三篇论文

SCIENCE AND TECHNOLOGY

《An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models》	2020	TSE
《CC2Vec: Distributed Representations of Code Changes》	2020	ICSE
《Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction》	2017	TSE

GRADUATION DEFENSE

Explanation

(1) the event to be explained, also called the explanandum(e.g., file A is defective);

(2) a set of similar events that are similar to the explanandum but did not occur (e.g., file A is clean);

(3) a request for information that can distinguish the occurrence of the explanandum from the non-occurrence of the other similar events (e.g., a large number of changes made to file A).

An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models

2020 TSE

Model-Agnostic Techniques

LIME、
LIME-HPO、
Breakdown

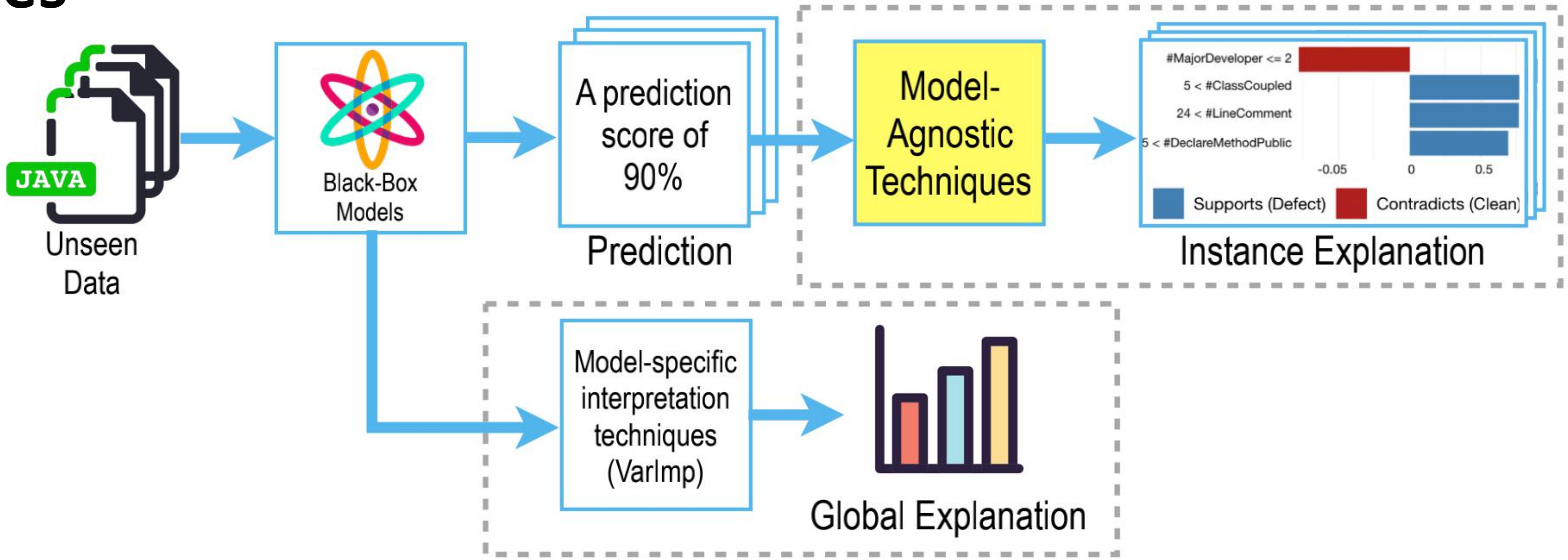
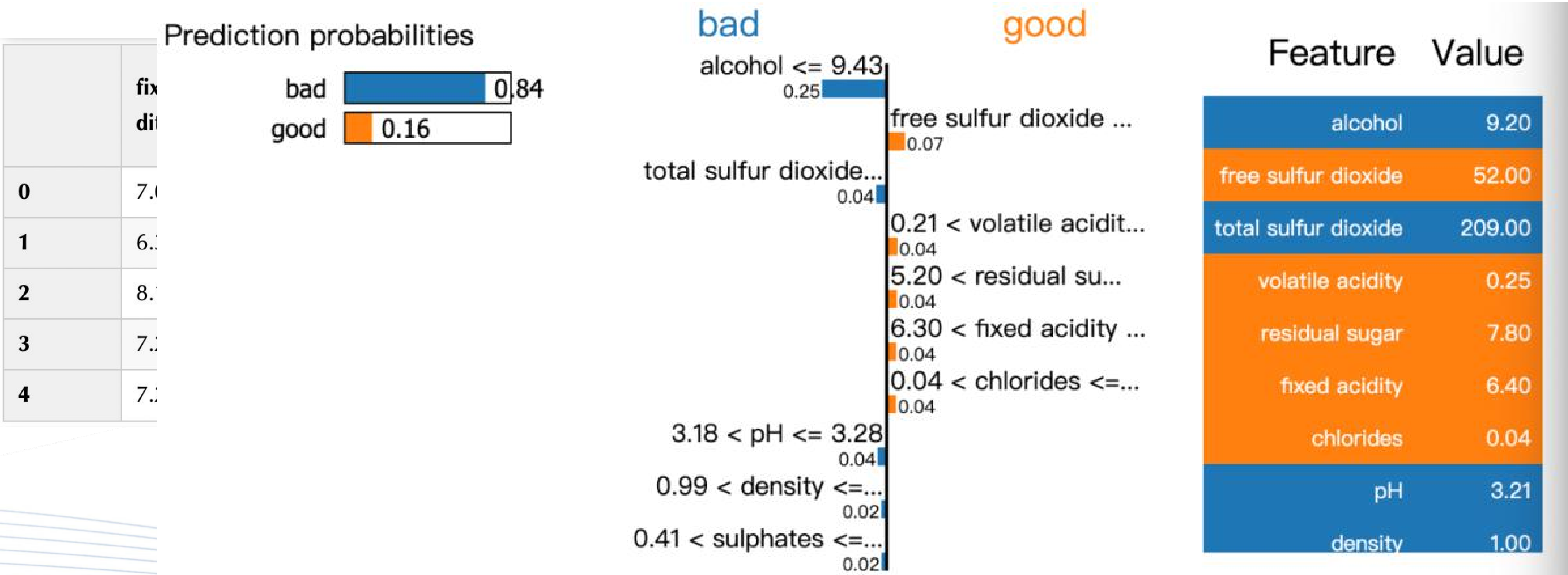


