JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

A Variance-Based Drift Metric for Inconsistency Estimation in Model Variant Sets

Karl Kegel, Sebastian Götz, Ronny Marx, and Uwe Aßmann
Technische Universität Dresden, Germany

ABSTRACT

Nowadays, iterative development has become a state-of-the-art engineering process. The shared artifacts, i.e., models, must be edited collaboratively to make iterative development work in large engineering teams. The typical non-real-time collaboration workflow consists of three phases: The collaborators create copies of the model; Each collaborator edits their variant (copy) of the model; The collaborators merge the edited models back into one. During the merge phase, conflicting changes become apparent and must be resolved. This is a time and resource-intensive task. However, if potential merge conflicts can be detected during the editing phase, the collaborators can take suitable measures in time. This work proposes an early warning system for merge conflicts in model-based development projects. We introduce the novel metric *Drift* to quantify the inconsistency between all co-existing variants of a model. An increase in *Drift* indicates an increase in potential merge conflicts. We evaluate the correctness of the *Drift* for synthetical modeling projects and syntactical model differences. We develop an openly available tool for calculating the *Drift* for arbitrary *Git* repositories.

KEYWORDS Software Metrics, Model Metrics, Model Evolution

1. Introduction

In today's development processes, models are edited concurrently. This applies not only to software engineering but engineering disciplines in general. A model can be any textual artifact following a formal or (semi-)formal specification. UML diagrams (Object Management Group 2024) or Petri-Nets (Reisig 2012) are models in the sense of software engineering. However, the civil engineer's CAD file and the accountant's EXCEL sheet are models, too. In all domains of engineering, collaboration is necessary to improve productivity. This naturally includes collaborative work on models. In real-time collaboration, all collaborators access a single instance of the model. Changes are synchronized as soon as possible (Franzago et al. 2018). *Google Docs*[1] is a prominent example. The two main approaches for realizing real-time collaboration are *Conflict-free Replicated Data Types (CRDT)* (Shapiro et al. 2011) and *Operational Transformations (OT)* (Ellis & Gibbs 1989). Although real-time collaboration is often a desired technique, it is unsuitable for all modeling tasks. For high-level models, real-time collaboration enables the joint development of ideas and concepts. However, for low-level models, modeling workflows become closer to those known from programming. Programmers implement their changes (features) individually before merging them with their colleagues' changes. This allows isolated implementation, testing, and debugging without the interference of unstable changes from other collaborators. Low-level models, e.g., executable automatons or state machines, database schemata, or transformation models, require the same isolated feature-wise development. For example, a robotics engineer must prove the safety of their feature via model-checking before it can be included in the main variant.

A collaborative, non-real-time development process starts with a single model. In the remainder of this work, we call this a fork-edit-merge workflow. After distributing tasks, each collaborator creates their variant (branch, fork) of the model. The

JOT reference format:
Karl Kegel, Sebastian Götz, Ronny Marx, and Uwe Aßmann. *A Variance-Based Drift Metric for Inconsistency Estimation in Model Variant Sets*. Journal of Object Technology. Vol. 23, No. 3, 2024.
http://dx.doi.org/10.5381/jot.2024.23.3.a2
[1] https://www.google.com/docs/about/

single model becomes a set of variants. The phase of individual editing can take from a few hours to multiple weeks. During this time, each collaborator performs an individual model evolution process. Lastly, the collaborators merge their changes back into a common base model. Conflicts and unanticipated results have become apparent and must be handled. Perry et al. conclude that preventing and managing conflicts is both a technical and an organizational challenge as it requires communication, clear responsibilities, consensus building, and support by management decisions (Perry et al. 2001). This makes merging, in general, a labor- and resource-intensive task.

Generally speaking, manifested or potential merge conflicts result from inconsistencies within the variant set. Vice versa, if it is possible to count merge conflicts, their number is a measure of inconsistency. From a practical standpoint, systematic inconsistencies are not introduced by a single action of a particular collaborator. Instead, they build up gradually. Any reduction in inconsistency during the editing phase leads to a decrease in merge effort.

1.1. Problem Statement

We argue that reducing the complexity and frequency of merge conflicts in model-based development is a relevant problem for research and practice. Openly available model repositories are rare and thus unsuitable for large-scale merge conflict analysis. However, such works exist for *Git* repositories. The survey of Zou et al. empirically underpins the relevance of branch-based workflows in *Git* projects (Zou et al. 2019). Ghiotto et al. conducted an automated analysis for merge conflicts in open-source *JAVA* software development projects. They found that 2.6% of all *Git* merge attempts fail with conflicts (Ghiotto et al. 2020). The *Git* merge algorithm is textual and not considered model merging. Merge algorithms that are aware of syntax are called *structured*. However, we argue that surveys of code merge conflicts are transferable to conflicts in modeling - at least for such models for which the fork-edit-merge workflow applies. We base this argument on studies comparing structured, semi-structured, and unstructured merges. A structured merge interprets code not as text but as an abstract syntax tree (AST). Merging two ASTs is a syntactical model merge. Works from Apel et al. and Cavalcanti et al. show that structured and semi-structured differences result in fewer and more fine-grained conflicts (Apel et al. 2012; Cavalcanti et al. 2019). However, the problem of merge conflicts does not vanish. We conclude that conflicting changes are an issue of any fork-edit-merge workflow, independent of the merge algorithm.

1.2. Solution Approach

The earlier the team knows of inconsistencies in their variants, the earlier they can act to prevent complex merge problems. Even if conflicts cannot be prevented, knowing them allows planning a sufficient amount of time to resolve them. In software engineering, checking for inconsistencies is achieved by consecutive merge attempts in the background (Guimarães & Silva 2012). Conflicts are then found and reported to each collaborator. This process can be applied to modeling as well. However, a quantification of the resulting conflict potential is not yet established. Thus, quantitatively monitoring the inconsistency from a management perspective is impossible. It requires gradual consistency metrics based on continuous consistency checking to realize a feasible early warning system. During surveying the literature and consulting with practitioners, we noticed the absence of such metrics. We aim for a novel metric for realizing an early warning system for inconsistency. Collaborators track the state of this metric during their work. The engineering team can act in time to prevent complex merges when the metric reaches a defined threshold or shows unanticipated behavior over time.

In machine learning, the continuous deviation of a model away from its initial nature is called *Concept Drift* (Widmer & Kubat 1996; Bayram et al. 2022). For example, by experiencing new data, a classifier changes its decision model - The model *drifts* over time. In domains where models are used for co-simulation and prediction, measures like correlation and variability are used to study similarities and causalities of model sets. For example, variability-based measures are used in geoinformatics to evaluate similarities in weather-forecast model ensembles (Abramowitz & Gupta 2008). Coping with consistency problems of individually evolved software instances (clone-and-own) in software product line engineering, Tinnes et al. coin the term *Drift* as the difference between an original and a modified copy of a system. We propose *Drift* as a general metric for describing the quantified inconsistency between all the model variant sets' variants. Each introduced inconsistency leads to an increasing drift value. A removed inconsistency leads to a decreasing drift value. The *Drift* is a leading metric (Parker 2000) because it indicates inconsistencies before their manifestation during the actual merge. The engineering team can act if large increases or unexpected changes in the drift metric happen, e.g., conduct a review, fix accidental issues, or reschedule the merge.

1.3. Research Question

This work proposes the novel metric *Drift* for quantifying the inconsistency in a model variant set. Therefore, we aim to answer the following research questions:

- RQ1 What is a definition of *Drift* as a metric for inconsistency that is explainable, robust, and transferable to different modeling paradigms?
- RQ2 How can *Drift* be calculated for pairs and sets of models?
- RQ3 Is the *Drift* metric applicable to branch-based software development projects managed with *Git*?

We define robustness, transferability, and explainability as follows: We define *robustness* as numerical stability. A numerically stable relation maps similar inputs to similar outputs, e.g., introducing a few conflicts leads to a small change in the metric. "Small" and "similar" are, of course, relative and must be discussed in the context of the concrete scenario. We define *explainability* as the ability to interpret results intuitively. The reason why the *Drift* has a certain value or trend must be understandable to the metric user. In consequence, the *Drift* metric must not rely on a black-box algorithm. We define *transferability* as the generality of the *Drift* definition and calculation to

apply to different modeling paradigms. Transferability particularly includes language-agnosticism. We define *applicable* as the (A) practical ability to be used and (B) the usefulness, i.e., plausibility, of the results.

1.4. Method and Contribution

This work employs an engineering-oriented method and defines the drift metric using a constructive approach. We start with a formal abstraction of the problem space in sec. 3 *Abstraction*. We continue with building an intuitive metaphor for drift in sec. 4. Section 5 defines the drift based on the introduced formalism and intuition. We evaluate our approach in the *Evaluation* in sections 6 and 7. In 6, we use a configurable model generator to simulate fork-edit workflows in variant sets. In 7, we evaluate the transferability of the drift metric to software development projects. Therefore, we develop the *Driftool* application for the drift calculation of *Git* repositories. This work contributes the novel *Drift* metric for model variant sets. Minor contributions are the *GraphGentool* graph model generator and the *Driftool* analysis tool.

