

Investigating the impact of the release cycle on bug priority prediction

Yutaro Kashiwa

PhD student
Wakayama University, Japan

16 October, 2018

Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca

Abstract

*Predicting the incidence of faults in code has been com-
monly associated with measuring complexity. In this paper*

2. Studies by Graves *et al.* [11] and Leszak *et al.* [22], on
commercial systems, and recently by Herraiz *et al.* [18]
on open source systems show that most code complexity
metrics highly correlate with LOC – a much simpler met-



Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca



Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects

Hongyu Zhang¹, Liang Gong¹, Steve Versteeg²
¹Tsinghua University
Beijing 100084, China
hongyu@tsinghua.edu.cn
²CA Technologies
Melbourne, Australia
steve.versteeg@ca.com

Abstract—For a large and evolving software system, the project team could receive many bug reports over a long period of time. It is important to achieve a quantitative understanding of bug-fixing time. The ability to predict bug-fixing time can help a project team better estimate software maintenance efforts and better manage software projects. In this paper, we perform an

time from the creation of a bug record to the time the bug is resolved as fixed.

Although each bug is assigned with a severity and a priority, we find that we cannot simply predict bug-fixing time based merely on bug severity or priority. Bug-fixing time is affected



Predicting the incidence
mainly associated with me

Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca



Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects

Hongyu Zhang¹, Liang Gong¹, Steve Versteeg²
¹Tsinghua University
Beijing 100084, China
hongyu@tsinghua.edu.cn
²CA Technologies
Melbourne, Australia
steve.versteeg@ca.com



Abstract—For a large and evolving software system, team could receive many bug reports over a long period. It is important to achieve a quantitative understanding of bug fixing time. The ability to predict bug-fixing time of project team better estimate software maintenance cost and better manage software projects. In this paper, we

2013 IEEE International Conference on Software Maintenance

DRONE: Predicting Priority of Reported Bugs Multi-Factor Analysis

Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore
²National University of Singapore, Singapore
{yuan.tian.2012,davidlo}@smu.edu.sg, suncn@comp.nus.edu.sg

Abstract—Bugs are prevalent. To improve software quality, developers often allow users to report bugs that they found using a bug tracking system such as Bugzilla. Users would specify among other things, a description of the bug, the component

and decide the appropriate priority levels.

To aid bug triagers in assigning priority, we new automated approach to recommend priority levels



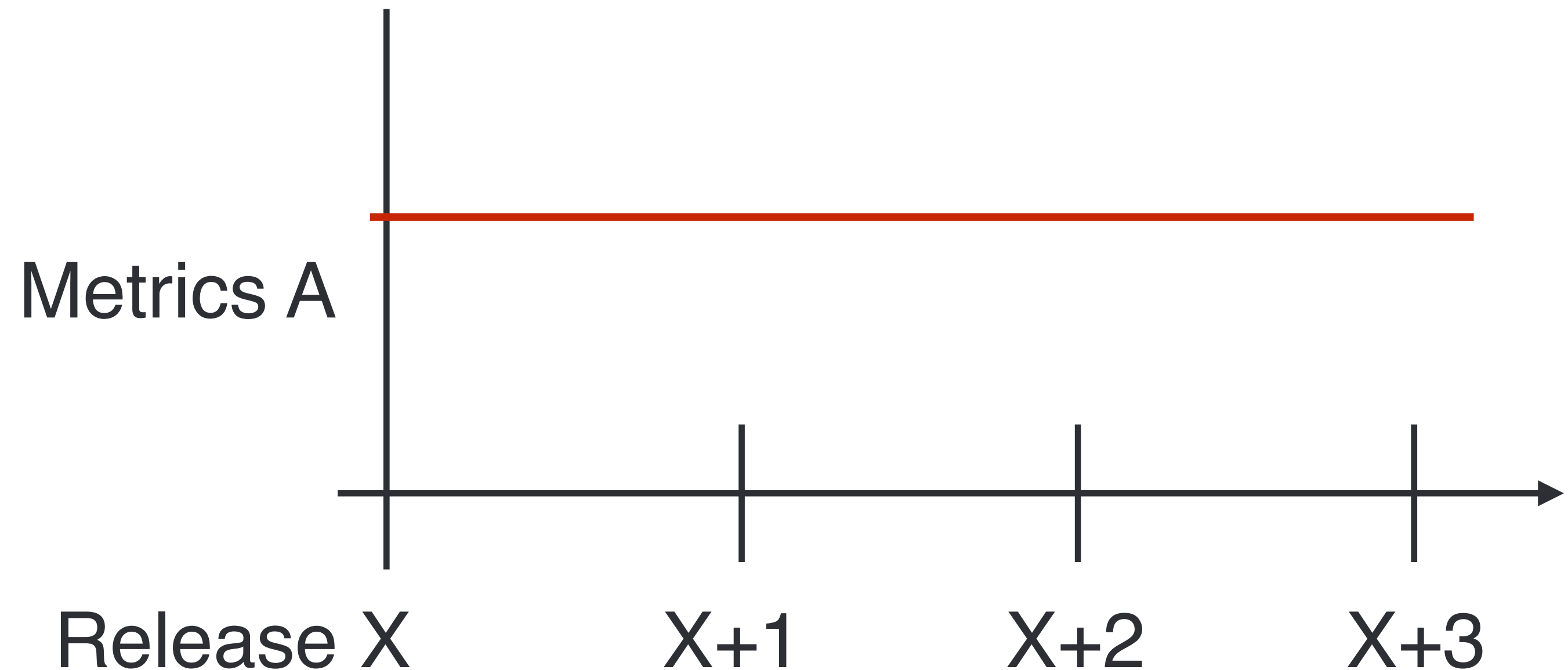
Most studies assume:

Data in a period are similar to one in other periods



bug report data
for **6 years**

1 prediction model



Are Fix-Inducing Changes a Moving Target?

A Longitudinal Case Study of Just-In-Time Defect Prediction

Shane McIntosh, *Member, IEEE* and Yasutaka Kamei, *Member, IEEE*

Abstract—Just-In-Time (JIT) models identify fix-inducing code changes. JIT models are trained using techniques that assume that past fix-inducing changes are similar to future ones. However, this assumption may not hold, e.g., as system complexity tends to accrue, expertise may become more important as systems age.

In this paper, we study JIT models as systems evolve. Through a longitudinal case study of 37,524 changes from the rapidly evolving QT and OPENSTACK systems, we find that fluctuations in the properties of fix-inducing changes can impact the performance and interpretation of JIT models. More specifically: (a) the discriminatory power (AUC) and calibration (Brier) scores of JIT models drop considerably one year after being trained; (b) the role that code change properties (e.g., Size, Experience) play within JIT models fluctuates over time; and (c) those fluctuations yield over- and underestimates of the future impact of code change properties on the likelihood of inducing fixes. To avoid erroneous or misleading predictions, JIT models should be retrained using recently recorded data (within three months). Moreover, quality improvement plans should be informed by JIT models that are trained using six months (or more) of historical data, since they are more resilient to period-specific fluctuations in the importance of code change properties.

Index Terms—Just-In-Time prediction, defect prediction, mining software repositories



old software development



The properties are similar?



future software development

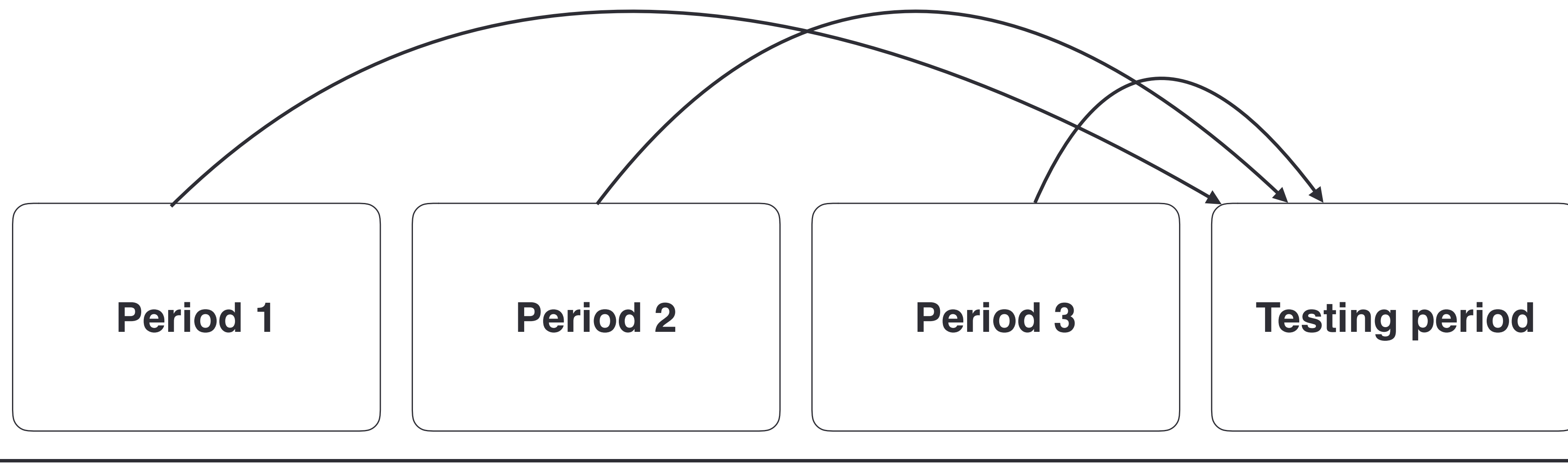
Are Fix-Inducing Changes a Moving Target?

A Longitudinal Case Study of Just-In-Time Defect Prediction

Shane McIntosh, *Member, IEEE* and Yasutaka Kamei, *Member, IEEE*

Abstract—Just-In-Time (JIT) models identify fix-inducing code changes. JIT models are trained using techniques that assume that past fix-inducing changes are similar to future ones. However, this assumption may not hold, e.g., as system complexity tends to accrue, expertise may become more important as systems age.

In this paper, we study JIT models as systems evolve. Through a longitudinal case study of 37,524 changes from the rapidly evolving QT and OPENSTACK systems, we find that fluctuations in the properties of fix-inducing changes can impact the performance and interpretation of JIT models. More specifically: (a) the discriminatory power (AUC) and calibration (Brier) scores of JIT models drop considerably one year after being trained; (b) the role that code change properties (e.g., Size, Experience) play within JIT models fluctuates over time.



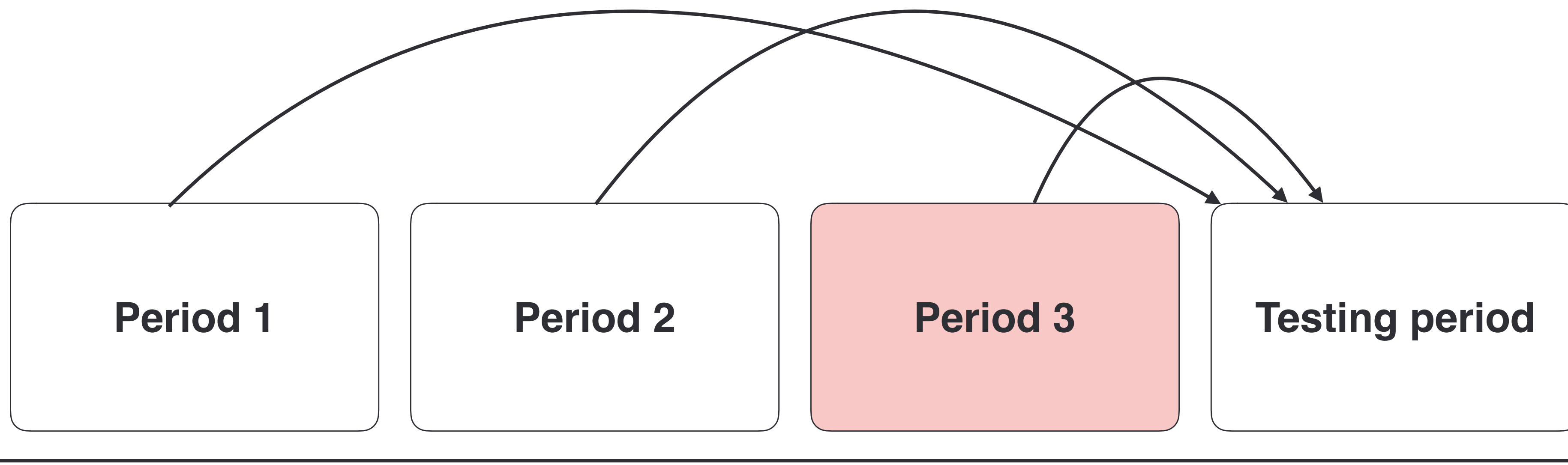
Are Fix-Inducing Changes a Moving Target?

A Longitudinal Case Study of Just-In-Time Defect Prediction

More recent data contributes the prediction accuracy

accrue, expertise may become more important as systems age.

In this paper, we study JIT models as systems evolve. Through a longitudinal case study of 37,524 changes from the rapidly evolving QT and OPENSTACK systems, we find that fluctuations in the properties of fix-inducing changes can impact the performance and interpretation of JIT models. More specifically: (a) the discriminatory power (AUC) and calibration (Brier) scores of JIT models drop considerably one year after being trained; (b) the role that code change properties (e.g., Size, Experience) play within JIT models fluctuates over time.

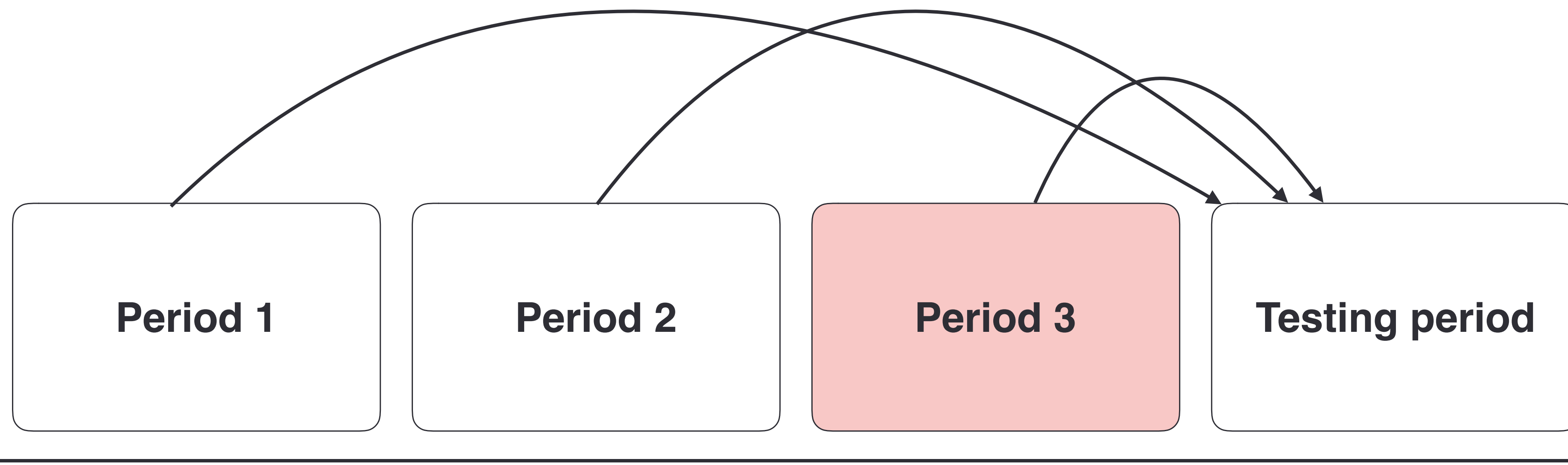


Are Fix-Inducing Changes a Moving Target?

A Longitudinal Case Study of Just-In-Time Defect Prediction

More recent data contributes the prediction accuracy

The magnitude of the importance scores fluctuate as systems evolve



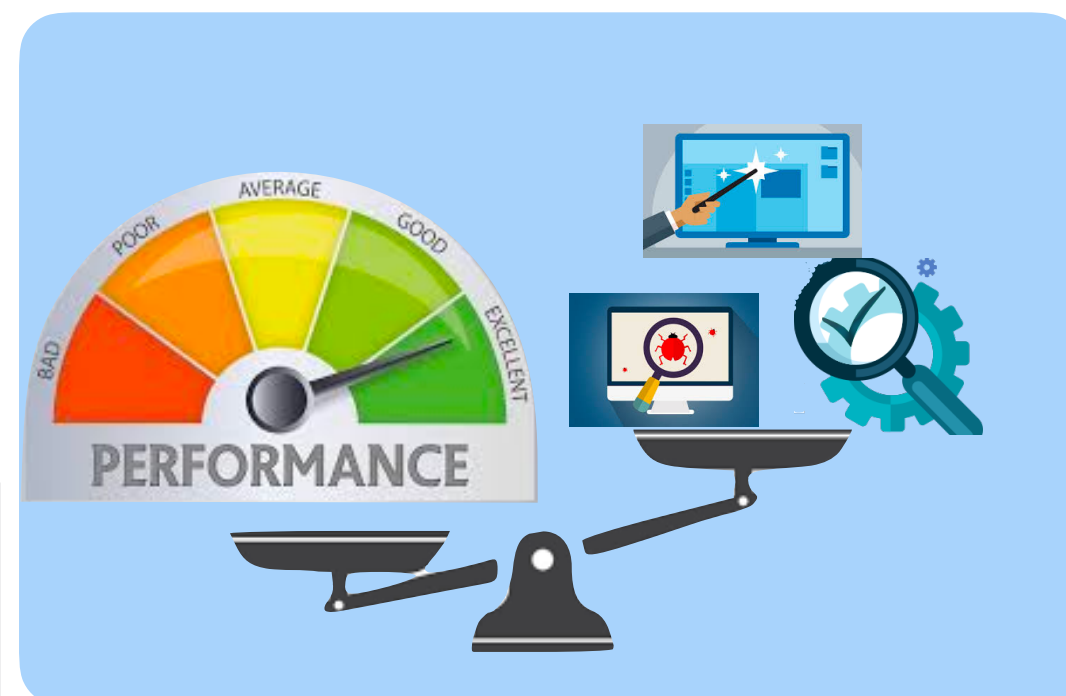
This paper assumes: Developers' activities might vary during Release Cycle (RC)

Developers focus on ...

Fixing
Serious bug?



Improving
performance?



Making
New features?



Testing?



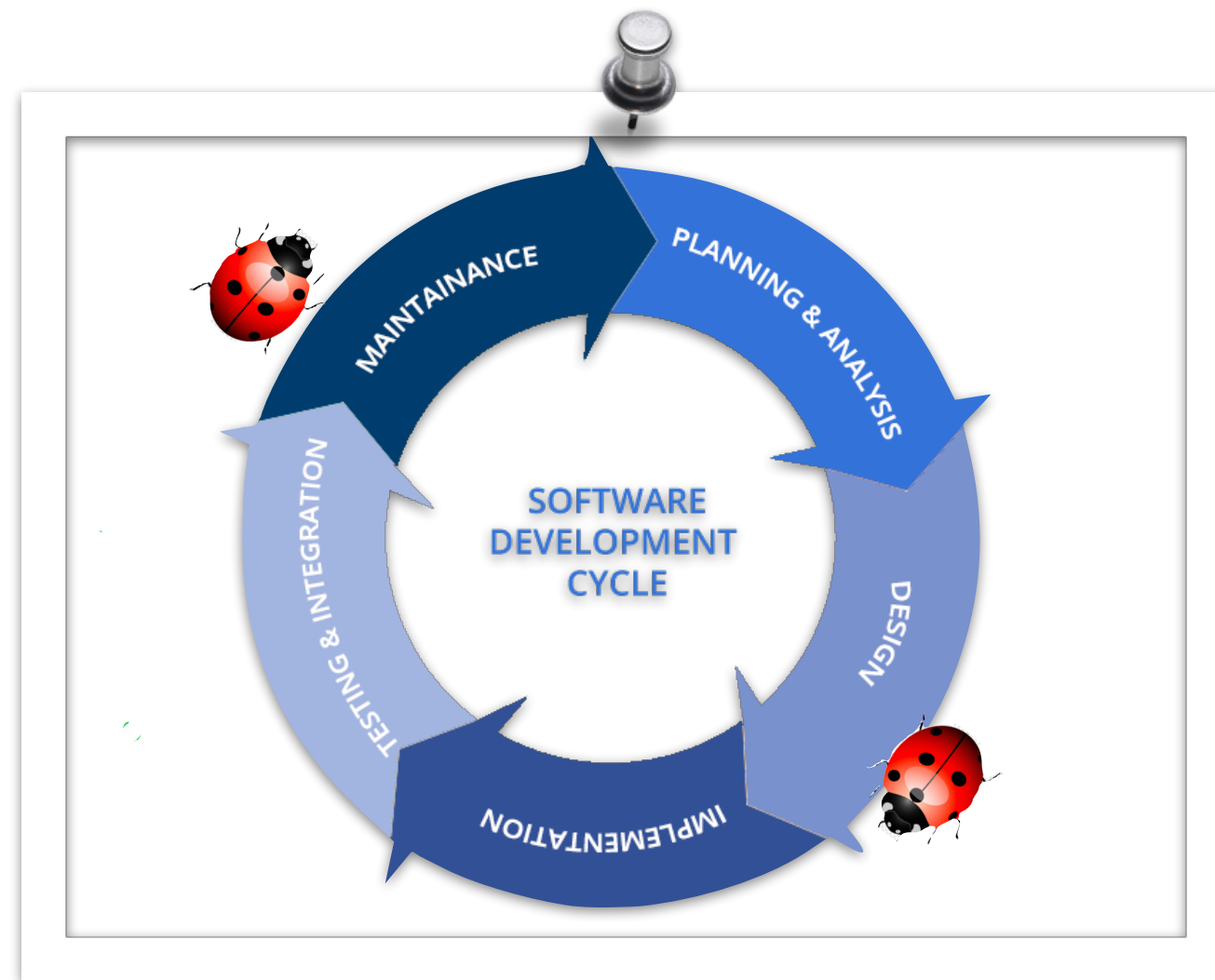
Release X

Minor Release
(X.1)

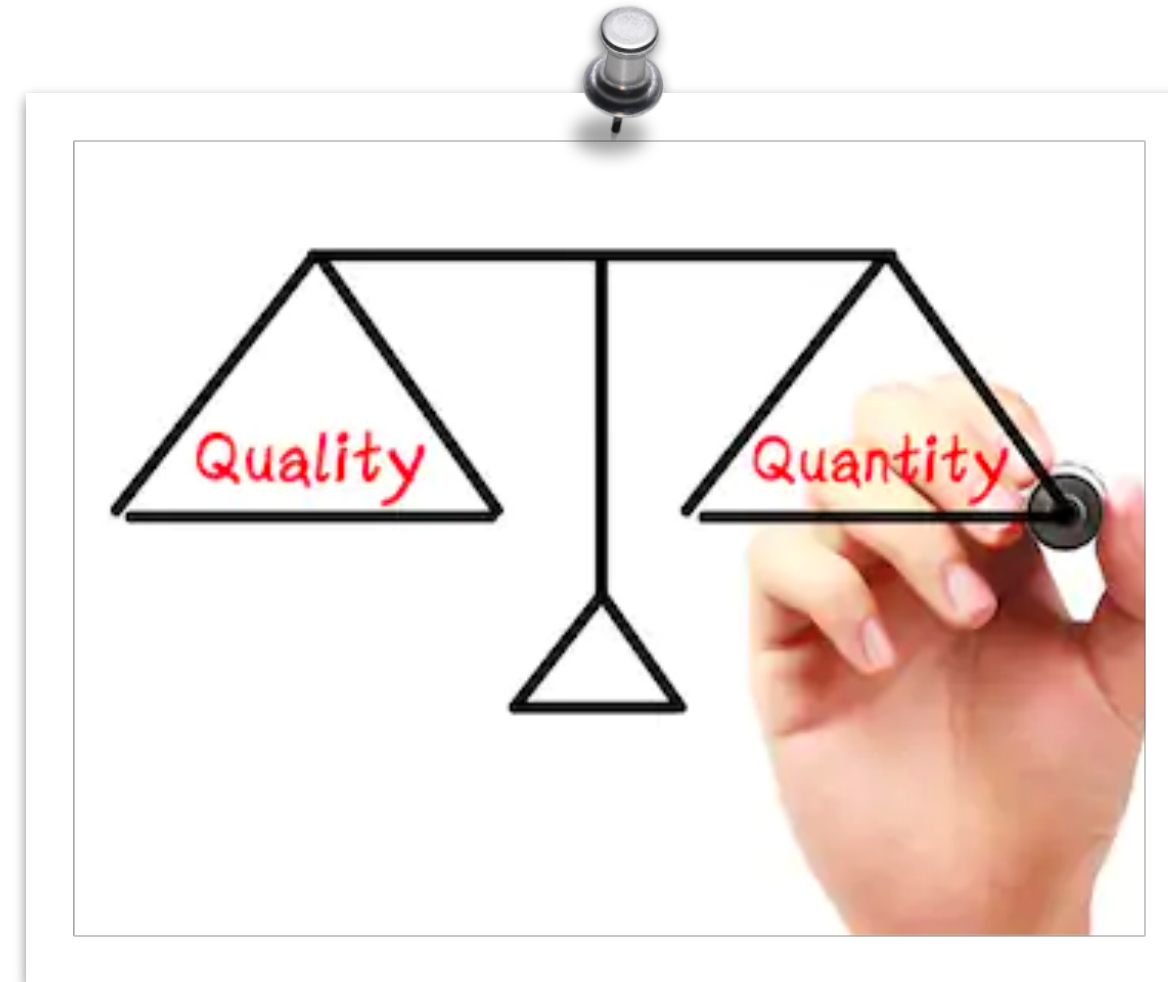
Minor Release
(X.2)

Release X+1

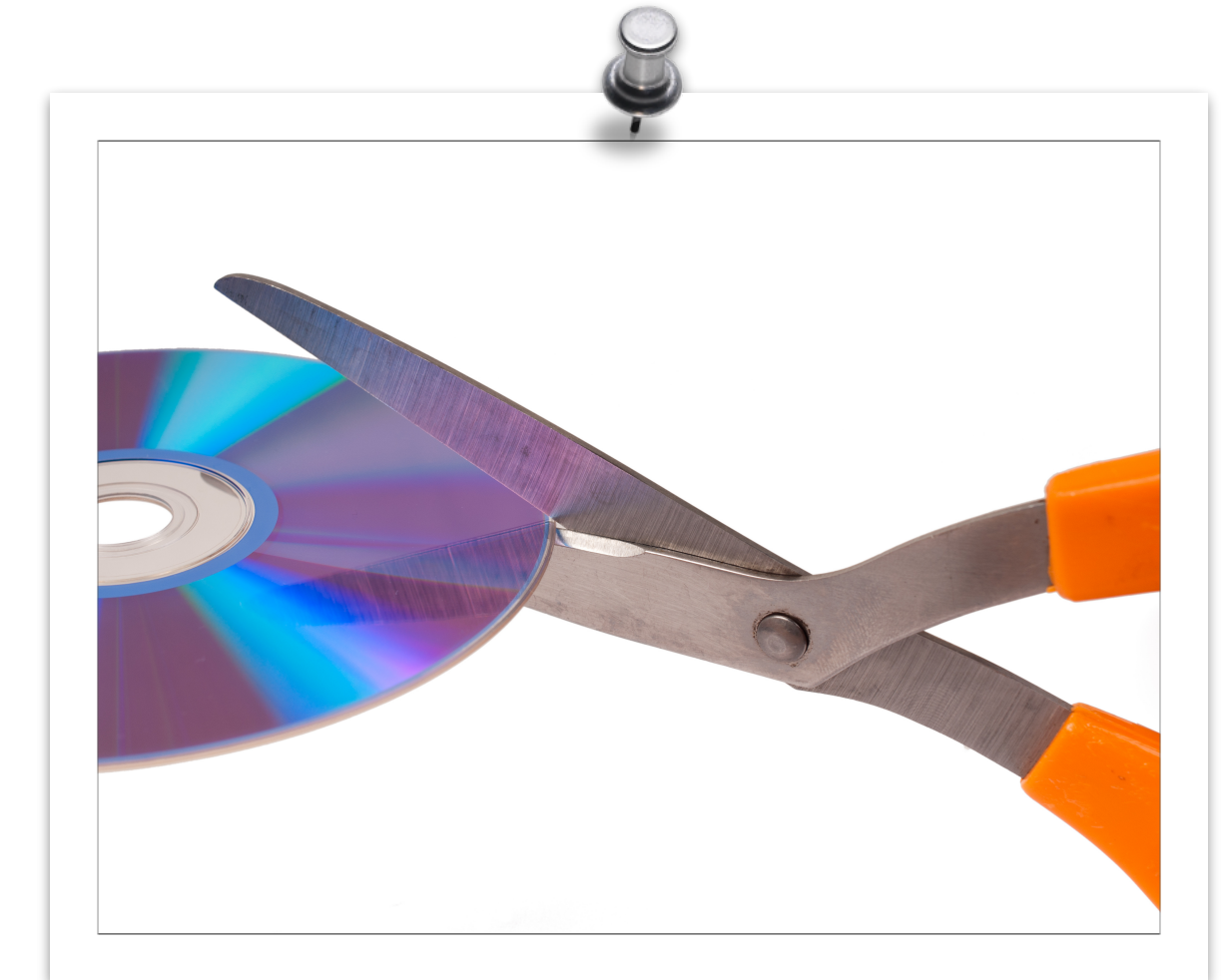
Our goal:
Understand the impact of the RC on bug priority prediction



RQ1: Do bug-fixing activities vary across within release cycle?



RQ2: Do Release-aware models for priority change outperform Global model?



RQ3: How fine-grained data should be used?

**Yearly 1 main release
and 2 service releases**

Release name	Platform version	Main release	SR1	SR2
Indigo	3.7	June 22, 2011	September 23, 2011	February 24, 2012
Helios	3.6	June 23, 2010	September 24, 2010	February 25, 2011
Galileo	3.5	June 24, 2009	September 25, 2009	February 26, 2010
Ganymede	3.4	June 25, 2008	September 24, 2008	February 25, 2009
Europa	3.3	June 27, 2007	September 28, 2007	February 29, 2008

**Yearly 1main release
and 2 service releases**

Release name	Platform version	Main release	SR1	SR2
Indigo	3.7	<u>June 22, 2011</u>	September 23, 2011	February 24, 2012
Helios	3.6	<u>June 23, 2010</u>	September 24, 2010	February 25, 2011
Galileo	3.5	<u>June 24, 2009</u>	September 25, 2009	February 26, 2010
Ganymede	3.4	<u>June 25, 2008</u>	September 24, 2008	February 25, 2009
Europa	3.3	<u>June 27, 2007</u>	September 28, 2007	February 29, 2008

Yearly synchronized

**Yearly 1main release
and 2 service releases**

Release name	Platform version	Main release	SR1	SR2
Indigo	3.7	June 22, 2011	September 23, 2011	February 24, 2012
Helios	3.6	June 23, 2010	September 24, 2010	
Galileo	3.5	June 24, 2009	September 25, 2009	February 26, 2010
Ganymede	3.4	June 25, 2008	September 24, 2008	February 25, 2009
Europa	3.3	June 27, 2007	September 28, 2007	February 29, 2008

Yearly synchronized

**Well-developed
8years**

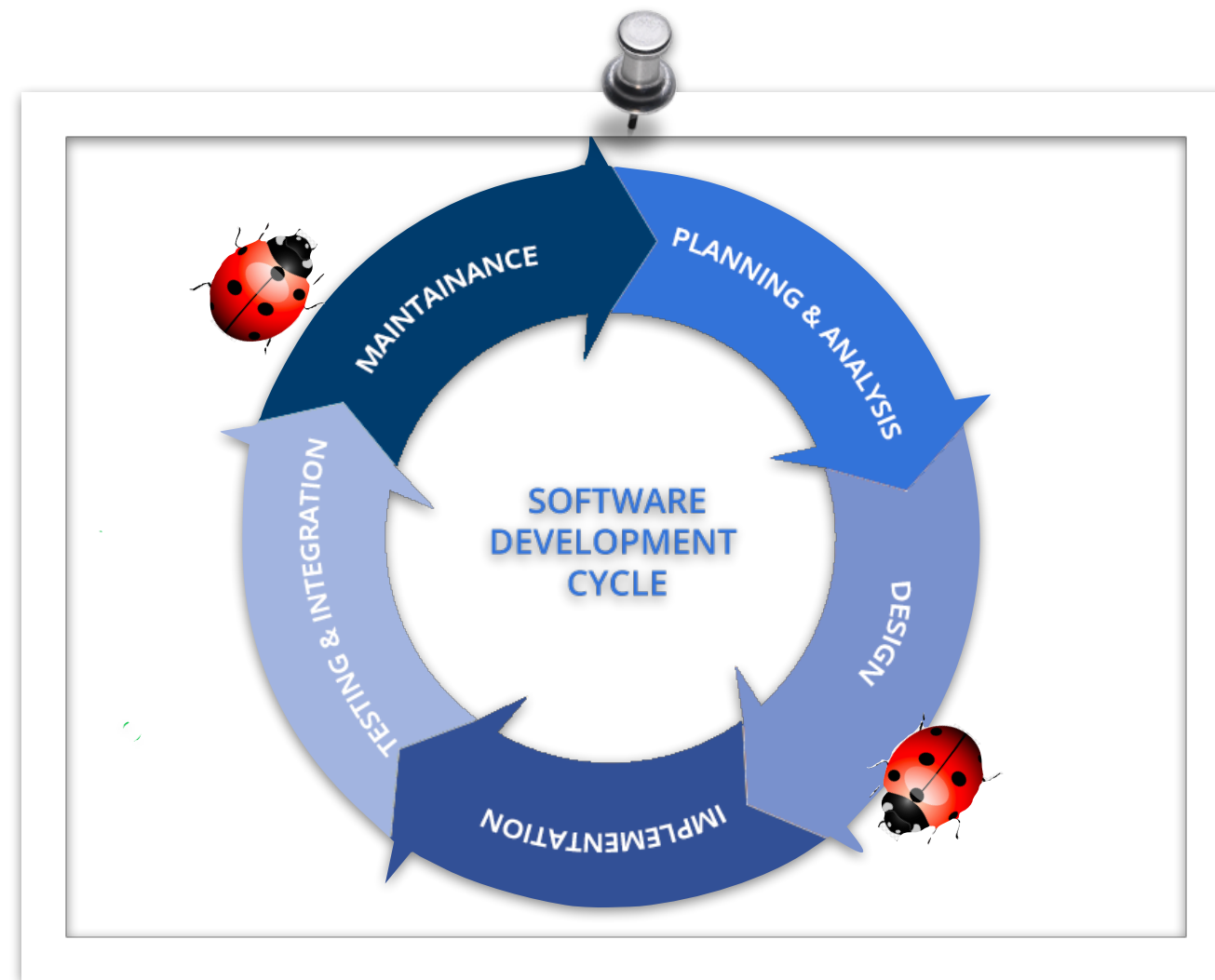
Release name	Platform version	Main release	SR1	SR2
Indigo	3.7	June 22, 2011	September 23, 2011	February 24, 2012
Helios	3.6	June 23, 2010	September 24, 2010	
Galileo	3.5	June 24, 2009	September 25, 2009	February 26, 2010
Ganymede	3.4	June 25, 2008	September 24, 2008	
Europa	3.3	June 27, 2007	September 28, 2007	February 29, 2008