Figure 1: An illustration of model-agnostic techniques. Model-agnostic techniques are used to explain the predictions of unseen data, while the global explanation is derived from the trained models from training data. In other words, one model can have only one global explanation, but should have multiple instance explanations.

An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models

2020 TSE

Model-Agnostic Techniques



An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models

2020 TSE

RF、LR…

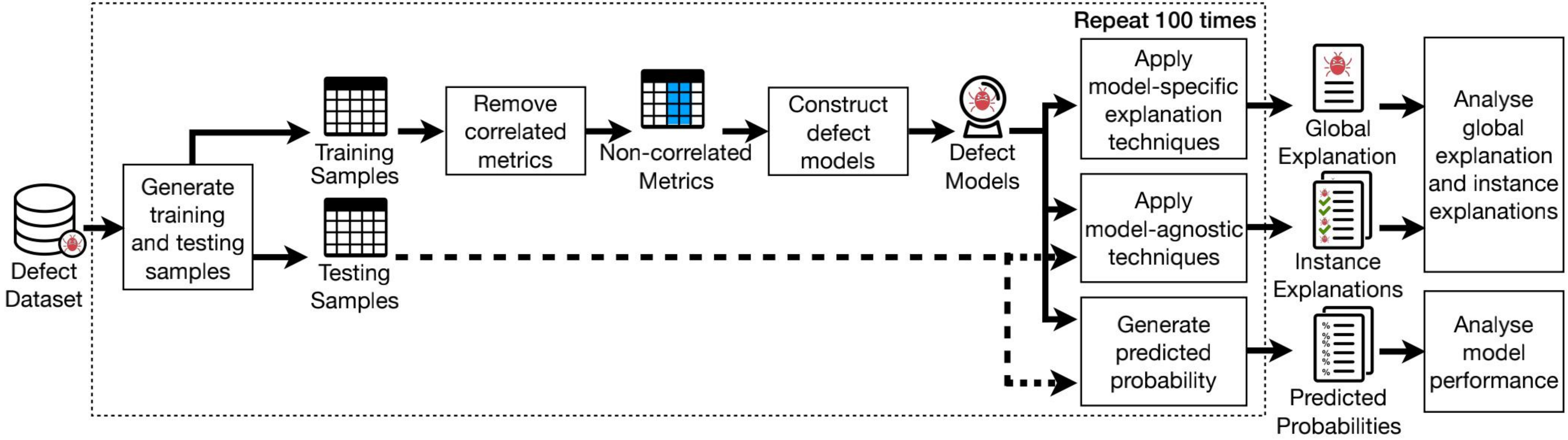


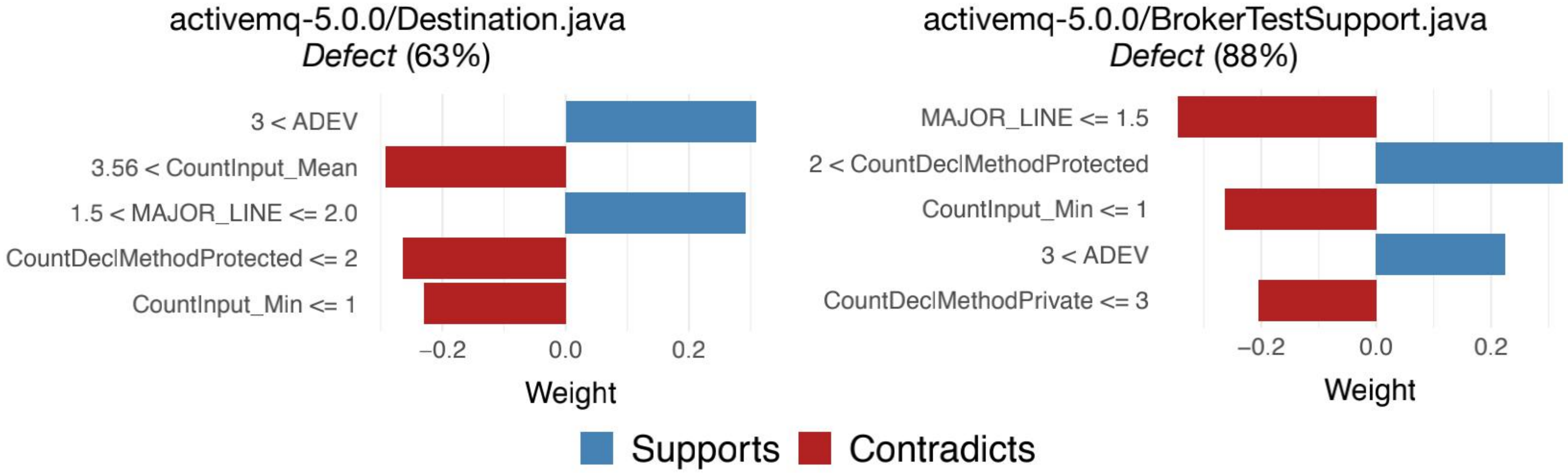
Figure 3: An overview diagram of the design of our case study.

LIME、
LIME-HPO、
Breakdown