2. Background

2.1. Model Merging and Differencing

Different classes of merge strategies are known in the literature (Mens 2002; Apel et al. 2011). The construction of a merge relies on the model differencing operation used. Selecting a particular differencing strategy significantly impacts the execution time and quality of the detected conflicts (Apel et al. 2012; Cavalcanti et al. 2019). Textual differencing, as used by *Git*, compares the textual representations of a model pair line by line. It ignores syntax and semantics but allows the comparison of arbitrary textual documents. Semi-structured and structured differencing operations work on the abstract syntax tree of the model. Semantic differencing includes the semantics of the model. Showing the semantic identity of two models is generally undecidable (Berzins 1986). Tools such as *EMFCompare* (Brun & Pierantonio 2008) and *UMLDiff* (Xing & Stroulia 2005) use a structured difference. Some approaches extend the AST comparison with simple semantic analysis, e.g., for matching synonyms of identifiers. Important research areas are model difference identification (matching) (Kolovos et al. 2009) and difference representation (Cicchetti et al. 2007). Cicchetti et al. propose the language-based representation and management of model differences. They represent model differences as difference models (Cicchetti et al. 2008). Model comparisons can be advanced to suggest repair operations. Such techniques are feasible for homogenous and heterogenous model sets, as shown by Diskin and König (Diskin & König 2016; König & Diskin 2017). Brindescu et al. show an AI method to predict the difficulty of merges (Brindescu et al. 2020). Owhadi et al. investigate using artificial intelligence to predict and resolve merge conflicts (Owhadi-Kareshk et al. 2019). However, human assistance is still required to resolve complex conflicts.

2.2. Metrics and Technical Debt

The number of potential merge conflicts in a modeling project is a form of technical debt. Technical debt is a metaphor for the number of deficiencies and shortcomings in an engineering project that negatively influences its future quality (Seaman & Guo 2011; Kruchten et al. 2012). Metrics are used to measure the technical debt of a project. Numerous metrics exist for code-based software development (programming) projects, as collected by Fenton and Bieman (Fenton & Bieman 2014). Although model metrics are less prominent as code metrics, state-of-the-art modeling tools also support similar metrics on models. The *IBM DevOps Model Architect*[2] can retrieve different metrics for UML models' size, coupling, inheritance structure, and visual representation. Berger and Guo propose model metrics for analyzing variability models (Berger & Guo 2014). El-Sharkawy et al. survey model-based metrics for software product lines (El-Sharkawy et al. 2019). Doan and Gogolla show how to integrate model quality metrics at the metamodel level to support quality assurance while modeling (Doan & Gogolla 2019). One can use metrics to get information about whole engineering projects besides single models and code files. Yamamoto et al. investigate metrics based on meta-information, e.g., the number of stars and issue number on *Github*, to estimate the quality of a repository without an in-depth code analysis for research purposes (Yamamoto et al. 2020). Furrer introduced the concept of *Managed Evolution* (Furrer 2019). Managed evolution aims to continuously improve the quality of a software product according to several criteria like business value, changeability, and dependability. Therefore, metrics for measuring these criteria are required. Liliental presents such metrics for sustainable software architectures (Lilienthal 2019).

3. Abstraction

The target of the collaboration is a model $m \in \mathcal{M}$. $\mathcal{M}$ denotes the set of all valid models of a metamodel $\mathbf{M}$. $\mathcal{M}$ contains the empty model $\epsilon \in \mathcal{M}$. Each m denotes a set of model elements itself. The model m edited by particular collaborator $c \in \mathcal{C}$ is indexed m_c. Collaborators evolve their models through atomic changes $\delta : d_1, d_2, ..., d_j$ with $\delta \in \mathcal{D}$. $\mathcal{D}$ is the set of all valid modification sequences (delta sequences). A δ executed by a specific collaborator is indexed δ_c. Conceptually, δ is a partially ordered set of atomic change operations. Individual d_i within a δ can be reordered as long as they are not dependent on each other. This concept is known as a *Delta Operation Sequence* (Clarke et al. 2010). We can use it to express incremental model transformations $\delta : \mathcal{M} \rightarrow \mathcal{M}$. A further definition of $\mathcal{D}$ depends on the nature of $\mathbf{M}$. An evolution function $\alpha : \mathcal{M} \times \mathcal{D} \rightarrow \mathcal{M}$ applies a delta sequence to a model. We call an evolution *stable* if $|\delta| << |m|$ with $|\_|$ as the size of contained elements. In other words, a δ is stable if it only changes a "small" subset of the model's elements. A collaborator transforms their initial model m_c^0 into the final model m_c^τ by executing $\alpha(m_c^0, \delta_c)$. There exist a number of intermediate models m_c^i with $0 \leq i \leq \tau, i \in \mathbb{N}$. An intermediate model m_c^i is reached by applying a δ up to its i-th element d_i to m_c^0. We call m_c a variant and m_c^i

[2] https://www.ibm.com/docs/en/dma

a version. The set of the latest version of each variant is called $\mathcal{M}^\tau$ with $\mathcal{M}^\tau \subseteq \mathcal{M}$ We simplify the abstract collaboration scenario so that all m_c^0 are equal, i.e., all individual model copies are created simultaneously. This does not impact the general problem because all $m \in \mathcal{M}$ have a common infimum m_{root} in $\mathcal{M}$. It is at least $m_{root} = \epsilon$. Each model m_c^0 is an evolution of m_{root} with a specific δ from m_{root} to m_c^0. In the case of identical m_c^0, this delta sequence is empty.

The function $\Delta : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ calculates a measure of inconsistency between two models, e.g., the number of merge conflicts. Δ must calculate a measure of conflict growing monotonously with the conflict's severity, i.e., it requires "countable conflicts". Additionally, Δ must be bounded, i.e., its result must differ from ∞. For example, a semantic Δ may calculate the (possibly infinite) number of instances hat do not match. Such a Δ is either unsuitable in our context or must be constrained. We allow additional implicit input parameters for concrete implementations of Δ for allowing, e.g., 3-way comparisons or metamodel-aware comparisons.

We define the *Environment with Drift* as a tuple $\mathcal{E}$. In summary, $\mathcal{E}$ describes a set of individually evolving models based on a common root model.

$$\mathcal{E} = (\mathcal{C}, \mathcal{M}, \mathcal{M}^\tau, \mathcal{D}, \alpha, \Delta)$$

4. Euclidean Space Metaphor

This section introduces an intuition for drift based on $\mathcal{E}$ from sec. 3. We use this intuition as the foundation for the drift definition in sec. 5.

We use the Euclidean space as the space of variability. Points represent models m. Here, we use the two-dimensional Euclidean space. The metaphor can be transferred to any dimensionality > 0, although this may no longer lead to an intuitive visualization. The dimensions of the system are unitless and have a linear scale. The pairwise Euclidean distance of two points represents the number of inconsistencies (conflicts). An empty model ϵ is located in the origin point. An environment $\mathcal{E}$ can be represented in this metaphorical space as follows:

1. Each $m^\tau \in \mathcal{M}^\tau$ becomes a (model-)point.
2. Use Δ to calculate the distance for each point combination.
3. Align (embedd) the points so that their Euclidean distances match their Δ-distances. If such an embedding is not possible, find the embedding with minimal error.

We call this representation a metaphor because it simplifies the problem. First, the dimensions used for the variability space do not have a defined unit. One can interpret them as a linear proportional scale regarding the amount of inconsistency. However, there is no mathematically sound definition of their unit. Second, a conflict-free embedding of $\mathcal{M}^\tau$ is, in general, not possible. The system contains a certain amount of error (stress). Consequently, geometric points may violate the triangle inequality under the Δ-distance. However, the stressed geometric points fulfill the triangle inequality under the Euclidean distance. Although the representation of $\mathcal{M}^\tau$ as a point cloud in the two-dimensional Euclidean space is a strong simplification, it is intuitive to grasp and usable for visualization. If the point cloud of $\mathcal{M}^\tau$ is more scattered (large volume / low density), it contains many inconsistencies. If the point cloud is less scattered (small volume / high density), it contains fewer inconsistencies. Outlier points with large distances to all other points indicate model variants with many inconsistencies with the remaining variants.

Figure 1 visualizes three different $\mathcal{E}$ after several changes. The empty root model m^ϵ was evolved to the base model m^0 by applying a sequence of changes δ_0. Figure 1 (a) shows a scenario with three collaborators, a, b, and c. Collaborator a performs two delta operations, b performs three delta operations, and c performs one. The $m \in \mathcal{M}$ are depicted as points. The $m \in \mathcal{M}^\tau$ have an additional blue coloring. Figures 1 (b) and (c) do not show the individual delta operations. The Δ-distance of m_a^τ and m_b^τ is depicted as an orange arrow. Figures 1 (a) and (b) show an increasing inconsistency with each delta operation. The points are *drifting* away from each other. Assuming that a delta does not introduce new inconsistencies, the position of the point remains the same. Assuming that a delta decreases the inconsistency, the points move closer together. Figure 1 (b) represents a variant set with more inconsistencies as in fig. (c). The scattering is a metaphor for the inconsistency in a model variant set. A problem of the simplification is the stressed position of m^0. If we define Δ as the number of merge conflicts in a history-based merge, neither of the models in $\mathcal{M}^\tau$ have merge conflicts with their ancestor m^0. This leads to each distance $\Delta(m_x^\tau, m^0)$ being 0. However, the points of m_a^τ, m_b^τ, and m_c^τ have conflicts in between each other. As a result, m^0 is placed somewhere in the center of the point cloud trying to minimize the stress. The same applies to m^0 and ϵ. If we assume a history-based Δ, m^0 and ϵ must be located in the same position. The shown figures are correct for a $\Delta(m^0, \epsilon) > 0$.

