernhard Rumpe. 2021. *MontiCore Language Workbench and Library Handbook: Edition 2021*. Shaker Verlag.
- [Kehrer et al.(2011)] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2011. A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning. In *International Conference on Automated Software Engineering (ASE'11)*.
- [Kehrer et al.(2013)] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2013. Consistency-Preserving Edit Scripts in Model Versioning. In *International Conference on Automated Software Engineering (ASE)*.
- [Küster et al.(2009)] Jochen M. Küster, Christian Gerth, and Gregor Engels. 2009. Dependent and Conflicting Change Operations of Process Models. In *Model Driven Architecture - Foundations and Applications*.
- [Küster et al.(2008)] Jochen M. Küster, Christian Gerth, Alexander Förster, and Gregor Engels. 2008. Detecting and Resolving Process Model Differences in the Absence of a Change Log. In *Business Process Management*.
- [Maoz et al.(2010)] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2010. A Manifesto for Semantic Model Differencing. In *Proceedings Int. Workshop on Models and Evolution (ME'10) (LNCS 6627)*. Springer.
- [Maoz et al.(2011)] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2011. CDDiff: Semantic Differencing for Class Diagrams. In *ECOOP 2011 - Object-Oriented Programming*, Mira Mezini (Ed.). Springer Berlin Heidelberg.
- [Ohst et al.(2003)] Dirk Ohst, Michael Welle, and Udo Kelter. 2003. Differences between versions of UML diagrams. *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering* (2003).
- [Ringert et al.(2023)] Jan Oliver Ringert, Bernhard Rumpe, and Max Stachon. 2023. On Implementing Open World Semantic Differencing for Class Diagrams. *Journal of Object Technology (JOT)* 22, 2 (2023).
- [Rumpe(2016)] Bernhard Rumpe. 2016. *Modeling with UML: Language, Concepts, Methods*. Springer International.
- [Stahl and Völter(2013)] Thomas Stahl and Markus Völter. 2013. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley.
- [Taentzer et al.(2014)] Gabriele Taentzer, Claudia Ermel, Philip Langer, and Manuel Wimmer. 2014. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software & Systems Modeling* 13, 1 (2014).
- [Thüm et al.(2009)] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning about Edits to Feature Models. In *Proceedings of the 31st International Conference on Software Engineering*.
- [Zimmermann et al.(2003)] T. Zimmermann, S. Diehl, and A. Zeller. 2003. How history justifies system architecture (or not). In *Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings*.