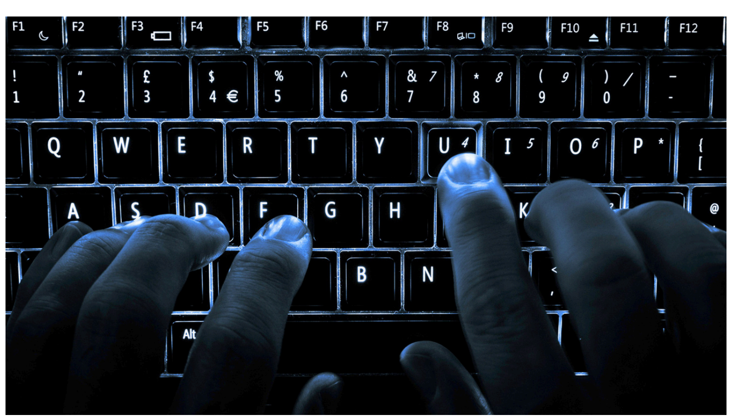
Class Imbalance Evolution and Verification Latency in Just-in-Time Software Defect Prediction

George Cabral[1,2], Leandro Minku[1], Emad Shihab[3], Suhaib Mujahid[3]

[1] University of Birmingham, UK
[2] Federal Rural University of Pernambuco, Brazil
[3] Concordia University, Canada

EPSRC SPDISC

Just-In-Time Software Defect Prediction (JIT-SDP)



Implementing a change

Your change is likely to induce **defects.**

Please inspect it further before commit.

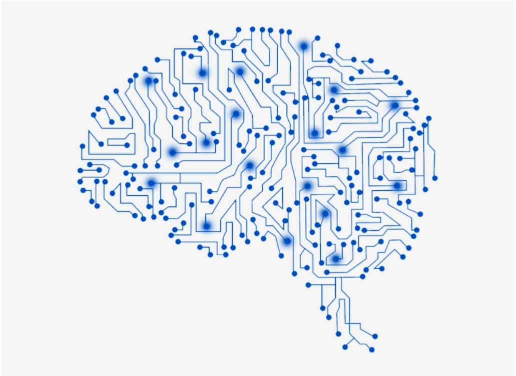
Versioning Repository

Machine Learning for JIT-SDP

SZZ algorithm

x1: #modified subsystems	x2: #LOC added	x3: fix?	x4: developer experience	…	y: Defect-inducing?
2	25	true	15		No
3	5	true	20		No
4	20	false	100		No
5	40	false	5		Yes
3	35	false	45		Yes
2	25	false	30		Yes
1	3	false	20		No
4	100	true	20		Yes

Machine Learning Algorithm



Predictive Model

In realistic scenarios, additional training data is produced over time, and should be used to avoid models becoming obsolete.

J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?", MSR 2005.
Kamei et al., "A Large-Scale Empirical Study of Just-in-Time Quality Assurance", TSE 2013.
S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction", IEEE TSE 2018.

Class Imbalance Evolution

- JIT-SDP is known to be a class imbalanced learning problem.
- When considering realistic learning scenarios, the imbalance status of the problem may vary over time, i.e., there may be class imbalance **evolution**.
- Proportion of defect-inducing and clean examples may vary over time.
  - E.g., the nature of a new release may increase / reduce the proportion of defects.

- Even though class imbalance has been studied, class imbalance evolution has not been studied in JIT-SDP, and it could potentially be detrimental to predictive performance.

Aim 1:

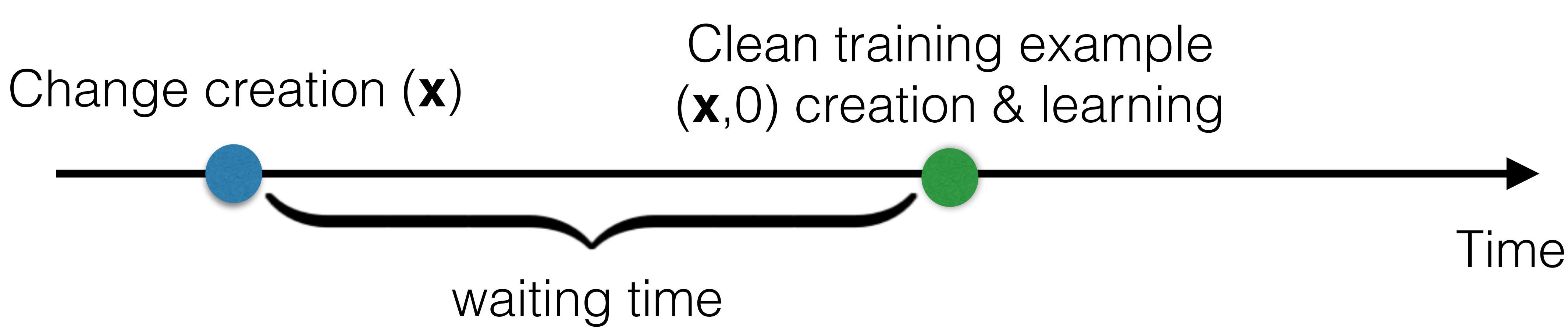
To provide the first investigation of whether class imbalance **evolution** occurs and negatively affects predictive performance in JIT-SDP.

Aim 2:

To propose a novel approach able to tackle class imbalance **evolution** in JIT-SDP.