Figure 2 visualizes the evolution of the scenario from fig. 1 (a). In 2 (t=1), a and b applied a single delta operation respectively. Because of the stress reduction, m^0 is located in the center. In 2 (t=2), c created their variant and added a conflicting delta. Now, the point cloud spans a 2D area, e.g., a triangle. This process continues in fig. 2 (t=3), (t=4), and (t=5). The triangle's area grows. Figure 2 *evolution* shows all steps of the model evolution process layered over each other.

This plot is our visual metaphor for an *Environment with Drift*. For a set of model variants, we understand the drift as a measure of the scatteredness of the point cloud of the model embeddings into an Euclidean space. We call the visual embedding of variants, as described in this section, a *Drift Plot*.

5. Drift Definition and Computation

This section proposes a bottom-up *Drift* definition based on the metaphor in section 4.

5.1. Definition by Construction

In the following, we define a process that aims to represent the intuition of sec. 4 closely. First, apply Δ to all model pairs in $\mathcal{M}^\tau$ for constructing the distance matrix $\mathbf{A}_{\mathcal{M}^\tau}$. Assume the

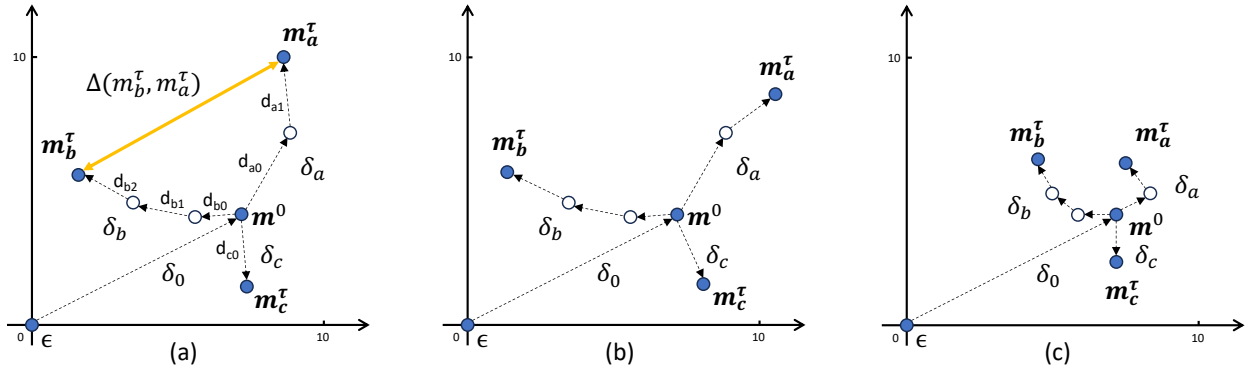

Figure 1 Three different *Environments with Drift* visualized in the metaphorical drift space.

models in $\mathcal{M}^\tau$ form a fully connected system. We write β for the computation function of the distance matrix $\beta(\mathcal{M}^\tau, \Delta) = \mathbf{A}_{\mathcal{M}^\tau}$. Second, choose a *Multi-dimensional Scaling (MDS)* function $\omega : \mathbf{A} \to \mathbb{R}^3$. *MDS* stands for a group of algorithms for dimensionality reduction. Different *MDS* algorithms are suitable for different domains (Borg & Groenen 2005). We embed the models into the $\mathbb{R}^3$. The embedding error (stress) of the *MDS* decreases with increasing dimensionality. However, a dimensionality higher than the $\mathbb{R}^3$ breaks our intuition of a visualizable point cloud. Third, apply ω on $\mathbf{A}_{\mathcal{M}^\tau}$. This results in a set of 3D points $P_{M^\tau} := \omega(\mathbf{A}_{\mathcal{M}^\tau})$. Fourth, let the *drift* γ be 0 if $|P_{\mathcal{M}^\tau}| < 2$ because they are trivial cases. Let the *drift* be the *mean absolute median deviation (MAD)* $\widetilde{d}_{0,5}$ of $P_{\mathcal{M}^\tau}$ otherwise. The MAD is a statistical measure of data points' variability (scatteredness) around their median (Toutenburg & Heumann 2008). The MAD of a dataset $\{x_1, x_2, ..., x_n\}$ is defined as

$$\widetilde{d}_{0,5}(x, \ldots, x_n) := \frac{1}{n} \sum_{i=1}^{n} |x_i - \tilde{x}|$$

This leads to the drift definition:

$$\gamma(\mathcal{M}^\tau, \Delta, \omega) := \begin{cases} 0 & \text{if } |\mathcal{M}^\tau| < 2 \\ \widetilde{d}_{0,5}(\omega(\beta(\mathcal{M}^\tau, \Delta)) & \text{if } |\mathcal{M}^\tau| \geq 2 \end{cases}$$

A fixed Δ and ω shortens the notation.

$$\gamma(\mathcal{M}^\tau) = \gamma(\mathcal{M}^\tau, \Delta, \omega)$$

5.2. Drift Trend

Let $\{\mathcal{M}_1^\tau, \mathcal{M}_2^\tau, ...\mathcal{M}_n^\tau\}$ be the set of $\mathcal{M}^\tau$ at different points in time, i.e., let it be the history of $\mathcal{M}^\tau$. Let $<$ be a strict total ordering relation based on the timestamp (lower index) of $\mathcal{M}_t^\tau$. We define the *Drift Trend* Γ as the application of γ on each version of $\mathcal{M}_t^\tau$. Γ defines a strict, totally ordered set.

$$\Gamma := (\{\gamma(\mathcal{M}_1^\tau), \gamma(\mathcal{M}_2^\tau), ...\gamma(\mathcal{M}_n^\tau)\}, <)$$

5.3. Drift Workflow

The practical drift-aware modeling workflow using the above definition looks as follows:

1. Find a suitable Δ. The Δ can be *Git* for line-based, *EMF-Compare* for *Ecore* syntax, tool-specific, a custom implementation, etc.
2. (Automatically) Calculate the adjacency matrix $\mathbf{A}_{\mathcal{M}^\tau}$ using Δ on a project-dependent regular basis, e.g., hourly, daily, weekly, or on change in the build process. For Git, we provide full tool support (sec. 7).
3. (Automatically) Calculate the drift value and plot from the matrix using the MDS. Given a CSV matrix, we provide full tool-support.
4. The team observes the drift value and plot and reacts to unanticipated changes, e.g., sudden drift increase. Possible reactions are an early merge, review, bugfix, rollback, or revision.

5.4. Conclusion

This section fulfills RQ2 by giving a formal algorithm for the drift calculation for pairs and sets of models. The drift definition fulfills RQ1, as stated below.

5.4.1. Explainability The drift metric is explainable as it relies on the visual metaphor from sec. 4. Additionally, the drift computation produces the drift plot (Euclidean point cloud) as an intermediate result. Changes in the drift can be traced back to changes in the drift plot. Individual variants can be identified. Consequently, the drift definition fulfills the explainability requirement of RQ1.

5.4.2. Robustness The drift metric is robust because its calculation is based on numerically stable measures. The $\widetilde{d}_{0,5}$ is a measure of variability. $\widetilde{d}_{0,5}$ is numerically stable because it is not prone to large fluctuations by outliers as it (A) uses the median as a reference point and (B) uses linear average distances. Consequently, the drift definition fulfills the robustness requirement of RQ1.

5.4.3. Transferability The drift definition is not dependent on a concrete Δ. This makes the drift definition transferable to different modeling languages, i.e., language-agnostic. The only requirement to transfer the drift metric to a specific language is the existence of a suitable Δ. The internals of a concrete implementation of Δ do not change the conceptual properties of the