Yearly 1main release
and 2 service releases

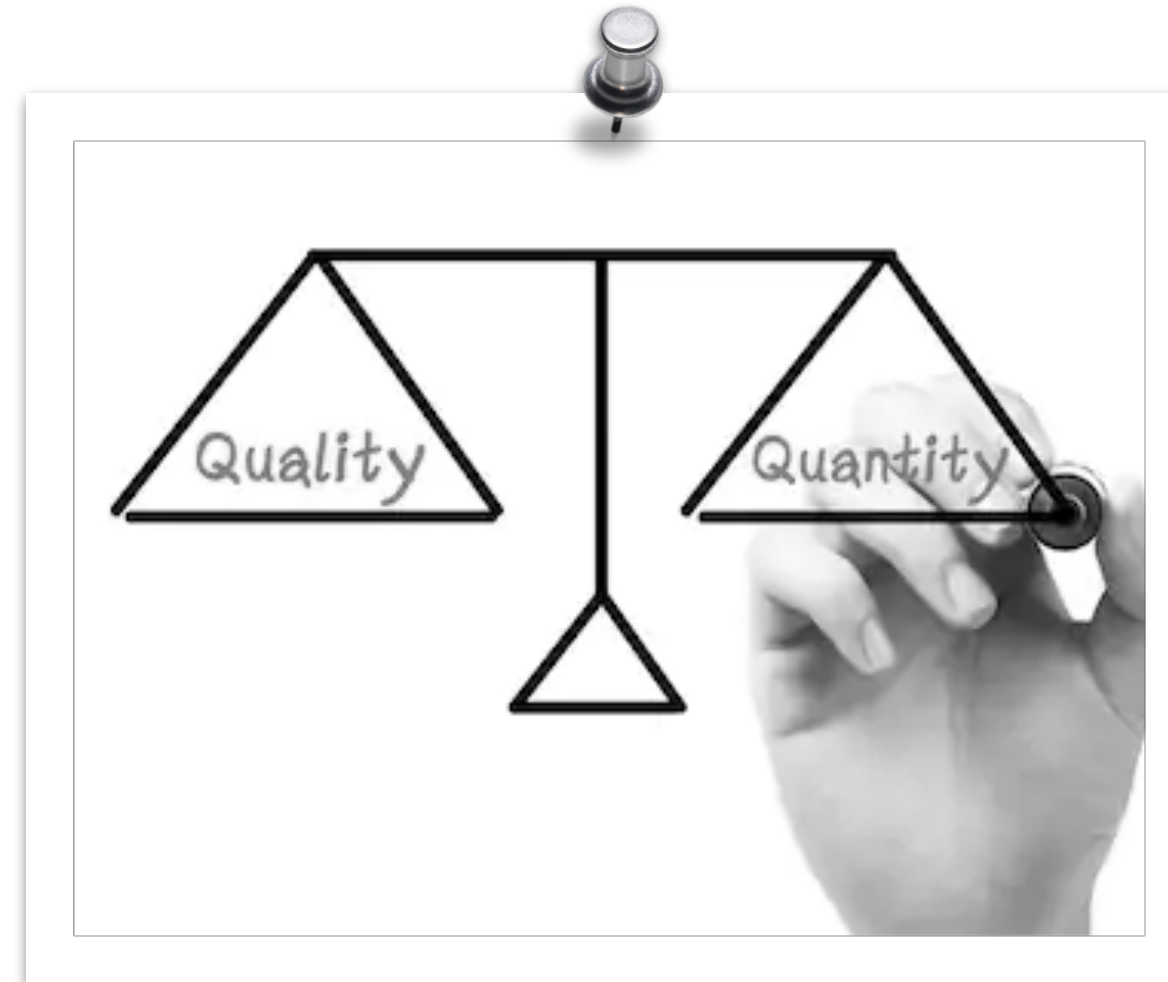
Yearly synchronized

Well-developed
8years

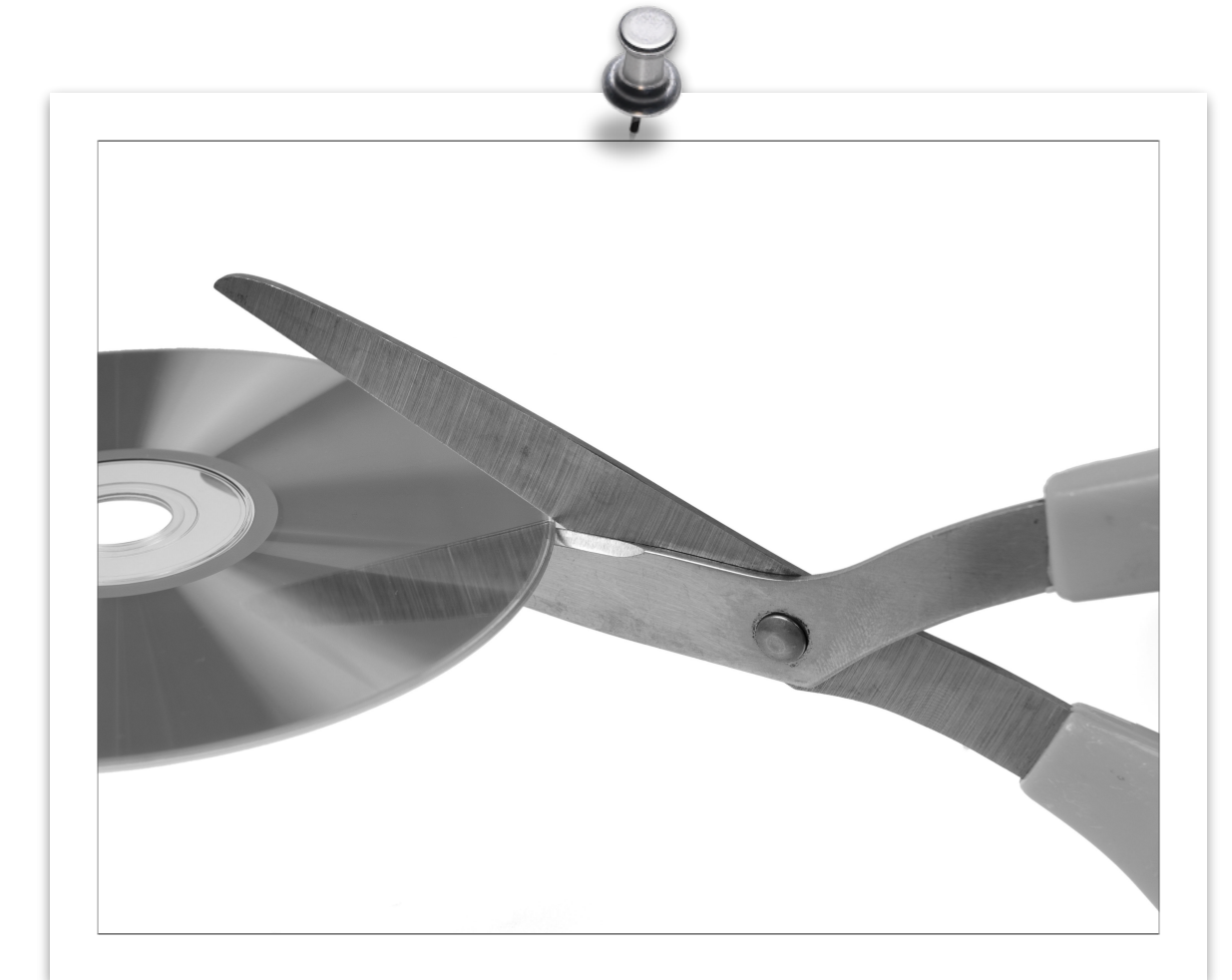
plenty of bug reports
28,924



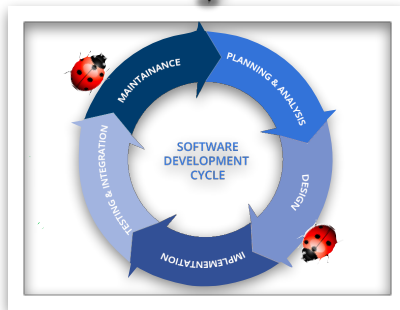
RQ1: Do bug-fixing activities vary across within release cycle?



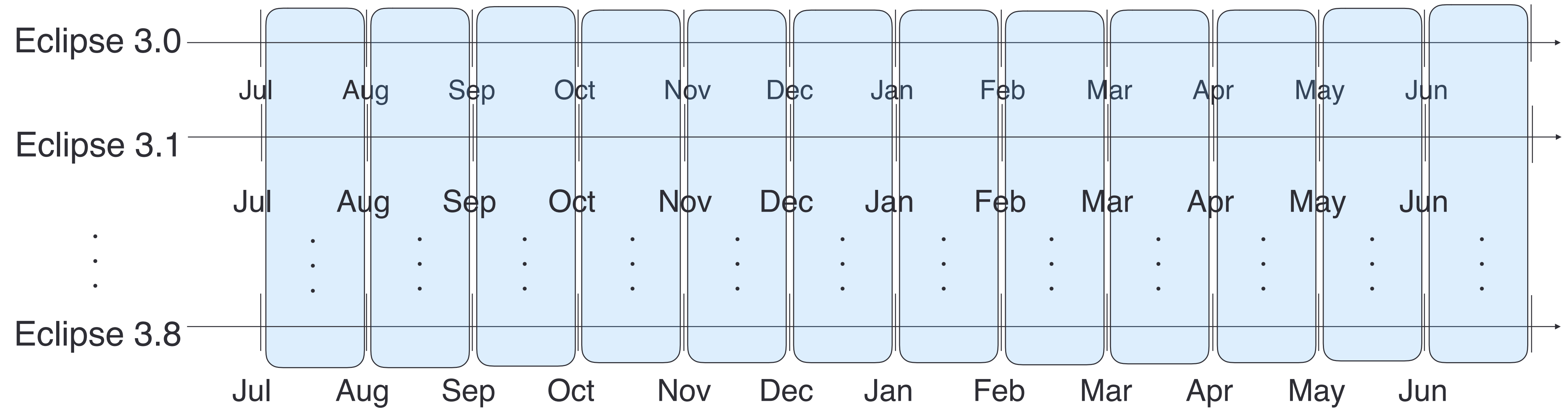
RQ2: Do Release-aware models for priority change outperform Global model?



RQ3: How fine-grained data should be used?



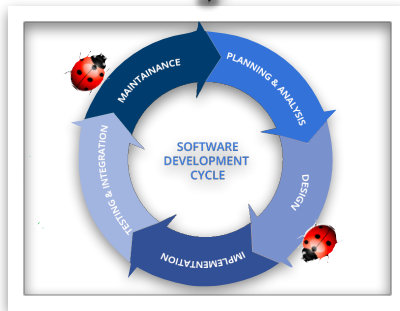
Aggregate metrics by each month



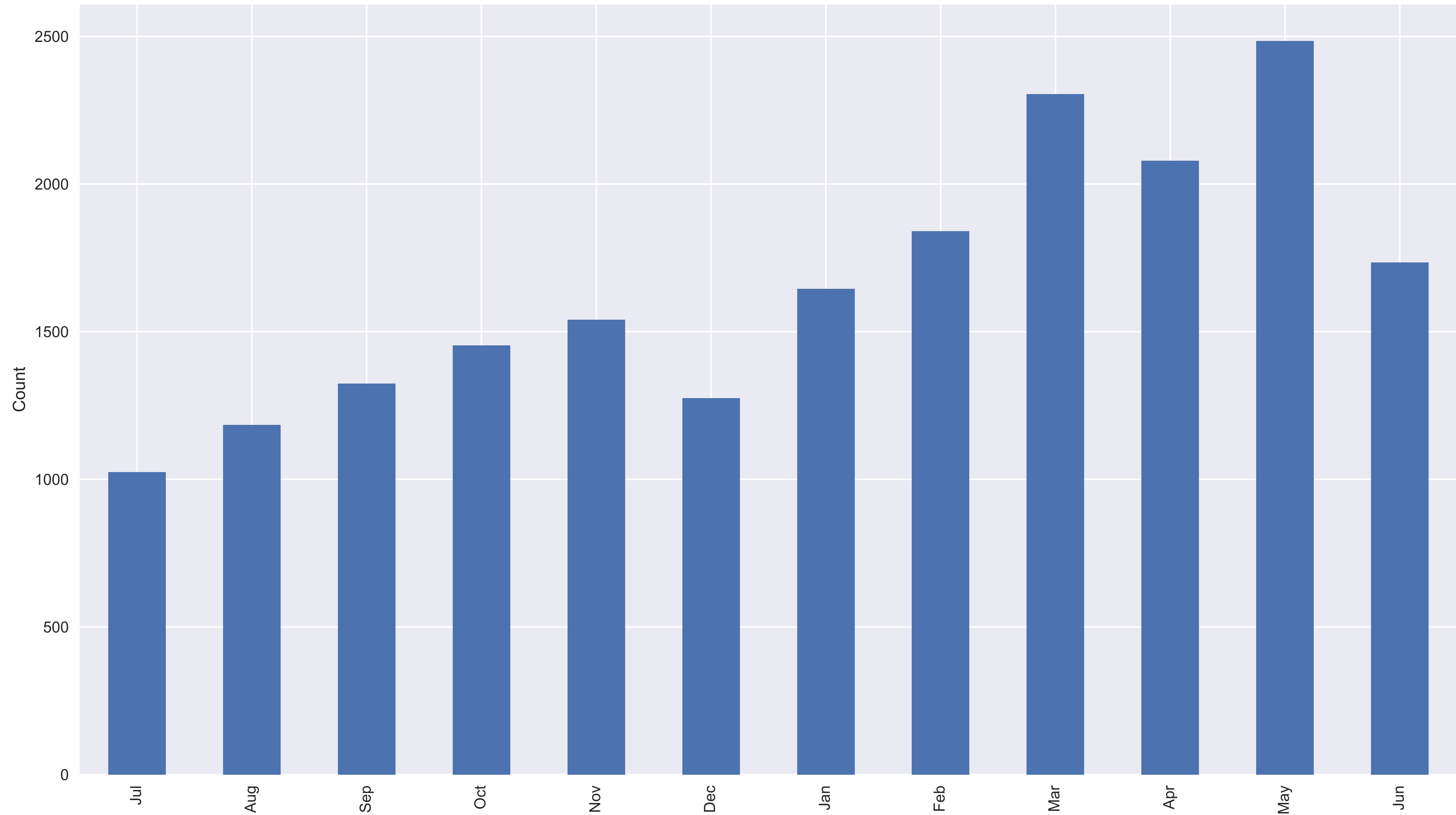
Bug reports

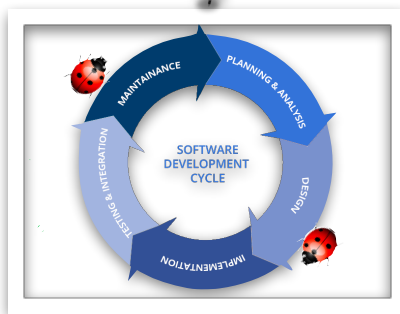
Fixing time

Priority changes

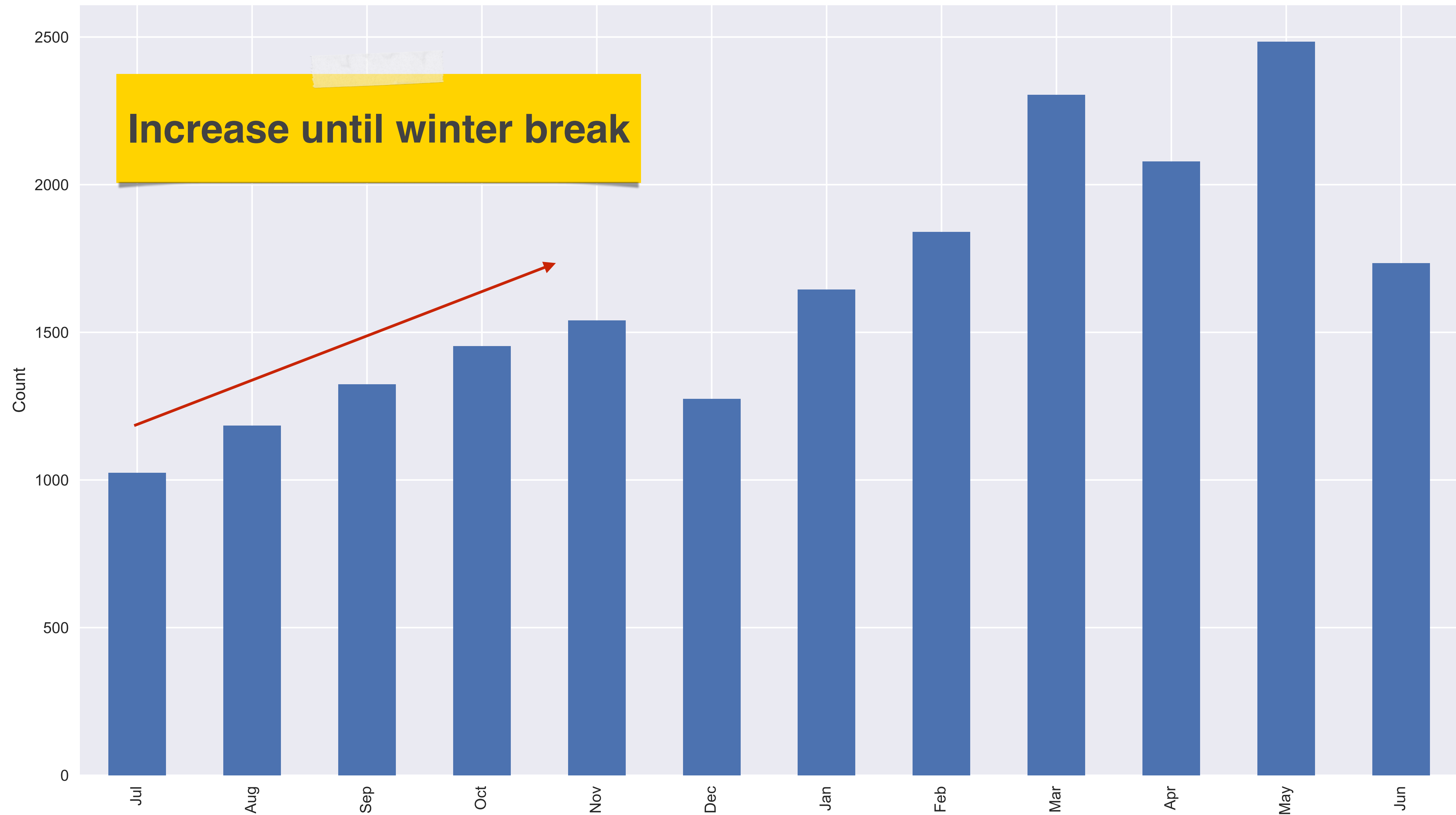


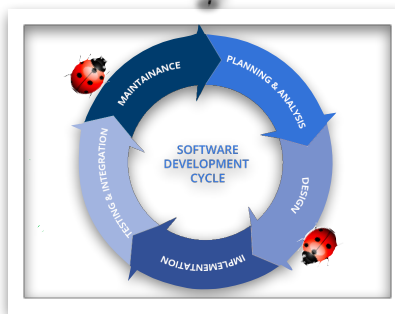
Growing number of reports towards end of the cycle



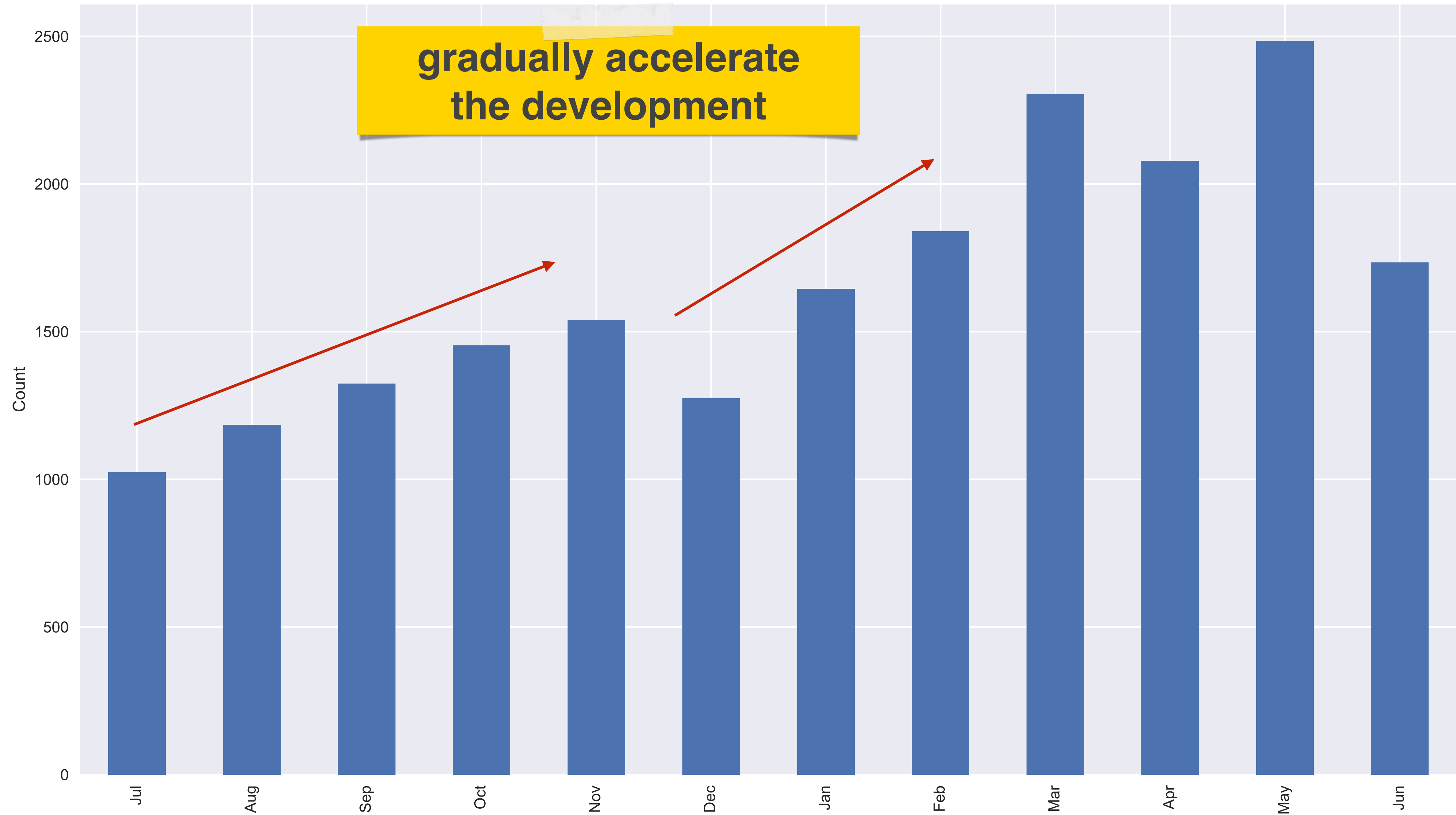


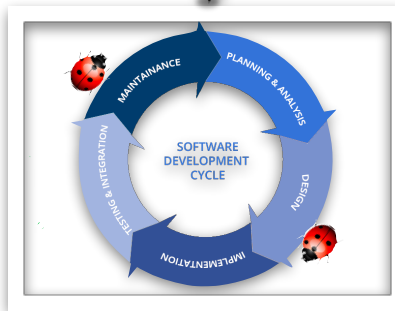
Growing number of reports towards end of the cycle



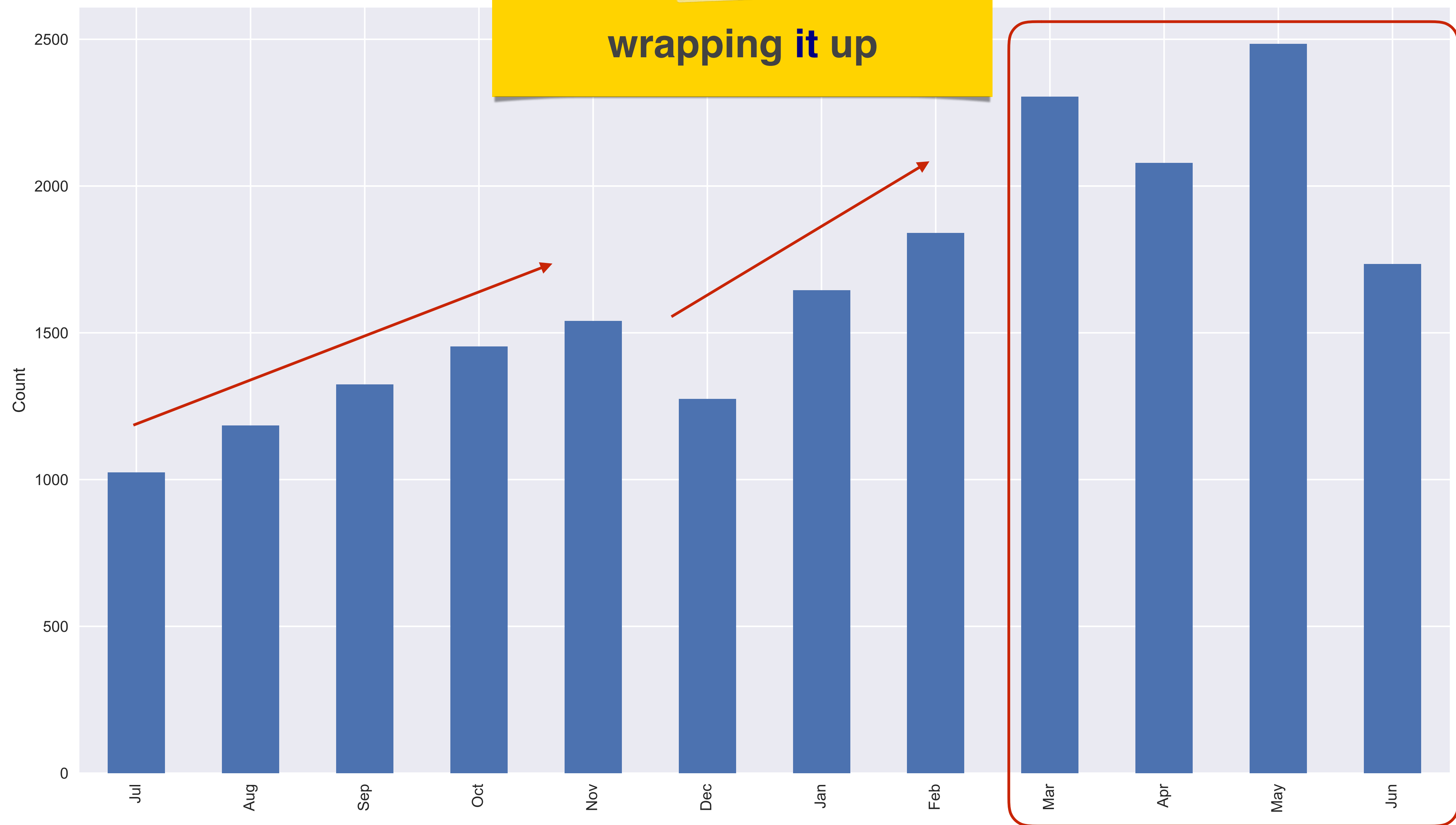


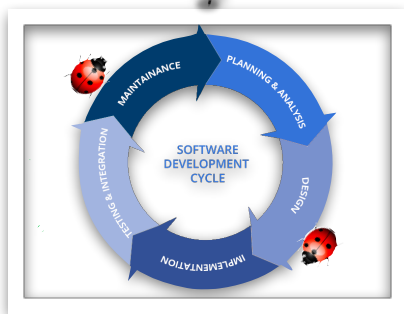
Growing number of reports towards end of the cycle



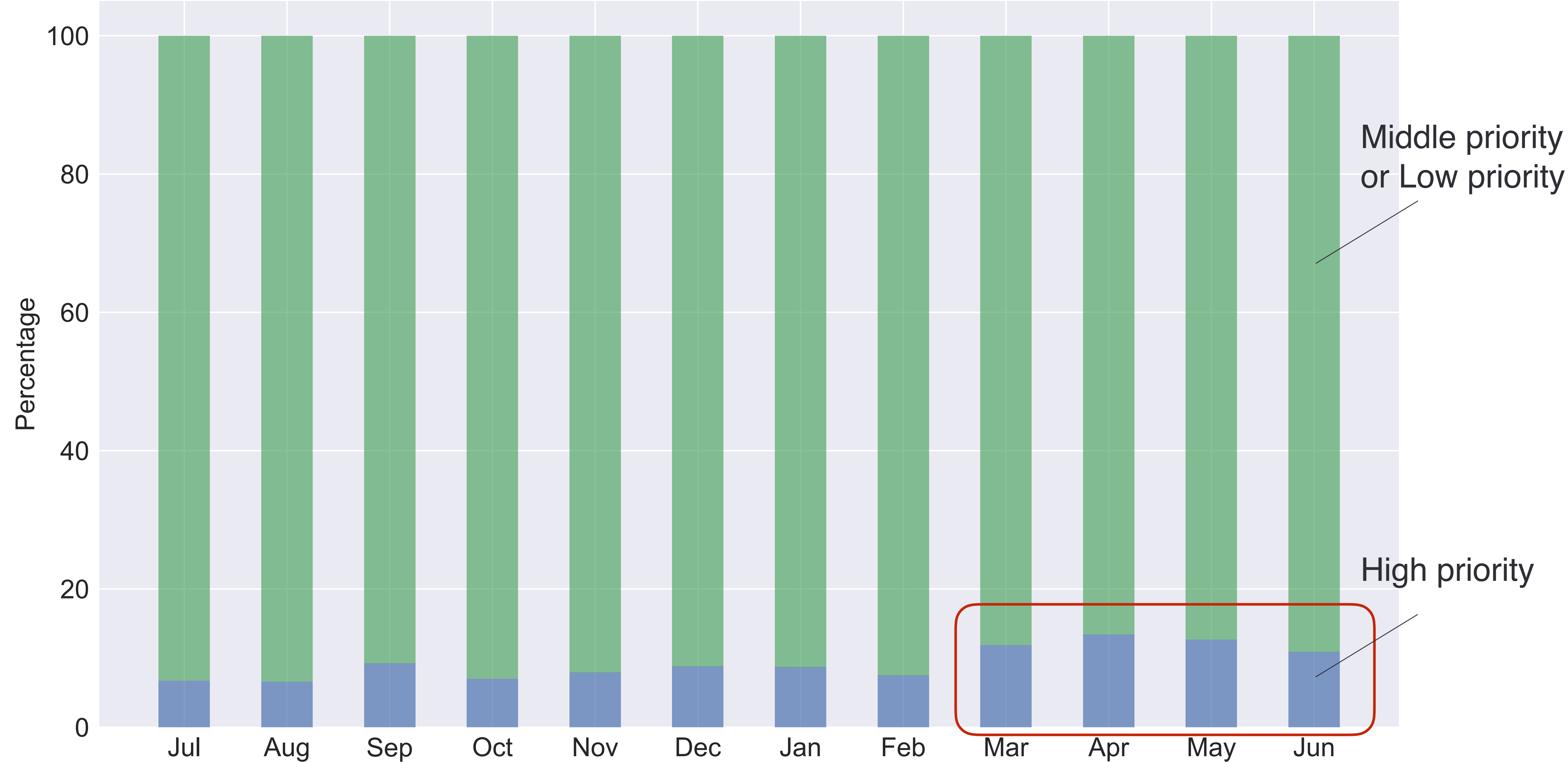


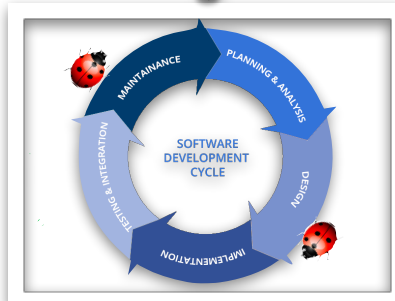
Growing number of reports towards end of the cycle



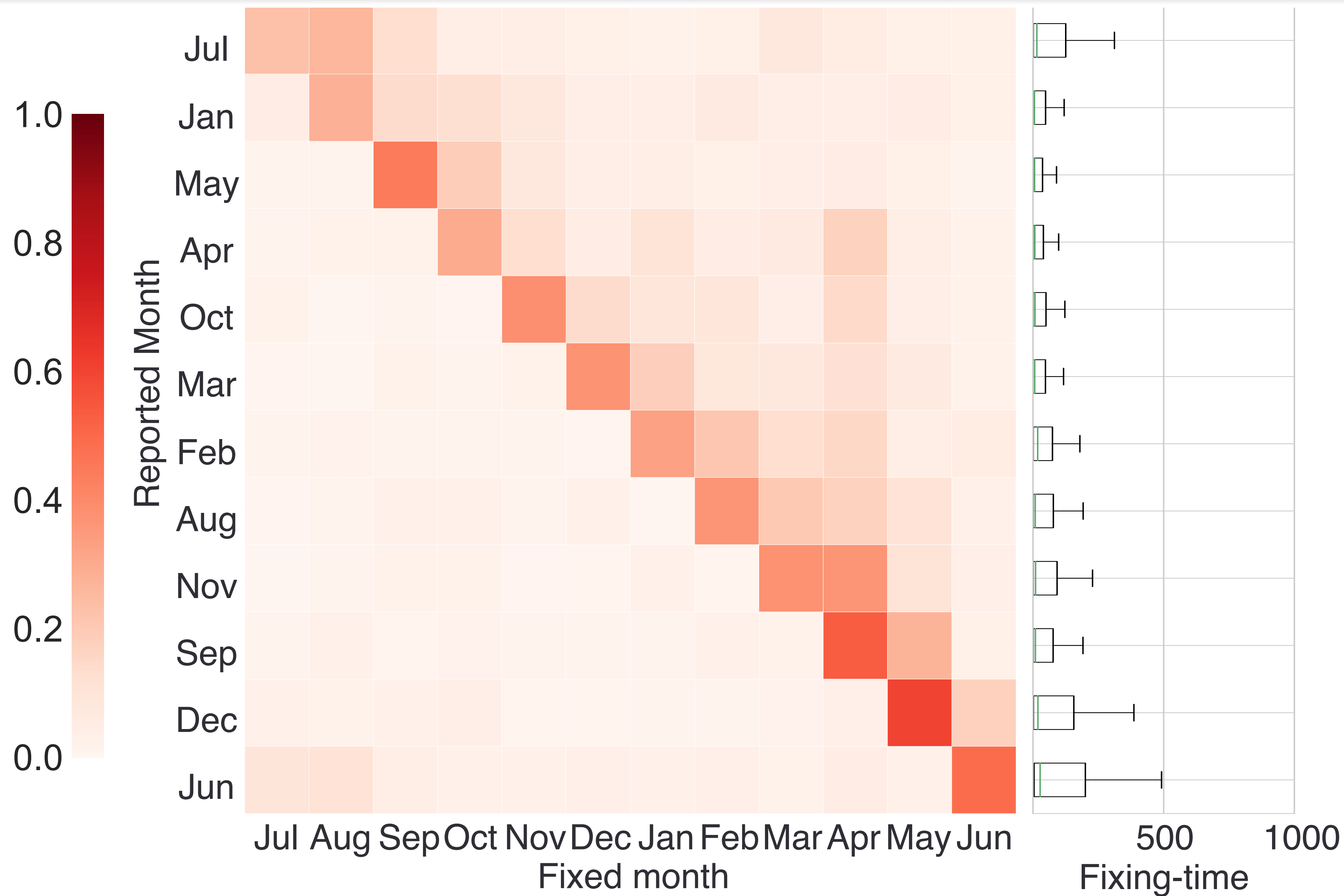


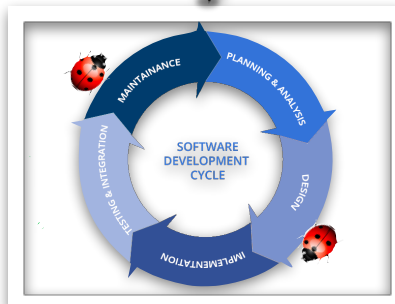
growing percentage of reports that will receive high priority at some point in time