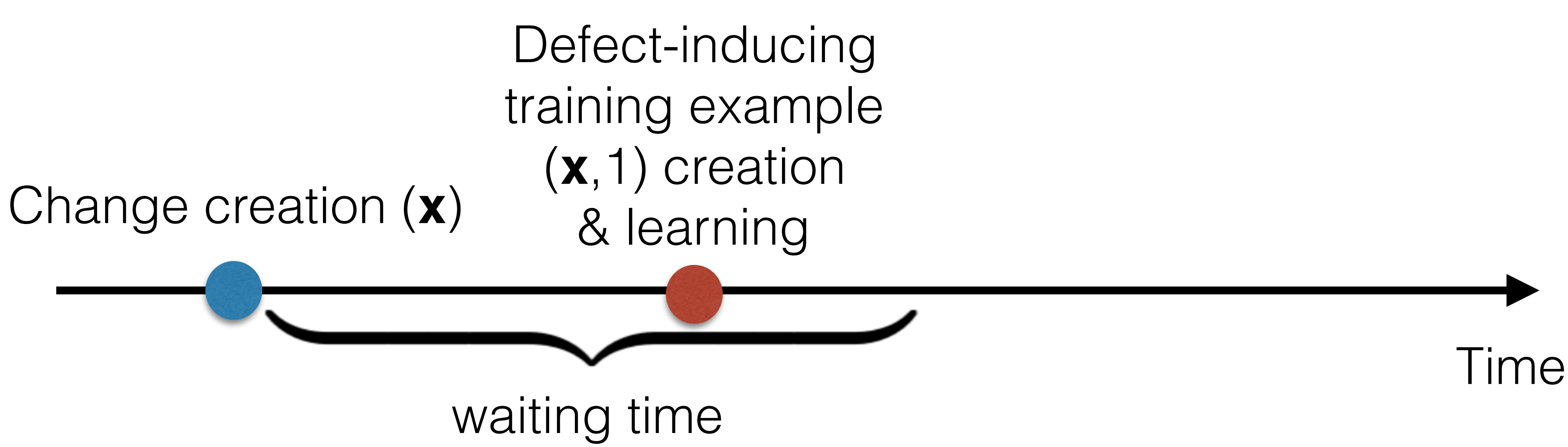
Verification Latency

- Labels of training examples arrive with a delay.
- We propose a framework that considers three cases.
- Case 1: no defect found during waiting time.



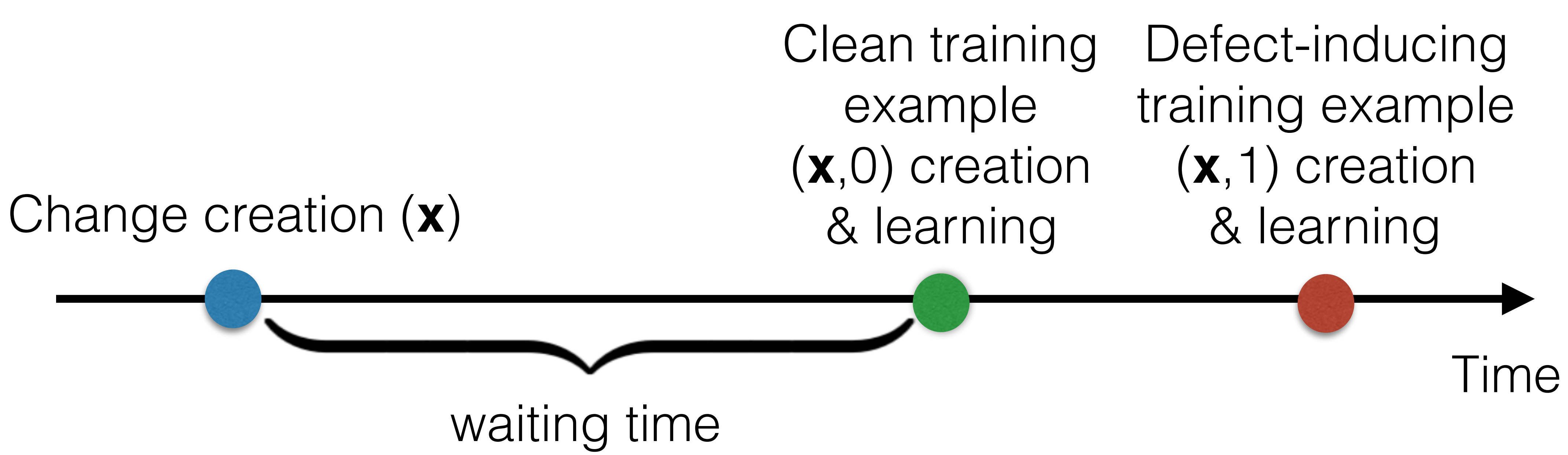
Verification Latency

- Labels of training examples arrive with a delay.
- We propose a framework that considers three cases.
- Case 1: no defect found during waiting time.
- Case 2: defect found during waiting time.



Verification Latency

- Labels of training examples arrive with a delay.
- We propose a framework that considers three cases.
- Case 1: no defect found during waiting time.
- Case 2: defect found during waiting time.
- Case 3: defect found after waiting time.