An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models

2020 TSE

Can model-agnostic techniques explain the predictions of defect models?

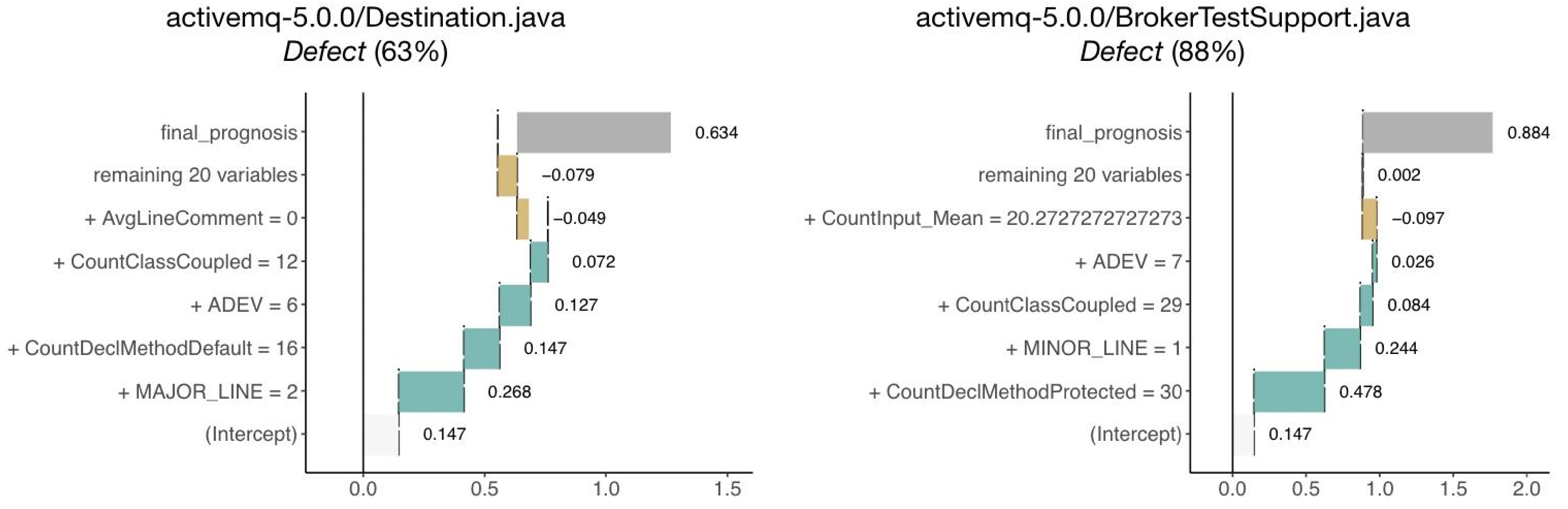


(a) An example instance explanation of LIME. The blue bars indicate supporting (positive) scores towards a file being predicted as defective, while the red bars indicate contradict (negative) scores towards its prediction.

An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models

2020 TSE

Can model-agnostic techniques explain the predictions of defect models?



(b) An example instance explanation of BreakDown. The x-axis presents the contribution (probability score) of each metric in the y-axis.