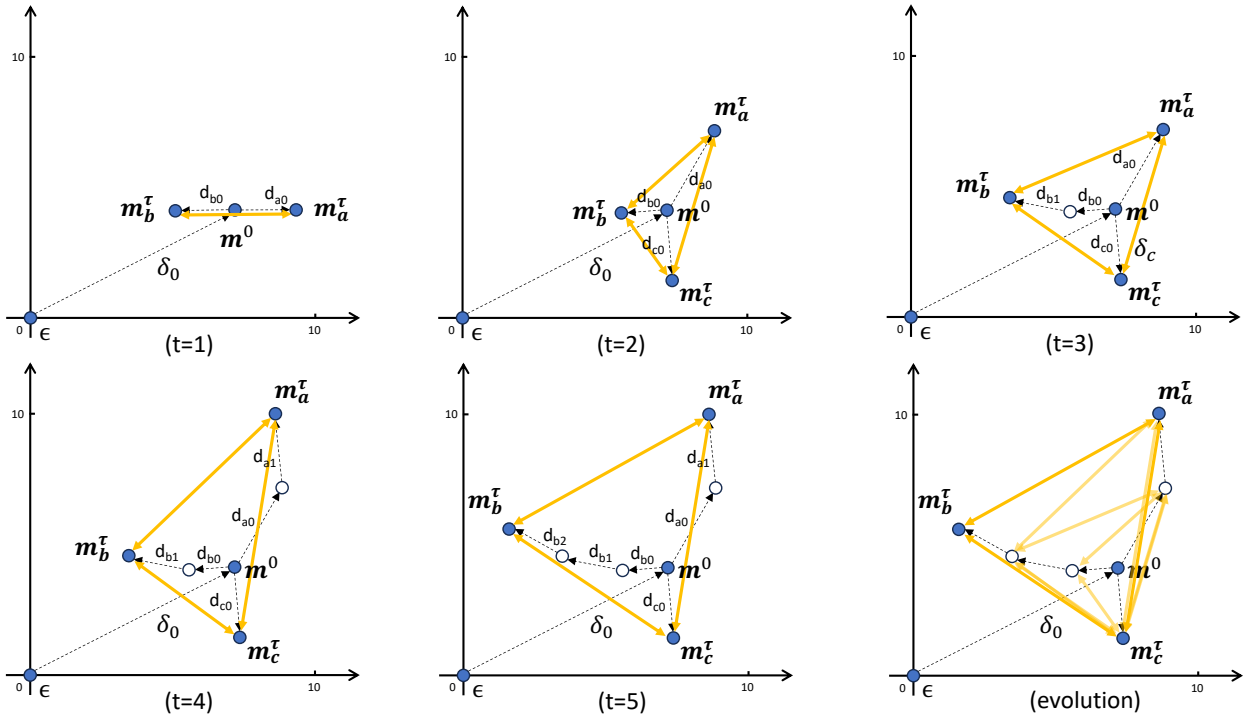

Figure 2 Visualization of the step-wise drift increase in the metaphorical drift space.

drift metric. It changes their absolute values. A different Δ may change the interpretation of the concrete drift values because different Δ could weigh merge problems differently or use different merge approaches. It can be syntactical or semantic as long as the requirements of the *Environment with Drift* are met. For a concrete project, the Δ has to be determined by experienced modelers. Denoting or communicating a concrete drift value, one must always name the Δ function used. Consequently, the drift definition fulfills the transferability requirement of RQ1.

6. Drift Trend Analysis of Synthetic Model Variant Sets

This section evaluates the drift metric's correctness experimentally using a general example. We developed the model generator *GraphGentool* as an evaluation tool and a minor contribution to this work. We provide a reproduction package for this evaluation (Kegel 2024) containing a copy of the developed tools and automation scripts.

6.1. Evaluation Goal

In this evaluation, we aim to validate the correctness of the proposed mathematical drift definition experimentally, i.e., does the drift metric behave as expected for known inputs? The drift metric is, per definition, language-agnostic, i.e., it needs to be instantiated for a particular language. It is impossible to experimentally evaluate the metric on the infinite number of possible modeling languages and Δ-functions. Therefore, we decided to select a representative language and Δ. We use an *Ecore* metamodel of a hierarchical labeled graph as the language because it abstracts common modeling languages. The fact that *Ecore* is used on M2 does not influence the results, i.e., the used graph language could be designed in any other metamodeling framework. However, *Ecore* comes with *EMFCompare* (Brun & Pierantonio 2008) - a well-established syntactical model comparison tool. We argue that the experimental results from such an abstract metamodel and syntactical Δ are generalizable to a large class of practical modeling scenarios.

We investigate the behavior of the drift metric compared to the following list of hypotheses.

- H1 The consecutive editing of the model variants leads to an increasing drift over time.
- H2 A higher number of variants leads to a higher amount of drift in the variant set. We assume that each variant receives equally many deltas.
- H3 Randomly distributed edits in each model variant lead to more drift than edits focusing on disjoint subsets of the models. In other words: if the collaborators make their edits randomly all over their models, many inconsistencies occur. If the collaborators make their edits in structurally distinguished regions of their models only, a few inconsistencies occur..
- H4 With an increasing amount of structural separation (modules, hierarchies) in the models, the effects of H3 are amplified.

We found these hypotheses in discussions with practitioners from research and practice. We use the experiments presented in the following to collect empirical evidence and, consequently, to either reject or accept the hypotheses. If the drift metric behaves as expected in the hypotheses, we consider it correct.

6.2. Tooling: Synthetic Graph Generation

For conducting this evaluation, we require exemplary model variant sets with specific properties, e.g., varying number of variants, specific model sizes, different model modularities, etc. The properties need to be configurable to check against the different hypotheses. Using real-world modeling projects is not feasible because we need full control of the variant set's parameters to investigate cases of interest in isolation. As a result, we developed a configurable generator for model variant sets. The generator, called *GraphGentool*[3] is openly available.

The *GraphGentool* uses a custom *Ecore* metamodel. The metamodel is shown in fig. 3. The root element is the *Graph*. The *Edge* has no direction but a fixed order of the *Nodes*. Each *Node* has a unique name. A *Region* contains a *Graph*. Thus, it is the structuring element that builds up hierarchies. We defined the following implicit rules which are not visible in the metamodel diagram: An *Edge* is located in the *Graph* where the first *Node* is located. The composition structure of the *Regions* must form a tree. As a result, the composition structure of the *Graph* is cycle-free. However, the *Edge* structure allows cycles. The metamodel is purposefully designed to be a simple and abstract representation of many modeling languages, e.g., UML class diagrams, Statecharts, or Petri-Nets. Such a general-purpose metamodel reduces the evaluation bias toward the specifics of a particular modeling language. The concretely chosen names of the metamodel elements are transferable to other domains. For example, *Graph* could be called *ClassDiagram*, *SimpleNode* could be called *Class*, and *Region* could be called *Package*. Such renamings have no impact on this experiment's results. The *Label* enum provides the *SimpleNode* with a type. We use colors for simplicity, but semantic types could be used instead. For example, a green *SimpleNode* could have the semantics of a hyper-edge. We further discuss generality concerns in sec. 6.5.

The *GraphGentool* works in two phases. The first phase is the generation of a base model m^0. It is saved as an XMI file. Several properties can be configured:

- Model size as the sum of *Nodes* and *Edges*.
- Average ratio of *Edges* per *Node*.
- Ratio of *Edges* that connect *Nodes* of the same *Region* and *Edges* that connect *Nodes* of different *Regions*.
- Ratio of *SimpleNodes* and *Regions* (structuredness).
- Random seed.

An important aspect of the graph generation algorithm is the distribution of *Regions*. The composition structure of *Regions* forms a tree. During construction, *Regions* are added to the composition tree iteratively. Each iteration adds a new *Region* to a random *Region* already present in the tree. Consequently, the average number of sub-*Regions* per *Regions* decreases the farther away from the root.

The second phase is the generation of a variant set. The *GraphGentool* takes the base model and creates a specified number of copies (variants). The variants are edited by iteratively applying delta operations of a generated delta operation sequence. The edit selection is configured with a probability per edit. As

[3] https://github.com/convidev-tud/emf-graph-gen

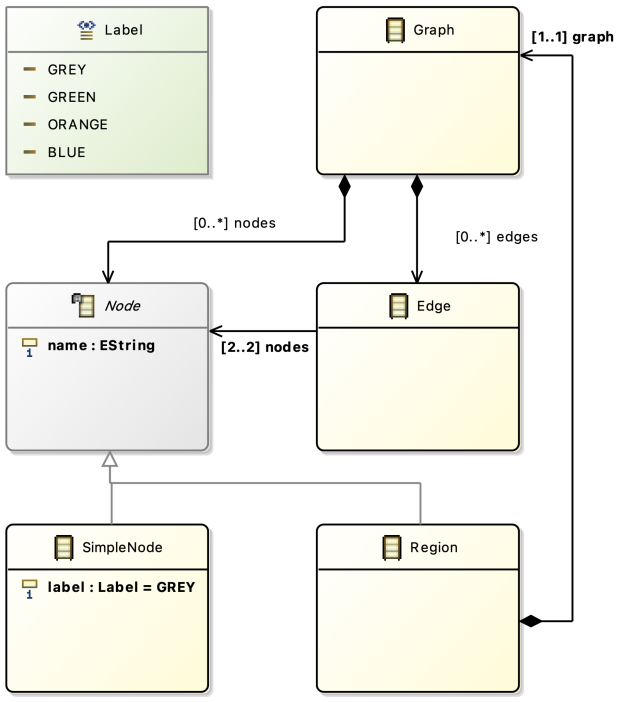

Figure 3 *Ecore* graph metamodel used by the *GraphGentool*

a consequence of editing a variant, the graph configuration can be violated, e.g., the ratio of model elements changes. The delta metamodel contains the operations: *AddNode*, *DeleteNode*, *MoveNode*, *ChangLabel*, *AddEdge*, *DeleteEdge*, and *MoveEdge*. The delta operations *DeleteNode* and *MoveNode* have implication associations to other move and delete operations. This models the required subsequent operations to transform the model into a valid state after an initial edit.

This second phase also has an input configuration:

- The number of variants.
- The number of edits.
- The way of counting the edits. Atomic counting considers each performed delta operation as one step. Non-atomic counting considers an atomic edit and the minimal number of implied atomic edits to reach a valid model as one step.
- The edit focus. A focus of 0.0 leads to randomly selecting the element for editing in the whole graph. An increasing focus up to 1.0 increases the probability that the next edit happens in the same *Region* as the last edit. However, an edit that moves an element can lead to changing the *Region* even if the focus is 1.0
- The selection probability for each edit operation.