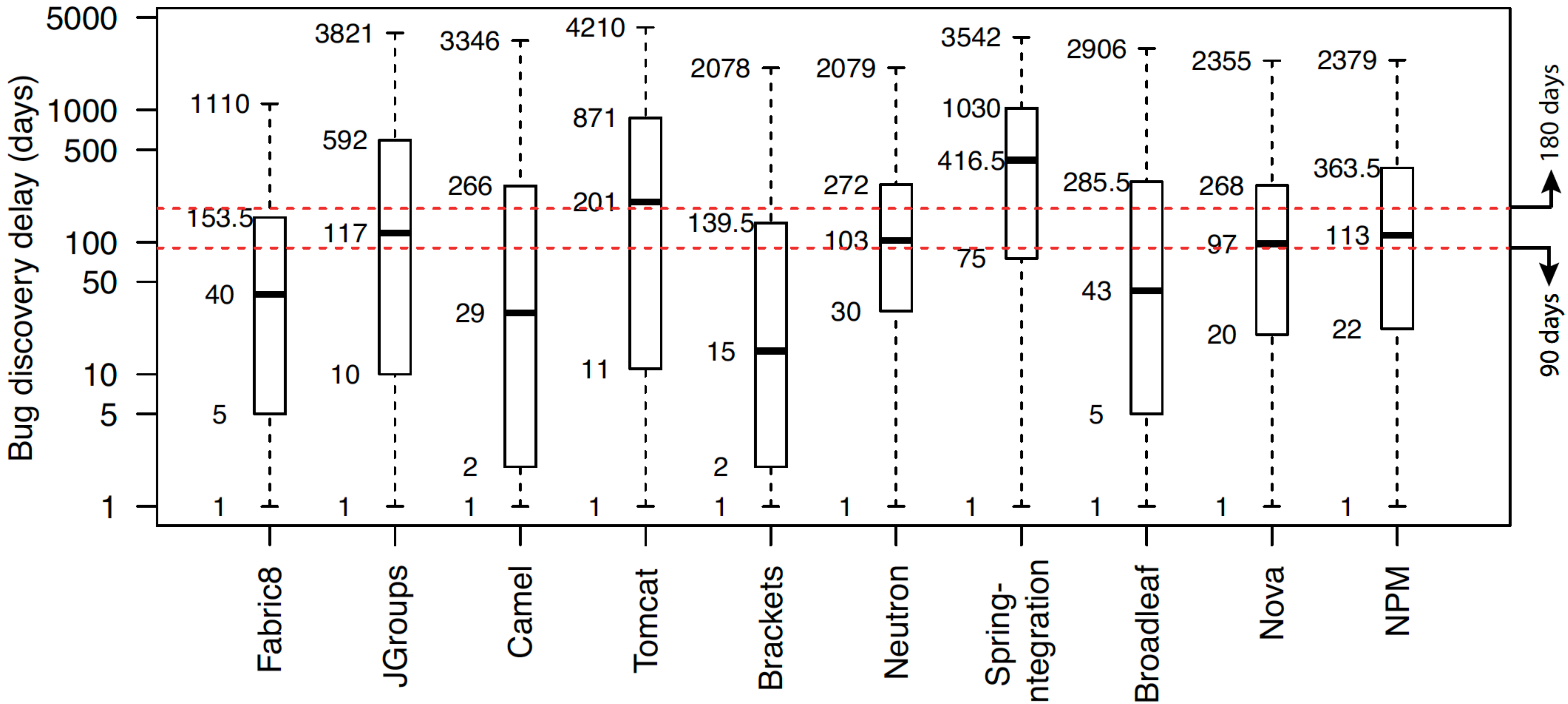
Data Sets - Ten Open Source Projects from GitHub

Dataset	#commits	%defect-inducing	Period	Language
Fabric8	13,004	20%	12/2011 - 12/2017	Java
JGroups	18,317	17%	09/2003 - 12/2017	Java
Camel	30,517	20%	03/2007 - 12/2017	Java
Tomcat	18,877	28%	03/2006 - 12/2017	Java
Brackets	17,311	23%	12/2011 - 12/2017	JavaScript
Neutron	19,451	24%	12/2010 - 12/2017	Python
Spring-integration	8,692	27%	11/2007 - 01/2018	Java
Broadleaf	14,911	17%	11/2008 - 12/2017	Java
Nova	48,938	25%	08/2010 - 01/2018	Python
NPM	7,893	18%	09/2009 - 11/2017	JavaScript

Rich history (#commits); good ratio of defect-inducing changes (~20% overall); > 5 years duration.