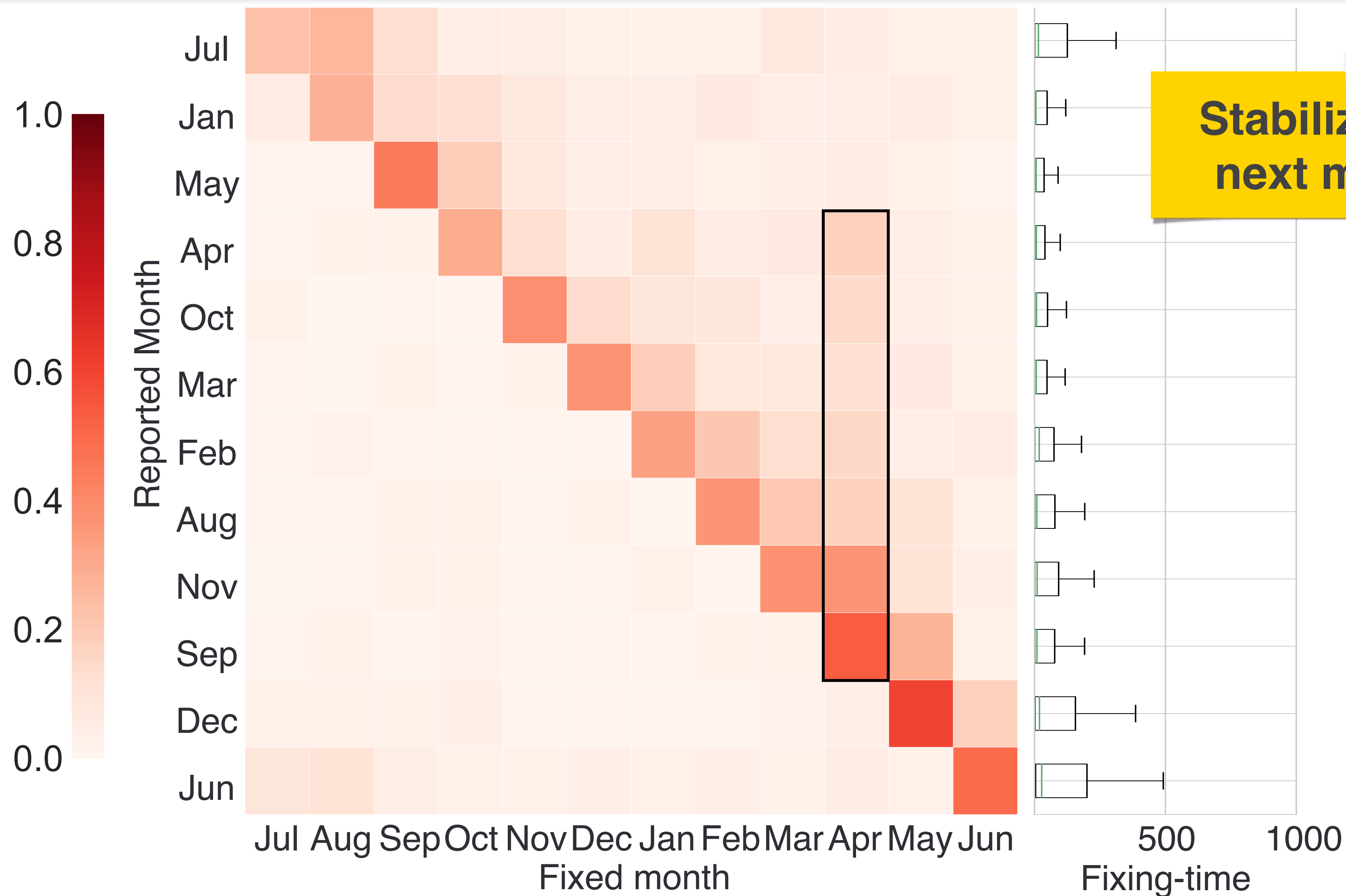


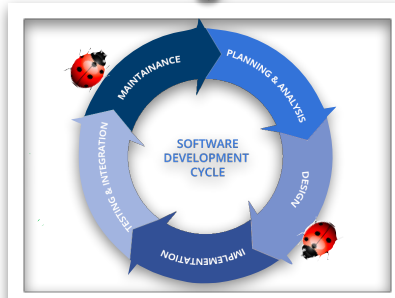
Bugs reported in June tend to be carried over into next release



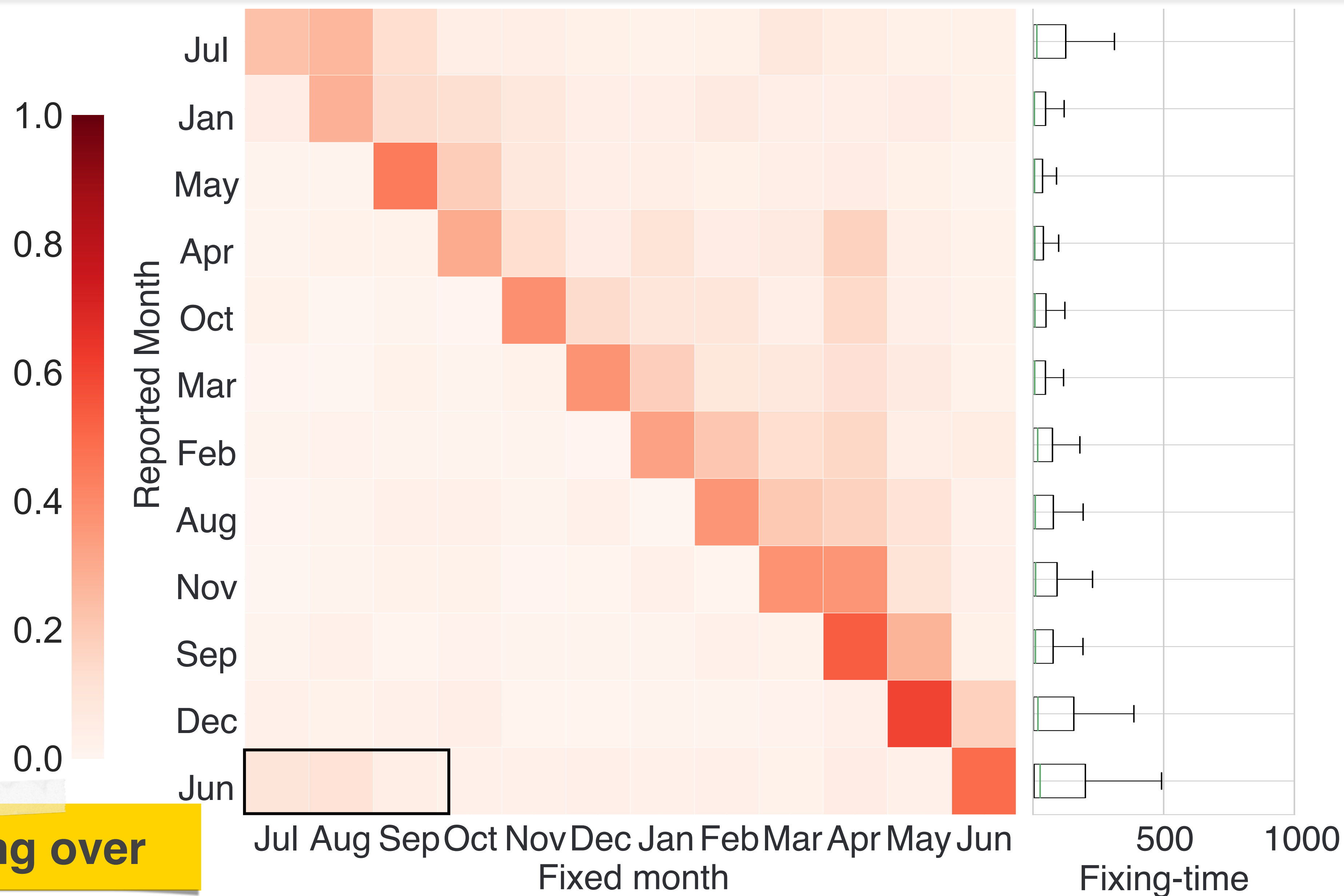


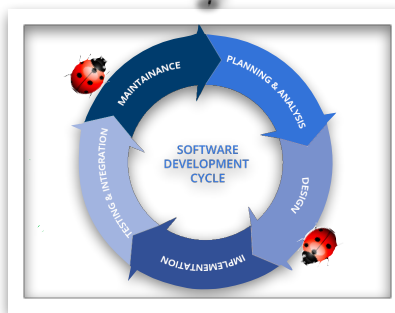
Bugs reported in June tend to be carried over into next release



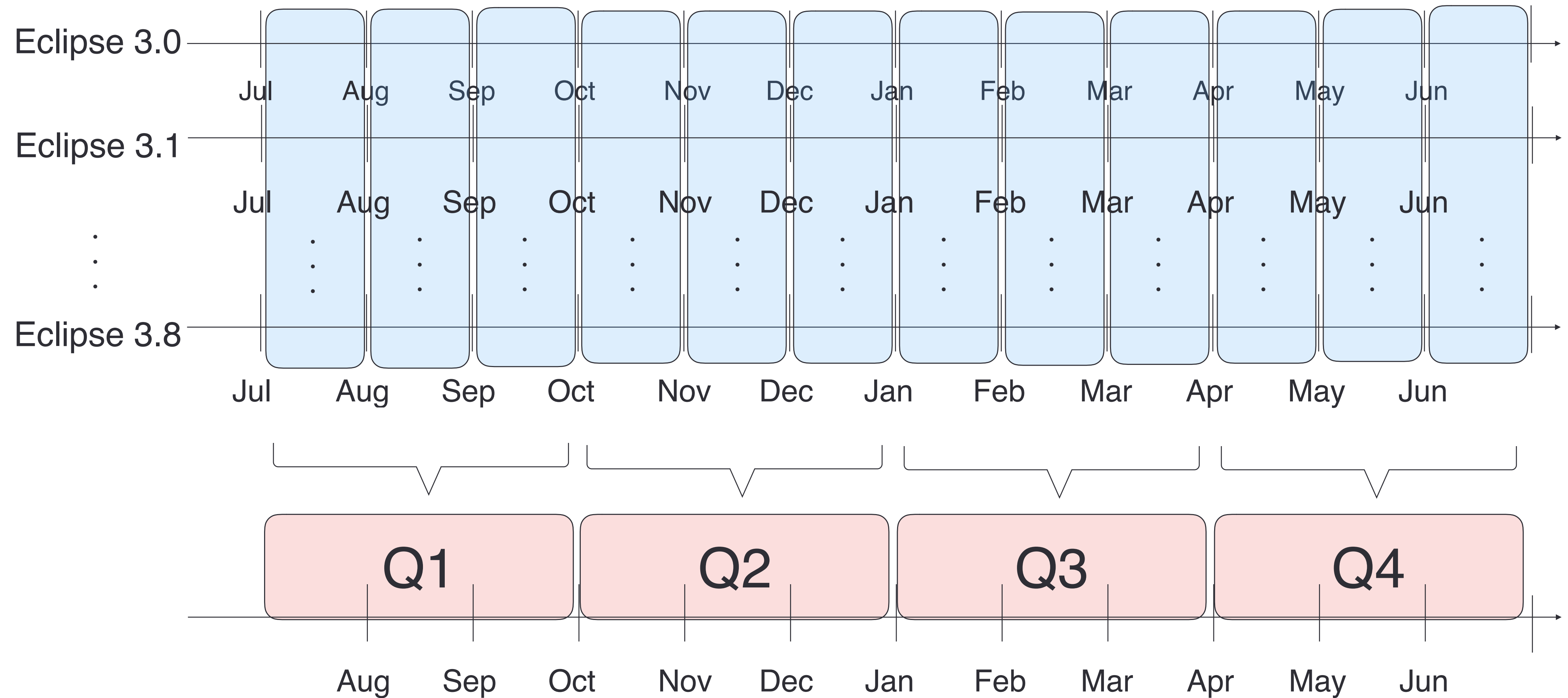


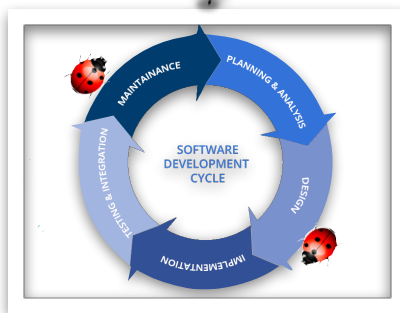
Bugs reported in June tend to be carried over into next release



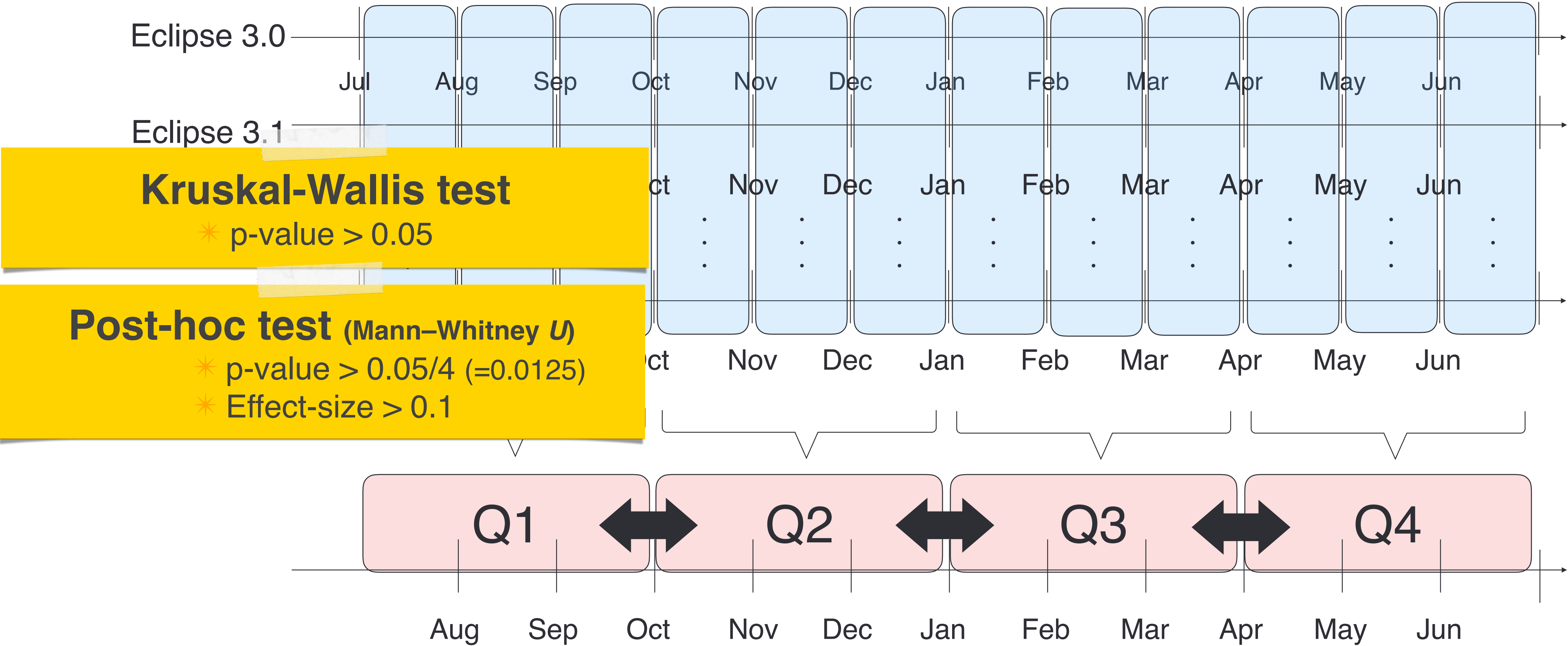


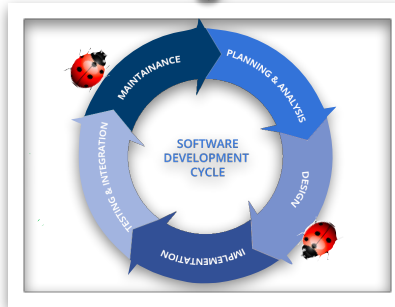
Statistical tests





Statistical tests

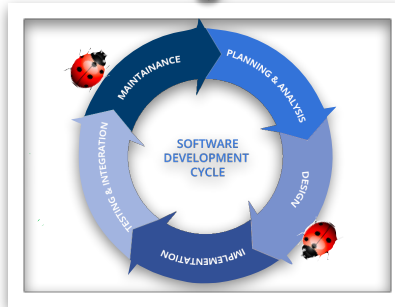




Metrics for the statistical tests

Unit	Dimension	Metrics
Daily	Report	# of reports
		# of High/Mid/Low priority reports
		# of High/Mid/Low severity reports
		# of enhancement
	Close	# of reports
		# of High/Mid/Low priority reports
		# of High/Mid/Low severity reports
		# of enhancement reports
	Comment	# of comments
		# of commentator
	Changes	# of changes priority low to high
		# of changes priority high to low
		# of changes severity low to high
		# of changes severity high to low
	Commits	# of commits
		# of files
		# of adds
		# of deletes

Unit	Dimension	Metrics
Bug	Report	#of description words
		bayes socre
		# of priority change
		# of assigned
		# of cc
		# of comments
		# of commentator
		time for assigning bug
	Commit	time for fixing bug
		# of commits
		# of files
		# of adds
		# of deletes
		median of adds per file
		median of delets per file



14 metrics are **not** affected by release cycle

Unit	Dimension	Metrics
Daily	Report	# of reports
		# of High/Mid/ Low priority reports
		# of High/Mid/Low severity reports
		# of enhancement
	Close	# of reports
		# of High/Mid/ Low priority reports
		# of High/Mid/Low severity reports
		# of enhancement reports
	Comment	# of comments
		# of commentator
	Changes	# of changes priority low to high
		# of changes priority high to low
		# of changes severity low to high
		# of changes severity high to low
	Commits	# of commits
		# of files
		# of adds
		# of deletes

Unit	Dimension	Metrics
Bug	Report	#of description words
		bayes score
		# of priority change
		# of assigned
		# of cc
		# of comments
		# of commentator
		time for assigning bug
	Commit	time for fixing bug
		# of commits
		# of files
		# of adds
		# of deletes
		median of adds per file
		median of deletes per file

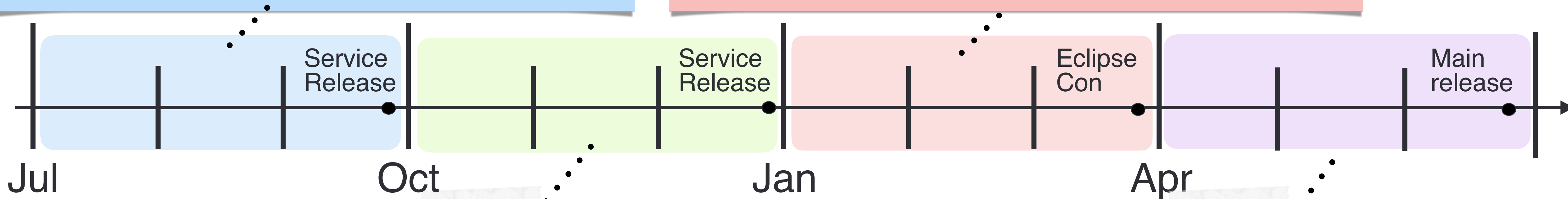
Developers are focusing on different activities during Release Cycle

Jul - Sep: Developers take longer time to discuss and fix bugs

of CC, comments, commentator ↑
Time for assigning, fixing ↑

Jan - Mar: Developers develop new features or make huge changes

of enhancement report close ↑
of commits, files, add, delete ↑

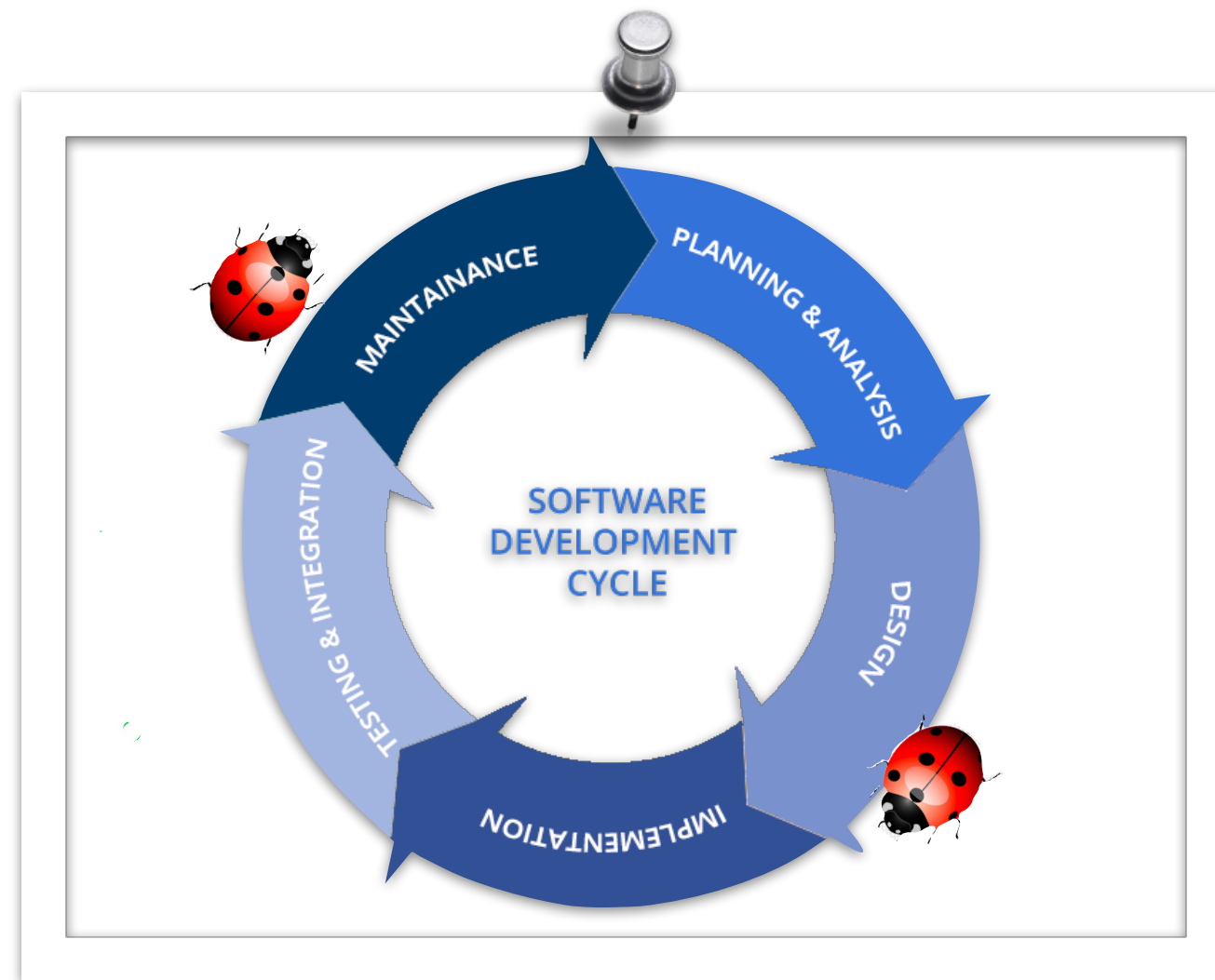


Oct - Dec: Developers are less active
most metrics ↓

Apr - Jun: Developers change priorities and fix high priority bugs

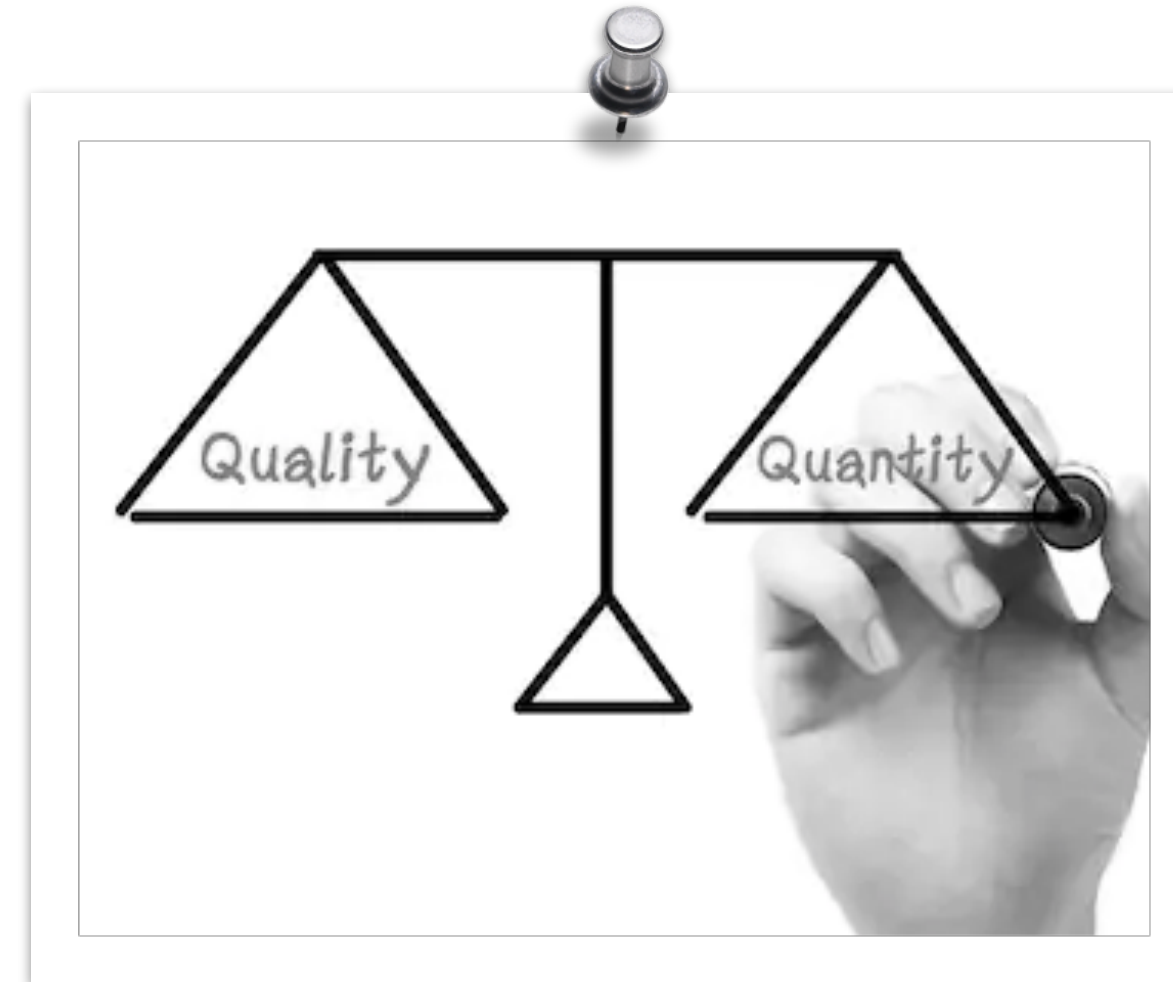
priority/severity change ↑
High priority close ↑

We might need to build **fine-grained, release aware** models

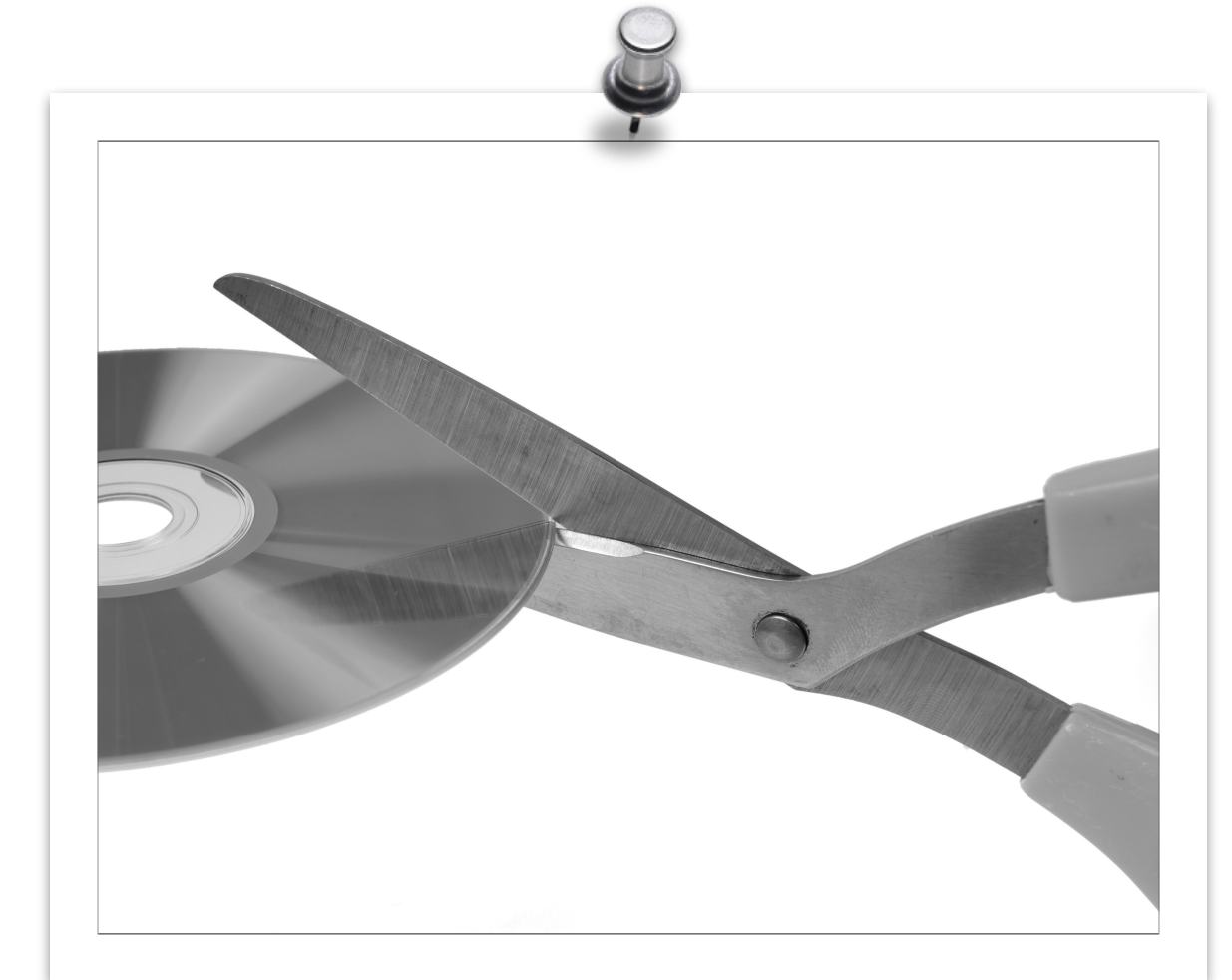


RQ1: Do bug-fixing activities vary across within release cycle?

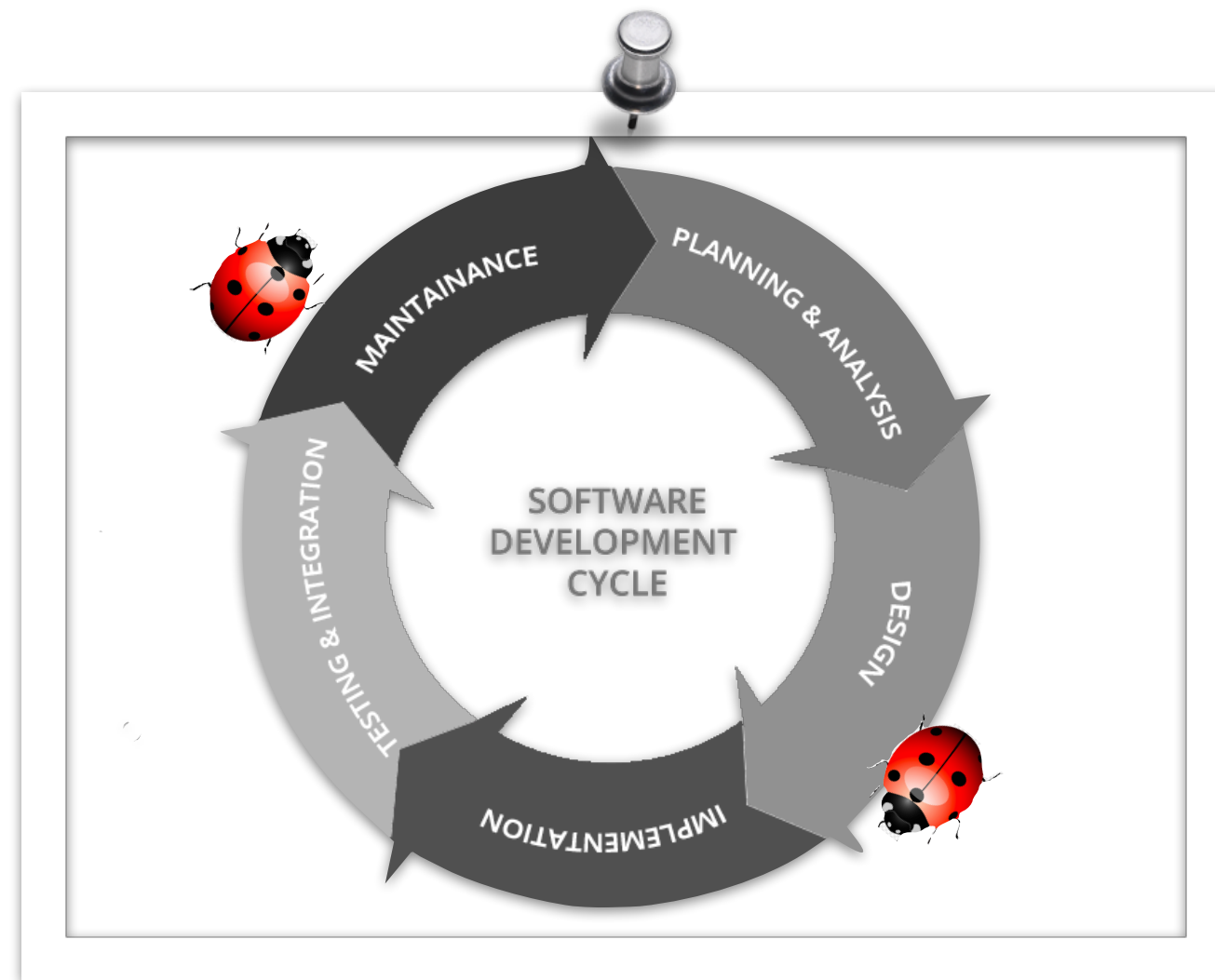
RQ2: Do Release-aware models for priority change outperform Global model?



RQ3: How fine-grained data should be used?