If required, an indexed version of the edited variant is saved as an XMI after each non-atomic edit step. The delta model representing the executed edits is also stored as an XMI file.

6.3. Environment & Process

We developed two experiments to evaluate the stated hypotheses H1-H4. Therefore, we analyzed different model variant set configurations (runs). For exp. (1), we repeated each run for

11 different seeds. For exp. (2), we repeated each run for 5 different seeds.

1. Use the *GraphGentool* to generate the model variant set for each configuration of interest.
2. Calculate the Δ-distance matrix **A** after each evolution step of each variant set. We used the *EMFCompare* three-way merge conflict count as Δ function.
3. Use the *Driftool* (sec. 7) to calculate the drift value of each matrix.
4. Collect and group all drift values and export the result as a CSV table.
5. Parse the CSV file for analysis and visualization of the results.

Table 1 Main experiment (1) configuration for H1, H3, H4

Parameter	Values
Non-atomic deltas	30
Number of variants	(5), (9), 13
Model Size	100, 500, 1000
Region Probability	0.05 (shallow), 0.2 (deep)
Edit Focus	0 (none), 0.8, 1 (max.)

Table 2 Experiment (2) configuration for H2

Parameter	Values
Non-atomic deltas	16
Number of variants	3, 5, 7, ..., 25
Model Size	200
Region Probability	0.15, 0.1, 0.05
Edit Focus	0.0, 0.5, 0.75, 1.0

Table 3 Experiment edit probabilities

Delta	Probability
Add SimpleNode	0.15
Add Region	0.05
Delete Node	0.05
Move Node	0.05
Change Label	0.25
Add Edge	0.25
Delete Edge	0.2

The used model variant set configurations are depicted in table 1 and 2. We conducted a separate experiment for H2 with a single model size to achieve a feasible computation time with many branches. Analyzing the drift for different variant set configurations over time (over increasing edits) evaluates H1. We analyze specific parameter configurations to evaluate the drift metric against H1-H4. Varying the number of variants for a fixed model size evaluates H2. Varying the edit focus for a fixed combination of the remaining parameters evaluates H3. Varying the hierarchy depth for each edit focus and a fixed combination of the remaining parameters evaluates H4. We choose 4, 8, and 12 as realistic numbers of variants, i.e., the number of collaborators. Because the base variant is also part of the variant set, the resulting numbers are 5, 9 and 13 variants. We choose three models sizes up to 1000 elements (*Nodes* + *Edges*) for the main experiment. *EMFCompare* performed too slow to conduct experiments with larger models in a feasible timeframe. We choose 0.05 as a suitable *Region*-probability for shallow models and 0.2 for deep models. Assuming a shallow model with 100 *Nodes*, it contains 5 *Regions* with, on average, 20 *Nodes* each. Assuming a deep model with 100 *Nodes*, it contains 20 *Regions* with on average 5 *Nodes* each. We choose an edit-focus of 0 for simulating random edits, 0.8 for simulating a realistic high but not optimal focus, and 1.0 for simulating a maximum focus. We choose 30 as a reasonable number of non-atomic deltas. The number of 30 non-atomic deltas is applied per variant. For each configuration, we observe the whole drift trend so that all edit counts smaller than 30 are covered as well. We argue that larger delta counts are potentially unstable for the chosen model size of 100. In this case, the drift may stagnate at a high level because a maximum number of conflicts (inconsistency) is reached. We do not consider this case because it indicates not a limitation of the drift metric but of the overall modeling workflow. Changes made by a single collaborator in a single variant must not change major parts of a model. The used delta operation probabilities for editing the models are depicted in table 3. Notably, these probabilities are only used to edit the variants and not generate the base model m^0. We argue that these probabilities reflect a practical edit process.

6.4. Results

Figures 5, 4, and 6 show the results of the experiment runs. We show the drift value for each second delta. Notably, this number of deltas is applied per model of a variant set, except m^0. We explain upper outliers with the occurrence of "unlucky" delta sequences. Examples are: One collaborator deletes a *Region* containing many other model elements, which are deleted recursively. Each edit targeting a model element another collaborator has deleted leads to a conflict; Two collaborators with a high edit focus choose the same *Region* for editing. The *GraphGentool* does not prevent these cases. We conducted no systematic time measurements during our experiments. The runtime of the drift metric grows quadratically with the number of variants. Observed runtimes are of a magnitude from a few seconds for smaller models and variant numbers to a few minutes for larger models and variant numbers. The runtime of the comparison algorithm, in this case, *EMFCompare* depends on the model size. *EMFCompare* has a very high time and memory consumption for large models. Depending on the metamodel and computing power, *EMFCompare* could be unfeasible for large models.

Figure 4 shows the results of the experiment for evaluating H1. The plots E1-E3 show the drift trend for 13 branches and fixed model sizes. The remaining variables vary in their range. The results provide support for hypothesis H1, as each plot shows a clear drift increase with an increasing number of deltas. The variance notably increases with the model size.

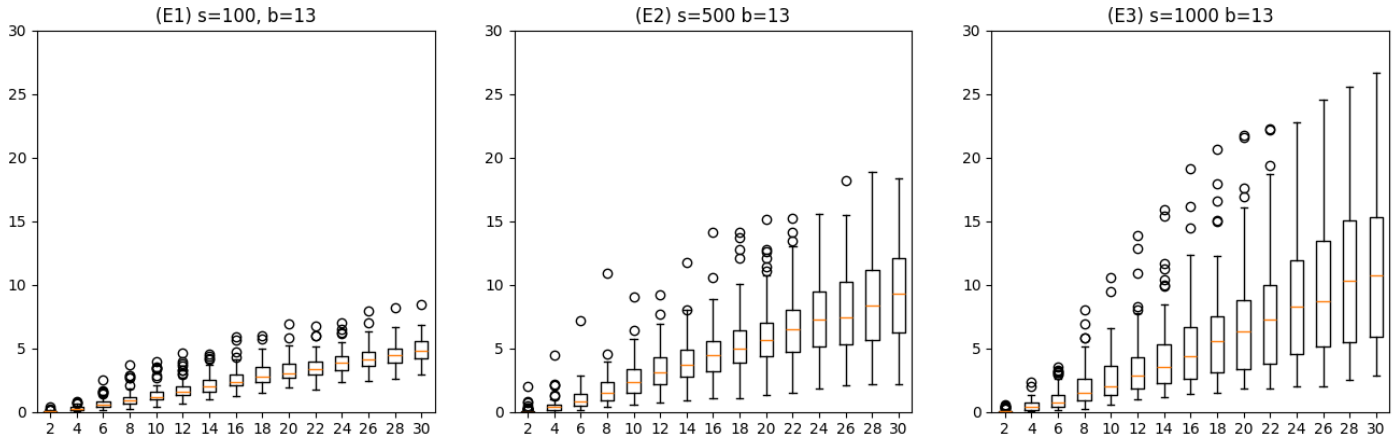

Figure 4 Experiment results for analyzing the drift for an increasing amount of edit operations for different model sizes and a fixed number of branches. The horizontal axis shows the amount of non-atomic edits that were performed. The vertical axis shows the drift. s is the model size in *Nodes* + *Edges*. b is the number of variants (branches).

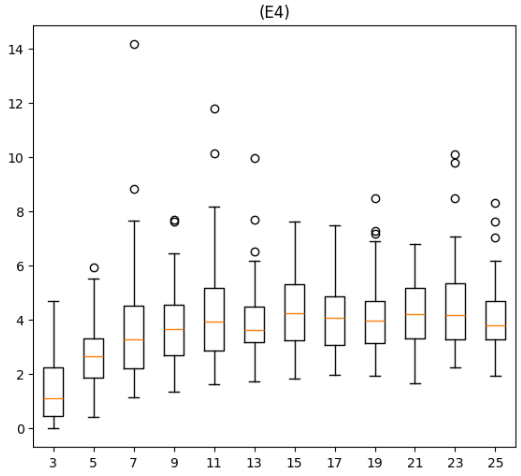

Figure 5 Experiment results for analyzing the drift for an increasing number of branches with constant model sizes and varying graph configurations. The horizontal axis shows the number of variants per set. The vertical axis shows the drift.

We explain this with an increasing influence of the variables *depth* and *focus* on the drift with increasing model size. Figure 5 shows the the results of the experiment for evaluating H2. Plot E4 shows the drift for an increasing number of branches. The remaining variables vary in their range. The results do support H2 for smaller variant sets (< 10 variants). The results do not support hypothesis H2 for larger variant sets because an increasing number of variants only slightly impacts the drift. Figure 6 shows the experiment results for evaluating H3 and H4. We selected a fixed number of 13 branches and a fixed model size of 500. For evaluating H3, we compare the median drift after 30 non-atomic deltas for variant sets of the same modularity. The drift of shallow models with a min. edit focus is 4.42 points higher then the max. edit focus as shown in E5-E7. The drift of highly modular models with a min. edit focus is 8.03 points higher than the max. edit focus as shown in E8-E10. In consequence, the drift metric fulfills hypothesis H3. The drift decrease for deep models is 1.83 times that for shallow models. In consequence, the experiment provides clear support for H4. The experiment also matches the experience that well-organized (focused) edits in a well-structured (hierarchical) model lead to the least amount of merge conflicts. Notably, the worst-case scenario is random edits on well-structured models. We explain this with the high risk of accidentally changing (or removing) many important structural elements in this scenario. The results in fig. 6 show an increasing drift trend with more deltas, which again supports for H1.