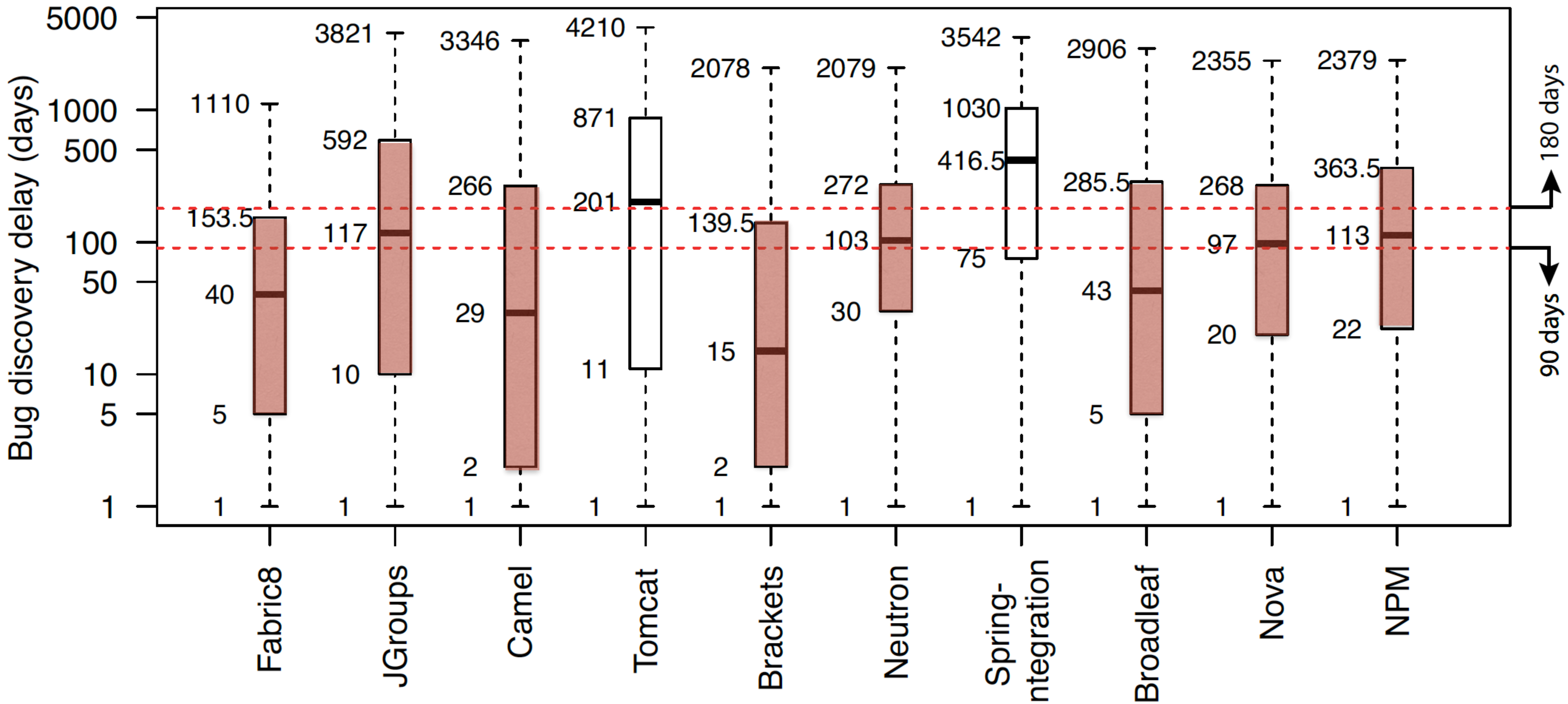
RQ1: To what extent **verification latency** occurs in JIT-SDP?
What are reasonable waiting times to use for creating training examples?

Defect Discovery Delay



Delays varied from 1 day to 11.5 years.

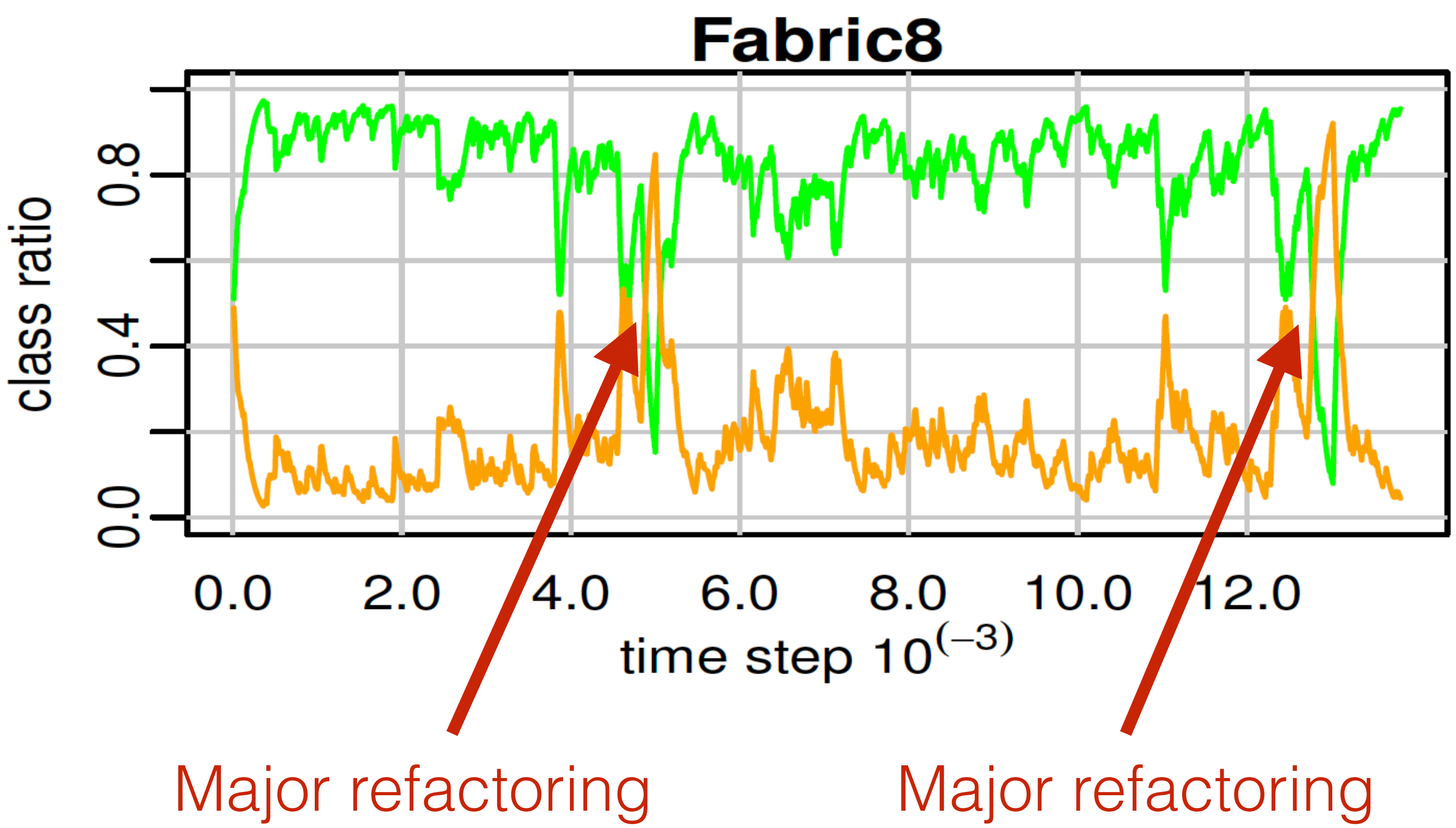
Defect Discovery Delay



A waiting time of 90 days could be considered as a good trade-off between correct labelling and obsolescence.