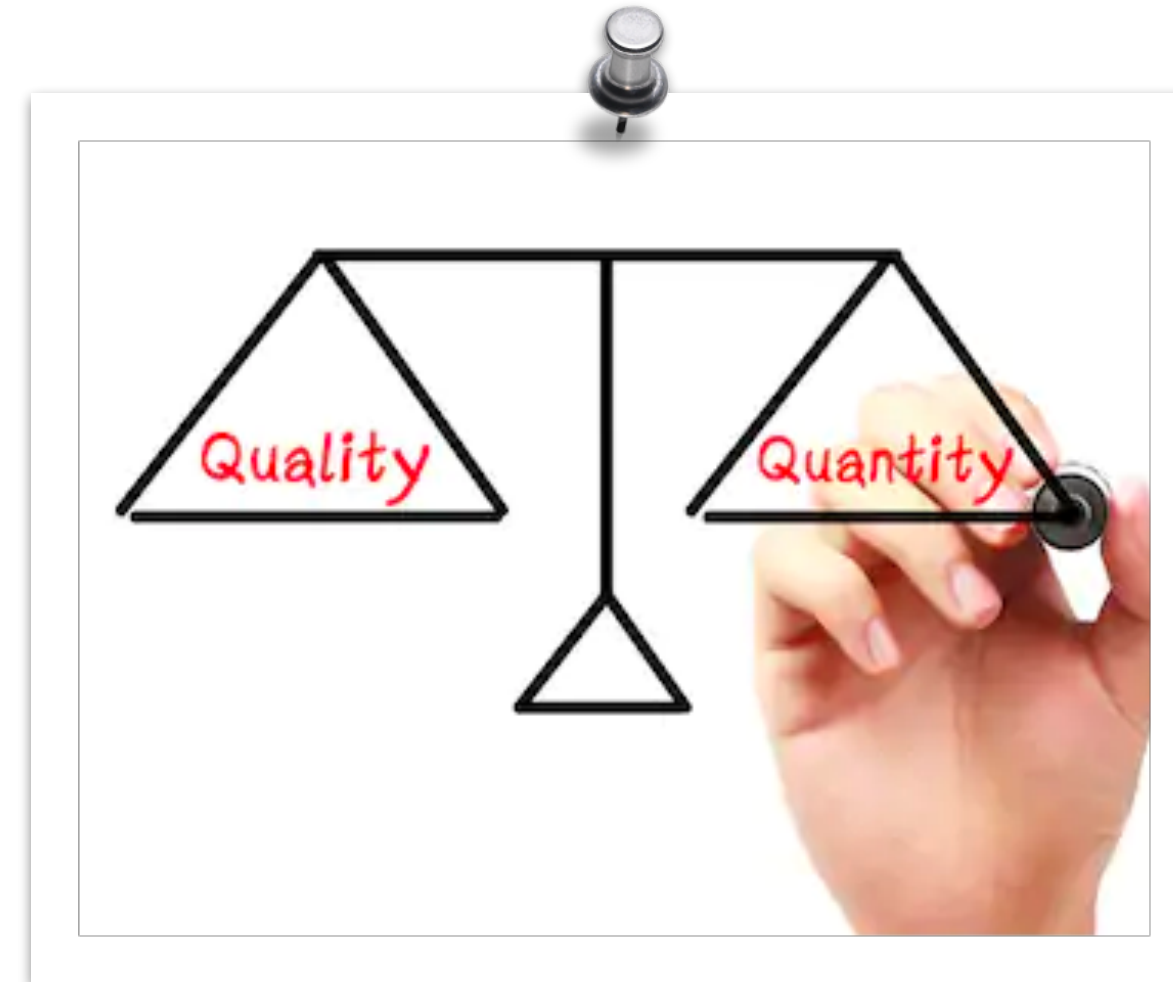


Yes, fixing activities are effected by release cycle (especially, bug reporting, bug-fixing, priority change).

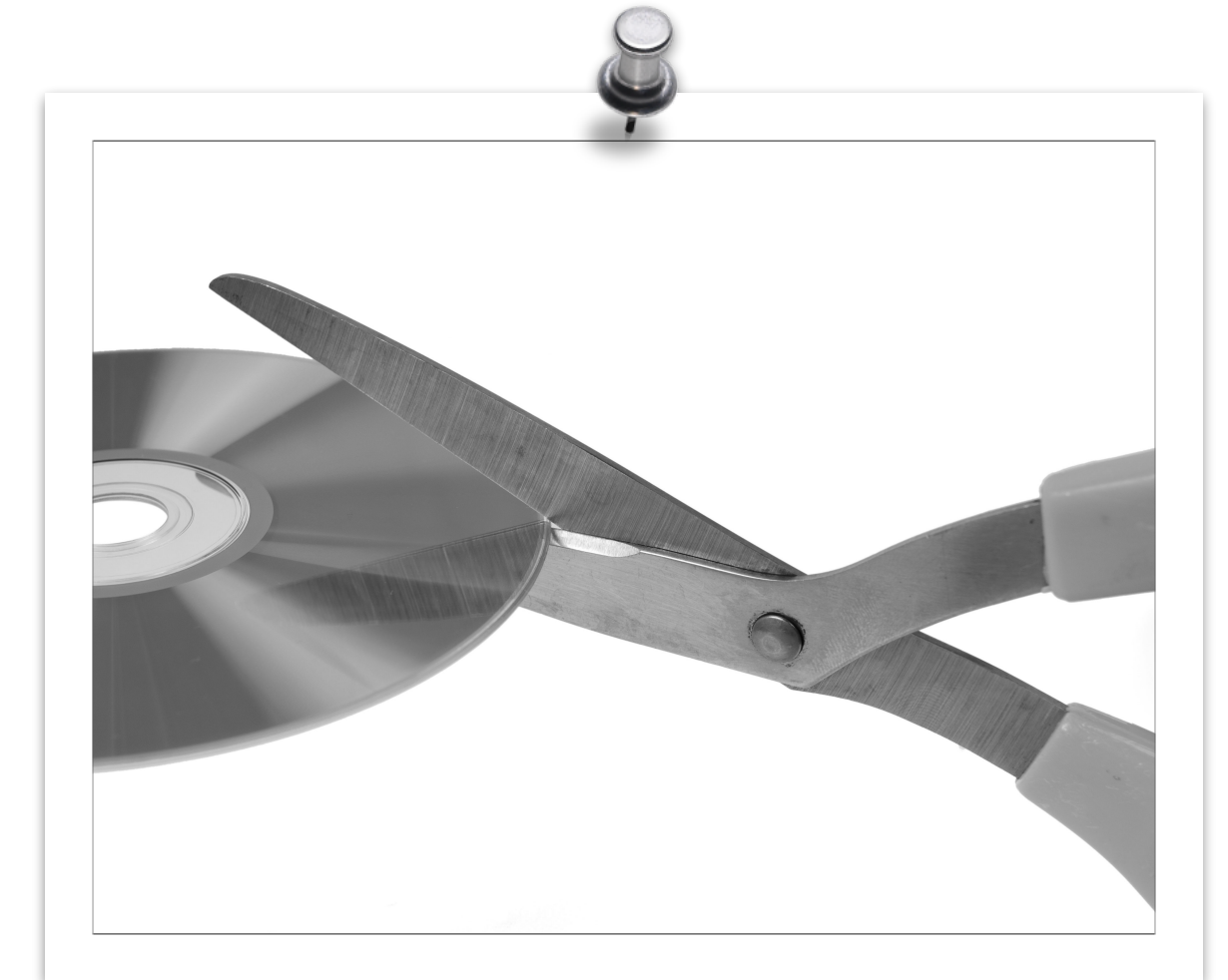


RQ1: Do bug-fixing activities vary across within release cycle?

RQ2: Do Release-aware models for priority change outperform Global model?



RQ3: How fine-grained data should be used?



Yes, fixing activities are effected by release cycle (especially, bug reporting, bug-fixing, priority change).



Predict priorities of bug reports whether high priority or not with Random Forest

P1

P2

P3

P4

P5

High

NOT High

bugs.eclipse.org

 eclipse

Bugzilla – Bug 33111

Bug 33111 - [Navigator] Copy/Paste projects doesn't enable

Status: RESOLVED FIXED

Alias: None

Product: Platform

Component: UI ([show other bugs](#))

Version: 2.1 

Hardware: PC Windows 2000

Importance: P1 major ([vote](#))

bugs.eclipse.org

 eclipse

Bugzilla – Bug 33333

Bug 33333 - Advanced Option dialog truncates drop-down list choic

Status: VERIFIED FIXED

Alias: None

Product: JDT

Component: Debug ([show other bugs](#))

Version: 2.1 

Hardware: Macintosh Mac OS X - Carbon (unsup.)

Importance: P3 normal ([vote](#))



Predict priorities of bug reports whether high



Metrics for the statistical tests



38

34

Unit	Dimension	Metrics
Daily	Report	# of reports
		# of High/Mid/Low priority reports
		# of High/Mid/Low severity reports
		# of enhancement
	Close	# of reports
		# of High/Mid/Low priority reports
		# of High/Mid/Low severity reports
		# of enhancement reports
	Comment	# of comments
		# of commentator
	Changes	# of changes priority low to high
		# of changes priority high to low
		# of changes severity low to high
		# of changes severity high to low
	Commits	# of commits
		# of files
		# of adds
		# of deletes

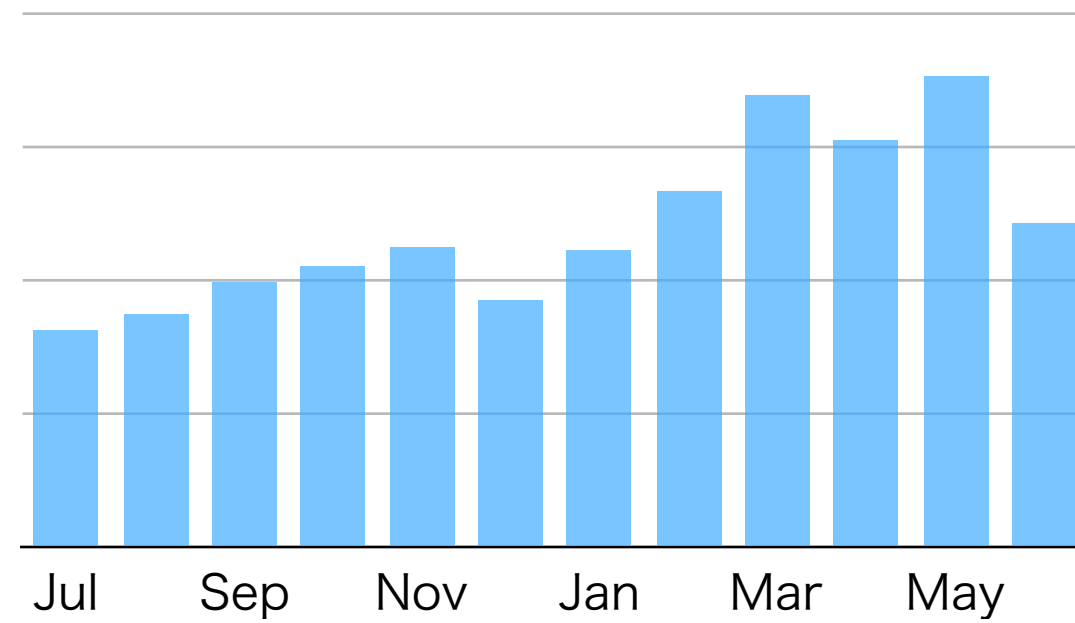
Unit	Dimension	Metrics
Bug	Report	#of description words
		bayes socre
		# of priority change
		# of assigned
		# of cc
		# of comments
		# of commentator
		time for assigning bug
	Commit	time for fixing bug
		# of commits
		# of files
		# of adds
		# of deletes
		median of adds per file
		median of delets per file

Hardware: Macintosh Mac OS X - Carbon (unsup.)

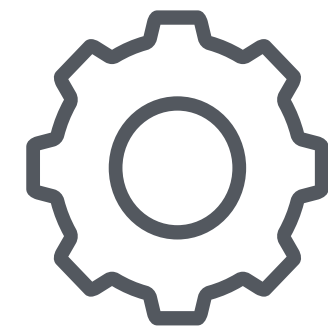
Importance: P3 normal (vote)



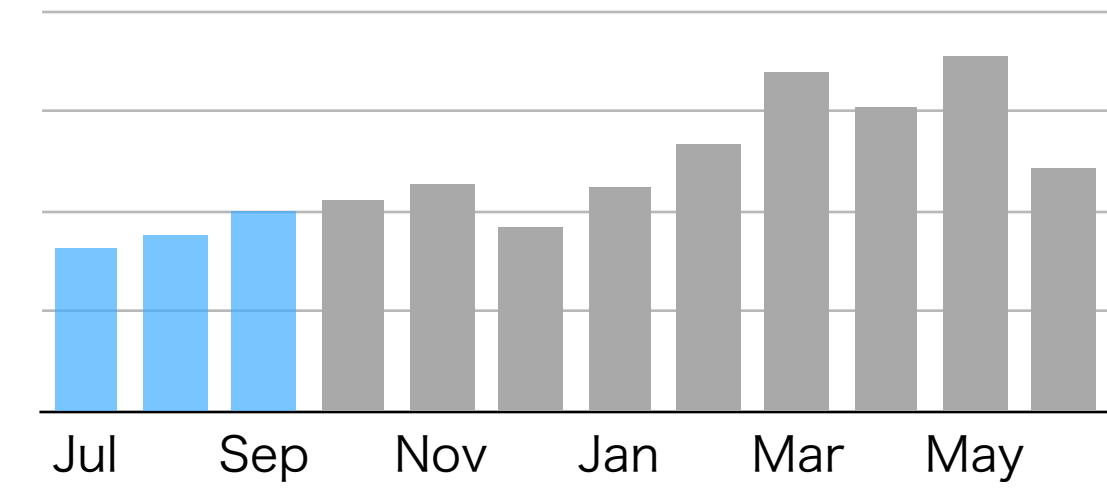
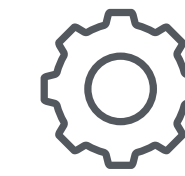
Methodology of evaluations



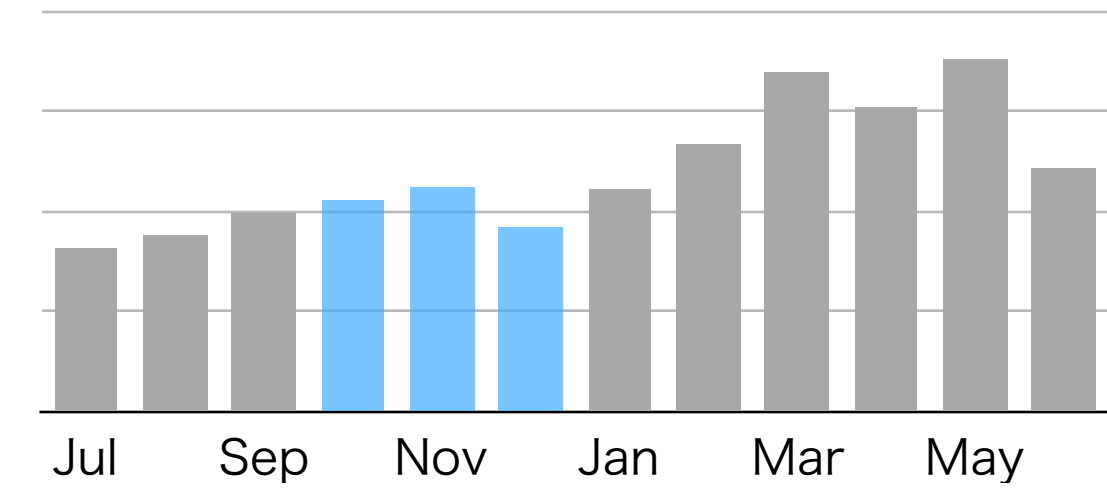
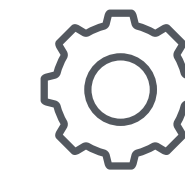
Global Model



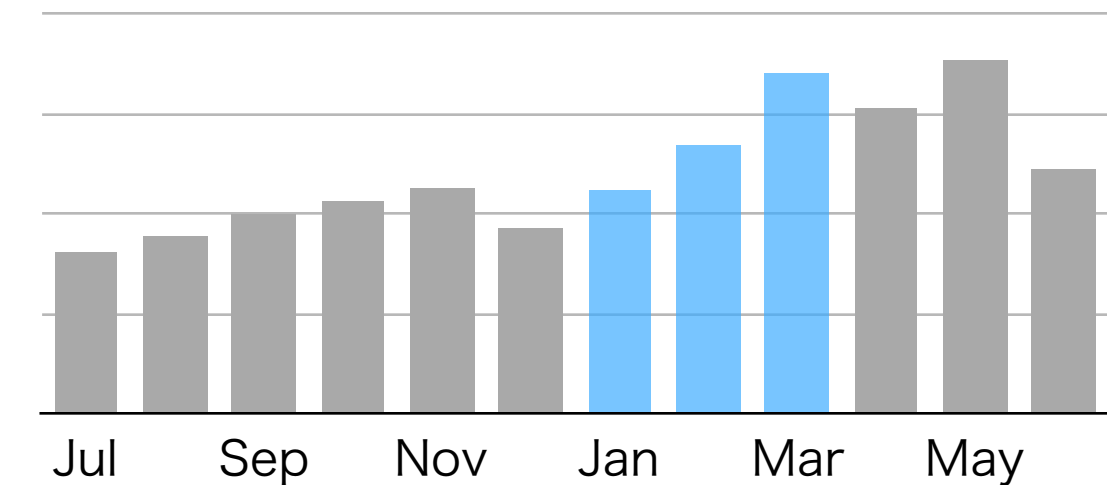
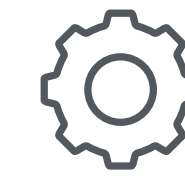
VS.



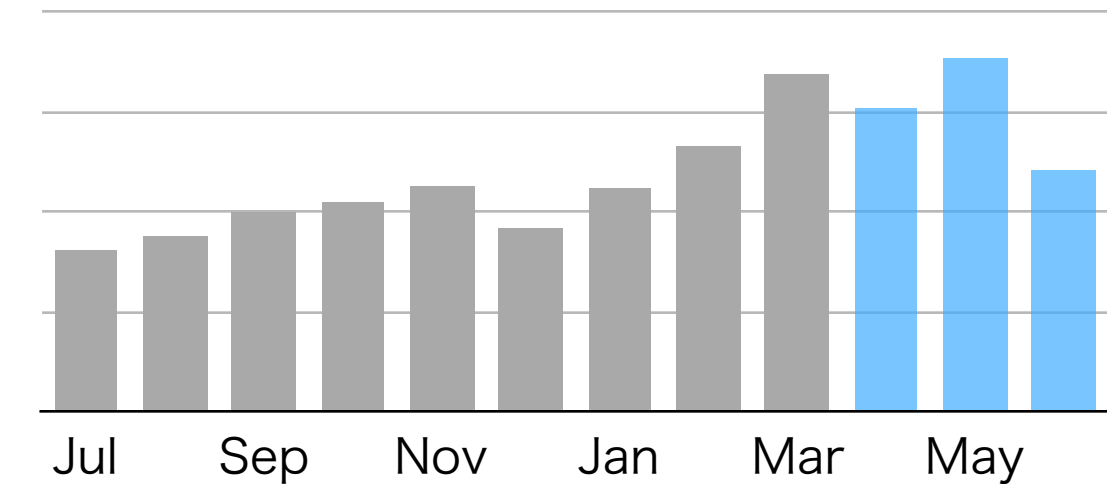
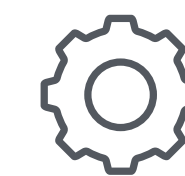
Q1



Q2



Q3



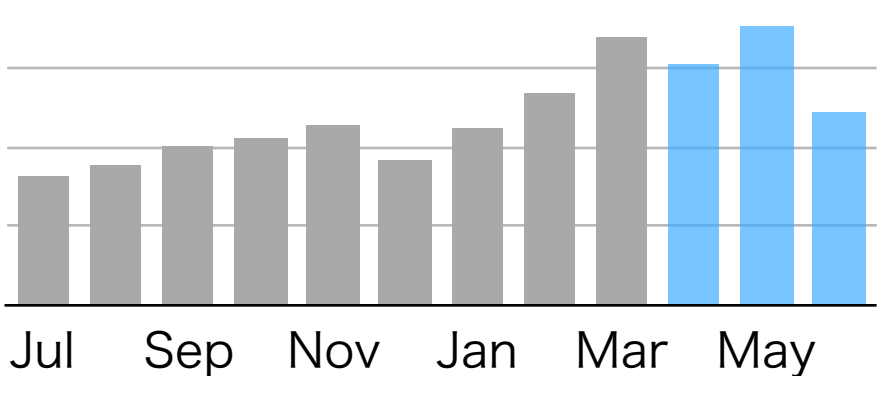
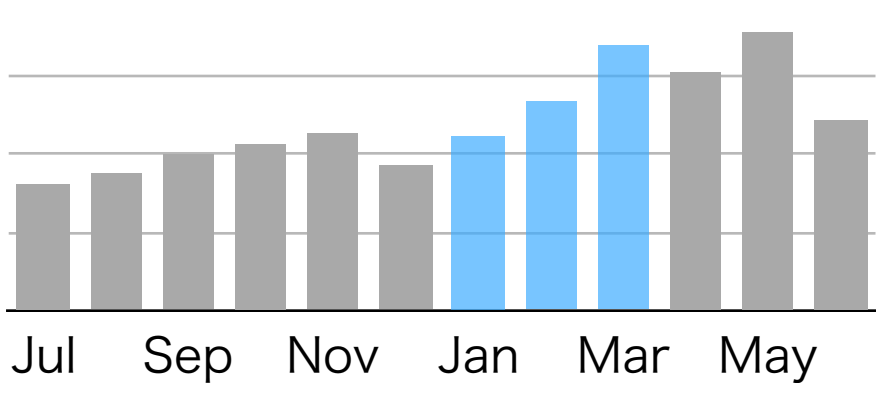
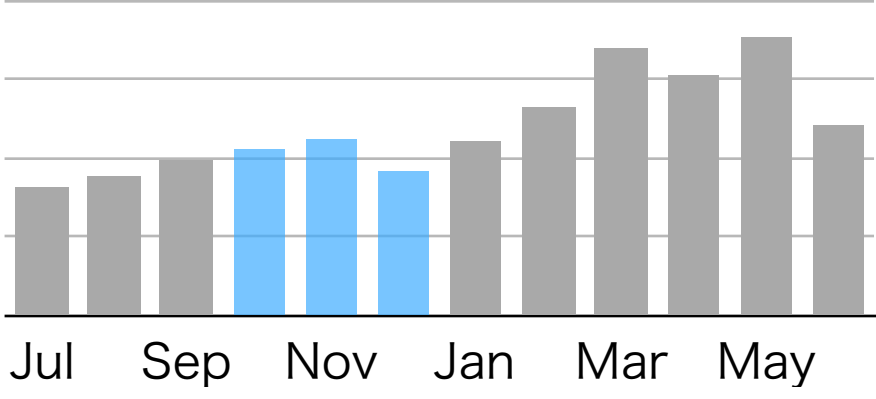
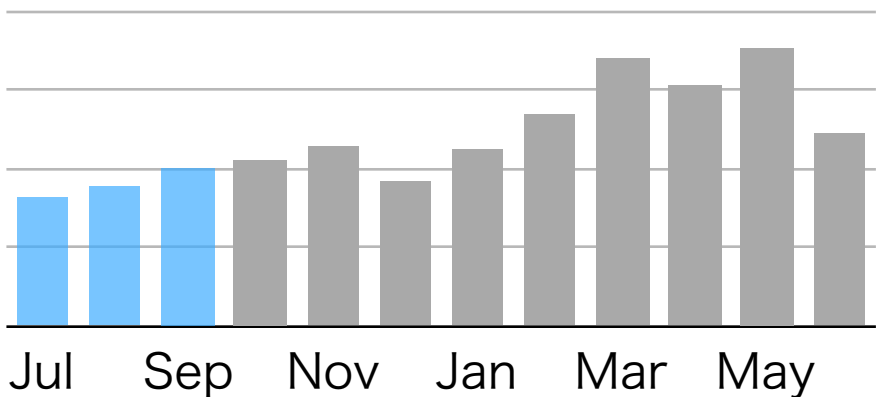
Q4

Quarter Models



Methodology of model building

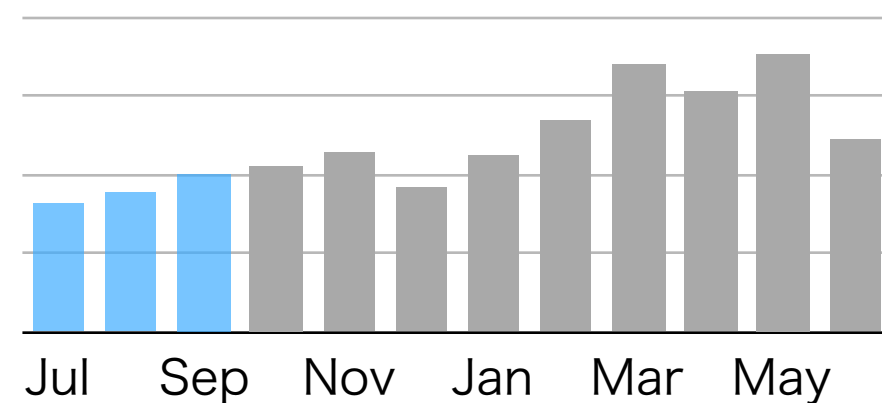
Dataset





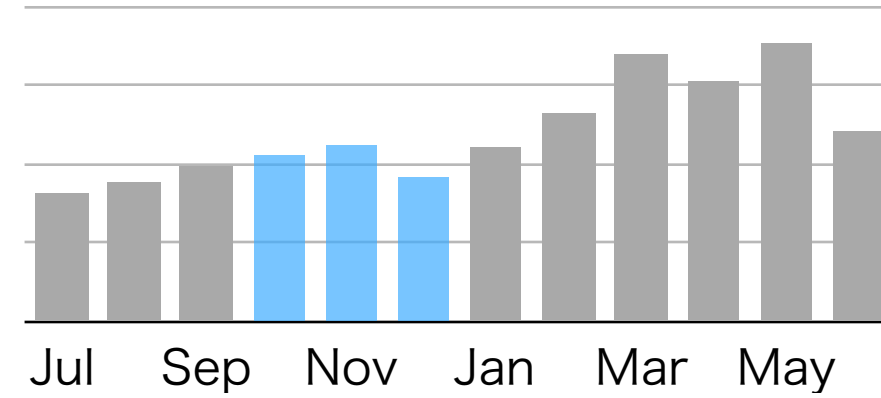
Methodology of model building

Dataset



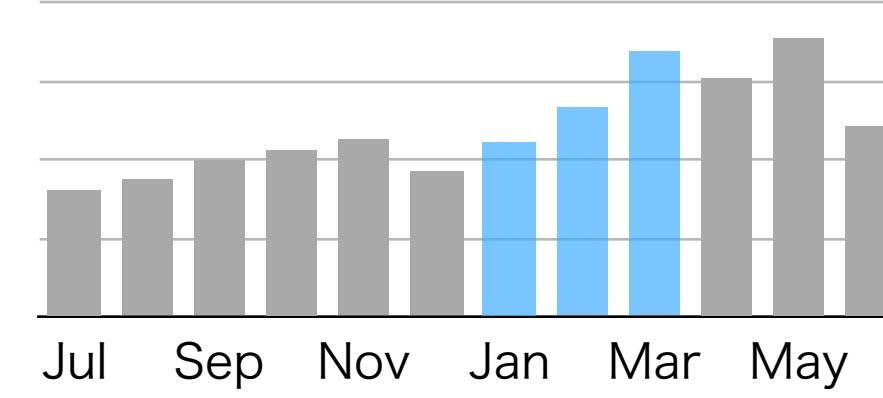
Training

Testing



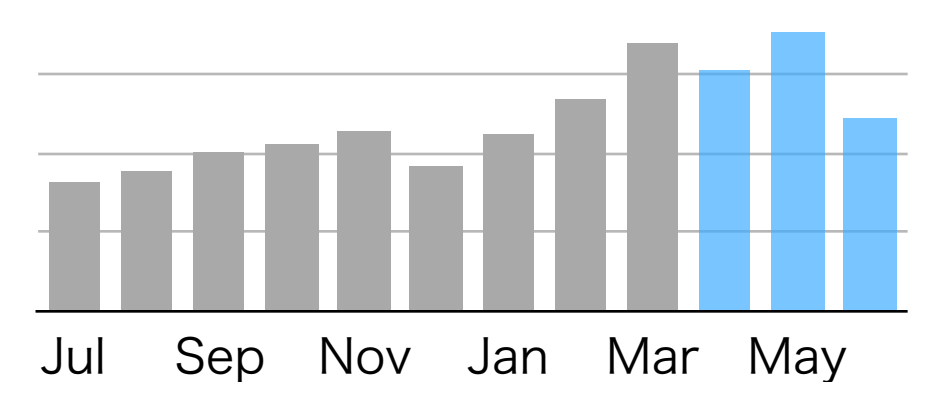
Training

Testing



Training

Testing



Training

Testing



Methodology of model building

Dataset

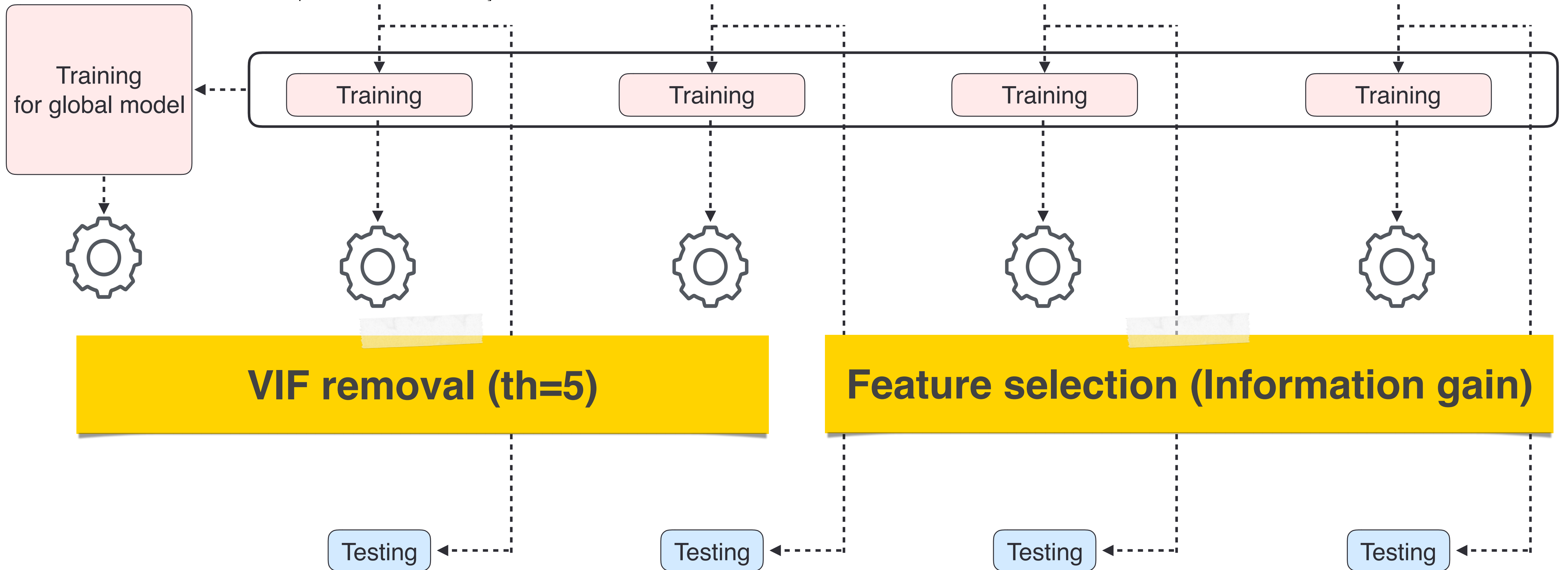
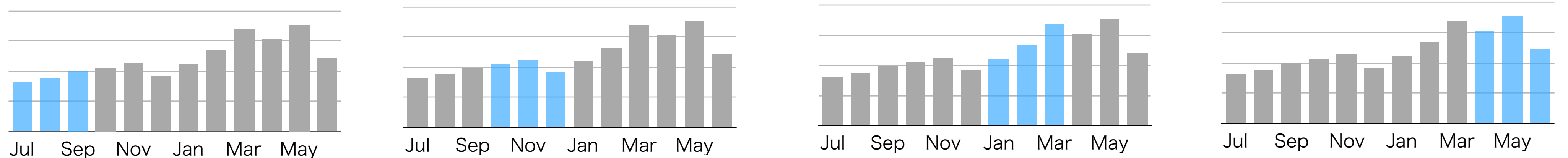