CC2Vec: Distributed Representations of Code Changes

2020 ICSE

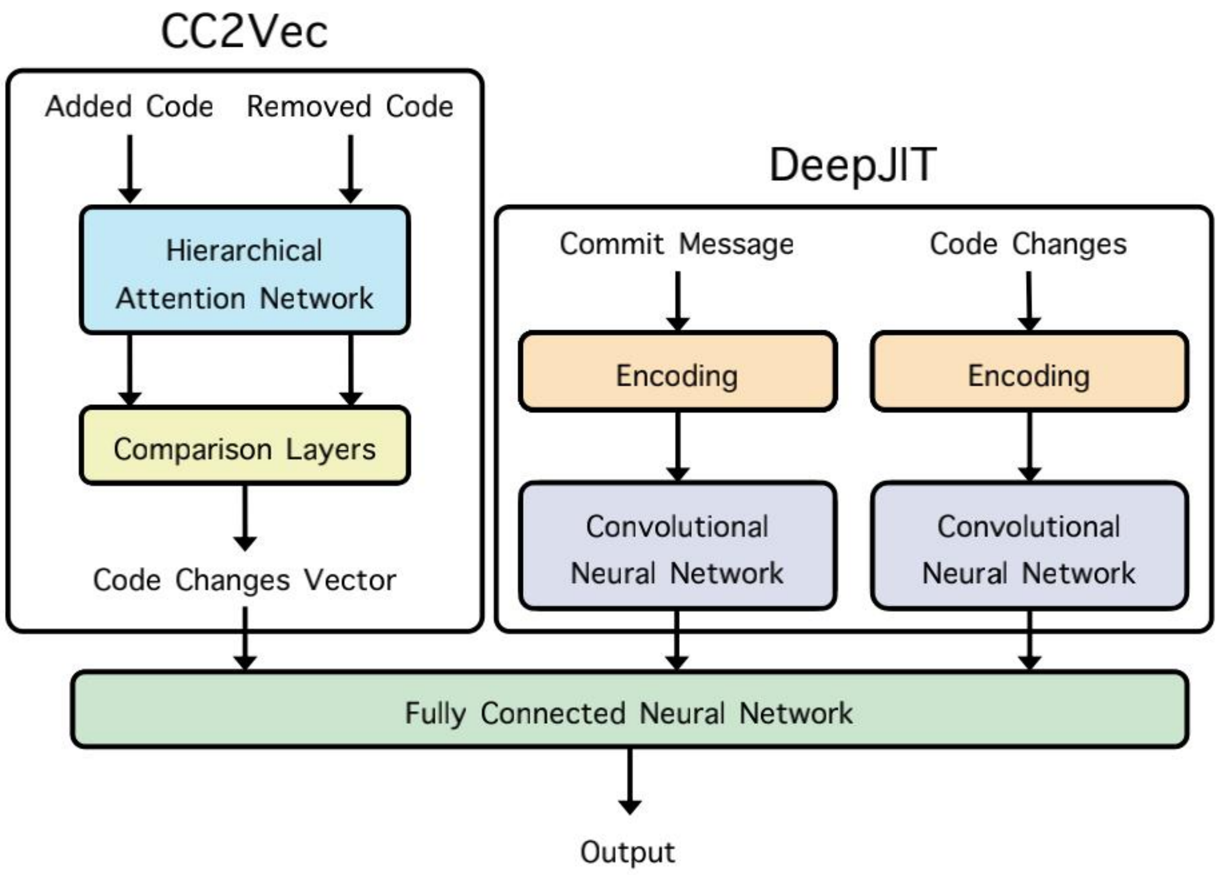
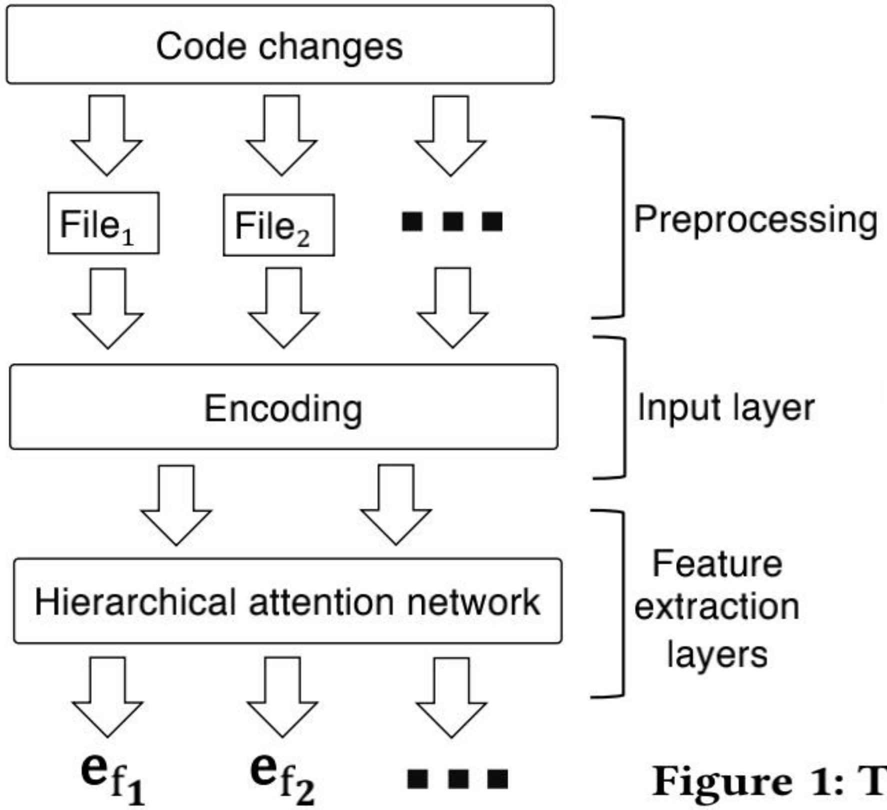