RQ2: Does JIT-SDP suffer from class imbalance **evolution**?

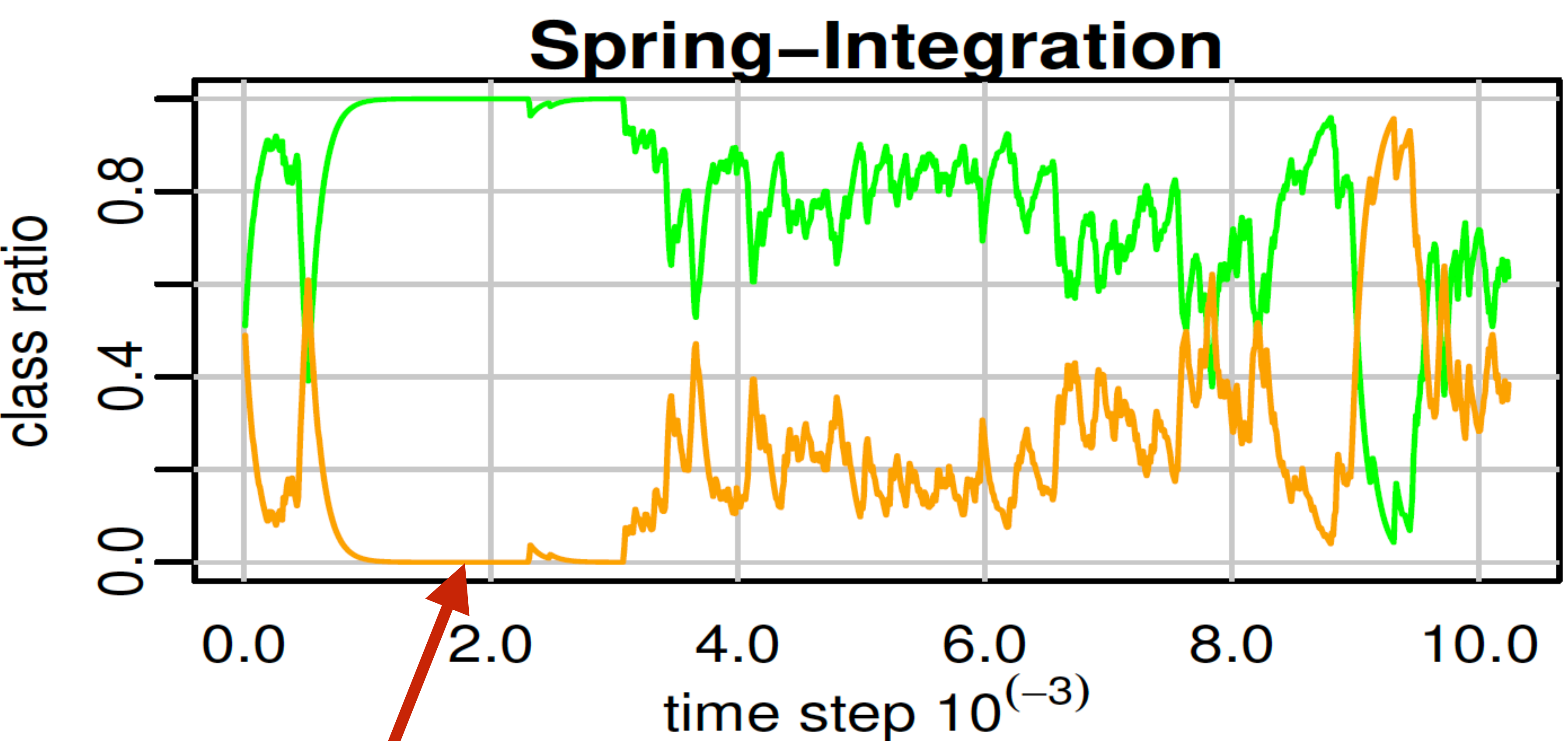
Proportion of Training Examples of Each Class Over Time



- The defect-inducing rate varies over time.
- Some increases even cause the clean class to become a minority.

Orange/lime: proportion of defect-inducing/clean changes.