In summary, the drift metric behaves correctly for measuring the inconsistency in model variant sets as our experiments showed clear support for hypotheses H1, H3 and H4. We haven't found clear support for H2 in our experiments, but we could see that a weaker hypothesis is supported. Namely, the hypothesis that with an increasing number of branches, the rate of increase of the drift metric decreases.

6.5. Threats to Validity

Our experimental evaluation contains potential threats to validity. Any experimental evaluation only allows conclusions within the observed boundaries. Our experimental setup contains a single metamodel with limited configuration parameters (e.g., the edit focus). This limitation threatens to cause the results to miss relevant cases and be biased toward the specific metamodel. We argue against this threat.

First, we claim that the used metamodel is a suitable representative example because it is abstract and general. Hierarchical labeled graphs are the foundation for many practical modeling languages. A metamodel different from the one presented would naturally change the absolute drift values. However, the relative tendencies in the drift trends are preserved for the same Δ. This also includes the consideration of the model's concrete syntax, e.g., diagrammatic representation. In our experiment, we only considered the abstract syntax of the models. If one defines a conflict counter for the concrete syntax, it can be incorporated in the Δ function. Although this adds a factor to the the drift, the monotony of the trends is preserved.

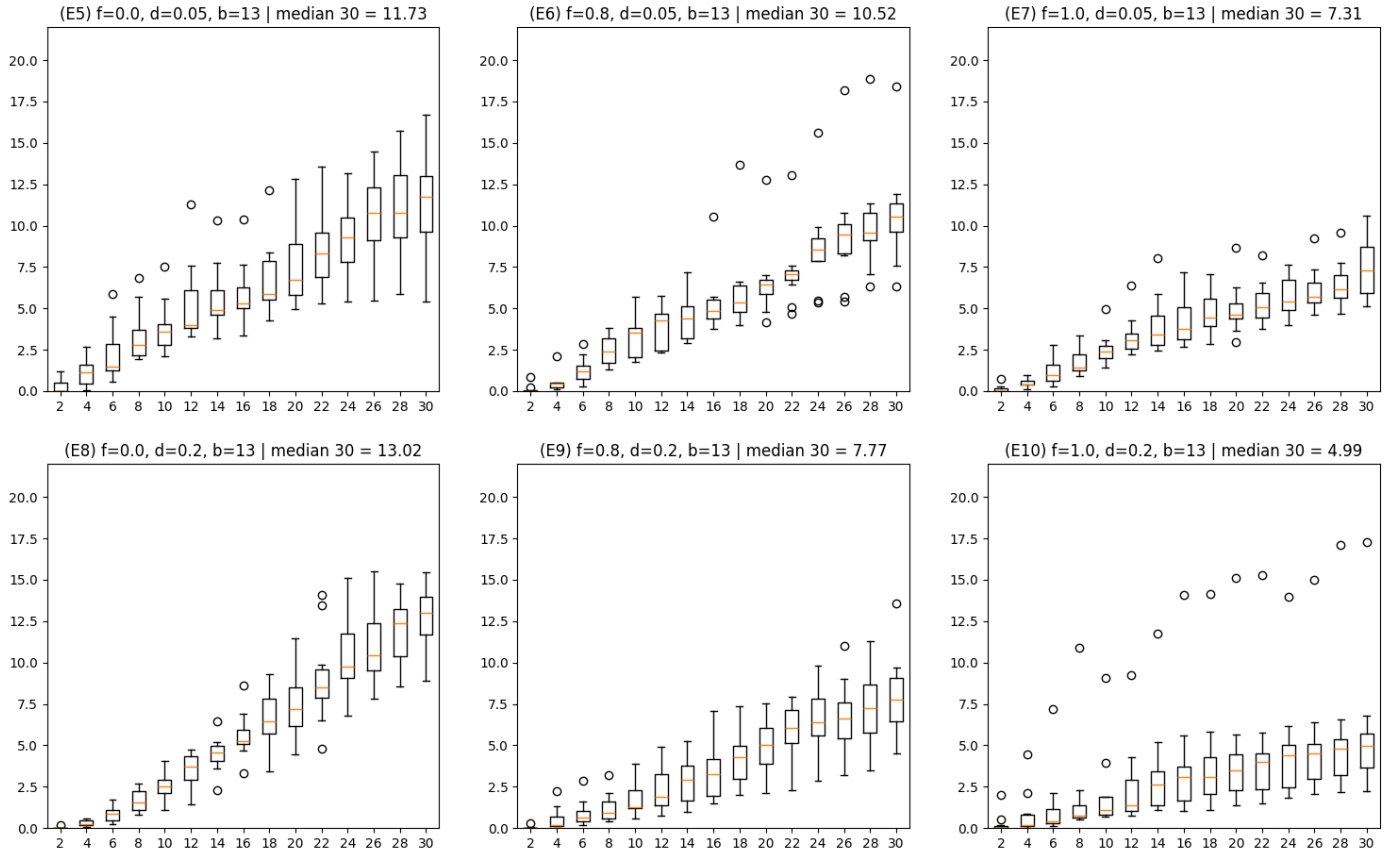

Figure 6 Experiment results for analyzing the drift for different amounts of edit focus and structuredness for 13 branches and a model size of 500 *Nodes* + *Edges*. The horizontal axis shows the amount of non-atomic edits that were performed. The vertical axis shows the drift. f is the probability to perform the next edit in the same *Region* as the last one. d is the parameter for hierarchy depth. $d = 0.5$ says that 50% of the nodes contain sub-graphs.

Second, we claim that the chosen Δ is representative as well. The syntactical model differencing of *EMFCompare* is state-of-the art tooling. Using a different Δ does not change the monotony of the drift trends. The Δ we used is a linear measure. The trend's shape will change if a different Δ reports conflicts using a logarithmic measure. However, the monotony requirement of the Δ function generally leads to the same monotony of the drift trends as observed in the experiments. Different Δ only impact the interpretation of the drift values, e.g., using a syntactical Δ shows a syntactic drift, and using a semantic Δ results in a semantic drift.

Third, we claim that the used configuration parameters and selected configurations are suitable to represent typical edit processes. Used configurations cover shallow and highly structured models, unstructured and well-structured edit operations, different model sizes, and different variant numbers. Varying configuration parameters did not influence the drift trend apart from linear factors. The only exception is an overly high probability for deletions. This leads to degenerated cases because the model size decreases towards the empty model ϵ.

We do not claim that an *Environment with Drift* can be constructed for every modeling scenario. The second part of the evaluation in sec. 7 is an addition for countering both threats. Although this does not eliminate the threat that observed trends might break for specific parameters, we show that the drift is explainable for real-world data.

7. Drift Analysis of Software Development Projects

This section evaluates if the drift metric applies to *Git* projects. To conduct this evaluation, we developed the *Driftool*. The reproduction package (Kegel 2024) contains all artifacts of this evaluation.

7.1. Evaluation Goal

The evaluation goal is to answer RQ1, i.e., if the drift metric is applicable for *Git* projects. In particular, the goal is to evaluate if it is (A) practically possible to calculate the drift for such projects and (B) investigate the usefulness, i.e., plausability, of the results.

7.2. Environment & Process

We adhere to the following process for reaching the evaluation goals:

1. Design and implement a tool for calculating the drift metric.

The tool must be able to use a Δ-function that is feasible for *Git* repositories. The tool must provide visual feedback to the user to ensure the explainability of the results.
2. Use the tool to calculate the drift metric for different open-source software repositories from *Github*[4].
3. Analyze the results regarding their plausibility. Find explanations for the calculated drift.

The drift values of different development projects are not comparable because of the different languages, file structures, etc.. However they are still usable for analysis as they show information on variant clusters and outliers. We aim to explain the observations in the plots with information derived from the repository metadata on *GitHub*.

7.3. The Driftool

We developed the *Driftool*[5] as a tool for automatic drift computation and visualization. The tool is implemented in *Python* and available under an open-source license. The *Driftool* is primarily designed to calculate the drift of *Git* repositories. Users can customize the analysis using a configuration file. Important configurable arguments are the exclusion of certain branches (RegEx), exclusion of certain files (RegEx), only inclusion of certain files (RegEx), The *Driftool* performs exactly one analysis per run, i.e., terminates after completing the analysis. Given a configuration, the analysis process is realized as follows.

1. Create a local copy of the input *Git* repository.
2. Go through each branch and apply the blacklist and whitelist rules. Create a commit to make the changes visible to *Git*.
3. Create the list of branches to analyze.
4. For each branch pair, calculate the Δ-distance. We use the count of conflicting lines as the distance. We attempt to merge and parse the conflict reports to get this number. This leads to $\mathbf{A}$.
5. Calculate the 3D embedding of $\mathbf{A}$. We use the MDS algorithm provided by *sklearn*[6].
6. Calculate the drift as defined in 5.
7. Generate visual and textual reports.

We realized a "bypass" feature for using the visualization and reporting abilities of the *Driftool* for use-cases apart from *Git* repositories. The user can input a precomputed distance matrix $\mathbf{A}$ represented as a CSV file. We used this feature for the experiment in sec. 6. Figure 7 shows an example screenshot of the interactive report created by the *Driftool*. This report shows the drift of a real-world project which we analyzed in the following sec. 7.4. If an HTML report is not required, the *Driftool* exports a simple scatterplot, as shown in fig. 8.