Figure 2: The overall framework of CC2Vec + DeepJIT

CC2Vec: Distributed Representations of Code Changes

2020 ICSE



NLTKLibrary

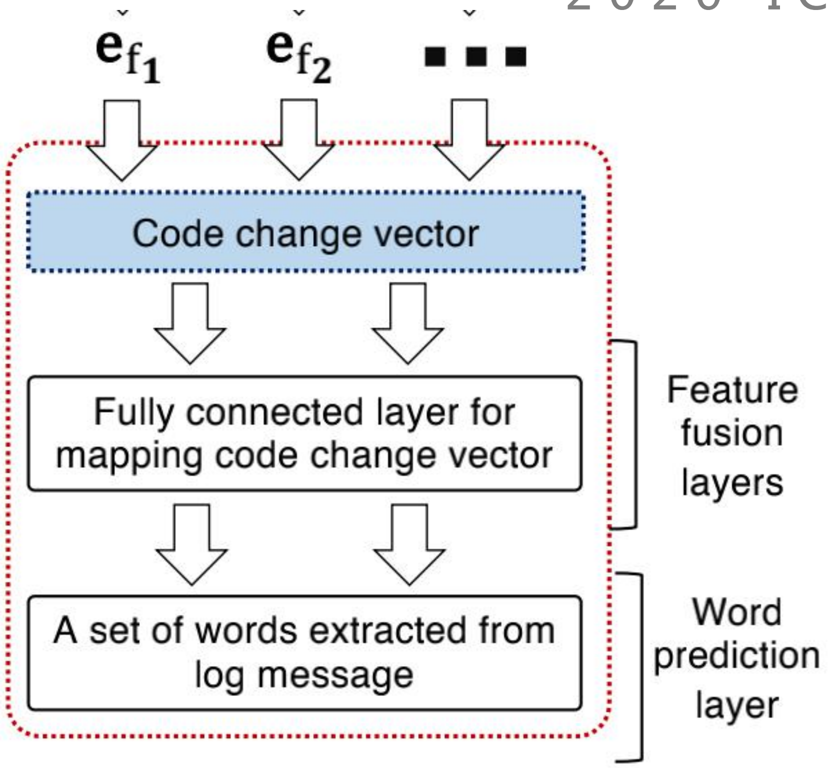


Figure 1: The overall framework of CC2Vec. Feature extraction layers are used to construct the embedding vectors for each affected file from a given patch (i.e., e_{f_1}, e_{f_2}, etc). The embedding vectors are then concatenated to build a vector representation for the code change in the patch (code change vector). The code change vector is connected to the fully connected layer and is learned by minimizing an objective function of the word prediction layer.