Proportion of Training Examples of Each Class Over Time

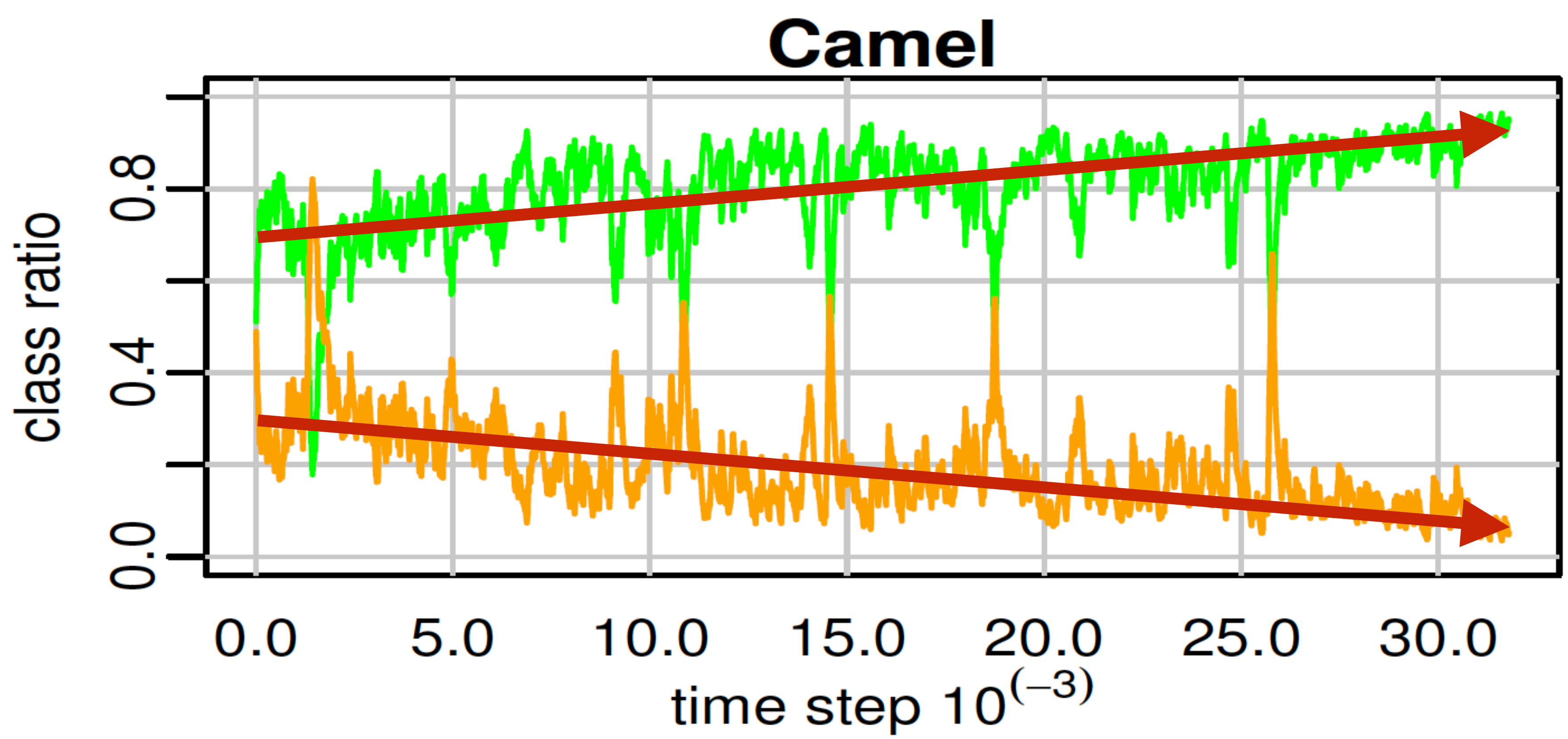


Extreme class imbalance

- There was one case where the ratio of defect-inducing examples became less than 0.001%.

Orange/lime: proportion of defect-inducing/clean changes.

Proportion of Training Examples of Each Class Over Time



- In several cases the problem became more imbalanced over time.

Increasing/decreasing trend

Orange/lime: proportion of defect-inducing/clean changes.

RQ3: Does class imbalance evolution negatively affect JIT-SDP's predictive performance?

Predictive Performance of Existing JIT-SDP Vs. Class Imbalance Evolution Approaches

Overall predictive performance across data sets

Classifier	R_0	R_1	$\|R_0 - R_1\|$	G-Mean
OOB	57.35 [3] (9.51)	**73.14** [1] (13.32)	33.21 [2] (9.61)	60.69 [2] (10.53)
UOB	57.23 [3] (10.91)	**71.13** [1] (16.60)	37.24 [3] (6.71)	59.21 [2] (8.93)
OOB(FixedIR)	70.26 [2] (18.67)	46.53 [4] (22.79)	55.34 [5] (20.48)	45.01 [4] (12.57)
OOB(FixedIR)*	**84.82** [1] (9.97)	36.89 [5] (23.77)	53.45 [5] (27.78)	45.89 [4] (23.70)
OOB-SW	66.75 [2] (11.32)	60.05 [3] (20.75)	45.69 [4] (15.16)	54.84 [3] (12.76)

- All approaches use Online Bagging of Hoeffding Trees as learners.
- Values in [] are ranks produced by Scott-Knott with Bootstrap sampling and A12 effect size.
- Values in () are standard deviations.
- Effect sizes of differences in performance were usually large when considering each data set separately.

Predictive Performance of Existing JIT-SDP Vs. Class Imbalance Evolution Approaches

Overall predictive performance across data sets

Classifier	R_0	R_1	$\|R_0$ - $R_1\|$	G-Mean
OOB	57.35 [3] (9.51)	**73.14** [1] (13.32)	33.21 [2] (9.61)	60.69 [2] (10.53)
UOB	57.23 [3] (10.91)	**71.13** [1] (16.60)	37.24 [3] (6.71)	59.21 [2] (8.93)
OOB(FixedIR)	70.26 [2] (18.67)	46.53 [4] (22.79)	55.34 [5] (20.48)	45.01 [4] (12.57)
OOB(FixedIR)*	**84.82** [1] (9.97)	36.89 [5] (23.77)	53.45 [5] (27.78)	45.89 [4] (23.70)
OOB-SW	66.75 [2] (11.32)	60.05 [3] (20.75)	45.69 [4] (15.16)	54.84 [3] (12.76)

- All approaches use Online Bagging of Hoeffding Trees as learners.
- Values in [] are ranks produced by Scott-Knott with Bootstrap sampling and A12 effect size.
- Values in () are standard deviations.
- Effect sizes of differences in performance were usually large when considering each data set separately.