Training
for global model



Methodology of model building

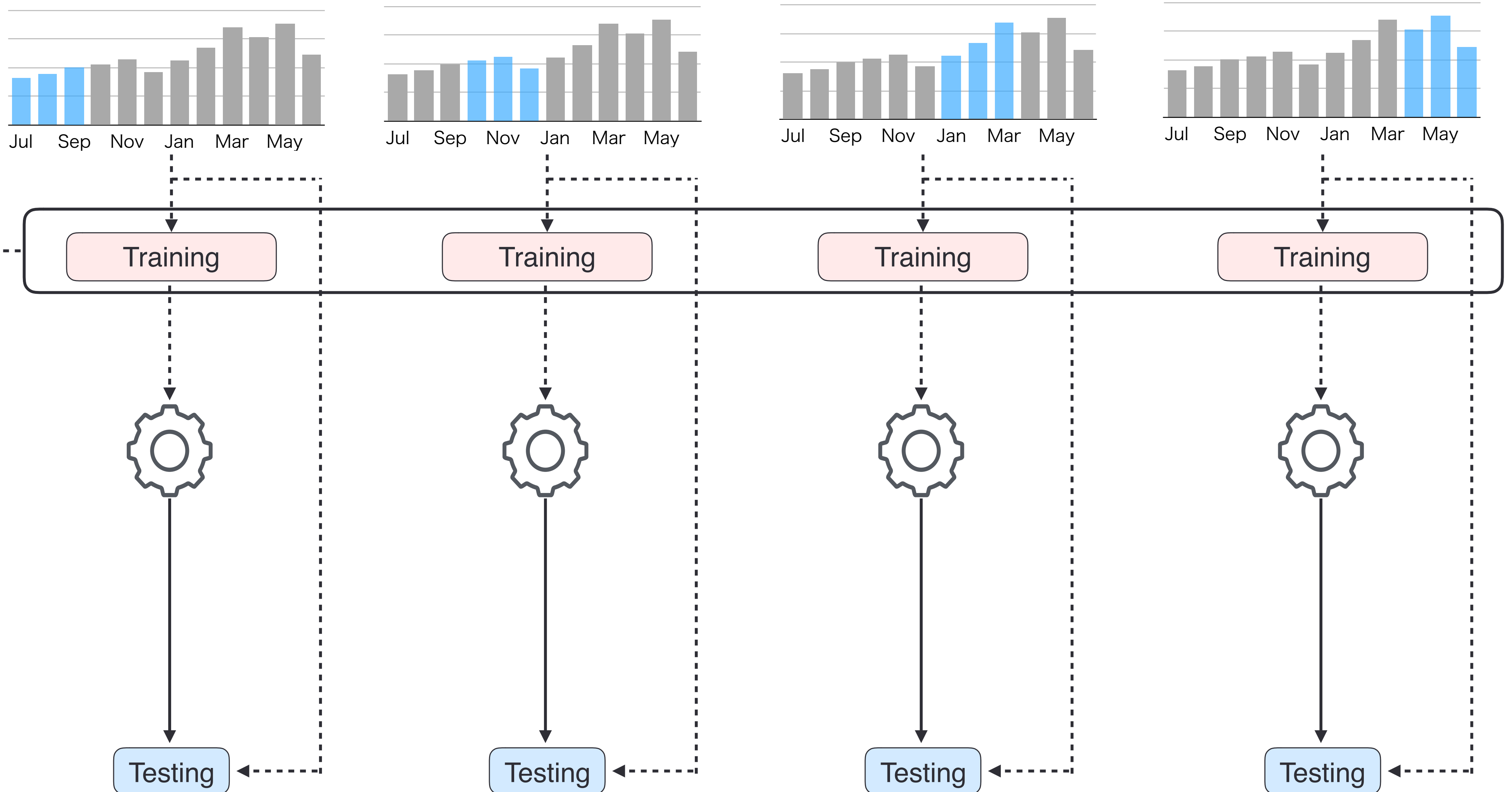
Dataset





Methodology of model building

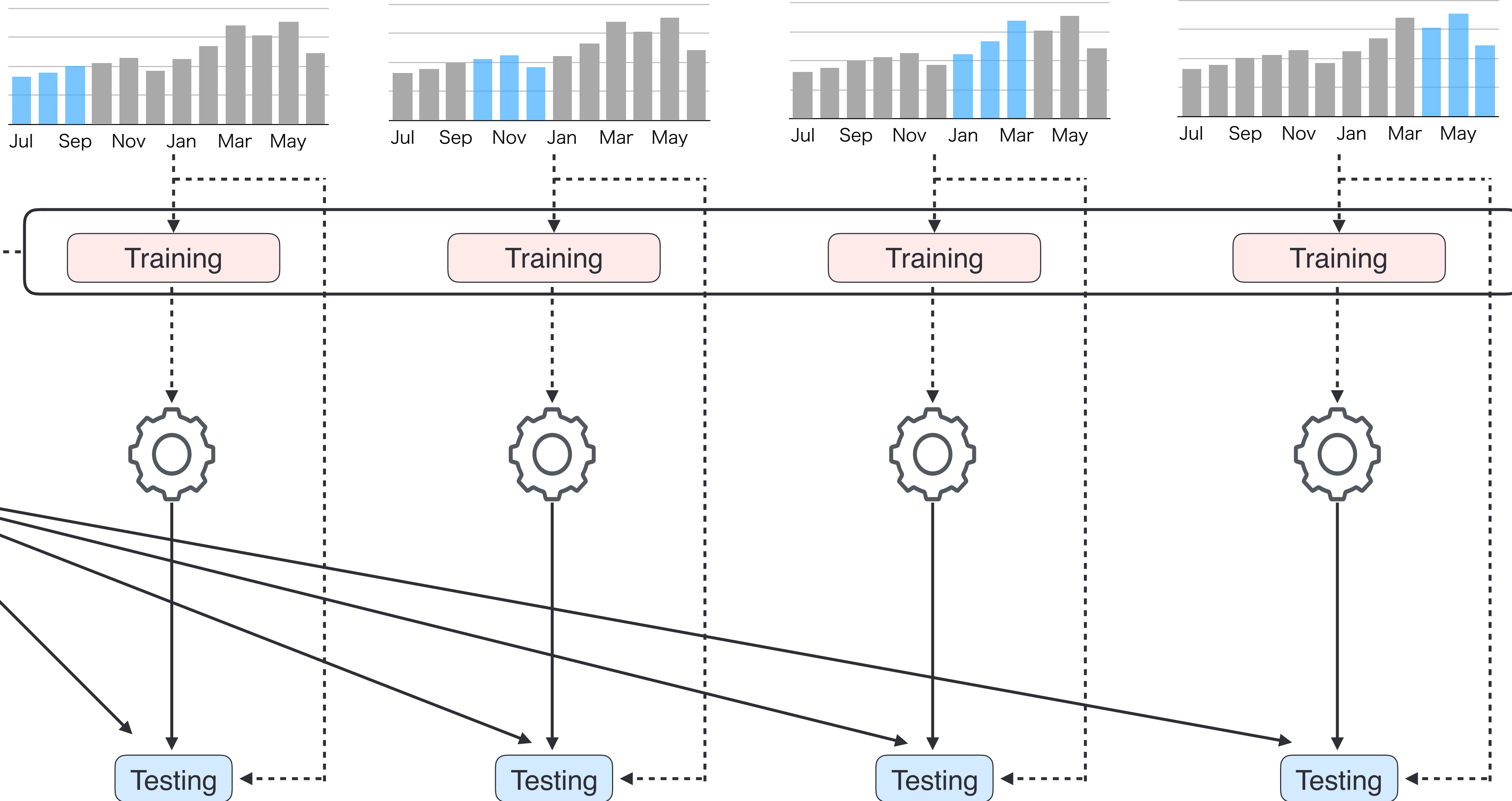
Dataset





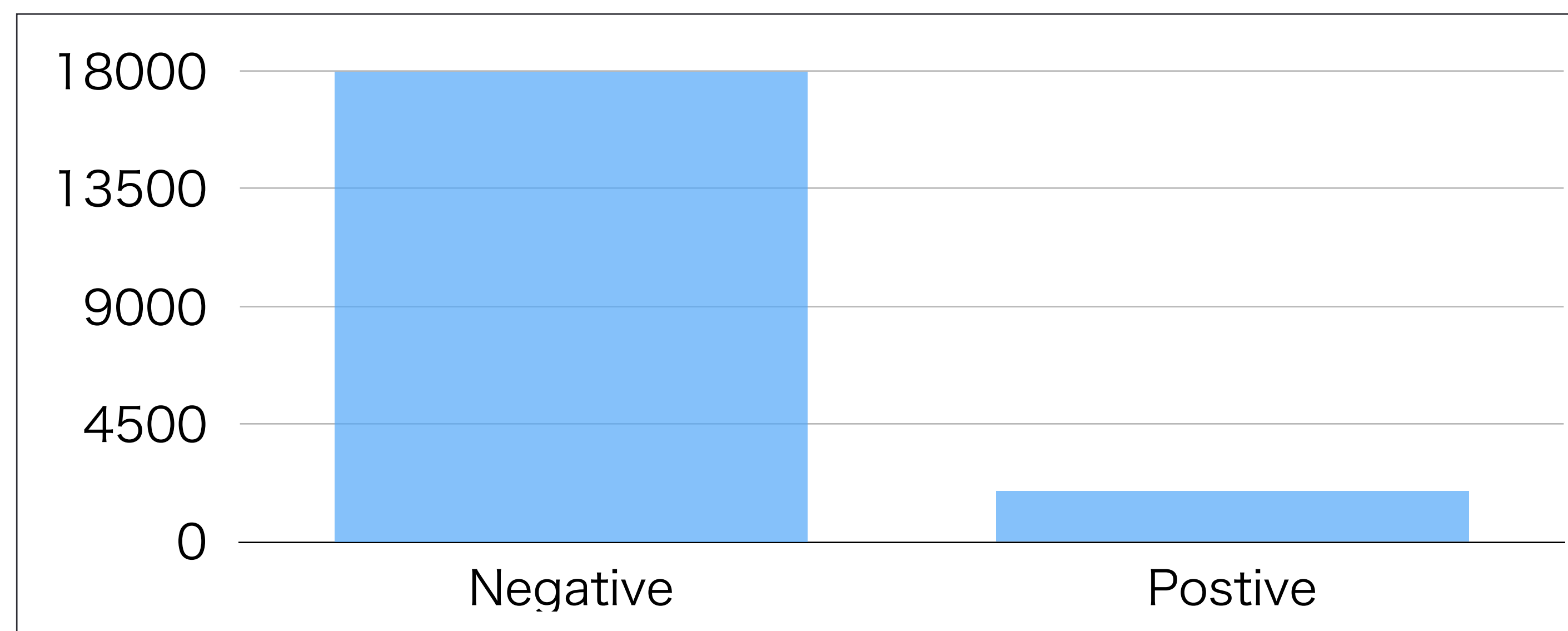
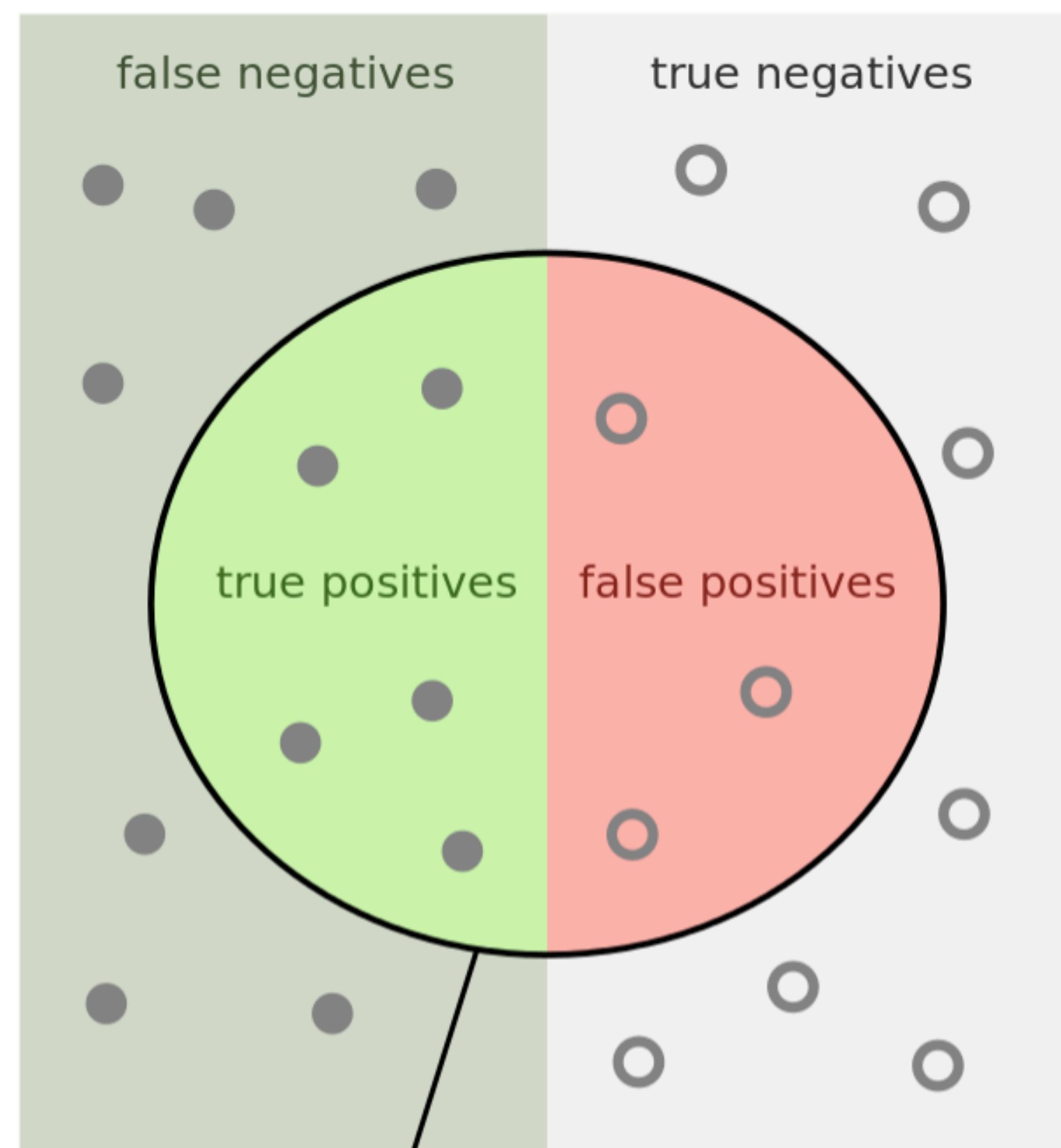
Methodology of model building

Dataset





Evaluation: Recall, G-Mean, AUC

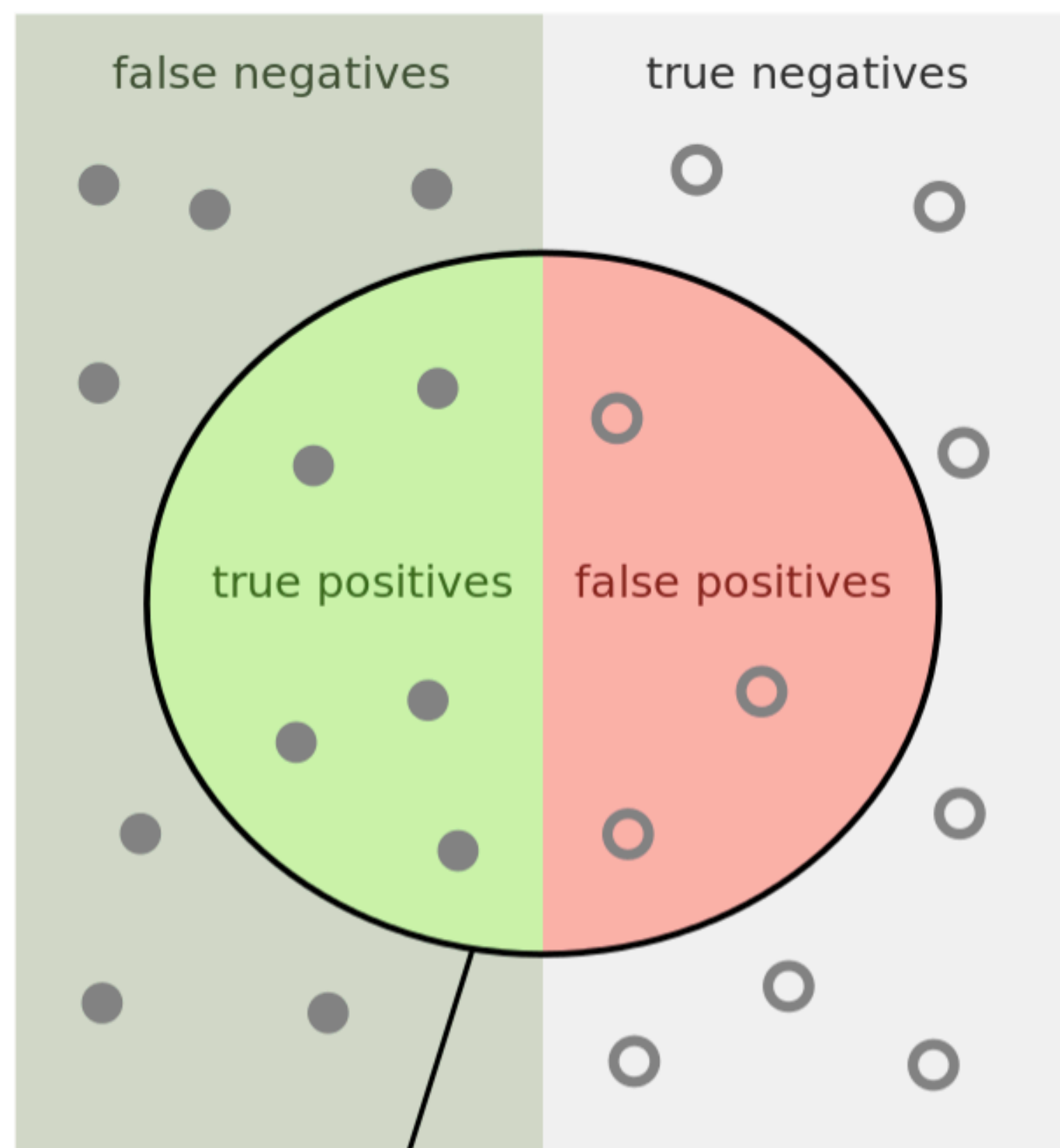


[1] Wang, S., Minku, L.L., Yao, X.: A systematic study of online class imbalance learning with concept drift. IEEE Transactions on Neural Networks and Learning Systems pp.1–20 (2018)





Evaluation: Recall, G-Mean, AUC



How many relevant
items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

How many negative
selected elements
are truly negative?

$$\text{Specificity} = \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}}$$

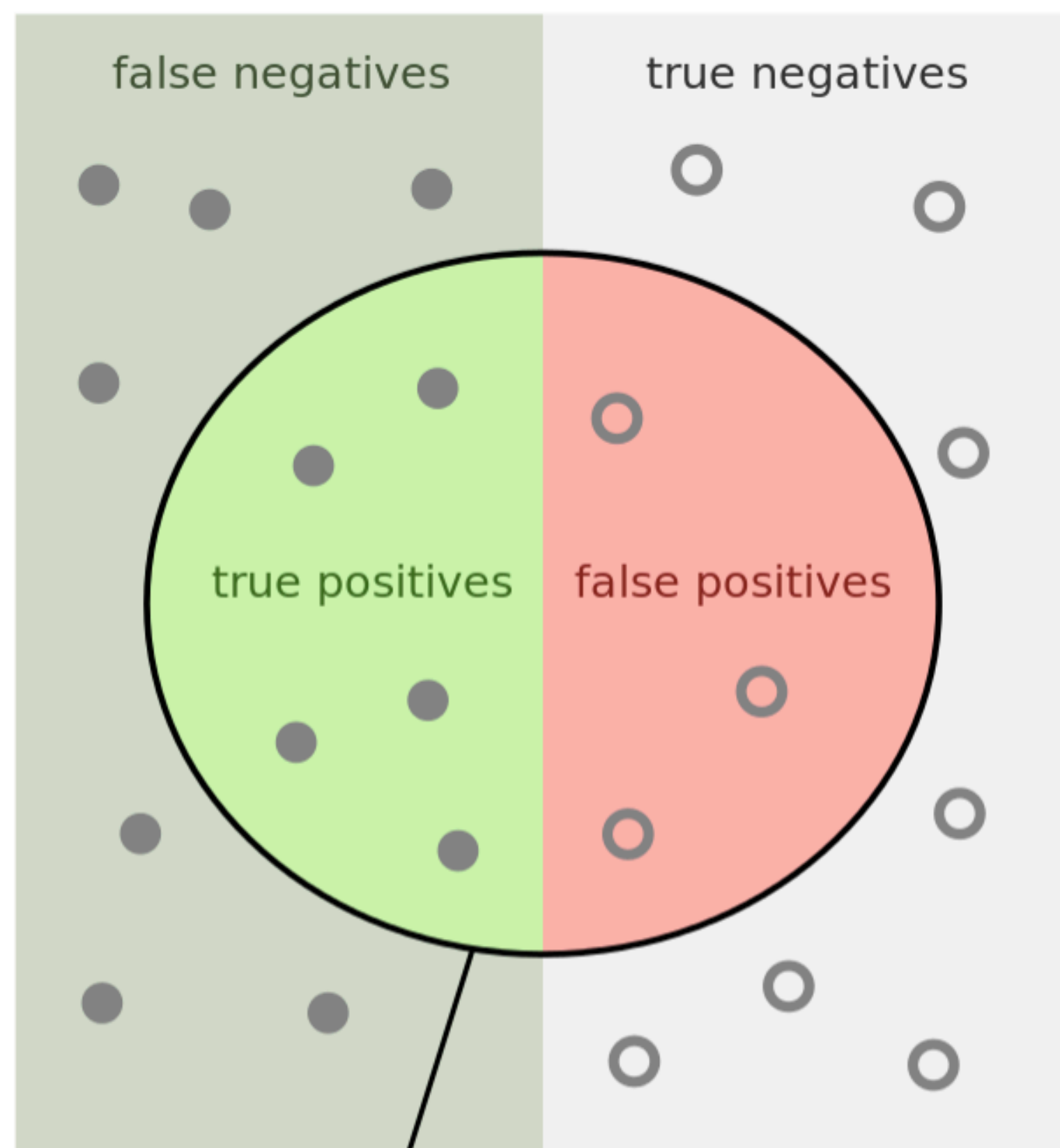
$$G - \text{mean} = \sqrt{\text{Recall} \times \text{Specificity}}$$





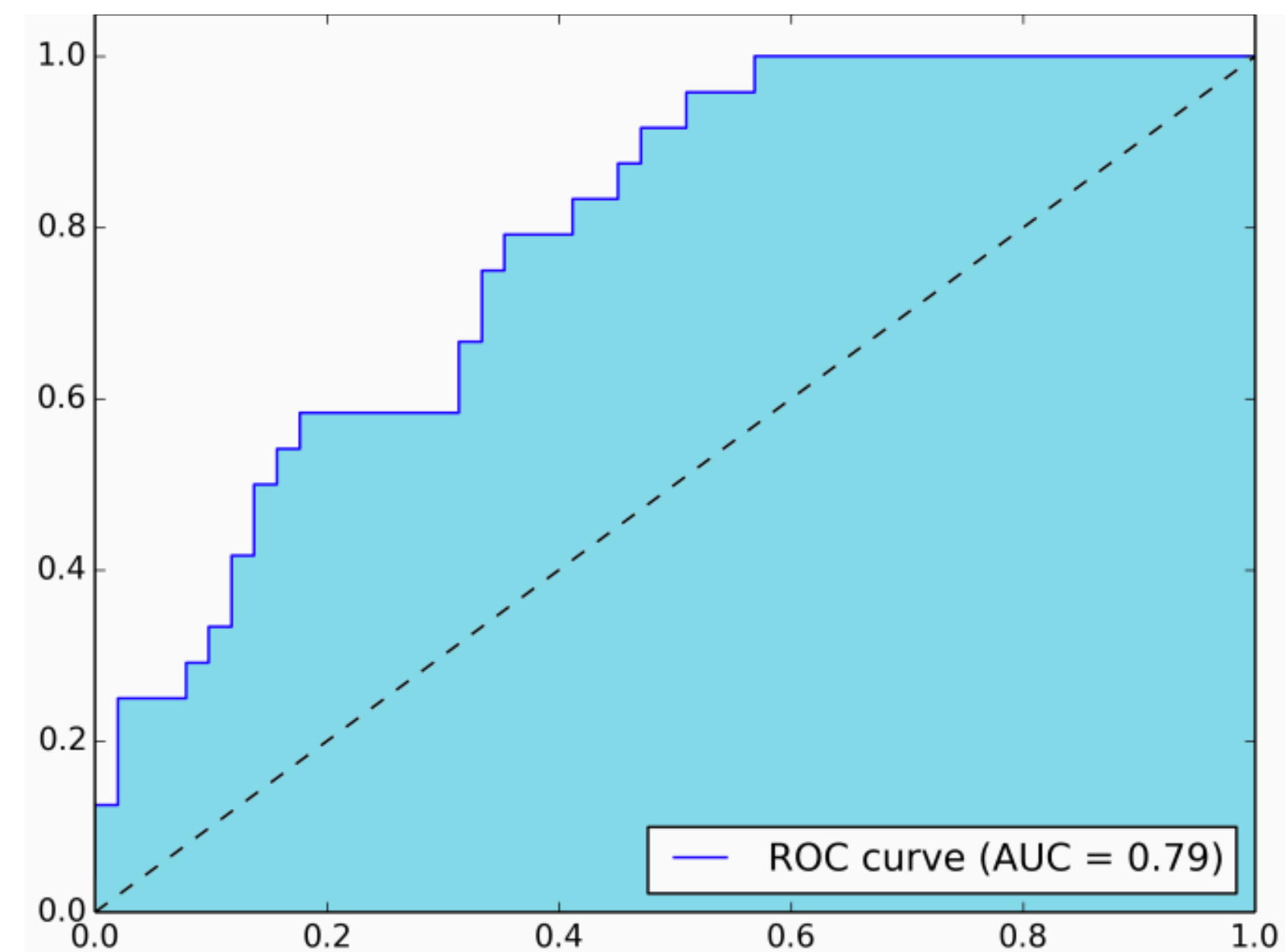
Evaluation: Recall, G-Mean, AUC

ROC-AUC



How many relevant
items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$



$$\text{Specificity} = \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}}$$

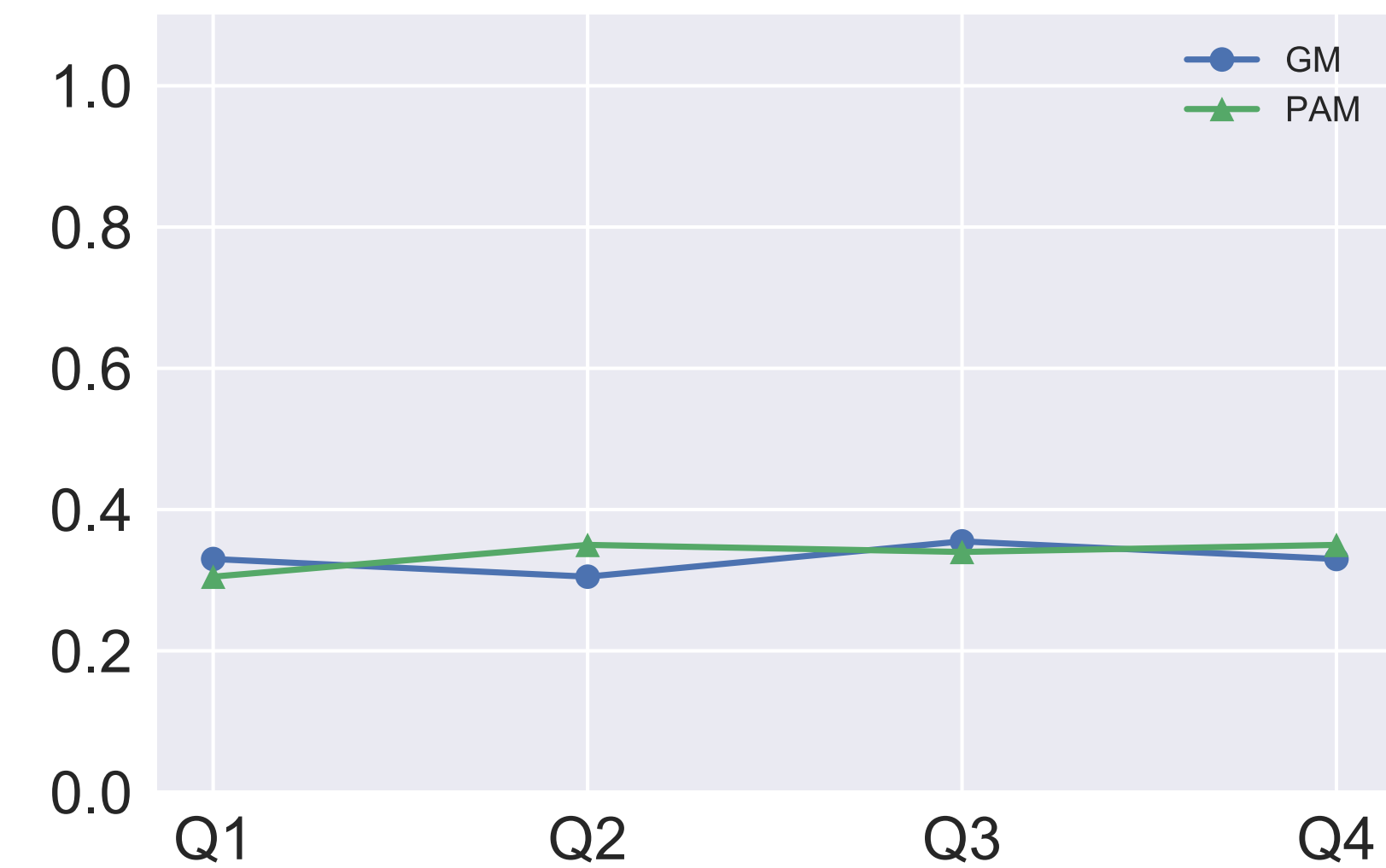
How many negative
selected elements
are truly negative?