7.4. Results

We created a *Driftool* configuration for each analyzed project. It ignores common file formats that "bloat" up repositories and are of low interest for the analysis, such as PDFs or precompiled JAR libraries. We ignored branches that contained past versions for maintenance or branches that exist purely for documentation. The presented numbers are based on the repository state of the 15th February 2024.

Figure 7 shows the drift plot of the *Vitruv*[7] repository. *Vitruv* is a research prototype developed at *Karlsruhe Institute of Technology*. The analysis covered 6 branches. The plot shows 2 comparably close points in the right area of the plot. One point has a medium distance to these two, and 2 points are further away. One of the 2 close points in the right area is doubled. This indicates no merge conflicts between the two branches. The four closer-located branches all received commits within the last 9 months. The three branches located further away received their last commits more than 2 years ago and thus are more outdated. This is supported by the large amount of commits these 3 branches are behind the main branch.

Figure 8 shows the drift plot of the *taiga-front*[8] repository. *Taiga* is a project management tool for agile development processes. The analysis covered 14 branches. 13 branches form a dense cluster, and 1 branch is a far outlier. We investigated the repository activity and reproduced merges to evaluate the plausibility of the results. The repository has a low activity, and most commits are made directly on the main branch. The scattered branches are feature implementations. 10 of them were already merged into the main in the past. We assume they still exist for feature maintenance purposes. Further investigation of the outlier branch found a larger number of merge conflicts compared to the other branches. Most of them resulted from syntax refactorings in *CoffeeScript* files.

Figure 9 shows the drift plot of the *semantic-kernel*[9] repository. The *semantic-kernel* is an automation tool suite from *Microsoft* to work with large language models. The analysis covered 57 branches. The drift plot shows a dense central cluster of points with a decreasing density with increasing distance from this central region. Outliers are located all around the dense center. The overall drift of 224.10 is high compared to the previous examples. We explain this drift plot and drift with the size and activity in the repository. 11 branches received modifications within the last month, and 218 people have contributed to the overall project. By experience, this leads to less control over the branches, explaining the high scattering in the drift plot. However, the majority of branches only contain small feature implementations. This is the reason for the dense central cluster in spite of the high activity.

The experiment results clearly indicate that the drift metric can be applied to branch-based software development projects. We consider the concrete drift value error-prone due to the line-based differencing algorithm *Git* uses. However, the 3D embeddings, as an intermediate result of the drift computation, provide useful and explainable insights.

[4] https://github.com/
[5] https://github.com/convidev-tud/driftool
[6] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html
[7] https://github.com/vitruv-tools/Vitruv
[8] https://github.com/kaleidos-ventures/taiga-front
[9] https://github.com/microsoft/semantic-kernel

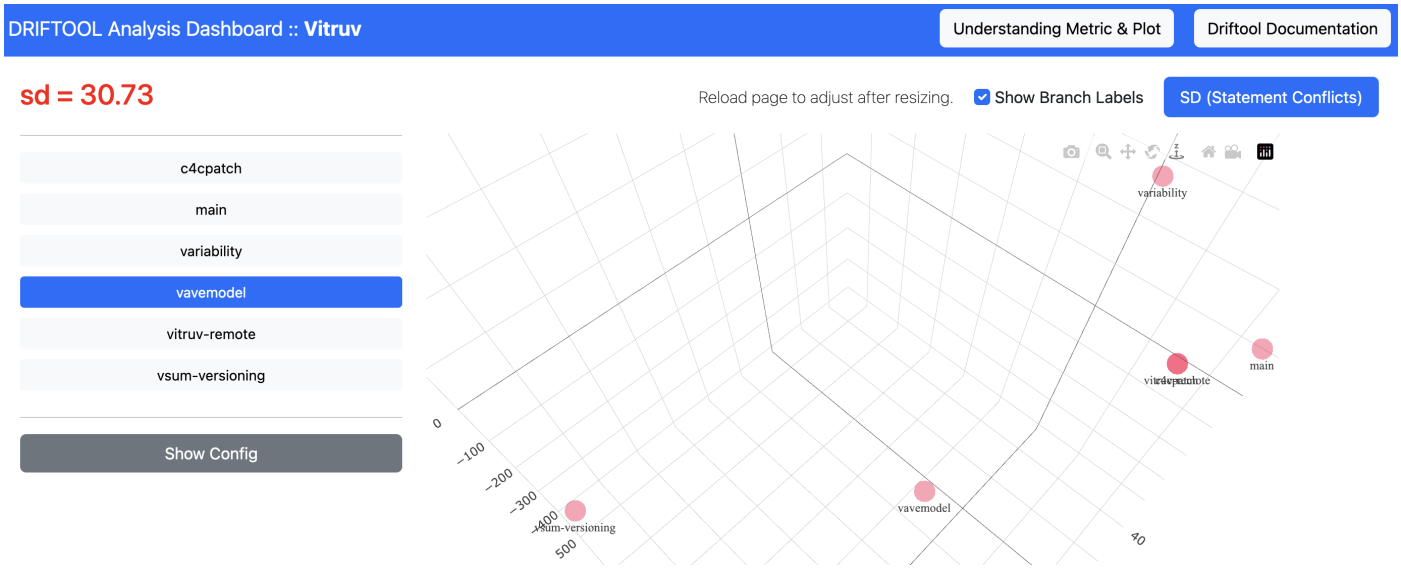

Figure 7 Screenshot of the *Driftool's* interactive HTML report. The shown drift plot results from an analysis of the *Vitruv* repository, presented in sec. 7.4

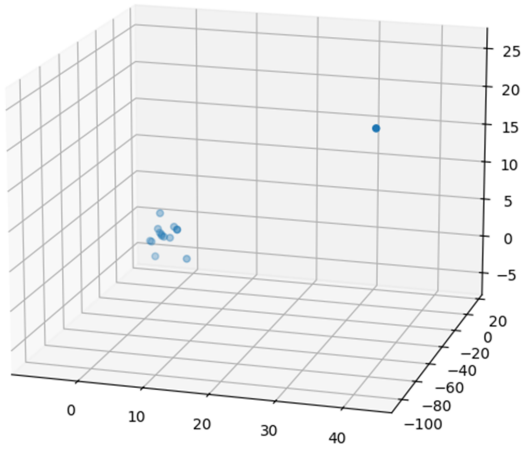

Figure 8 Drift plot of the *taiga-front* repository (drift=13.33).

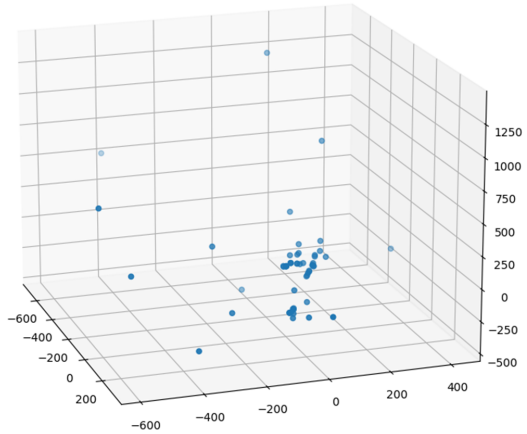

Figure 9 Drift plot of the *semantic-kernel* repository (drift=224.10).

7.5. Threats to Validity

We presented 3 examples of drift calculations for real-world software development projects. This poses the threat that the results are biased towards the analyzed repositories. To address this threat, we analyzed 16 repositories with the *Driftool*. Although this analysis was smaller than for the repositories in this section, we repeatedly found similar patterns in the drift plot and repository metadata. The interactive drift reports of the additional samples are provided in the reproduction package (Kegel 2024). A second threat is the potentially low accuracy of the drift due to *Git*'s line-based differencing. This has not influenced the explainability of the analyzed results. In practice, a repository expert must configure the *Driftool* using in-depth project knowledge.

8. Conclusion

8.1. Summary

This work proposes the novel drift metric as an estimator for the inconsistency in a model variant set. We envision the drift to become a metric for measuring the conflict potential during collaborative modeling. Consequently, the engineering team can take suitable measures to reduce costly merge conflicts early on. We started this work with an abstract problem description. We continued with an intuitive drift definition and a formal constructive definition. In this process, we proposed answers to the research questions RQ1 and RQ2. We conducted a twofold evaluation. First, we evaluated the correctness of the drift metric. The experiments showed that the metric's results align with a list of hypotheses. Second, we evaluated the feasibility of the drift metric from a tool-builder's perspective. We developed the *Driftool* to calculate and visualize *Git* repositories

to answer RQ3. Conducting the evaluation, we validated the proposed answers for RQ1 and RQ2. In particular, we showed transferability by calculating drift based on the two different Δ-distances: *EMFCompare* conflict count for models, and *Git* merge conflict counts for arbitrary text files.

In summary, we assess the proposed drift metric as an explainable, robust, and transferable metric to detect inconsistencies during collaborative modeling early on. It is based on an intuitive definition and can be implemented in practical analysis tools. The drift computation is compatible with state-of-the-art comparison tools such as *EMFCompare* and *Git*.