Predictive Performance of Existing JIT-SDP Vs. Class Imbalance Evolution Approaches

Overall predictive performance across data sets

Classifier	R_0	R_1	$\|R_0$ - $R_1\|$	G-Mean
OOB	57.35 [3] (9.51)	**73.14** [1] (13.32)	33.21 [2] (9.61)	60.69 [2] (10.53)
UOB	57.23 [3] (10.91)	**71.13** [1] (16.60)	37.24 [3] (6.71)	59.21 [2] (8.93)
OOB(FixedIR)	70.26 [2] (18.67)	46.53 [4] (22.79)	55.34 [5] (20.48)	45.01 [4] (12.57)
OOB(FixedIR)*	**84.82** [1] (9.97)	36.89 [5] (23.77)	53.45 [5] (27.78)	45.89 [4] (23.70)
OOB-SW	66.75 [2] (11.32)	60.05 [3] (20.75)	45.69 [4] (15.16)	54.84 [3] (12.76)

- OOB and UOB achieved the top G-Means.
    - Do not assume fixed level of imbalance and do not waste past knowledge.
- However, **all** approaches' $|R_0 - R_1|$ was very high.

Problems of a High $|R_0\text{-}R_1|$

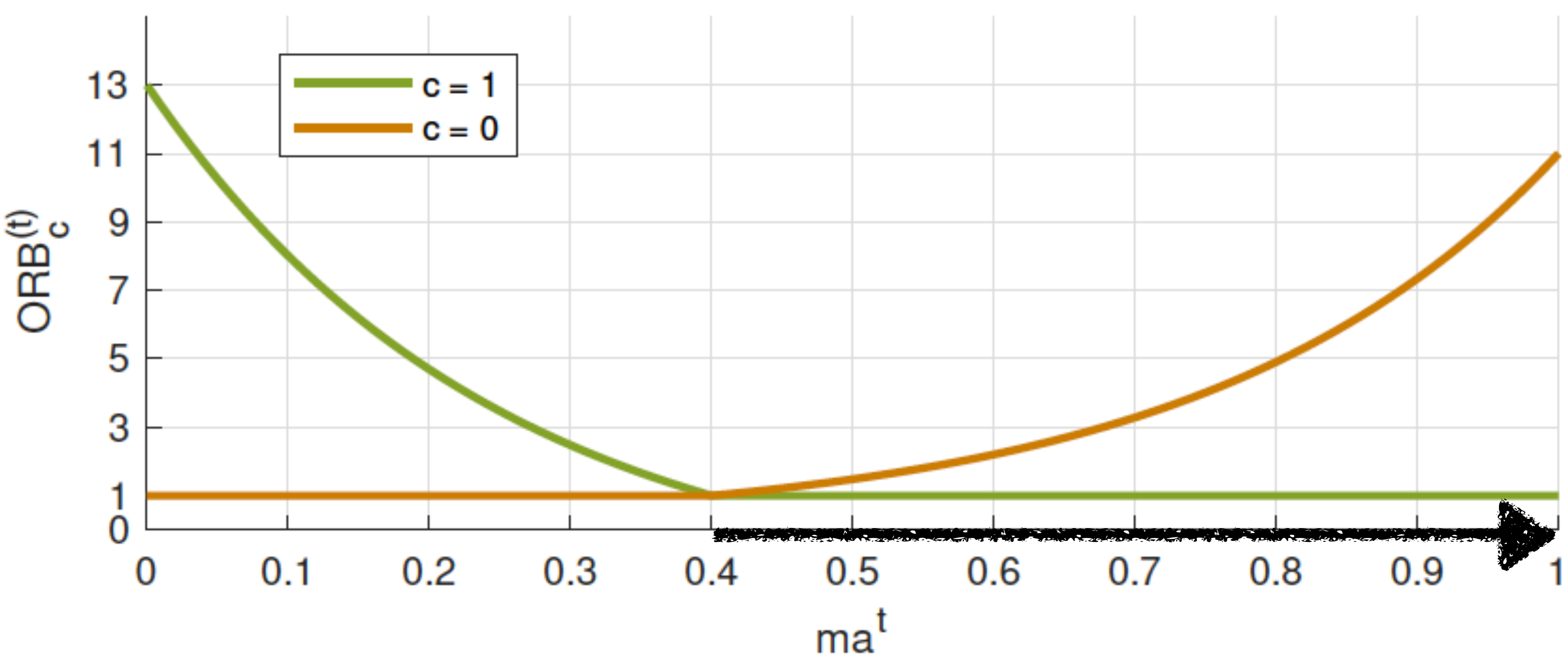
- Impact of higher R_1 at the cost of a low R_0:
  - Practitioners will have to carefully inspect many clean changes, and will loose trust in the approach.

- Impact of higher R_0 at the cost of a low R_1:
  - Many defect-inducing changes will be missed.

RQ4: How to improve JIT-SDP's predictive performance, especially $|R_0-R_1|$, in view of class imbalance evolution and verification latency?

Proposed Approach Oversampling Rate Boosting (ORB)

- We need to decide the resampling rate at the current point in time, without knowing the proportion of defect-inducing changes produced up to 90 (waiting time) days ago.
- Moving average of defect-inducing predictions (ma^t) gives an idea of whether we need to further emphasise a given class.