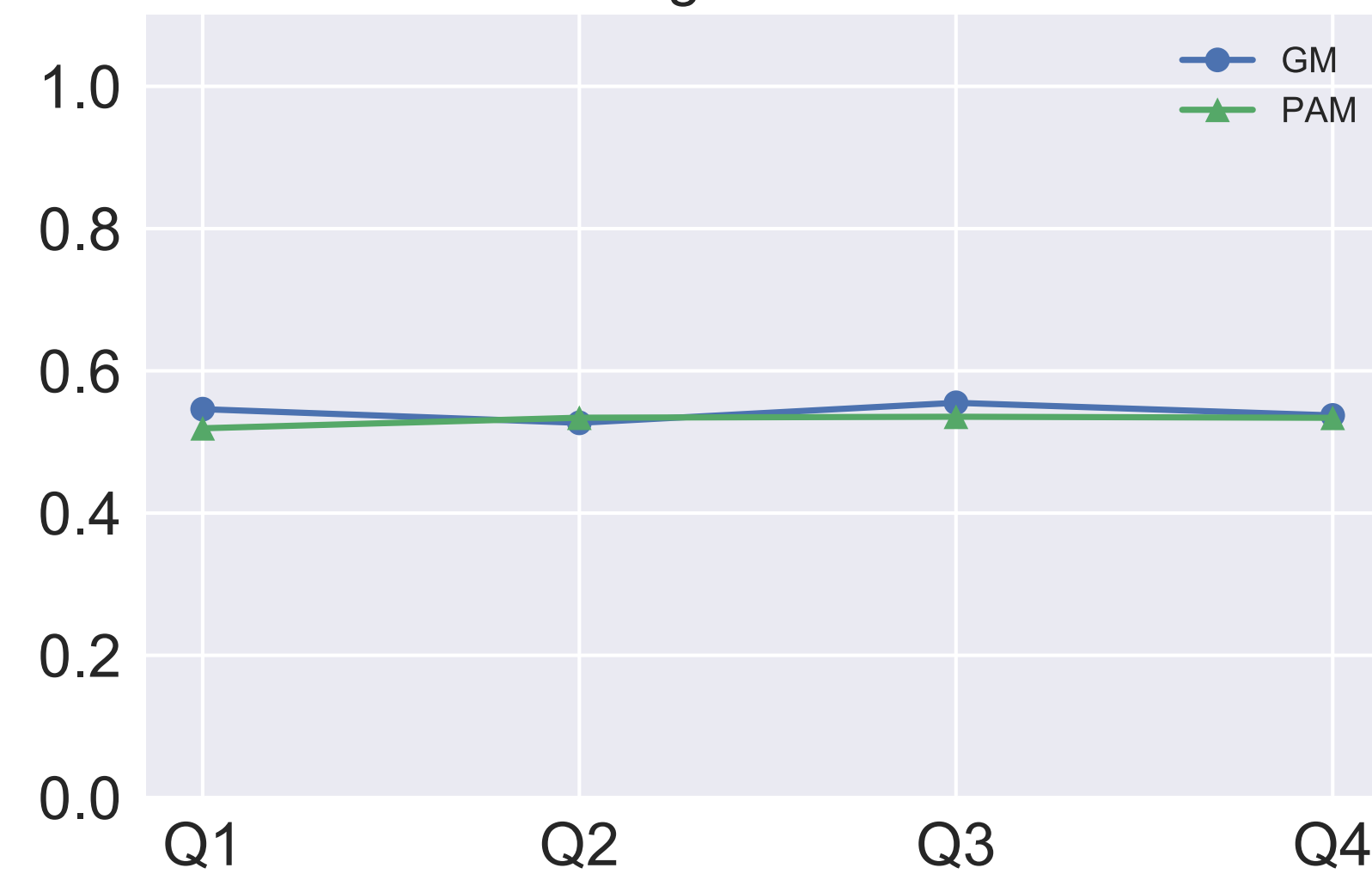


Release aware models can not totally outperform Global models

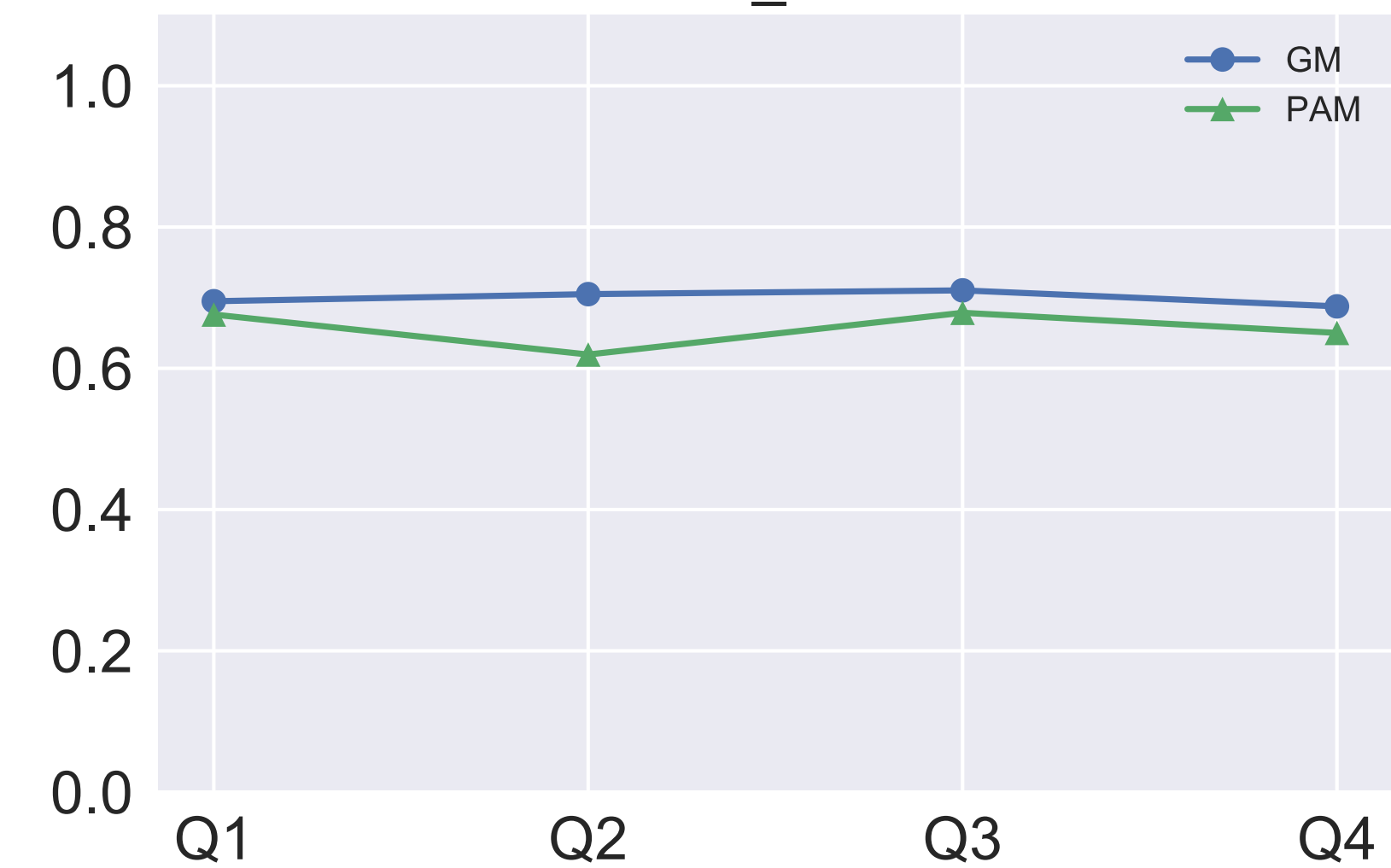
recall



g-score



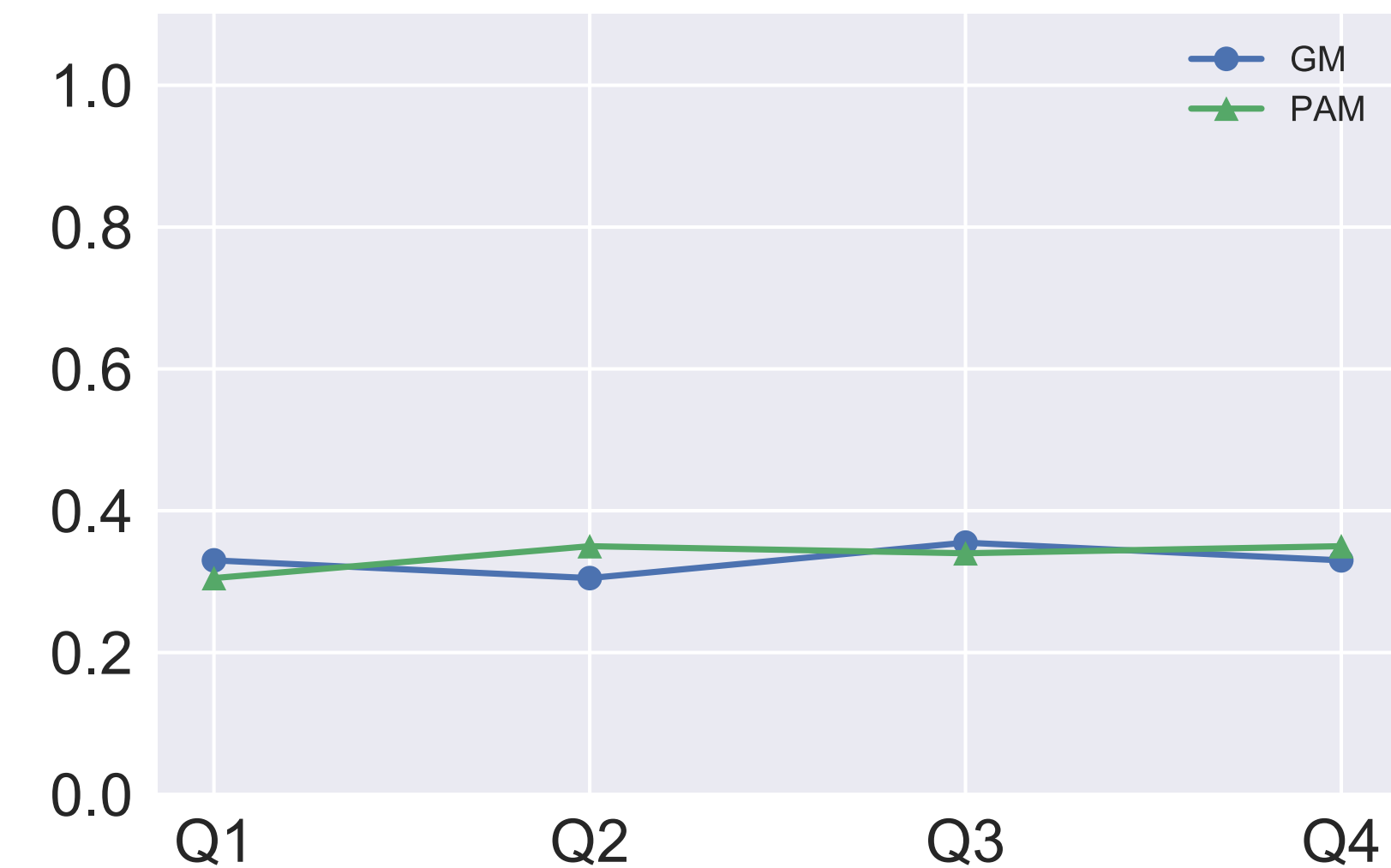
roc_auc



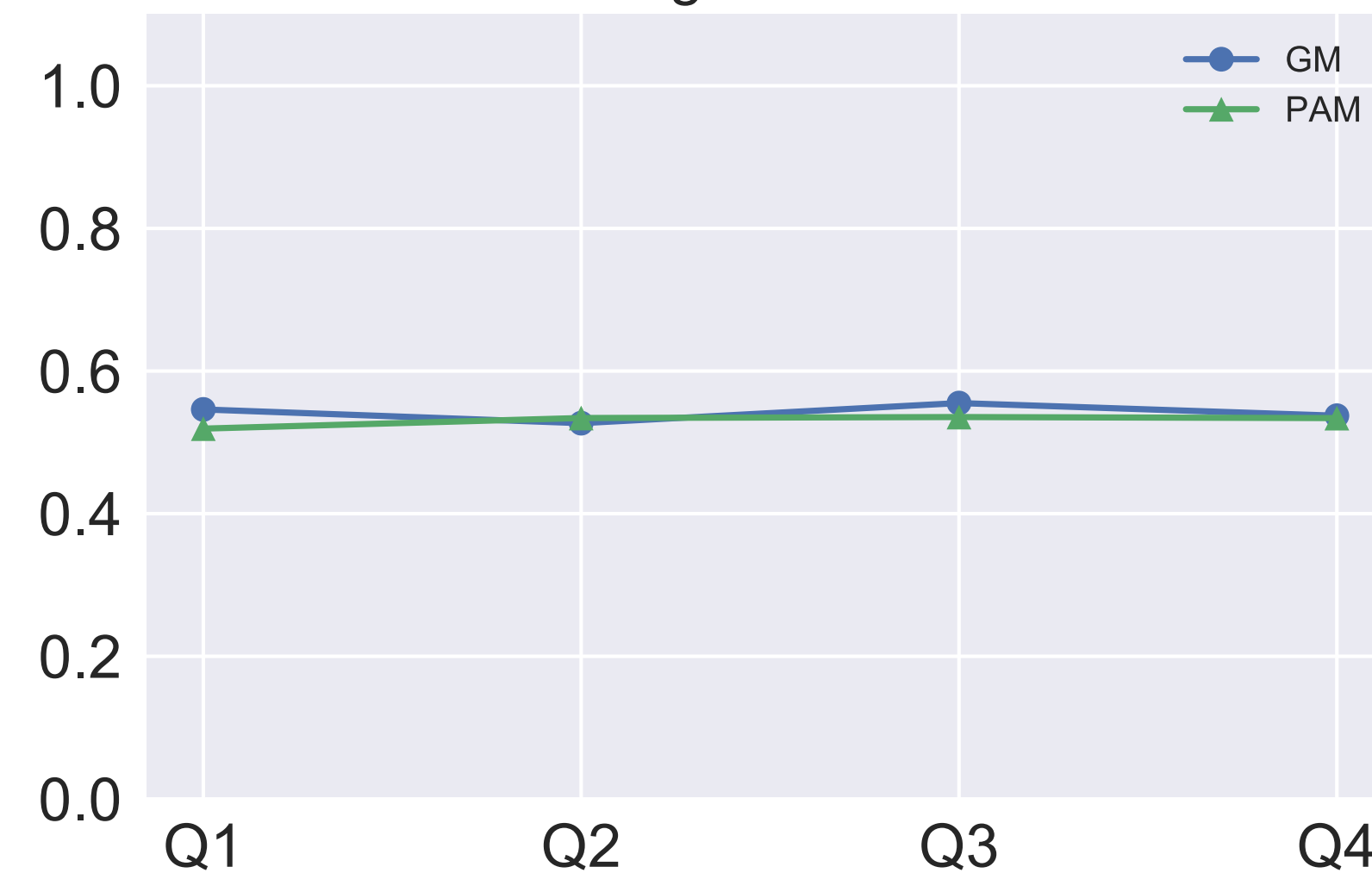


Release aware models can not totally outperform Global models

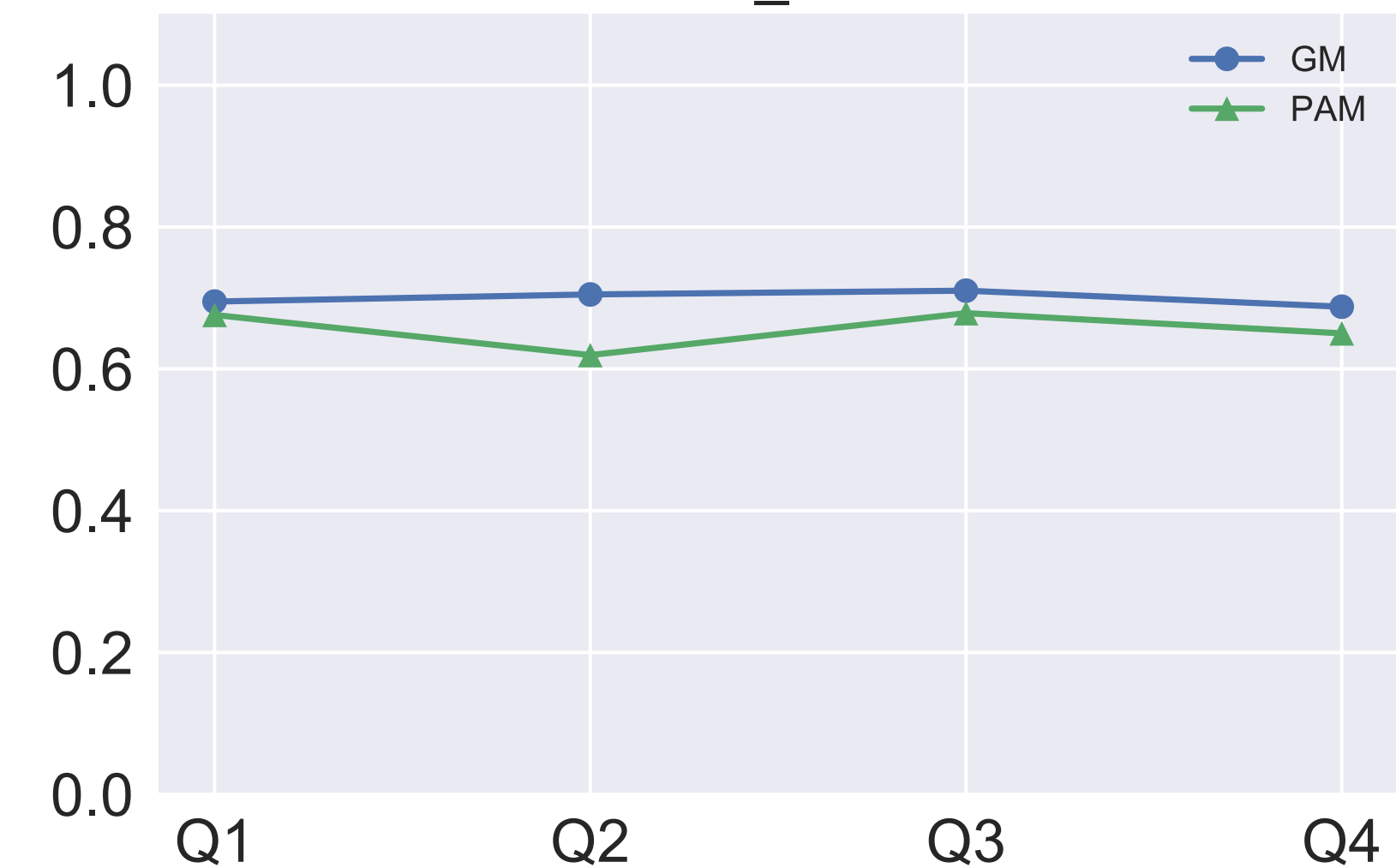
recall



g-score



roc_auc

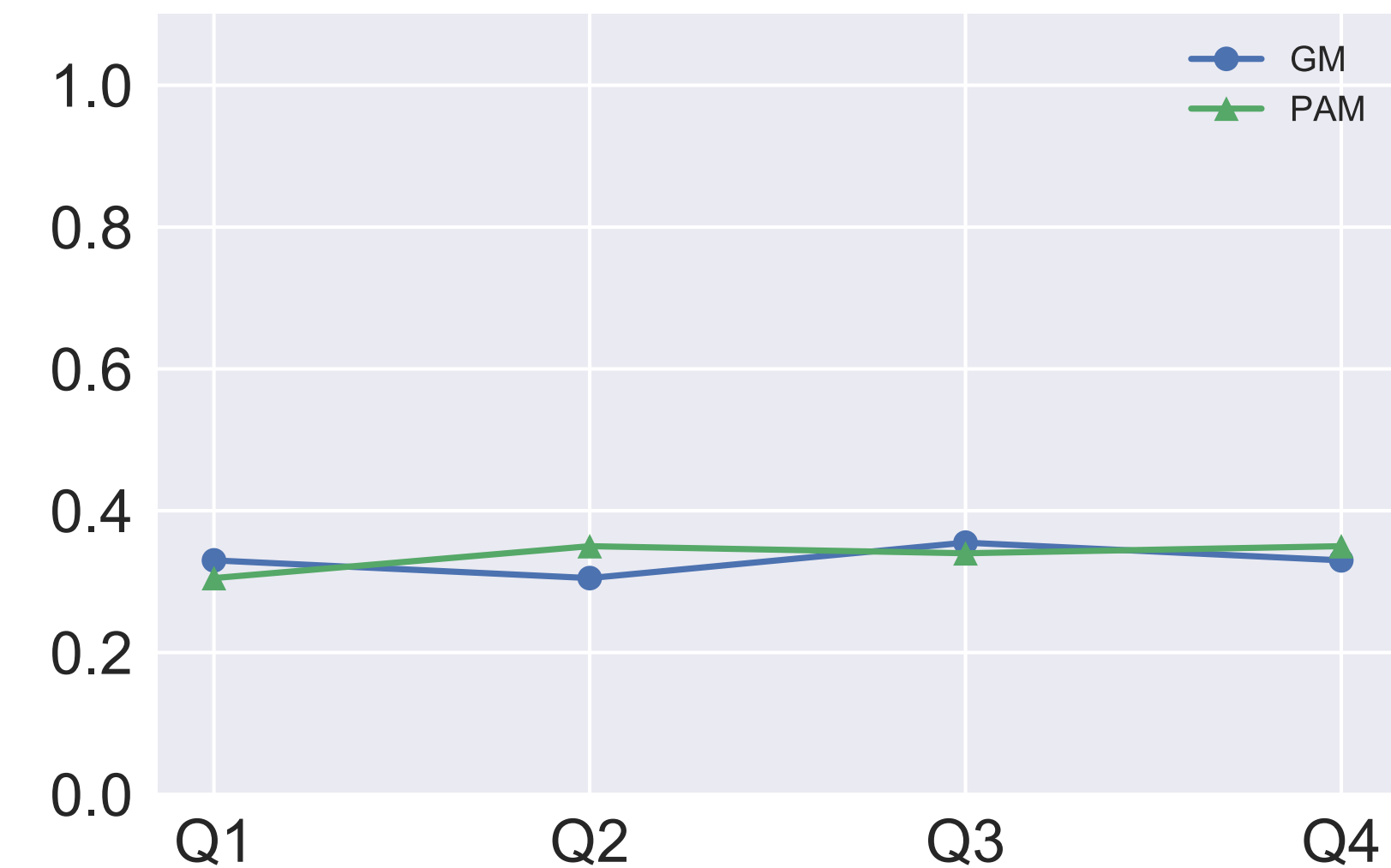


The recall of RAM in Q2 is only significantly higher

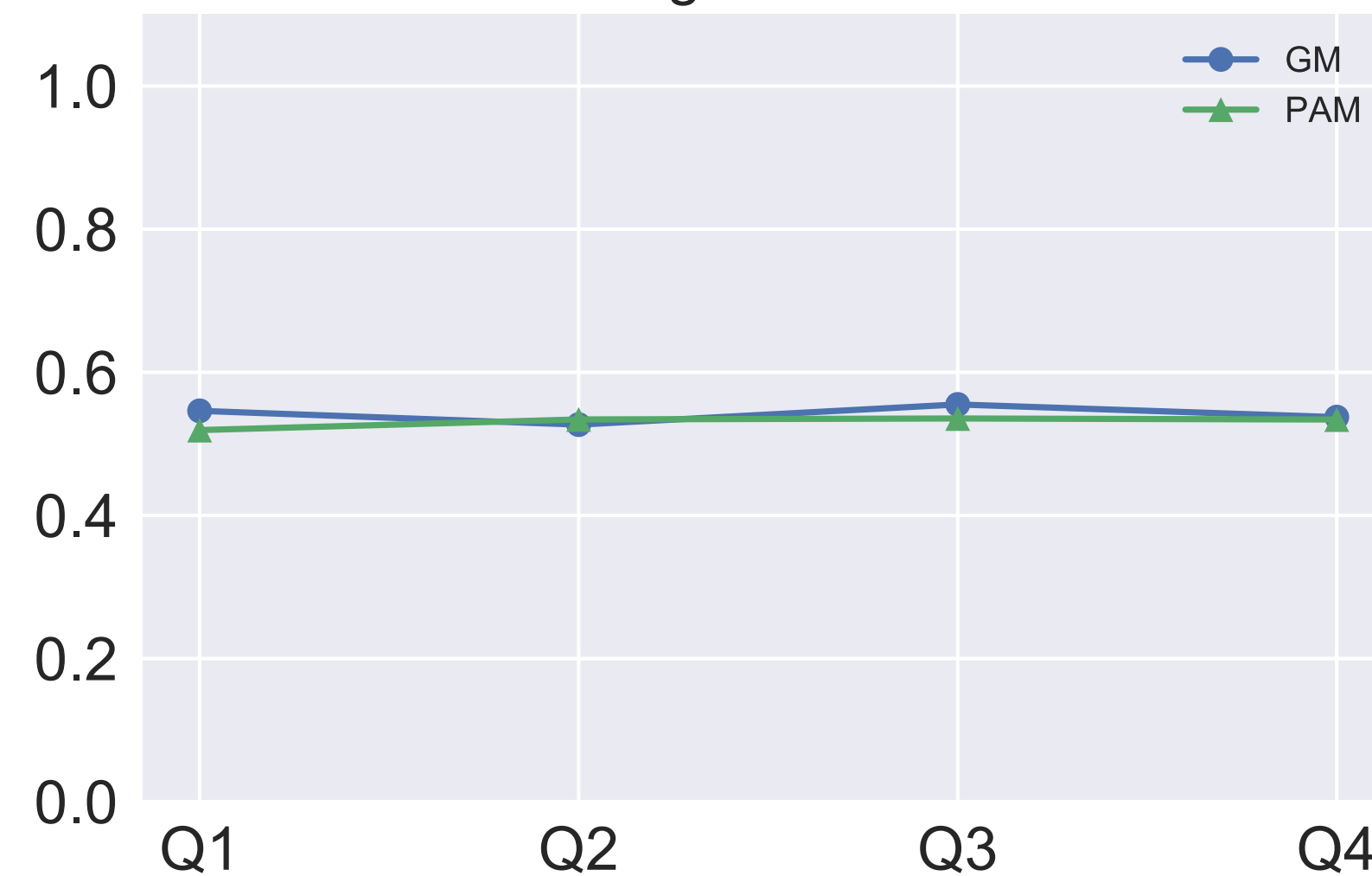


Release aware models can not totally outperform Global models

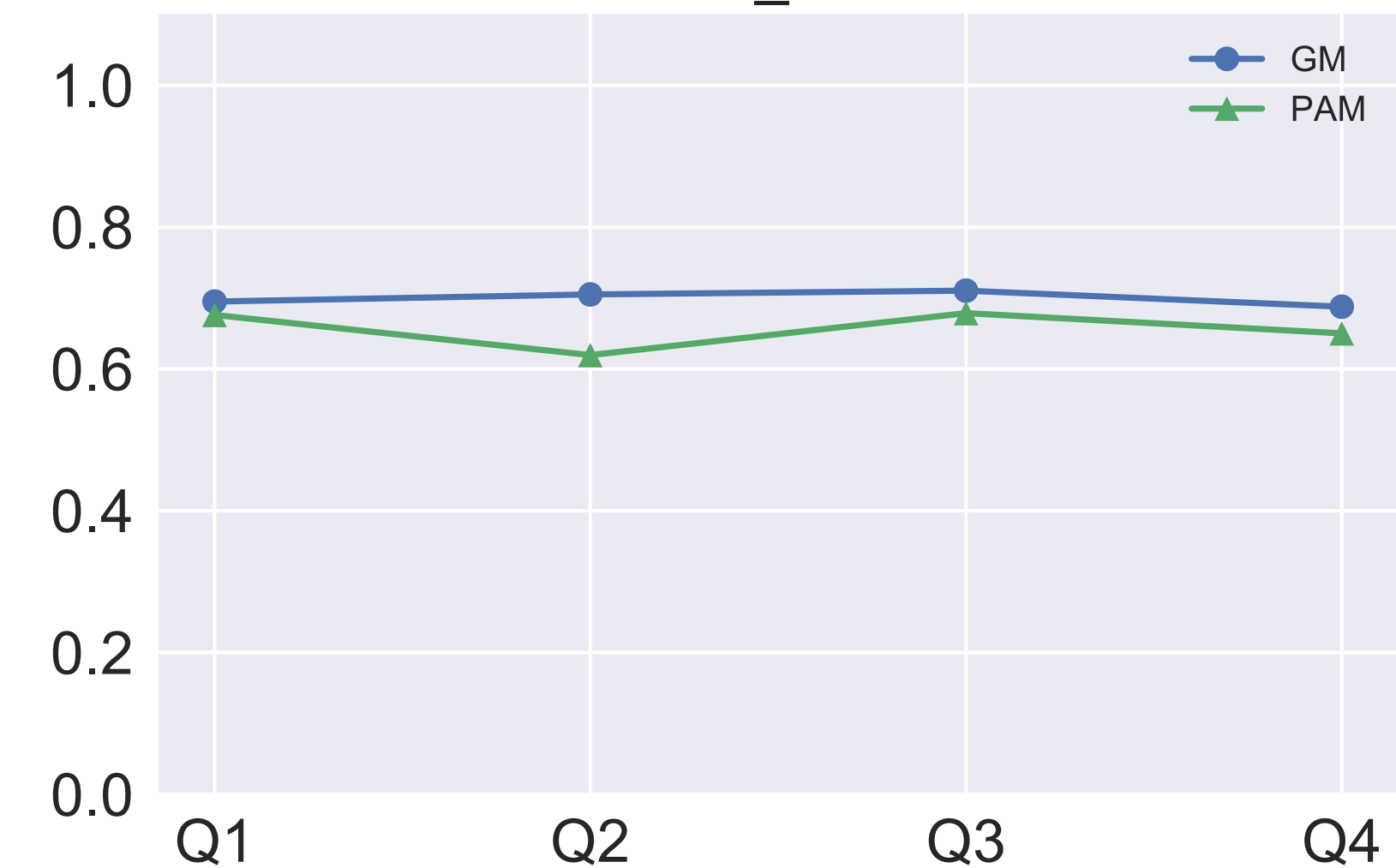
recall



g-score



roc_auc



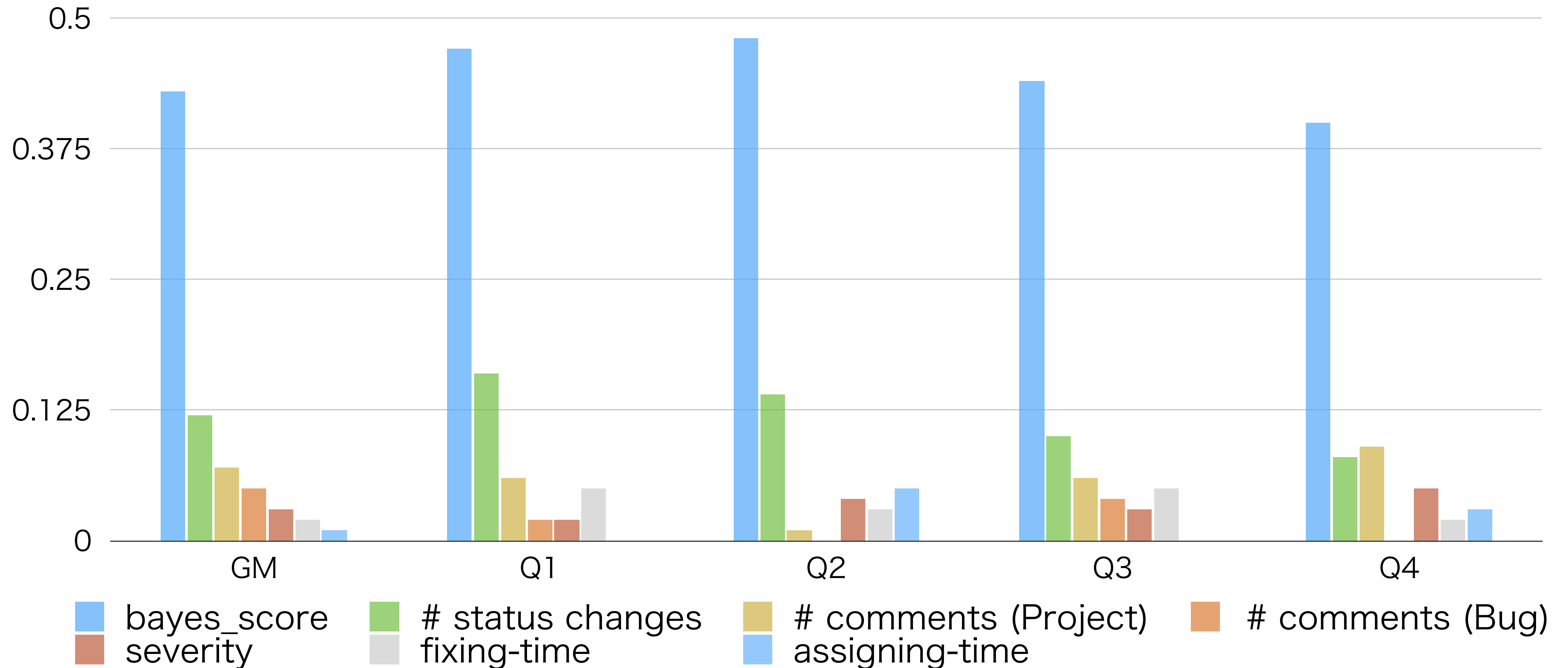
The recall of RAM in Q2 is only significantly higher

ROC-AUC of RAM are constantly lower



All models rely on bayes score but Global model made better predictions

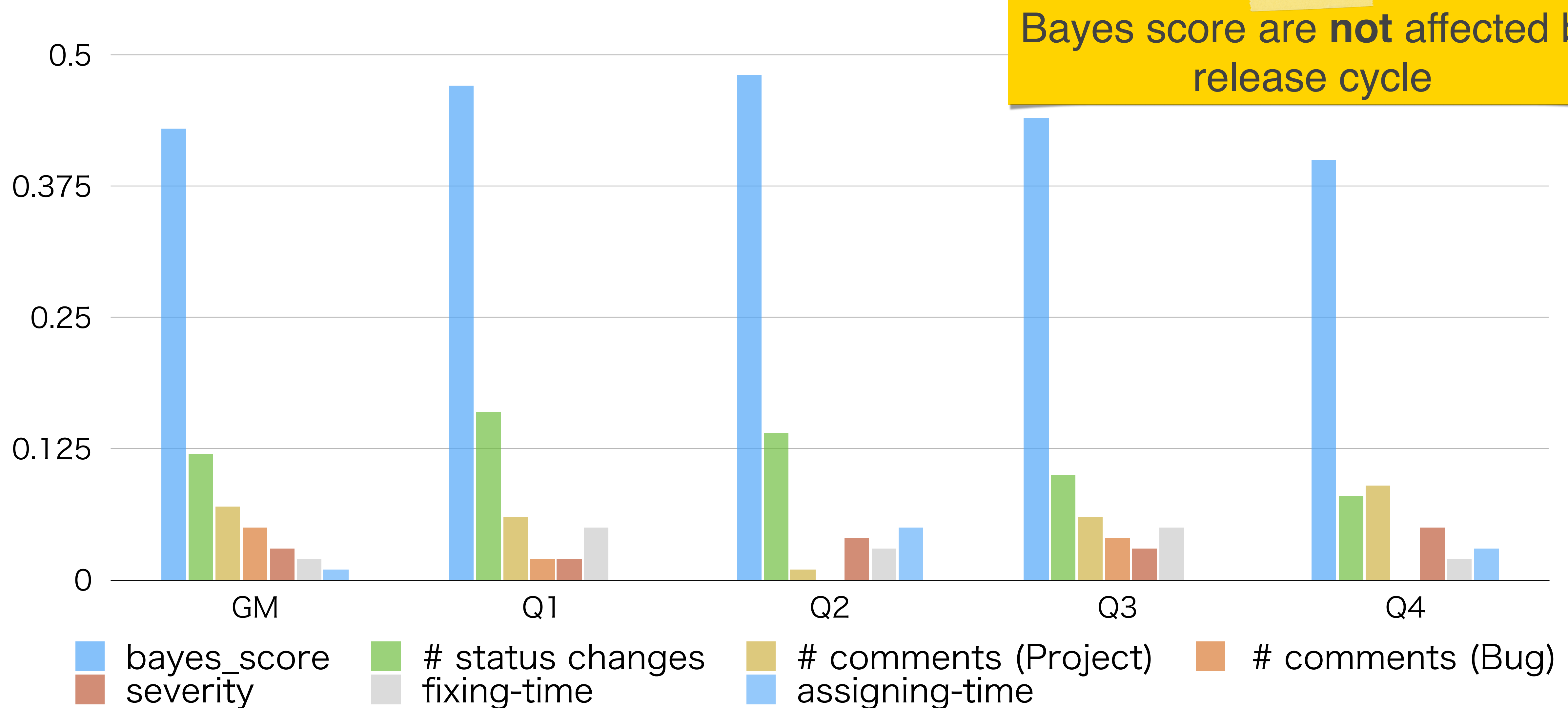
- Pick up variables (Importance > 0.05) for each prediction model





All models rely on bayes score but Global model made better predictions

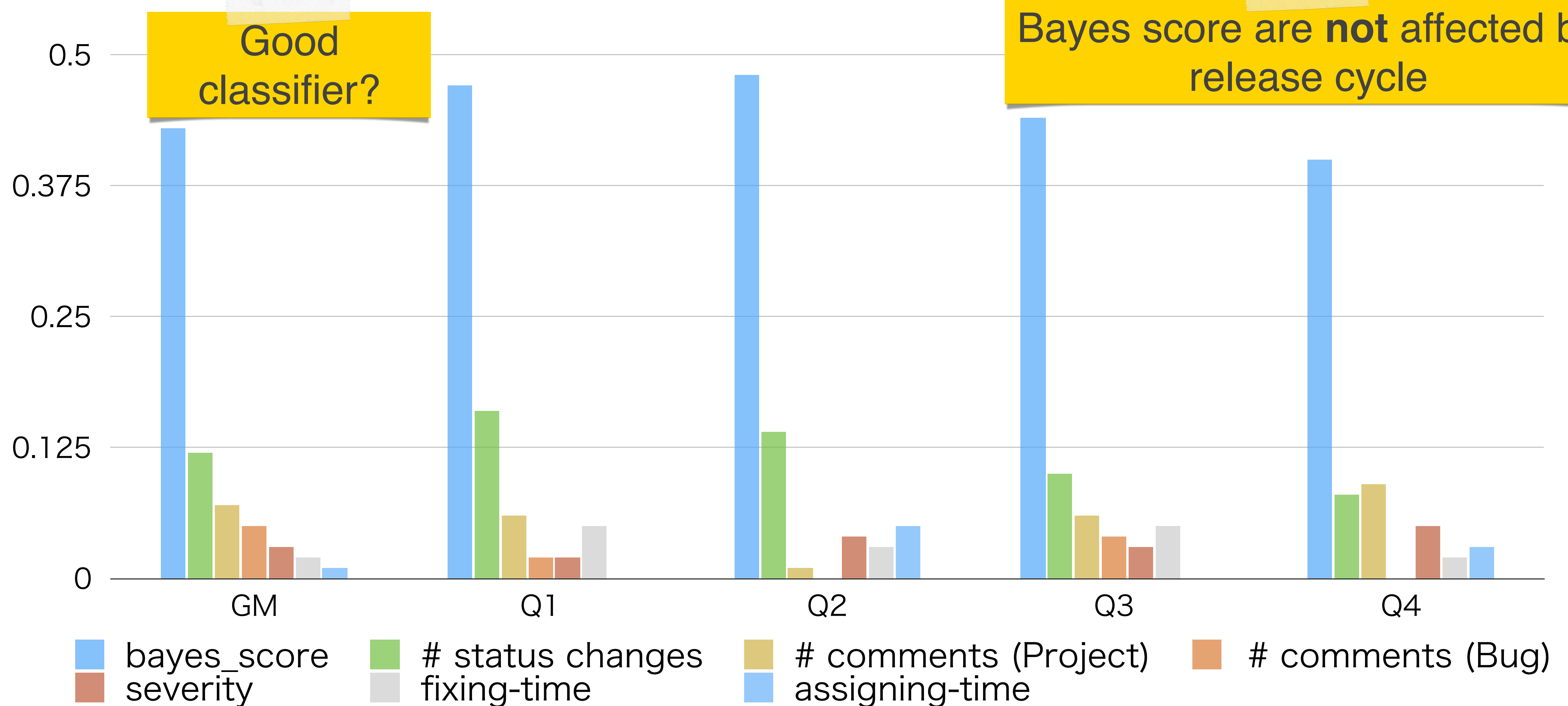
- Pick up variables (Importance > 0.05) for each prediction model





All models rely on bayes score but Global model made better predictions

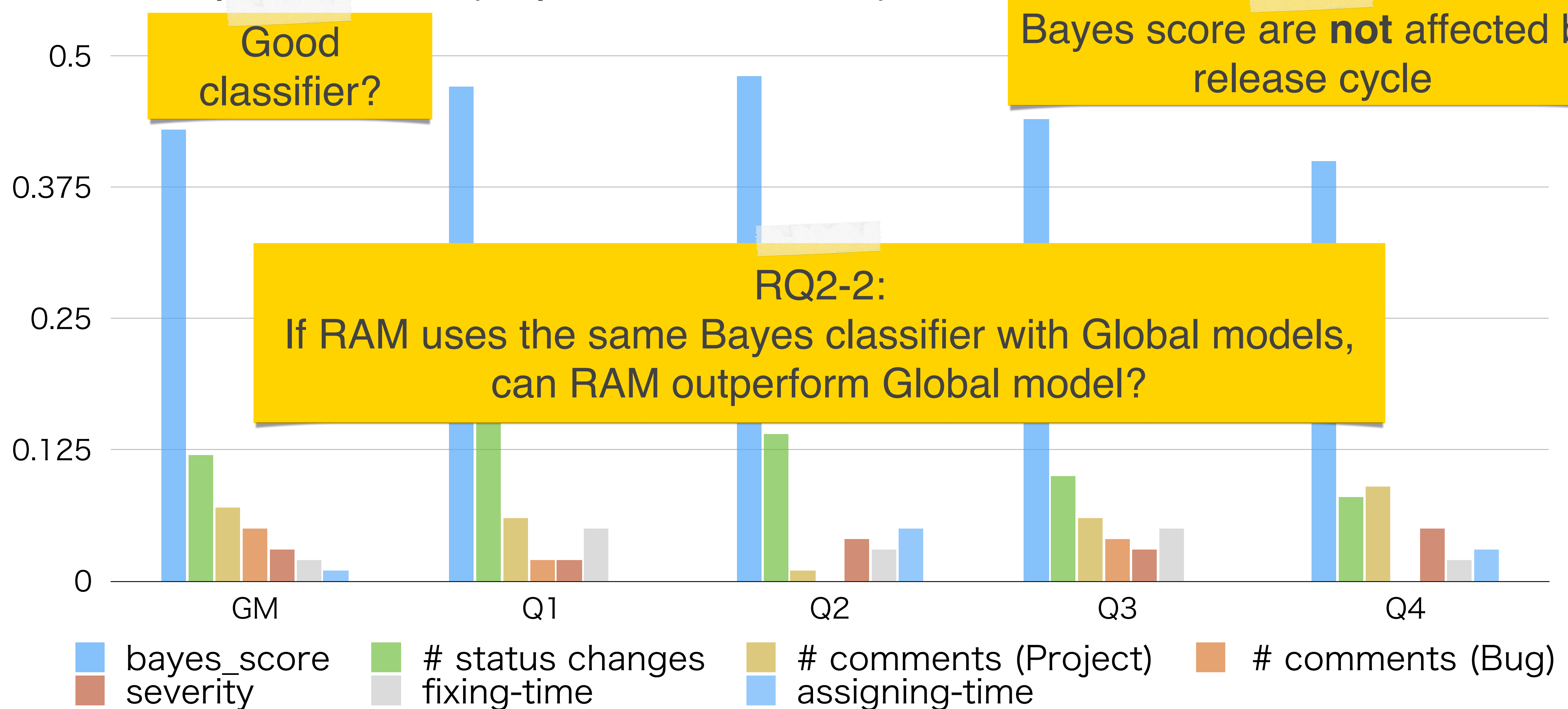
- Pick up variables (Importance > 0.05) for each prediction model