CC2Vec: Distributed Representations of Code Changes

2020 ICSE

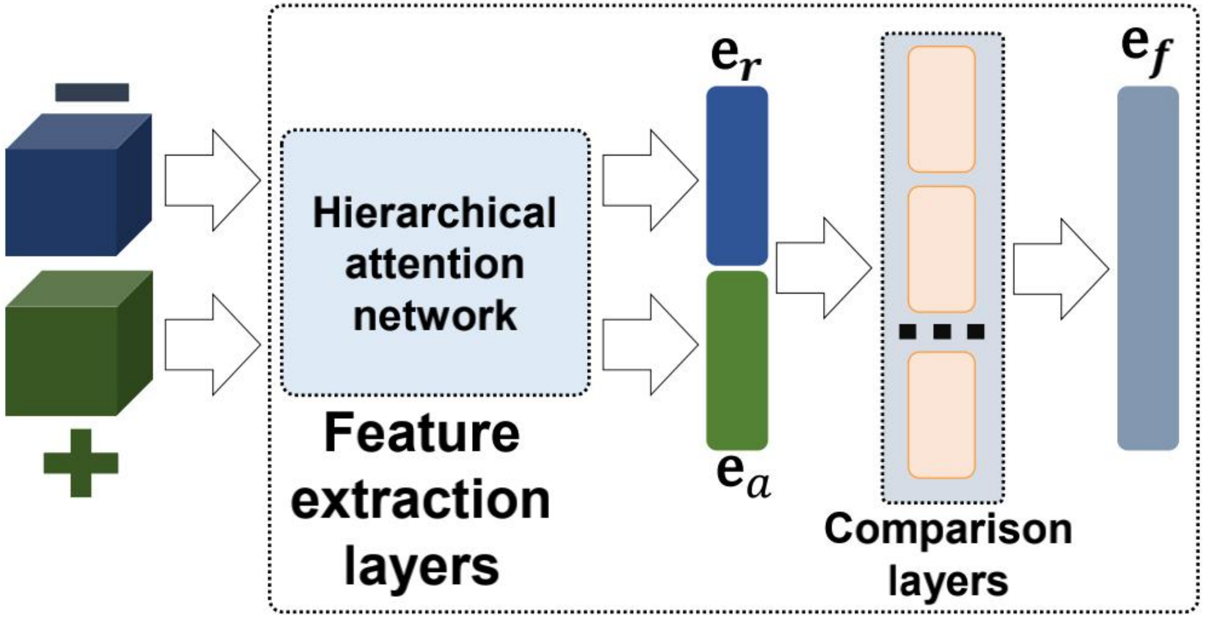


Figure 2: Architecture of the feature extraction layers for mapping the code change of the affected file in a given patch to an embedding vector. The input of the module is the removed code and added code of the affected file, denoted by "-" and "+", respectively.

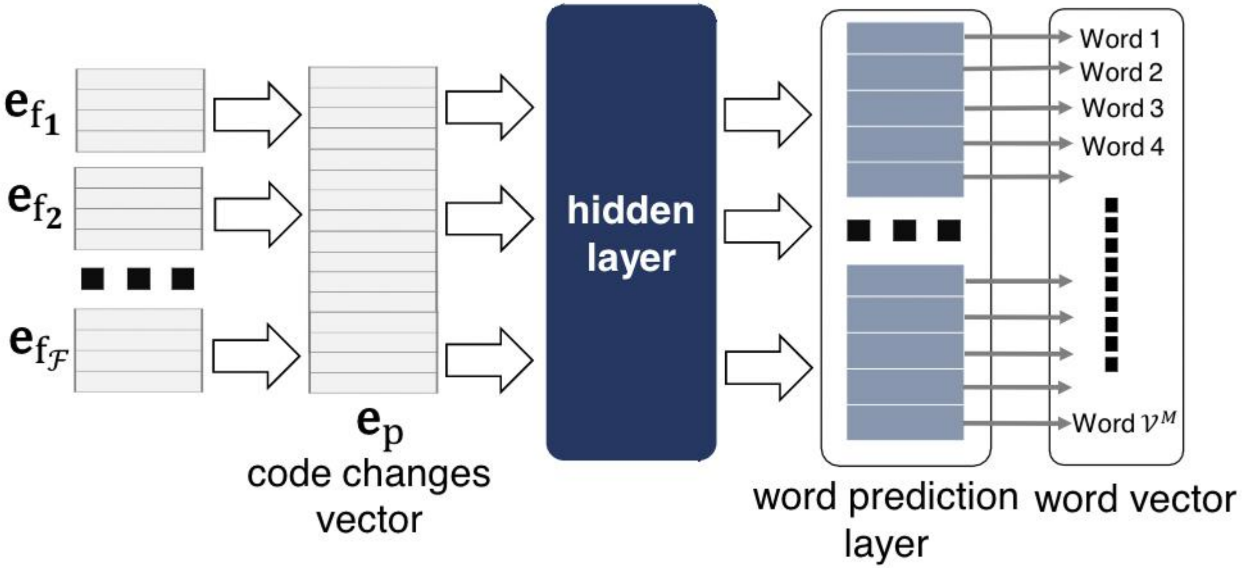


Figure 5: The details of the red dashed box in Figure 1. It takes as input a list of embedding vectors of the affected files of a given patch (i.e., $\mathbf{e}_{\mathbf{f}_1}, \mathbf{e}_{\mathbf{f}_2}, \ldots, \mathbf{e}_{\mathbf{f}_\mathcal{F}}$). $\mathbf{e}_\mathbf{p}$ is the vector representation of the code change and is fed to a hidden layer to produce the word vector (i.e., the probability distribution over words). $\mathcal{V}^M$ is a set of words extracted from the first line of the log messages.

Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction

2017 TSE

Central Question: ***Do the important properties of fix-inducing changes remain consistent as systems evolve?***

- Fix-inducing Changes

Despite the advantages of JIT defect prediction, like all prediction models, they assume that the properties of past events (i.e., fix-inducing changes) are similar to the properties of future ones.

Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction

2017 TSE

AUC、Brier

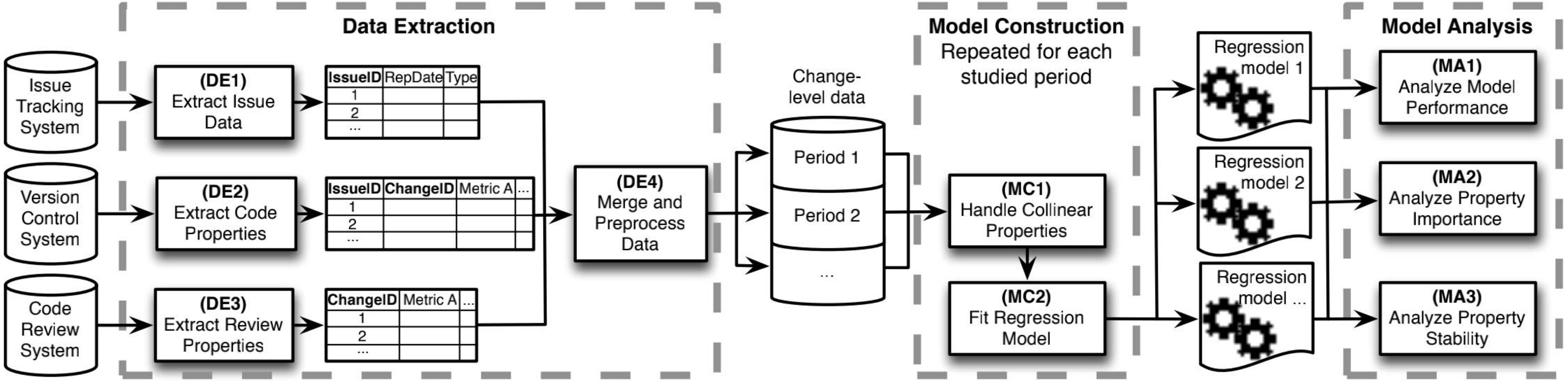


Fig. 2. An overview of the design of our case study.

SZZ Algorithm

Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction

2017 TSE

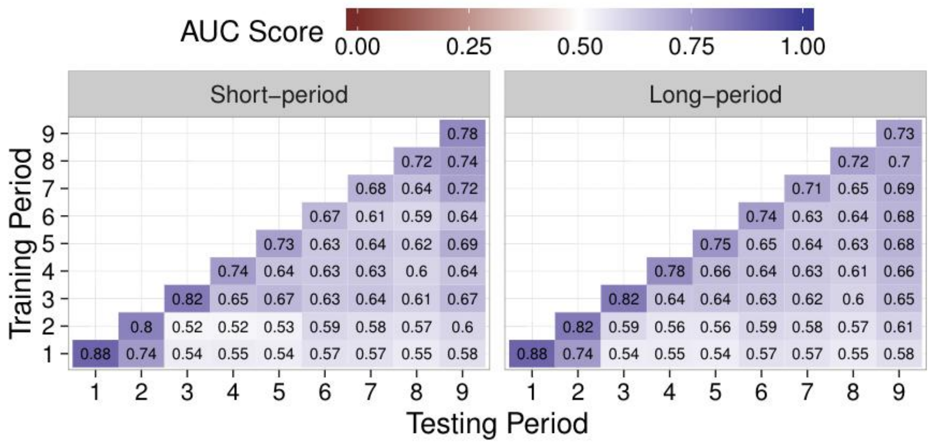
TABLE 2
A taxonomy of the studied families of code and review properties.

	Property	Description	Rationale
Size	Lines added	The number of lines added by a change.	The more code that is changed, the more likely that defects will be introduced [31].
	Lines deleted	The number of lines deleted by a change.	
Diffusion	Subsystems	The number of modified subsystems.	Scattered changes are riskier than focused ones because they require a broader spectrum of expertise [6, 14].
	Directories	The number of modified directories.	
	Files	The number of modified files.	
	Entropy	The spread of modified lines across file.	
History	Unique changes	The number of prior changes to the modified files.	More changes are likely more risky because developers will have to recall and track many previous changes [18].
	Developers	The number of developers who have changed the modified files in the past.	Files previously touched by more developers are likely more risky [24].
	Age	The time interval between the last and current changes.	More recently changed code is riskier than older code [10].
Author/Rev. Experience	Prior changes	The number of prior changes that an actor[1] has participated in.[2]	Changes that are produced by novices are likely to be more risky than changes produced by experienced developers [28].
	Recent changes	The number of prior changes that an actor has participated in weighted by the age of the changes (older changes are given less weight than recent ones).	
	Subsystem changes	The number of prior changes to the modified subsystem(s) that an actor has participated in.	
	Awareness[3]	The proportion of the prior changes to the modified subsystem(s) that an actor has participated in.	Changes that involve developers who are aware of the prior changes in the impacted subsystems are likely to be less risky than those that do not.
Review	Iterations	Number of times that a change was revised prior to integration.	The quality of a change likely improves with each iteration. Hence, changes that undergo plenty of iterations prior to integration may be less risky than those that undergo few [34, 42].
	Reviewers	Number of reviewers who have voted on whether a change should be integrated or abandoned.	Since more reviewers will likely raise more issues so that they may be addressed prior to integration, changes with many reviewers are likely to be less risky than those with fewer reviewers [36].
	Comments	The number of non-automated, non-owner comments posted during the review of a change.	Changes with short discussions may not be deriving value from the review process, and hence may be more risky [25, 26].
	Review window	The length of time between the creation of a review request and its final approval for integration.	Changes with shorter review windows may not have spent enough time carefully analyzing the implications of a change prior to integration, and hence may be more risky [34, 42].