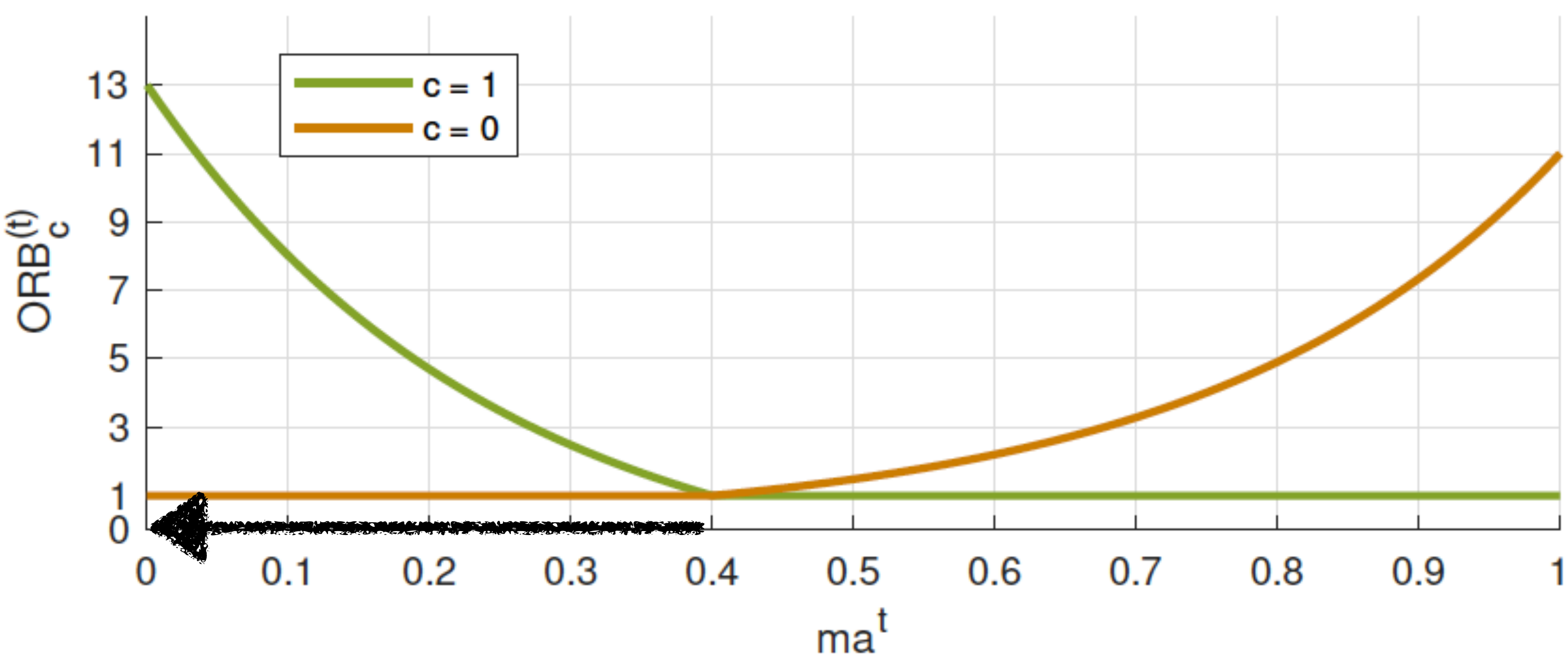


If ma^t is high, we need to further emphasise the clean class.

Proposed Approach Oversampling Rate Boosting (ORB)

- We need to decide the resampling rate at the current point in time, without knowing the proportion of defect-inducing changes produced up to 90 (waiting time) days ago.

- Moving average of defect-inducing predictions (ma^t) gives an idea of whether we need to further emphasise a given class.



If ma^t is low, we need to further emphasise the defect-inducing class.

ORB Evaluation

Classifier	R_0	R_1	$\|R_0$ - $R_1\|$	G-Mean
OOB	57.35 [3] (9.51)	**73.14** [1] (13.32)	33.21 [2] (9.61)	60.69 [2] (10.53)
UOB	57.23 [3] (10.91)	**71.13** [1] (16.60)	37.24 [3] (6.71)	59.21 [2] (8.93)
OOB(FixedIR)	70.26 [2] (18.67)	46.53 [4] (22.79)	55.34 [5] (20.48)	45.01 [4] (12.57)
OOB(FixedIR)*	**84.82** [1] (9.97)	36.89 [5] (23.77)	53.45 [5] (27.78)	45.89 [4] (23.70)
OOB-SW	66.75 [2] (11.32)	60.05 [3] (20.75)	45.69 [4] (15.16)	54.84 [3] (12.76)
ORB	65.12 [2] (8.22)	67.35 [2] (11.17)	**23.00** [1] (8.63)	**63.02** [1] (8.70)

- ORB managed to improve $|R_0 - R_1|$, which was up to 45.38% better than OOB's and up to 63.59% better than UOB's.

- ORB also managed to achieve top G-Means, even though the magnitude of the improvements in G-Mean w.r.t. OOB was not large.

Conclusions and Implications

- Aim1: class imbalance evolution occurs and negative affects predictive performance in JIT-SDP.
  - It is important for practitioners to apply online learning algorithms able to tackle class imbalance **evolution** in JIT-SDP (RQ2 and RQ3).
    - Otherwise, there is a risk of missing a substantial amount of defect-inducing software changes over time.
  - Even the sliding window strategy recommended in the JIT-SDP literature was not enough to cope with class imbalance evolution (RQ3).
    - So, simply rebuilding classifiers from scratch over time is not enough to achieve good predictive performance.
  - OOB and UOB can treat class imbalance evolution to some extent.
    - But $|R_0 - R_1|$ is still high, and can lead to a large number of false alarms.

Conclusions and Implications

- Aim2: we proposed a novel approach ORB to improve predictive performance in JIT-SDP.
    - Practitioners adopting ORB could potentially be less overloaded by false alarms while not missing too many defect-inducing software changes (RQ4).
        - Future studies in their company’s environment would be necessary to check whether our findings generalise to them.
- We further emphasise that it is important to take verification latency into account in JIT-SDP studies (RQ1).