All models rely on bayes score but Global model made better predictions

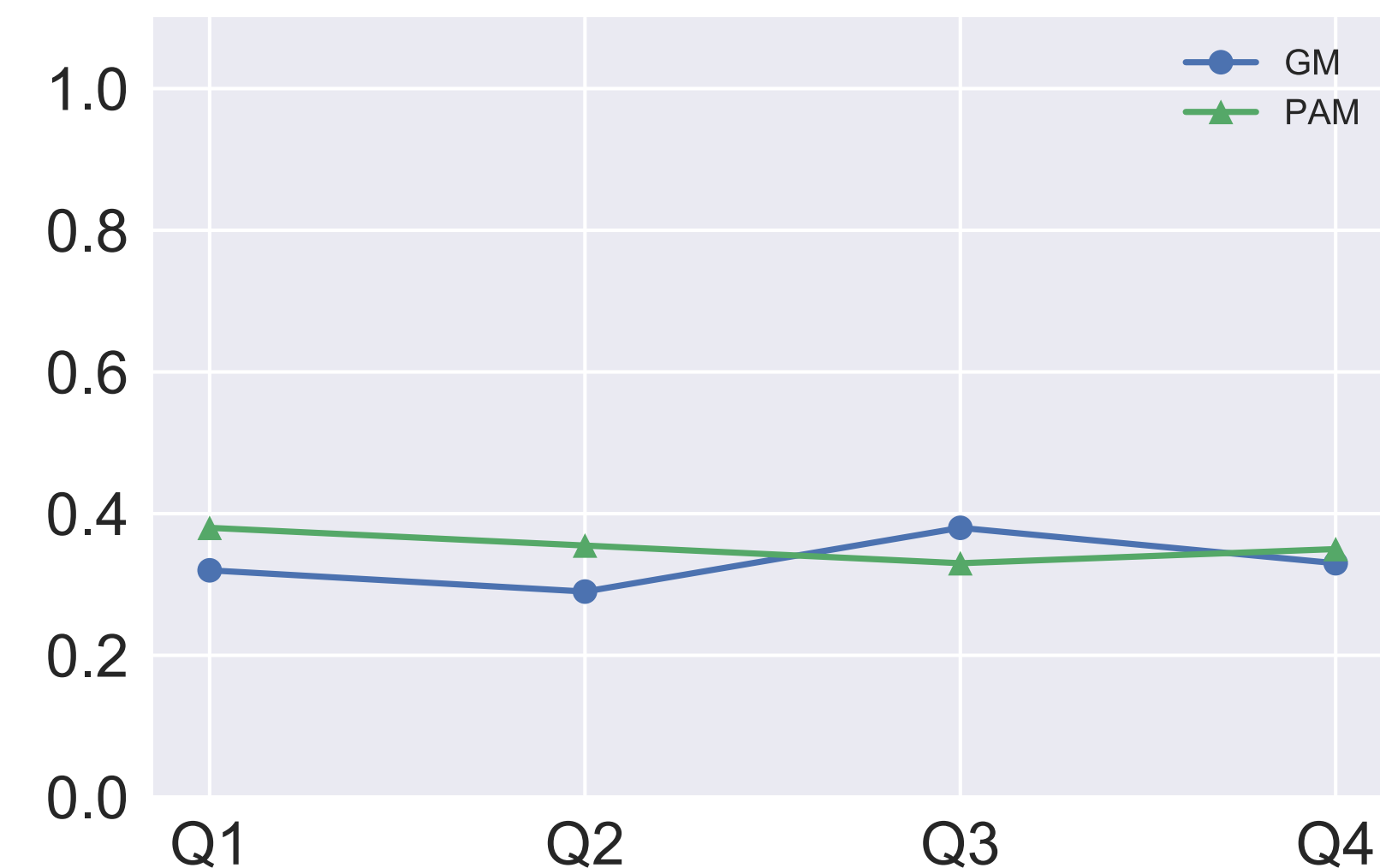
- Pick up variables (Importance > 0.05) for each prediction model



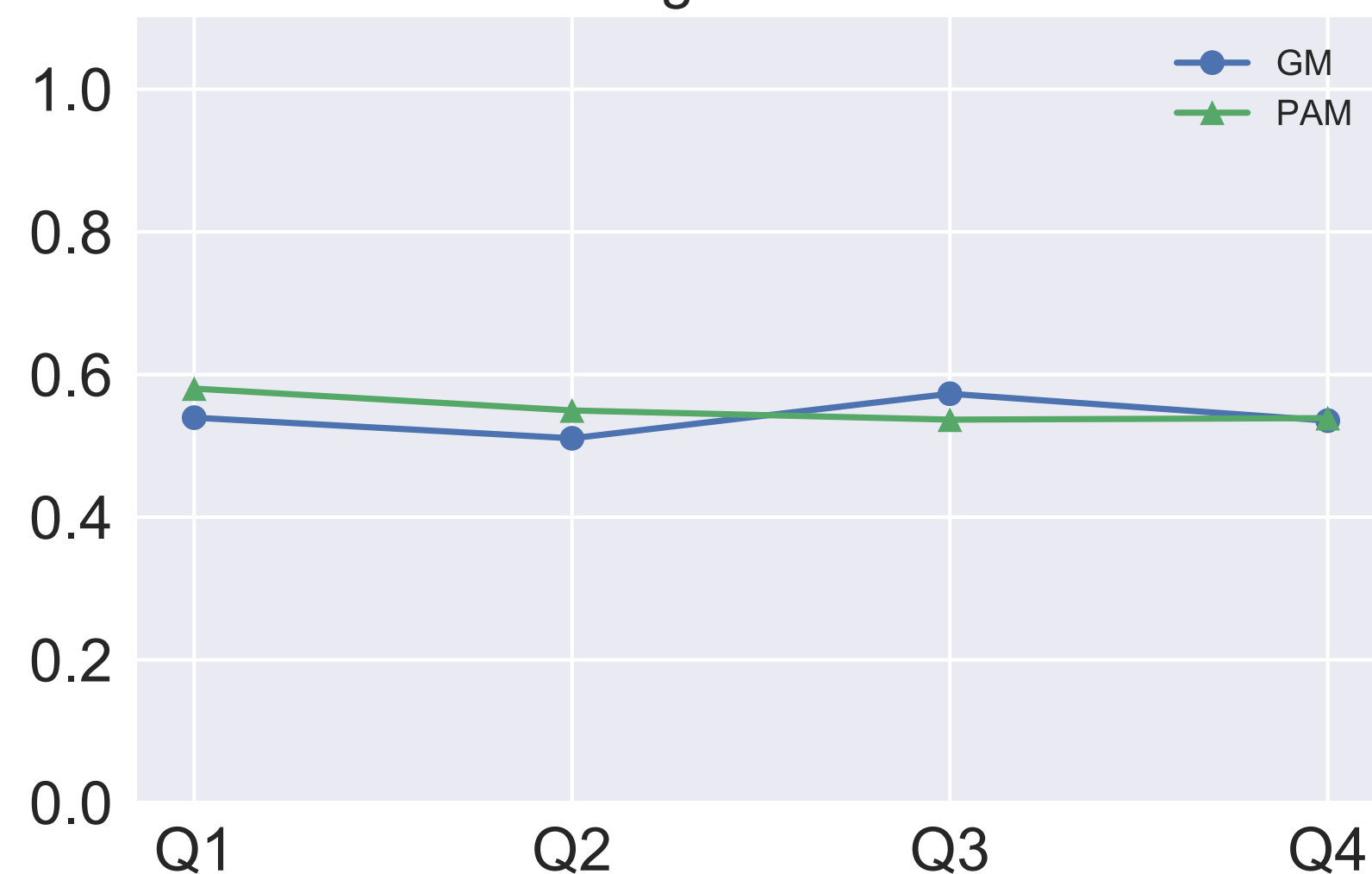


Hybrid Release aware models outperform Global model in Q1 and Q2 when have subtle in small percentages of whole data

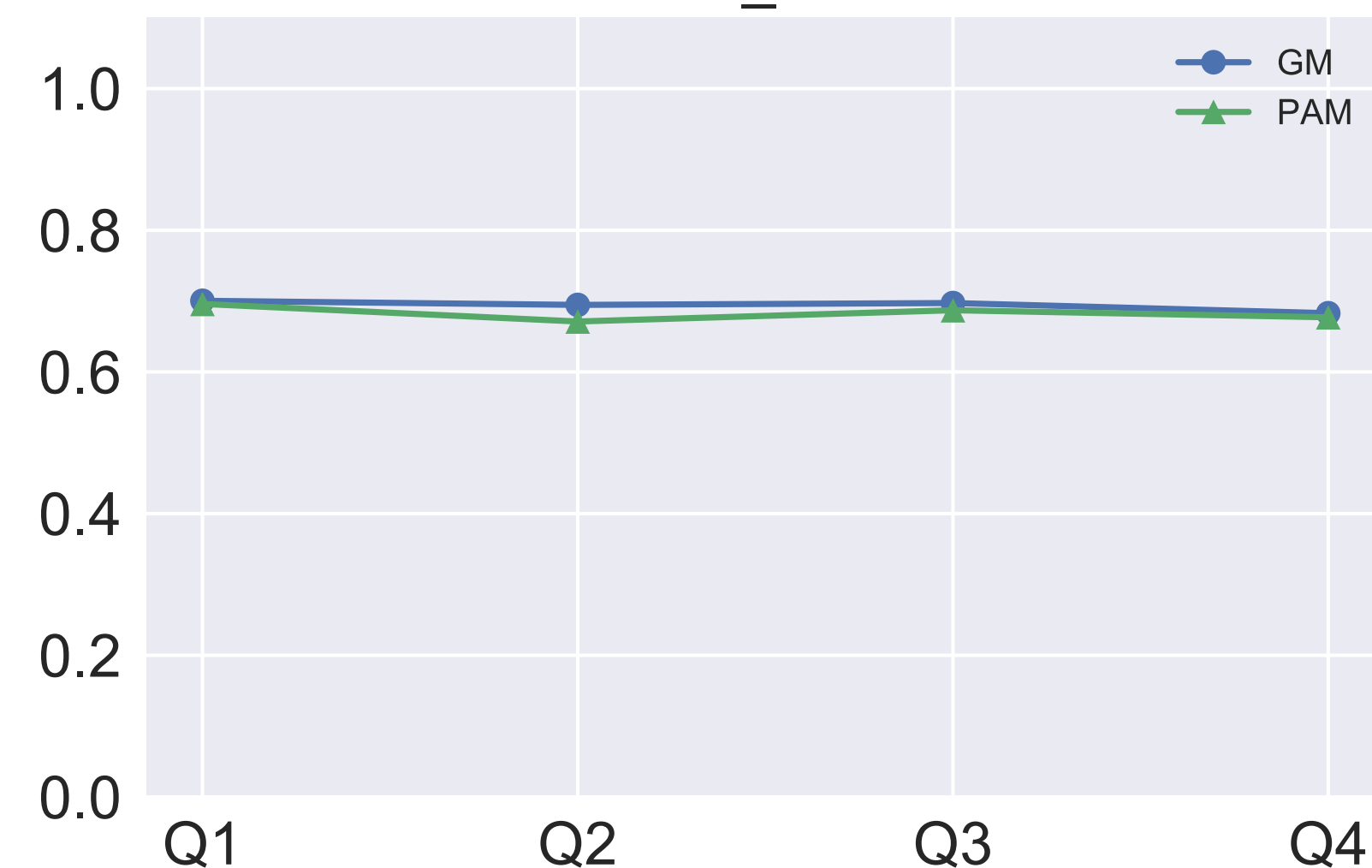
recall



g-score

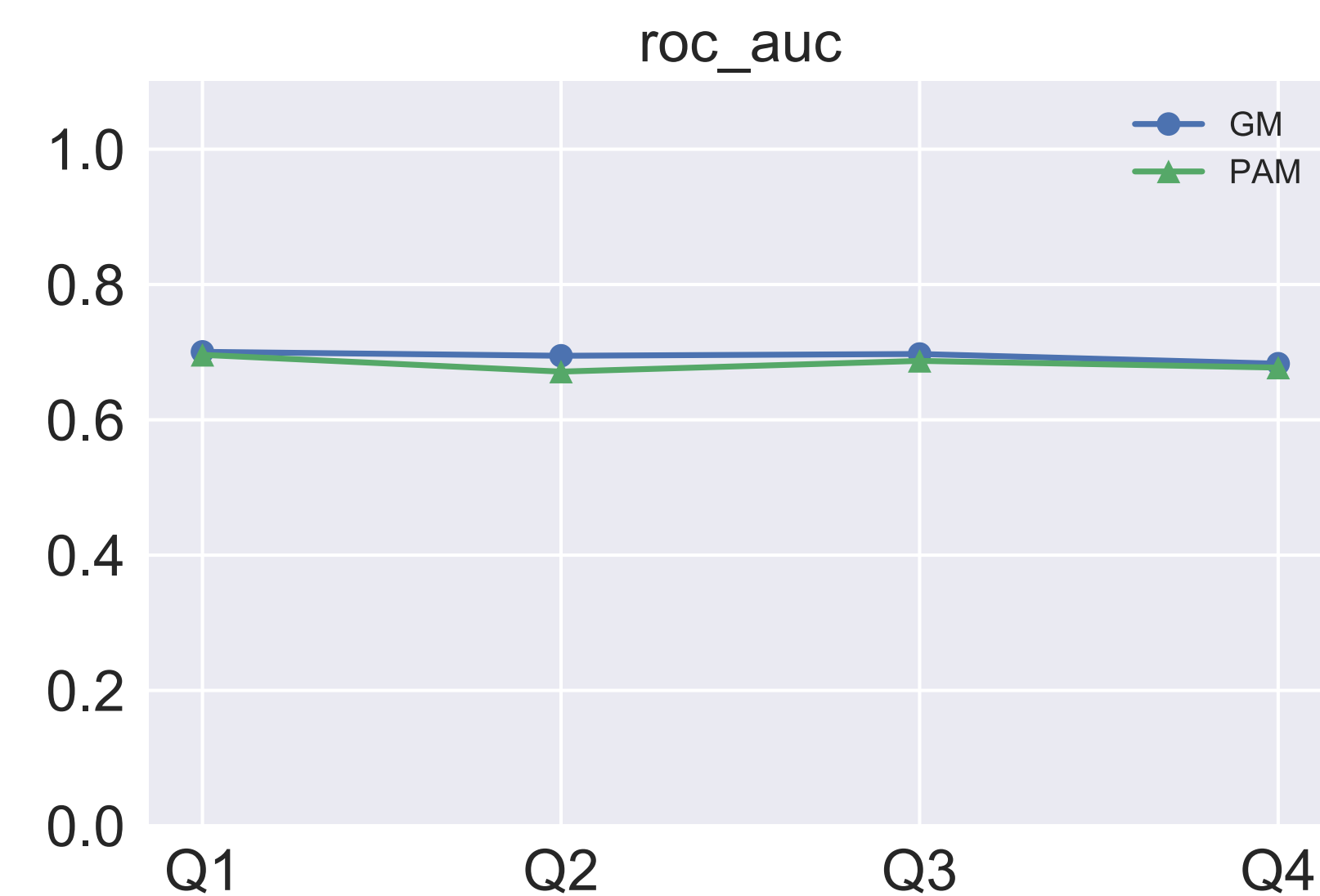
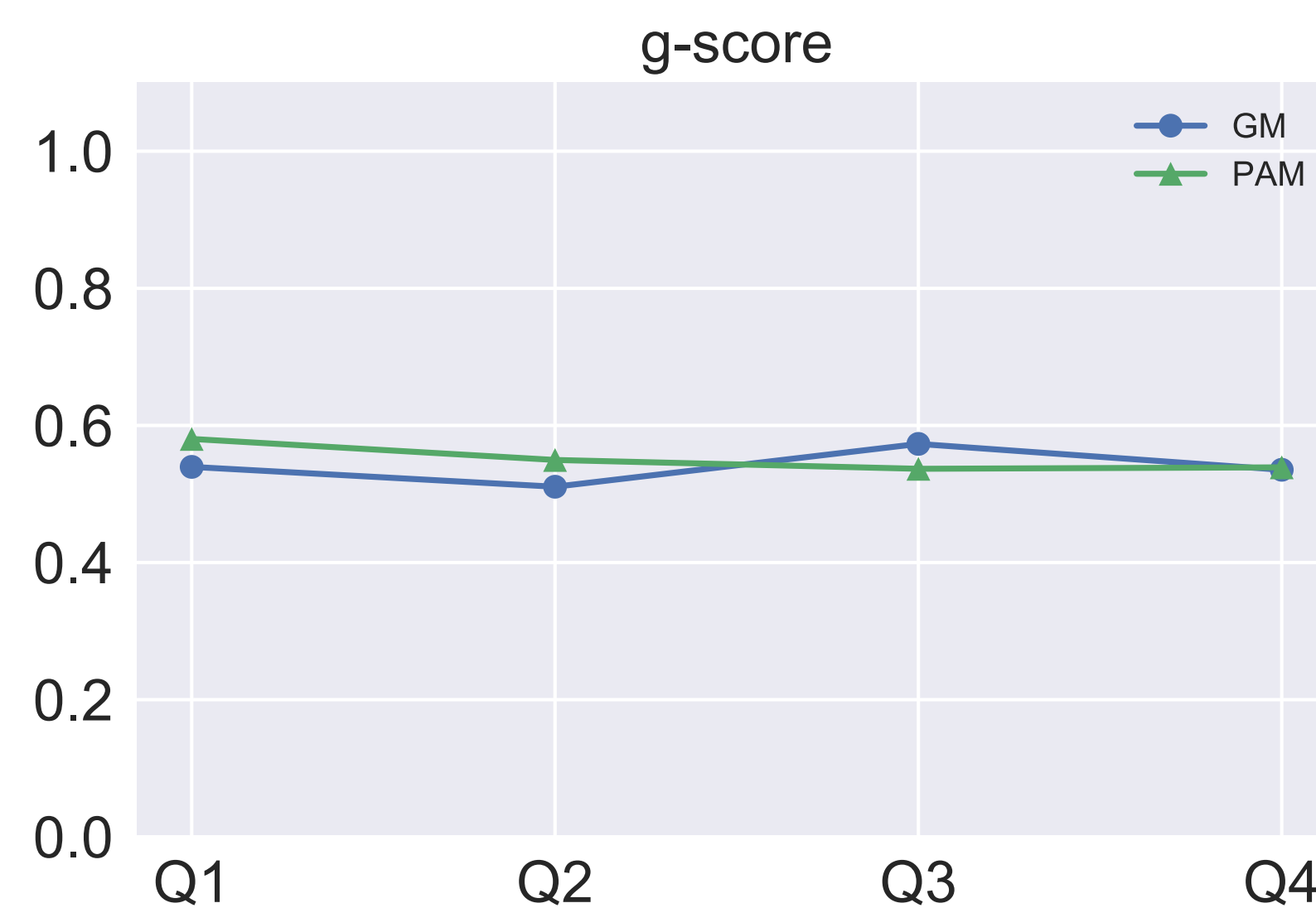
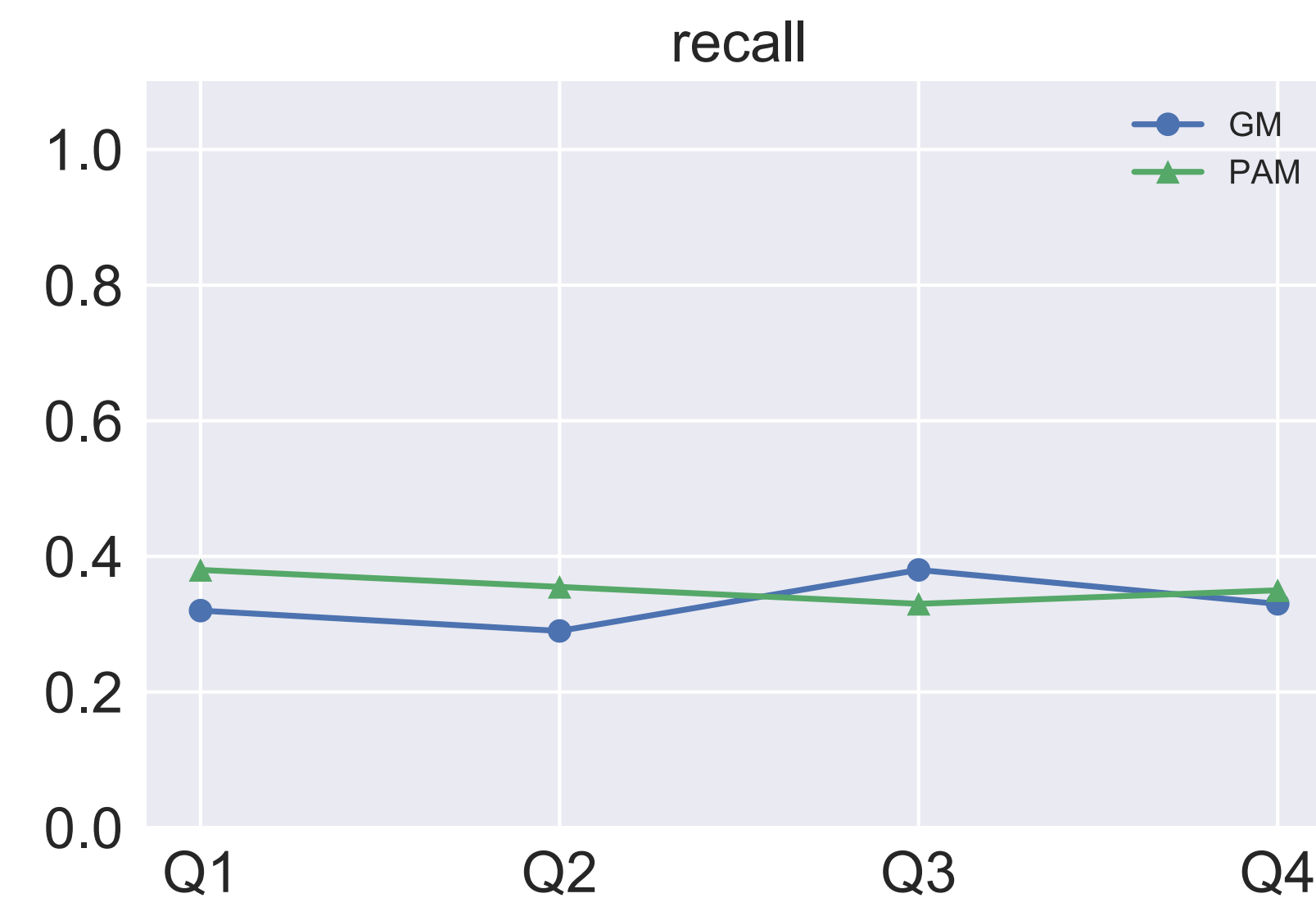


roc_auc





Hybrid Release aware models outperform Global model in Q1 and Q2 when have subtle in small percentages of whole data



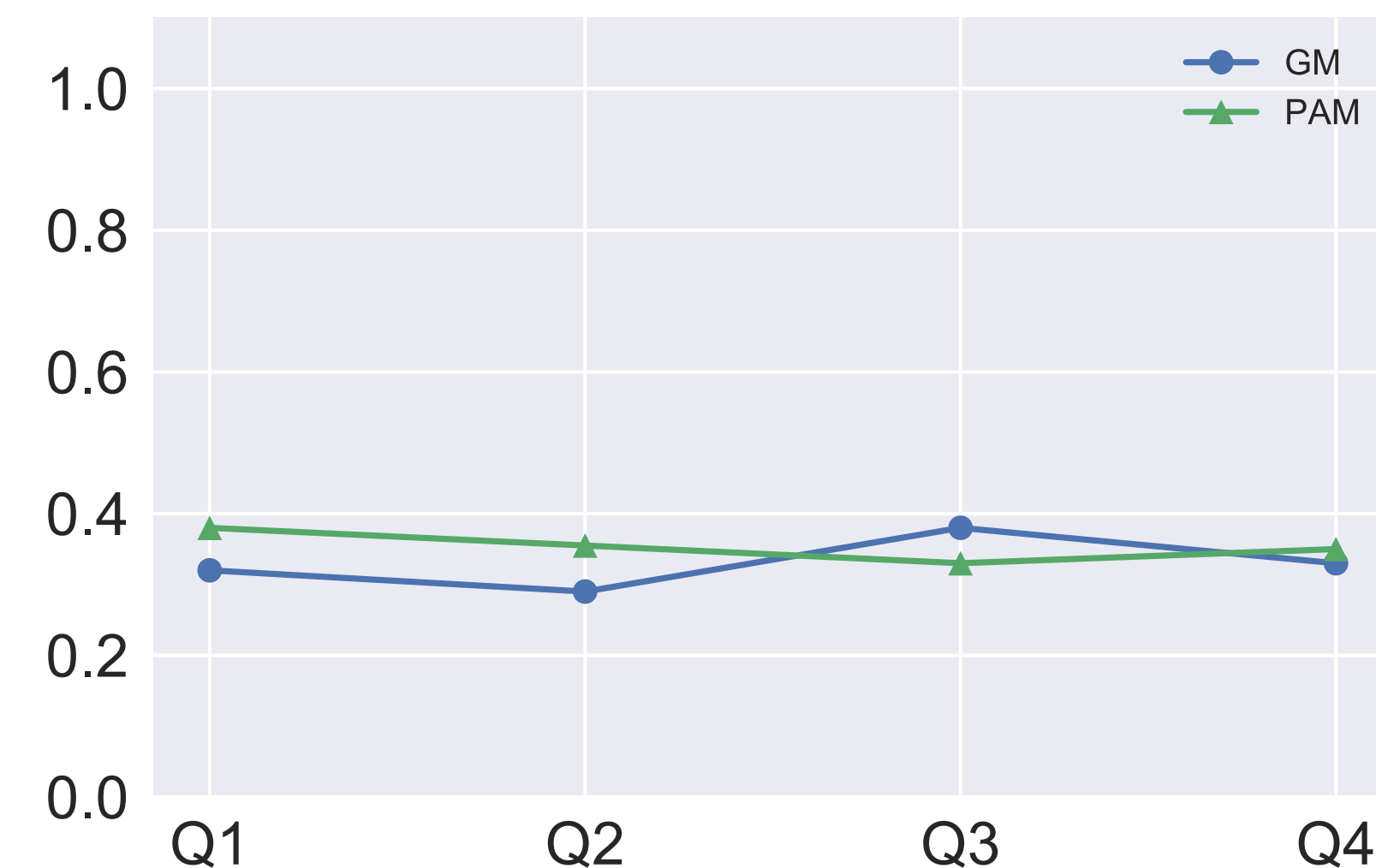
Addition to Q2, the recall of PAM in Q1 is also significantly higher



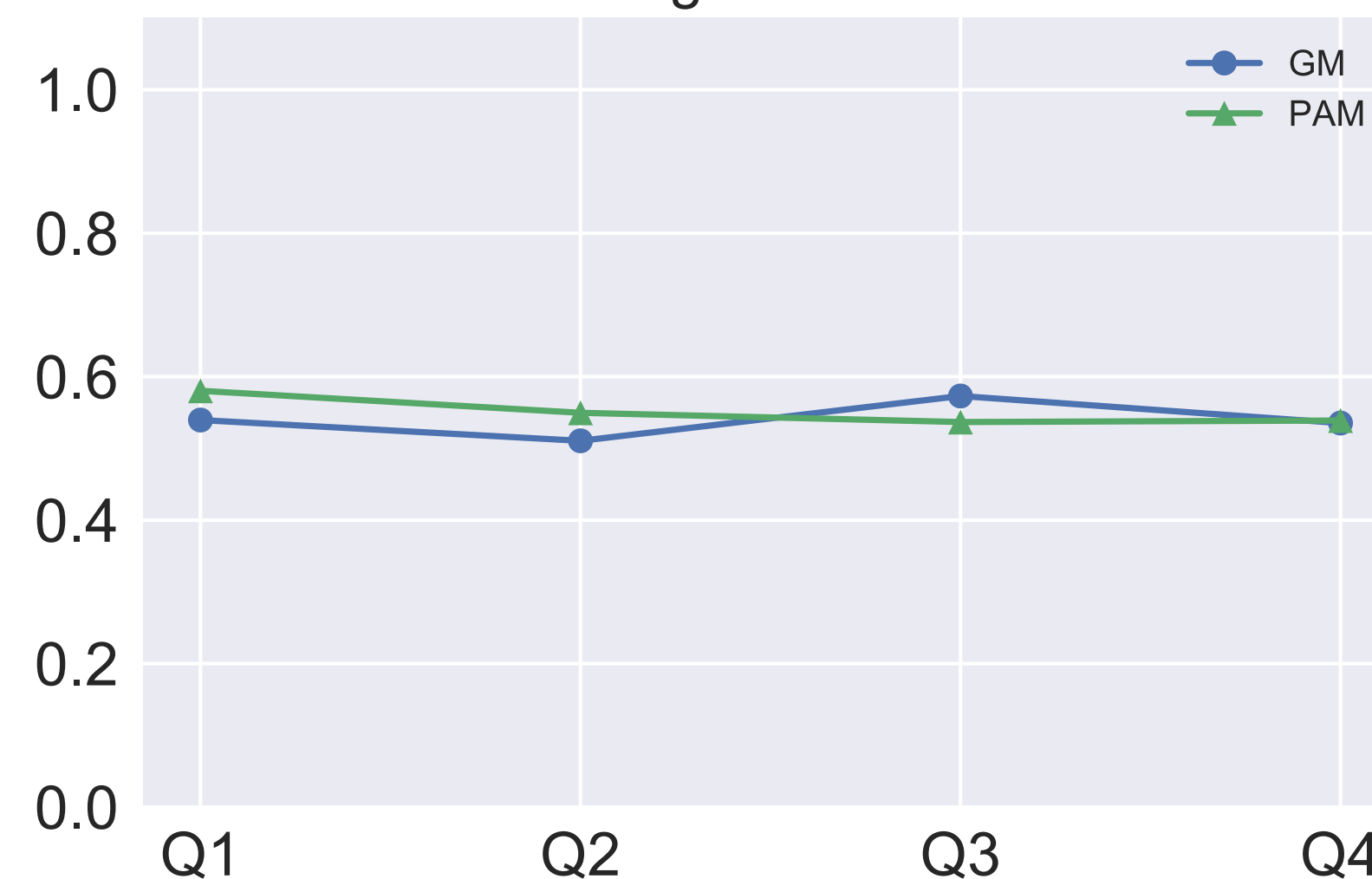
Hybrid Release aware models outperform Global model in Q1 and Q2 when have subtle in small percentages of whole data

The g-mean of PAM in Q1 and Q2 become significantly higher

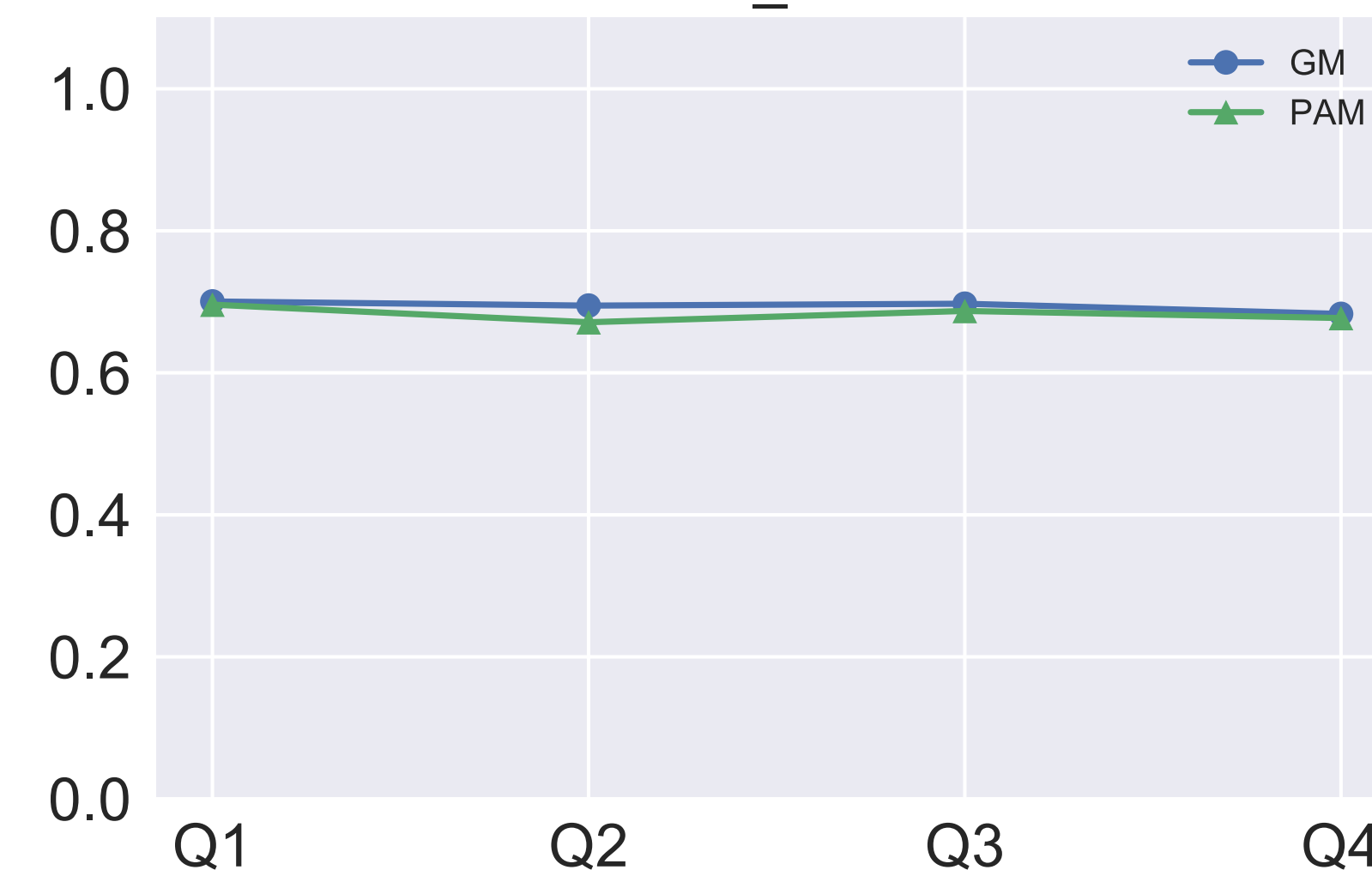
recall



g-score



roc_auc



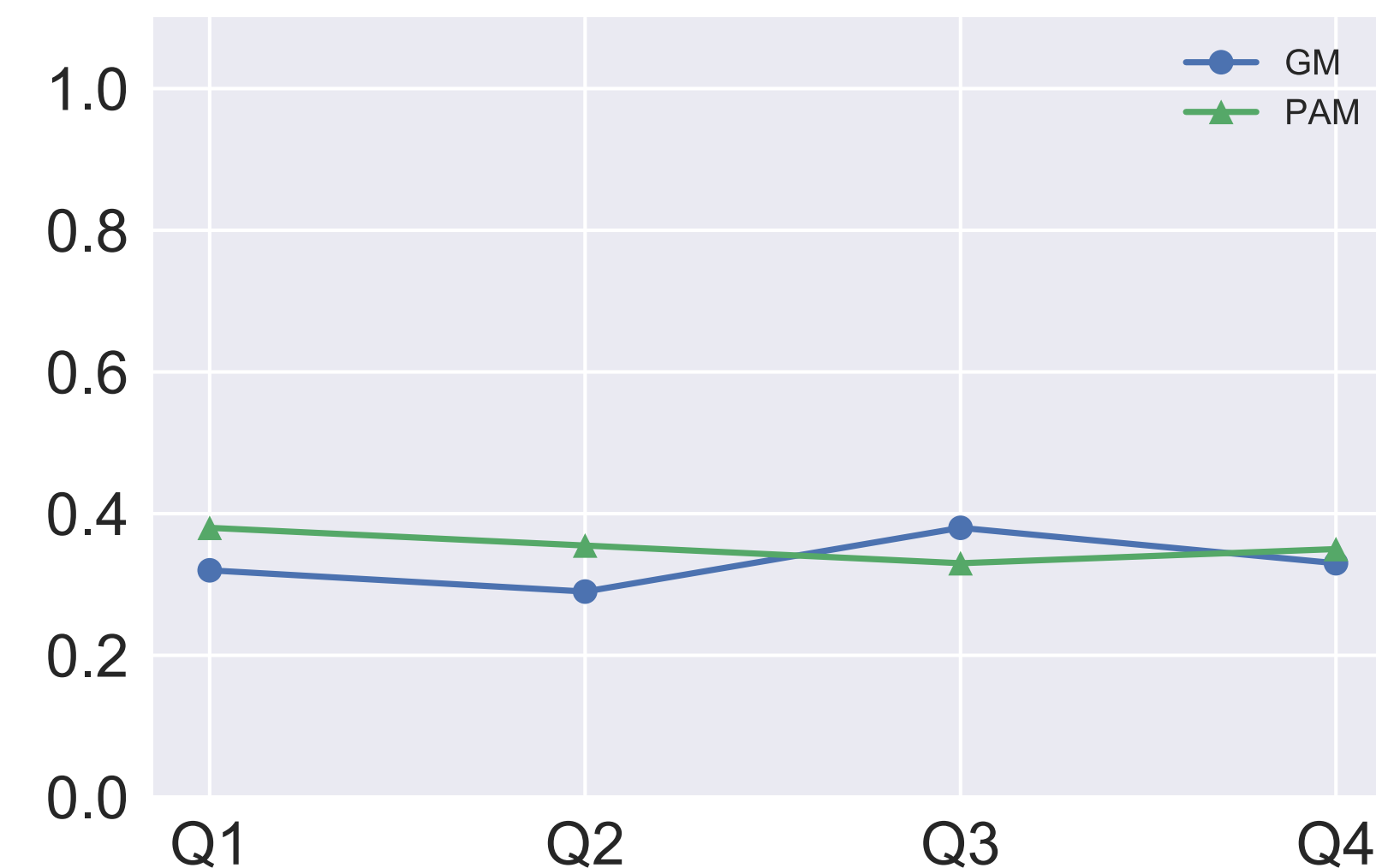
Addition to Q2, the recall of PAM in Q1 is also significantly higher