8.2. Future Work

This work proposes the drift as a gradual inconsistency measure for model variant sets. This is a restricted case of a homogeneous model set. However, we consider the drift metric transferable to general homogeneous and heterogeneous model sets. Investigations of suitable Δ for these scenarios are a target for future work. A promising application for heterogeneous drift metric is the *Virtual Single Underlying Model*, where so-called *Consistency Relations* relate heterogeneous models (Klare et al. 2021). The number of violated consistency relations may be a suitable Δ for a heterogenous model set. Furthermore, we aim to extend the *Driftool* to support different Δ concurrently. We assess this as a requirement for analyzing real-world development projects with different file formats. AI-based approches as presented in (Brindescu et al. 2020) could provide support for classical differencing algorithms. Lastly, future work has to investigate the drift metric in real-world usecases over an extended period of time.

9. Verifiability

The developed evaluation tools *GraphGentool* and *Driftool* are openly available on *Github*. A reproduction package is provided in (Kegel 2024). It contains a copy of the used version of the tools, the automation scripts used to conduct the evaluation, the raw evaluation results, and a set example *Driftool* reports of open-source projects.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263

References

Abramowitz, G., & Gupta, H. (2008). Toward a model space and model independence metric. *Geophysical Research Letters*, *35*(5). doi: 10.1029/2007GL032834

Apel, S., Leßenich, O., & Lengauer, C. (2012). Structured merge with auto-tuning: balancing precision and performance. In *Proceedings of the 27th ieee/acm international conference on automated software engineering* (p. 120-129). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2351676.2351694

Apel, S., Liebig, J., Brandl, B., Lengauer, C., & Kästner, C. (2011). Semistructured merge: rethinking merge in revision control systems. In *Proceedings of the 19th acm sigsoft symposium and the 13th european conference on foundations of software engineering* (p. 190-200). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2025113.2025141

Bayram, F., Ahmed, B. S., & Kassler, A. (2022). From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, *245*, 108632. doi: 10.1016/j.knosys.2022.108632

Berger, T., & Guo, J. (2014). Towards system analysis with variability model metrics. In *Proceedings of the 8th international workshop on variability modelling of software-intensive systems*. New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2556624.2556641

Berzins, V. (1986). On merging software extensions. *Acta Informatica*, *23*, 607–619.

Borg, I., & Groenen, P. J. (2005). *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media.

Brindescu, C., Ahmed, I., Leano, R., & Sarma, A. (2020). Planning for untangling: predicting the difficulty of merge conflicts. In *Proceedings of the acm/ieee 42nd international conference on software engineering*. ACM. doi: 10.1145/3377811.3380344

Brun, C., & Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, *9*(2), 29–34.

Cavalcanti, G., Borba, P., Seibt, G., & Apel, S. (2019). The impact of structure on software merging: Semistructured versus structured merge. In *2019 34th ieee/acm international conference on automated software engineering (ase)* (p. 1002-1013). doi: 10.1109/ASE.2019.00097

Cicchetti, A., Di Ruscio, D., & Pierantonio, A. (2008). Managing model conflicts in distributed development. In *Model driven engineering languages and systems: 11th international conference, models 2008, toulouse, france, september 28-october 3, 2008. proceedings 11* (pp. 311–325).

Cicchetti, A., Di Ruscio, D., Pierantonio, A., et al. (2007). A metamodel independent approach to difference representation. *Journal of Object Technology*, *6*(9), 165–185.

Clarke, D., Helvensteijn, M., & Schaefer, I. (2010). Abstract delta modeling. *ACM Sigplan Notices*, *46*(2), 13–22.

Diskin, Z., & König, H. (2016). Incremental consistency checking of heterogeneous multimodels. In P. Milazzo, D. Varró, & M. Wimmer (Eds.), *Software technologies: Applications and foundations* (pp. 274–288). Cham: Springer International Publishing.

Doan, K.-H., & Gogolla, M. (2019). Quality improvement for uml and ocl models through bad smell and metrics definition. In *2019 acm/ieee 22nd international conference on model driven engineering languages and systems companion (models-c)*. IEEE. doi: 10.1109/models-c.2019.00121

Ellis, C. A., & Gibbs, S. J. (1989). Concurrency control in groupware systems. In *Proceedings of the 1989 acm sigmod international conference on management of data* (pp. 399–407).

El-Sharkawy, S., Yamagishi-Eichler, N., & Schmid, K. (2019). Metrics for analyzing variability and its implementation in

software product lines: A systematic literature review. *Information and Software Technology*, *106*, 1-30. doi: 10.1016/j.infsof.2018.08.015

Fenton, N., & Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. CRC press.

Franzago, M., Ruscio, D. D., Malavolta, I., & Muccini, H. (2018). Collaborative model-driven software engineering: A classification framework and a research map. *IEEE Transactions on Software Engineering*, *44*(12), 1146-1175. doi: 10.1109/TSE.2017.2755039

Furrer, F. J. (2019). *Future-proof software-systems*. Springer Vieweg. doi: 10.1007/978-3-658-19938-8

Ghiotto, G., Murta, L., Barros, M., & van der Hoek, A. (2020). On the nature of merge conflicts: A study of 2,731 open source java projects hosted by github. *IEEE Transactions on Software Engineering*, *46*(8), 892-915. doi: 10.1109/TSE.2018.2871083

Guimarães, M. L., & Silva, A. R. (2012). Improving early detection of software merge conflicts. In *2012 34th international conference on software engineering (icse)* (p. 342-352). doi: 10.1109/ICSE.2012.6227180

Kegel, K. (2024). *Model drift experiment reproduction package.* Zenodo. doi: 10.5281/zenodo.10687311

Klare, H., Kramer, M. E., Langhammer, M., Werle, D., Burger, E., & Reussner, R. (2021). Enabling consistency in view-based system development — the vitruvius approach. *Journal of Systems and Software*, *171*, 110815.

Kolovos, D. S., Di Ruscio, D., Pierantonio, A., & Paige, R. F. (2009). Different models for model matching: An analysis of approaches to support model differencing. In *2009 icse workshop on comparison and versioning of software models.* doi: 10.1109/CVSM.2009.5071714

König, H., & Diskin, Z. (2017). Efficient consistency checking of interrelated models. In A. Anjorin & H. Espinoza (Eds.), *Modelling foundations and applications* (pp. 161–178). Cham: Springer International Publishing.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, *29*(6), 18-21. doi: 10.1109/MS.2012.167

Lilienthal, C. (2019). *Sustainable software architecture*. dpunkt.verlag.

Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, *28*(5), 449-462. doi: 10.1109/TSE.2002.1000449

Object Management Group. (2024). *Unified modeling language uml reference documentation 2.5.1.*

Owhadi-Kareshk, M., Nadi, S., & Rubin, J. (2019). Predicting merge conflicts in collaborative software development. *CoRR*, *abs/1907.06274*.

Parker, C. (2000). Performance measurement. *Work study*, *49*(2), 63–66.

Perry, D. E., Siy, H. P., & Votta, L. G. (2001). Parallel changes in large-scale software development: an observational case study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *10*(3), 308–337.

Reisig, W. (2012). *Petri nets: An introduction* (Vol. 4). Springer Science & Business Media.

Seaman, C., & Guo, Y. (2011). Measuring and monitoring technical debt. In *Advances in computers* (Vol. 82, pp. 25–46). Elsevier.

Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). Conflict-free replicated data types. In *Stabilization, safety, and security of distributed systems: 13th international symposium, sss 2011, grenoble, france, october 10-12, 2011. proceedings 13* (pp. 386–400).

Toutenburg, H., & Heumann, C. (2008). *Deskriptive statistik: eine einführung in methoden und anwendungen mit r und spss*. Springer-Verlag.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101. doi: 10.1023/A:1018046501280

Xing, Z., & Stroulia, E. (2005). Umldiff: An algorithm for object-oriented design differencing. In *Proceedings of the 20th ieee/acm international conference on automated software engineering.* ACM. doi: 10.1145/1101908.1101919

Yamamoto, K., Kondo, M., Nishiura, K., & Mizuno, O. (2020). Which metrics should researchers use to collect repositories: An empirical study. In *2020 ieee 20th international conference on software quality, reliability and security (qrs).* IEEE. doi: 10.1109/qrs51102.2020.00065

Zou, W., Zhang, W., Xia, X., Holmes, R., & Chen, Z. (2019). Branch use in practice: A large-scale empirical study of 2,923 projects on github. In *2019 ieee 19th international conference on software quality, reliability and security (qrs)* (p. 306-317). doi: 10.1109/QRS.2019.00047

About the authors

Karl Kegel is a doctoral researcher at the Chair of Software Engineering, Technische Universität Dresden. His research interests are long-living software, software- and model evolution, and software quality assurance. You can contact the author at karl.kegel@tu-dresden.de.

Sebastian Götz is a tenured senior researcher at the Chair of Software Engineering, Technische Universität Dresden. You can contact the author at sebastian.goetz@acm.org or visit https://st.inf.tu-dresden.de/sgoetz.

Ronny Marx is a doctoral researcher at the Chair of Software Engineering, Technische Universität Dresden. You can contact the author at ronny.marx@tu-dresden.de.

Uwe Aßmann is professor of the Chair of Software Engineering, Technische Universität Dresden. You can contact the author at uwe.assmann@tu-dresden.de.