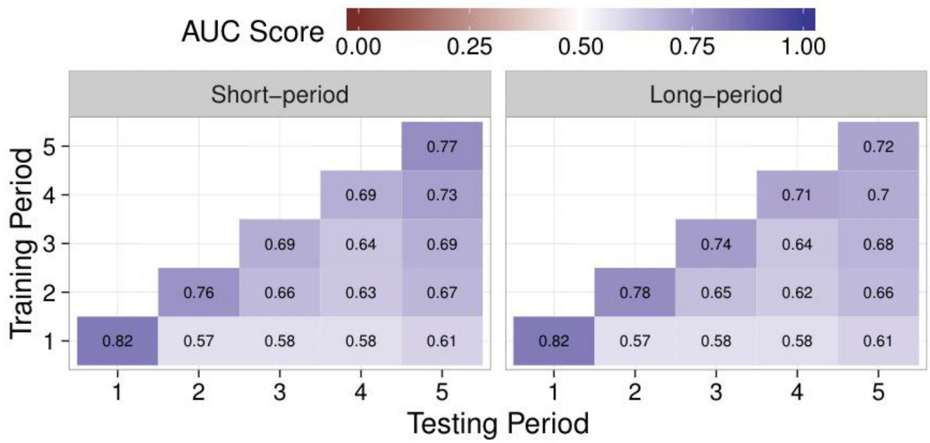
[1] Either the author or reviewer of a change. [2] Either authored or reviewed. [3] New property proposed in this paper.

Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction 2017 TSE

(RQ1) Do JIT models lose predictive power over time?

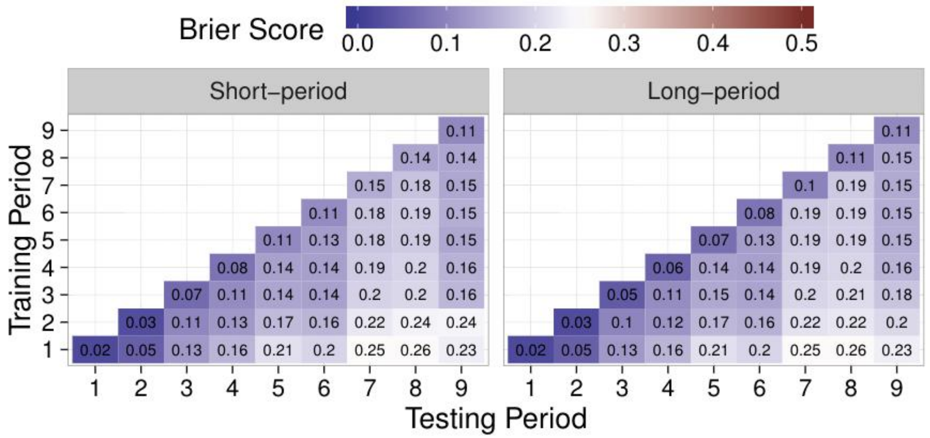


(c) AUC in three-month periods (OPENSTACK)

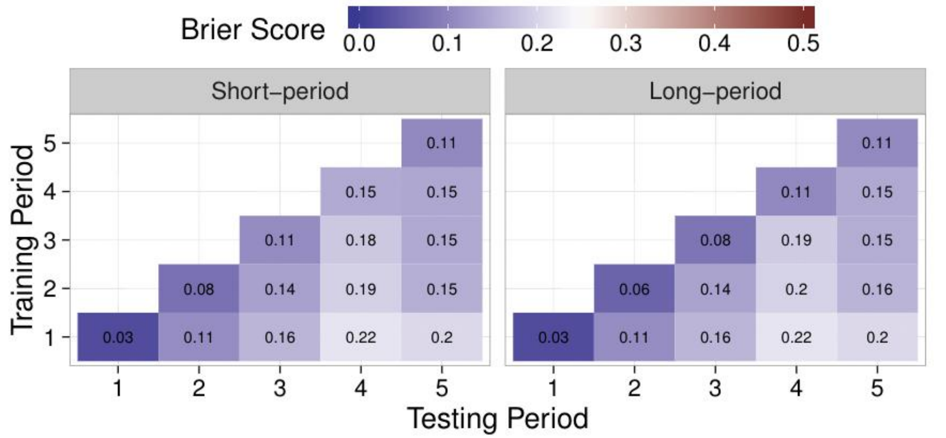


(d) AUC in six-month periods (OPENSTACK)

(RQ2) Does the relationship between code change properties and the likelihood of inducing a fix evolve?



(g) Brier score in three-month periods (OPENSTACK)



(h) Brier score in six-month periods (OPENSTACK)