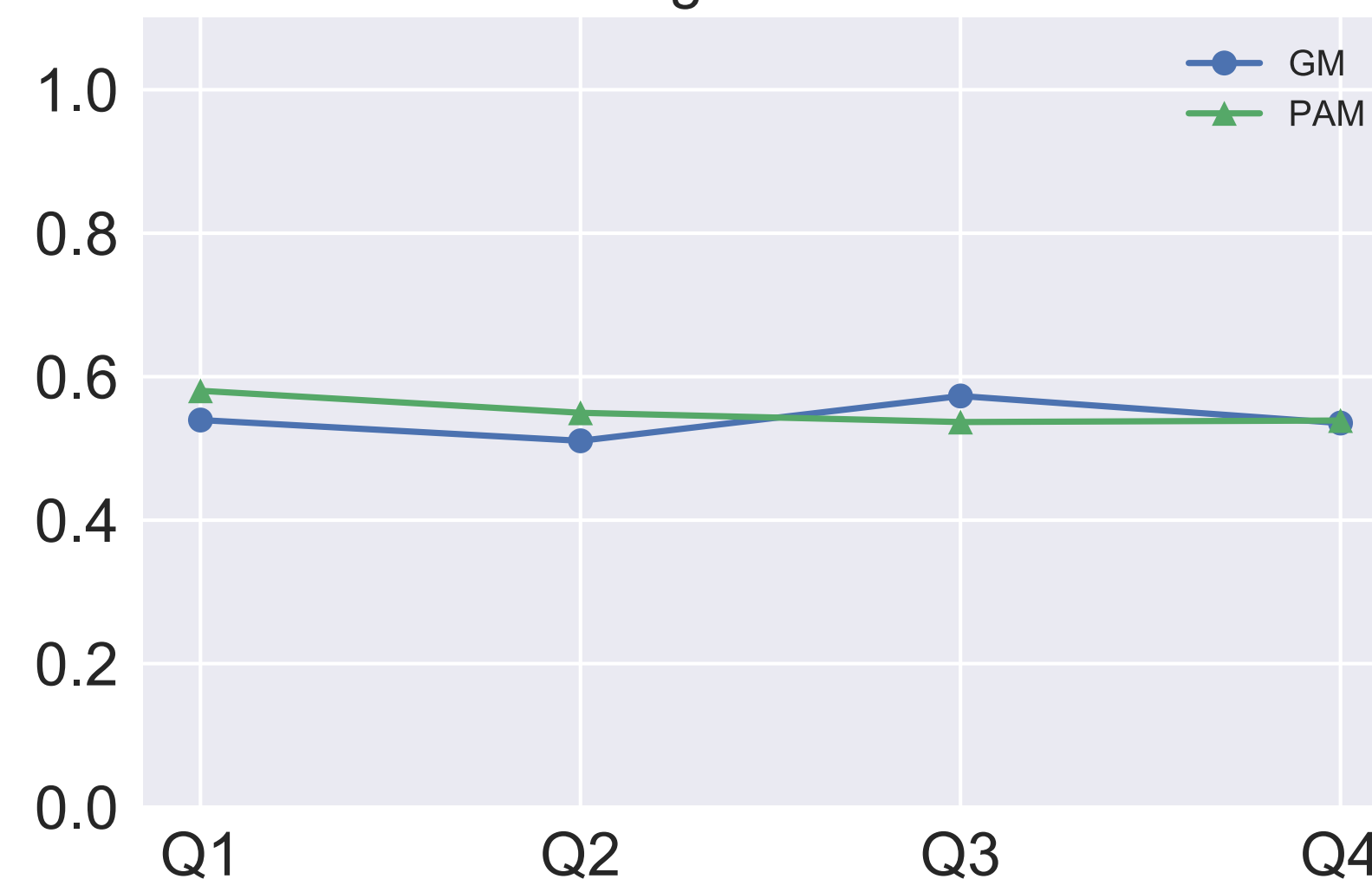
Hybrid Release aware models outperform Global model in Q1 and Q2 when have subtle in small percentages of whole data

The g-mean of PAM in Q1 and Q2 become significantly higher

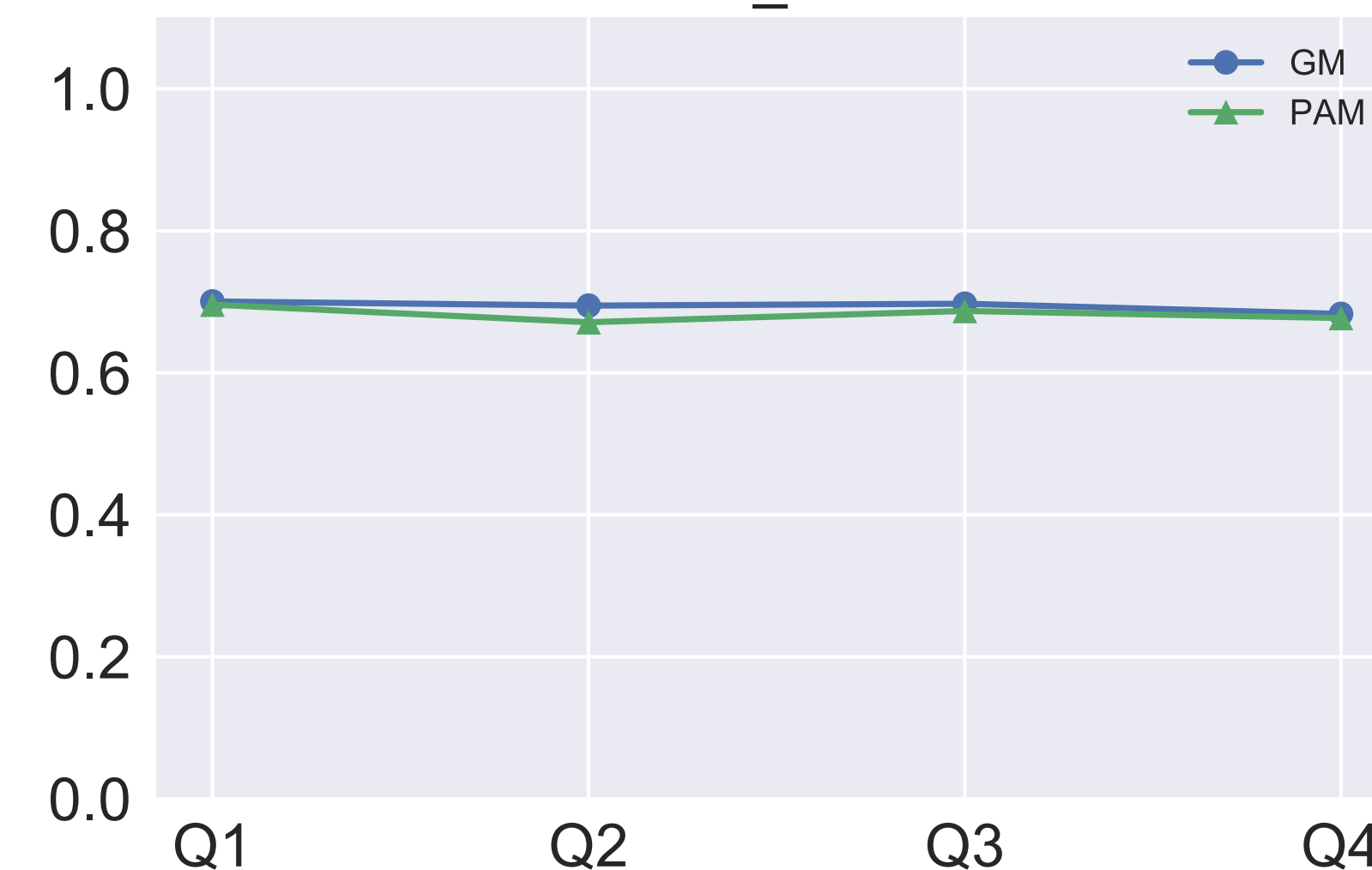
recall



g-score



roc_auc

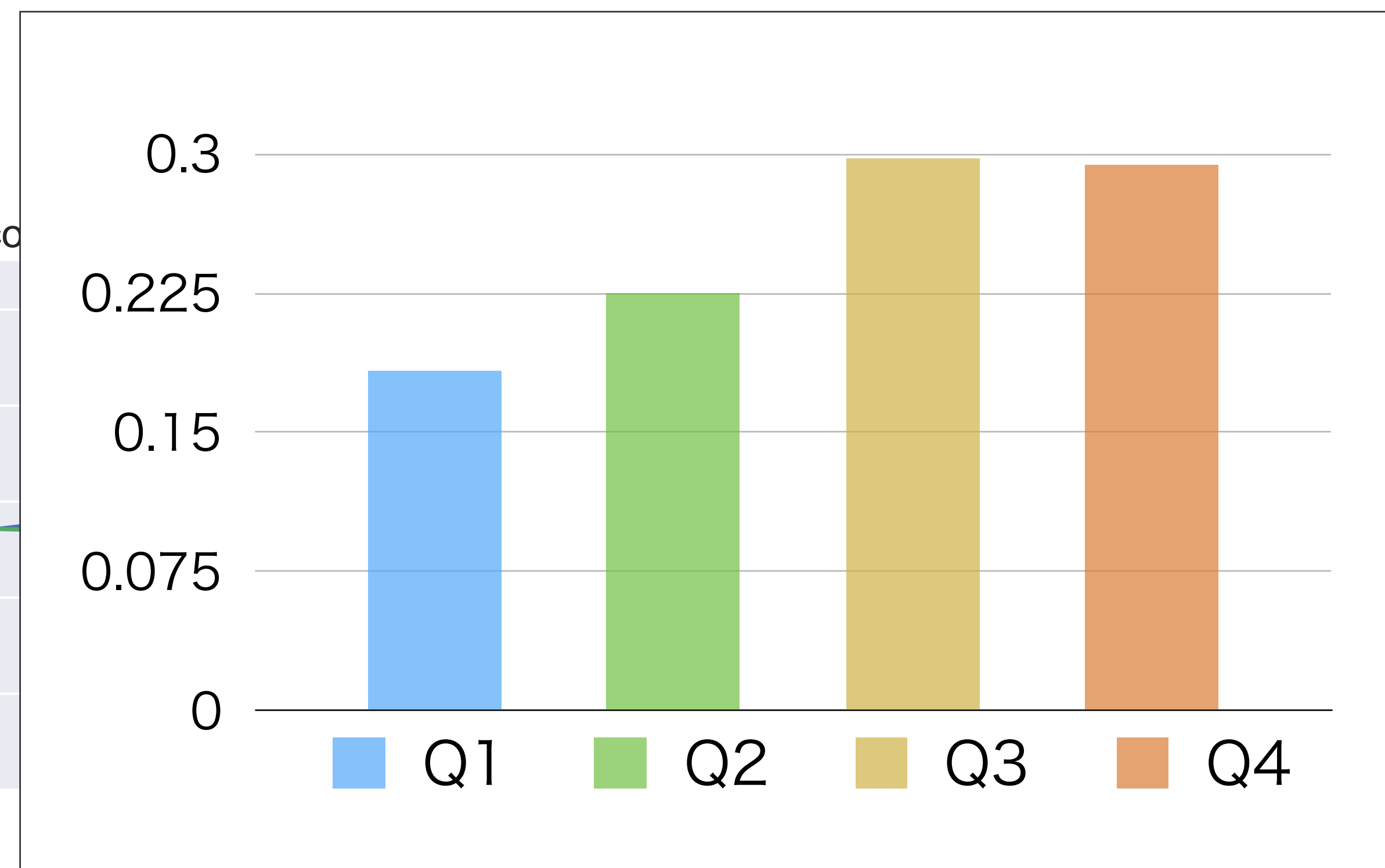
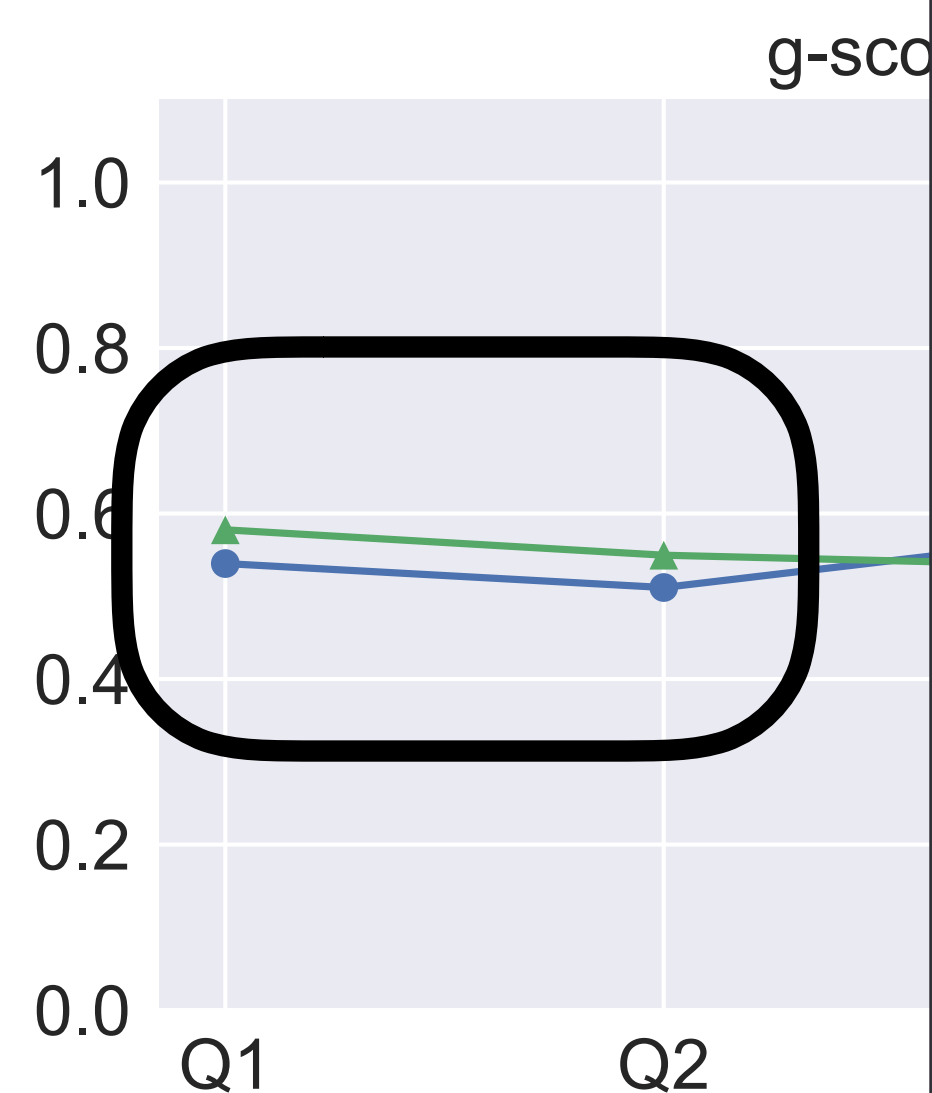
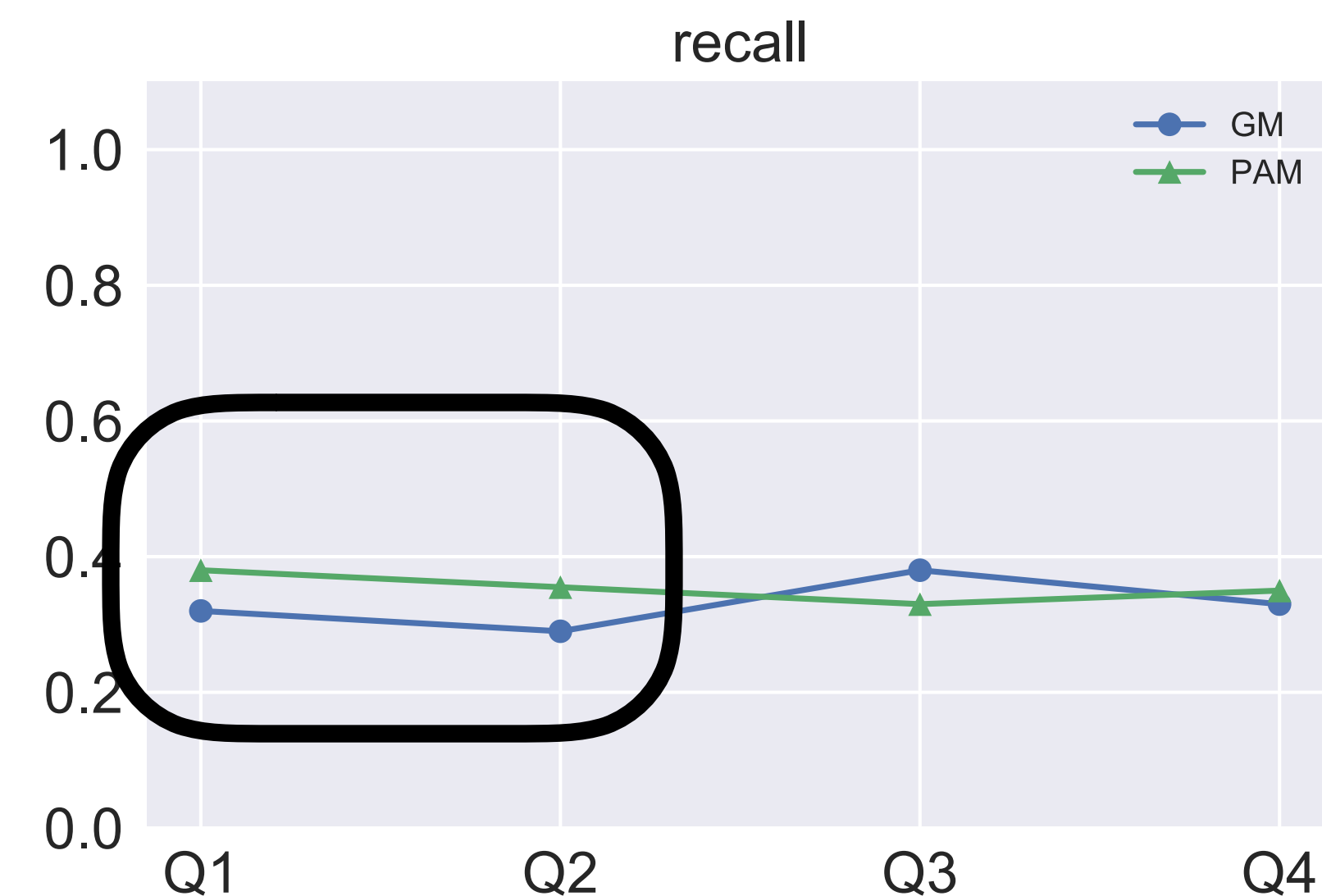


Addition to Q2, the recall of PAM in Q1 is also significantly higher

The difference of ROC-AUC become smaller

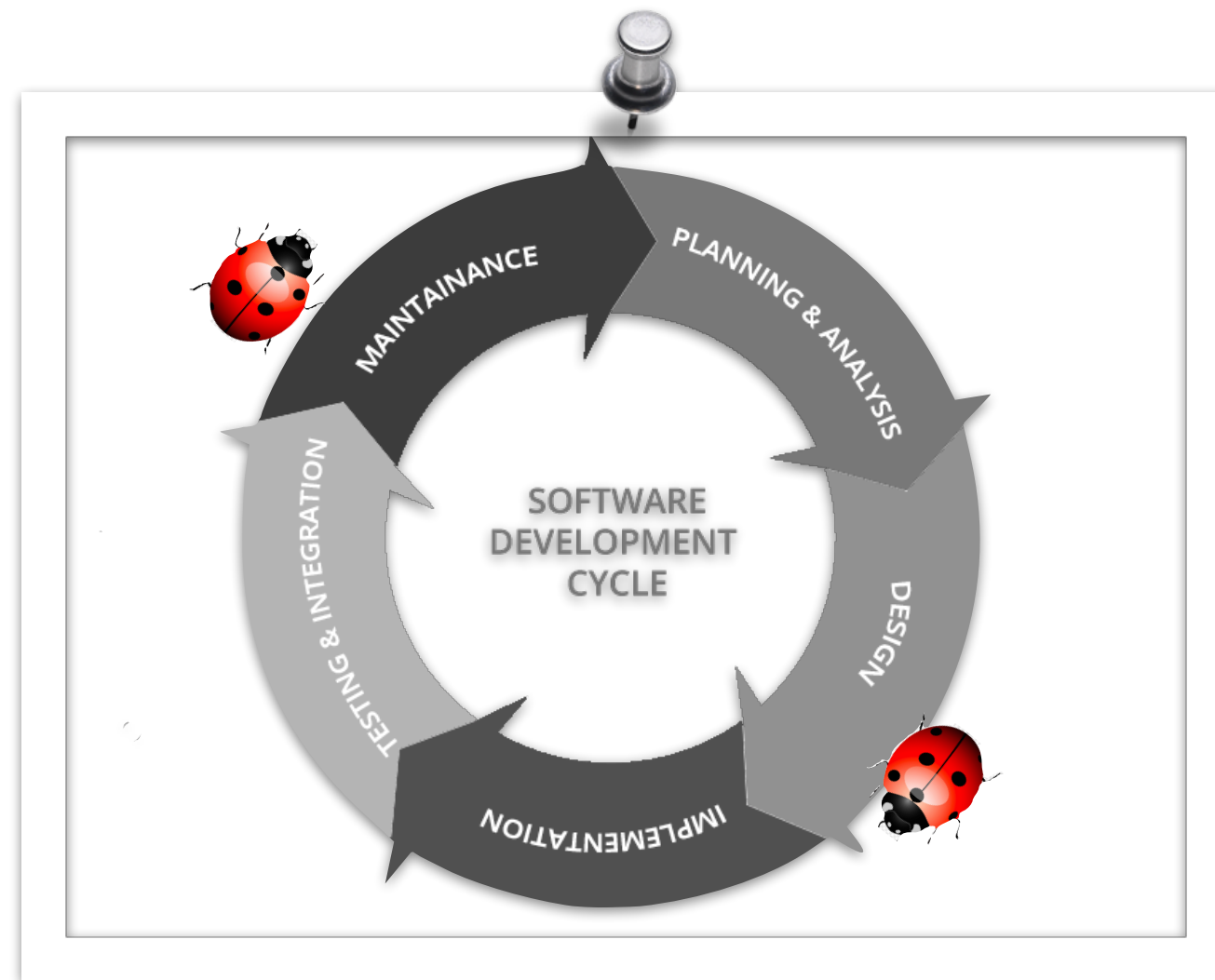


Hybrid Release aware models outperform Global model in Q1 and Q2 when have subtle in small percentages of whole data



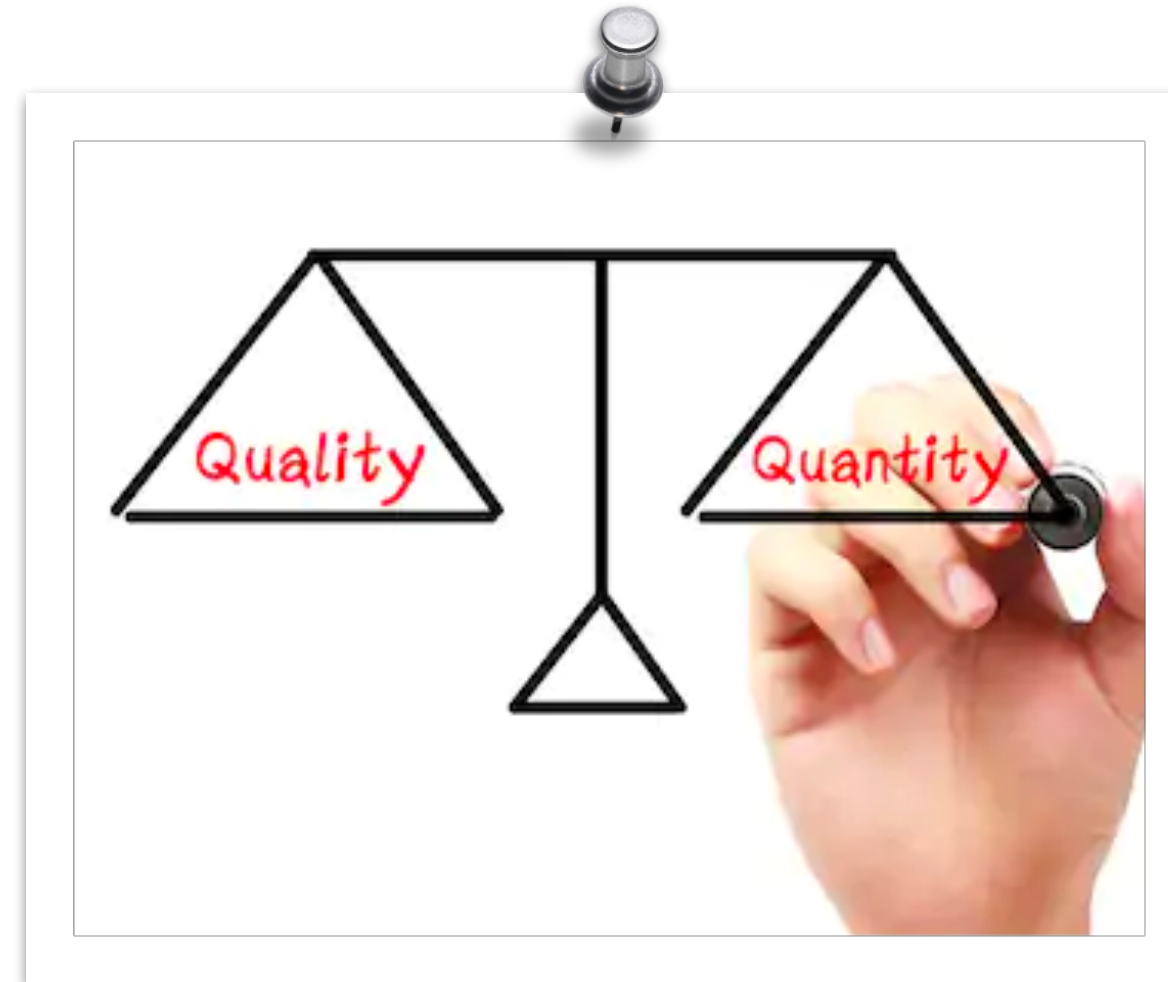
Metrics are affected by release cycle

the dataset is imbalanced



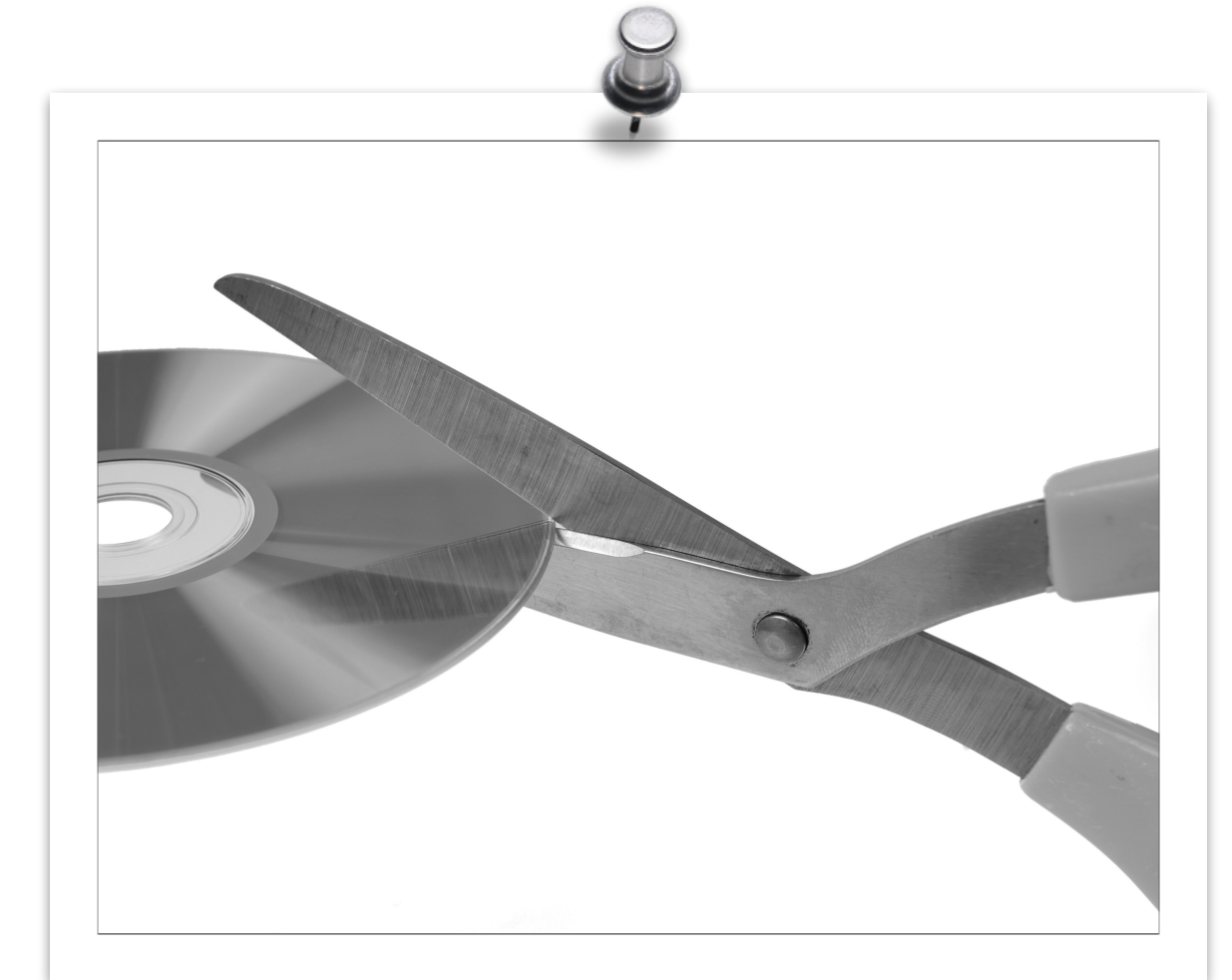
RQ1: Do bug-fixing activities vary across release cycle?

Yes, fixing activities are different across release cycle (especially, bug-fixing, priority change)

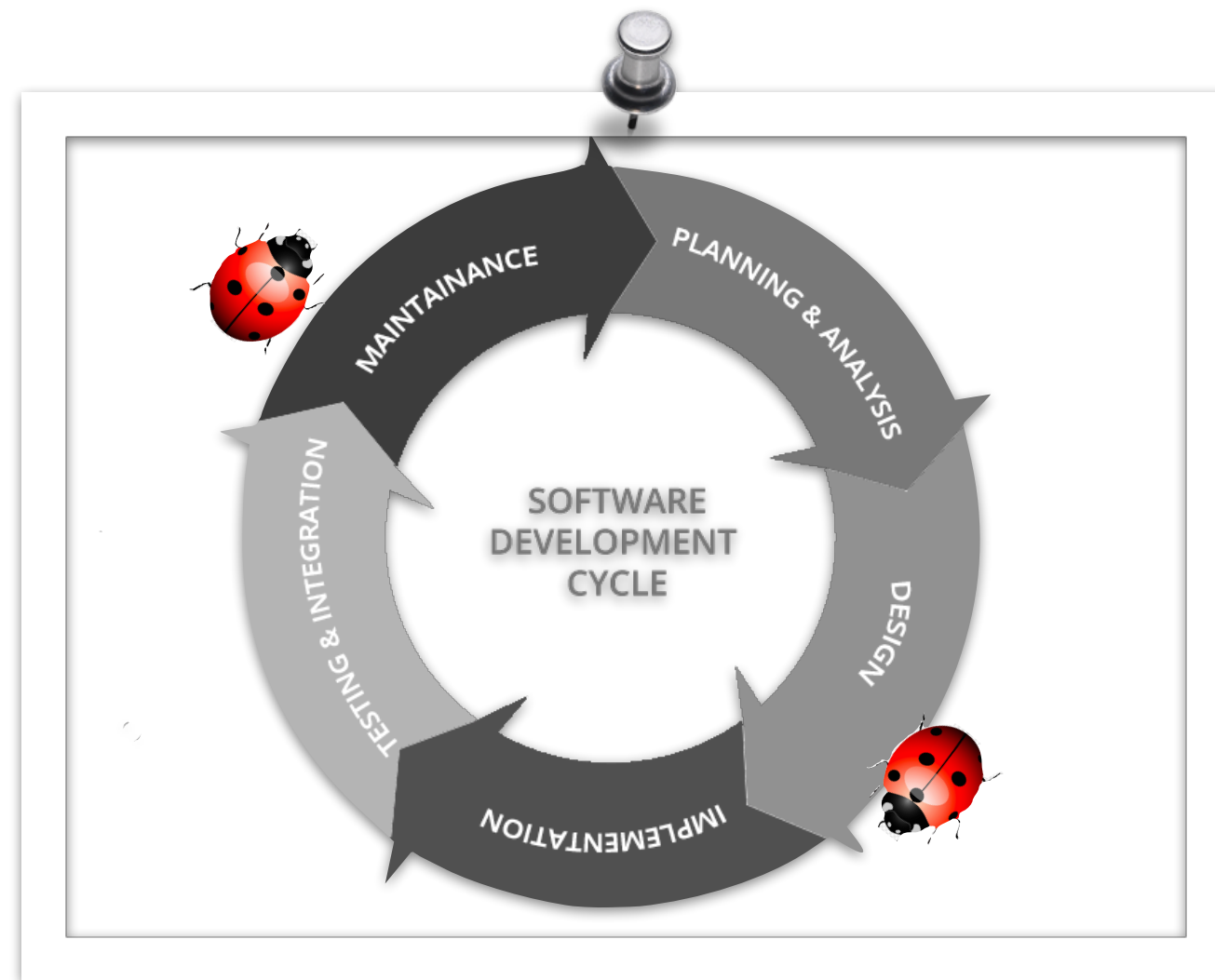


RQ2: Do Release-aware models for priority change outperform Global model?

Yes, release aware models leverage the characteristic of the release process

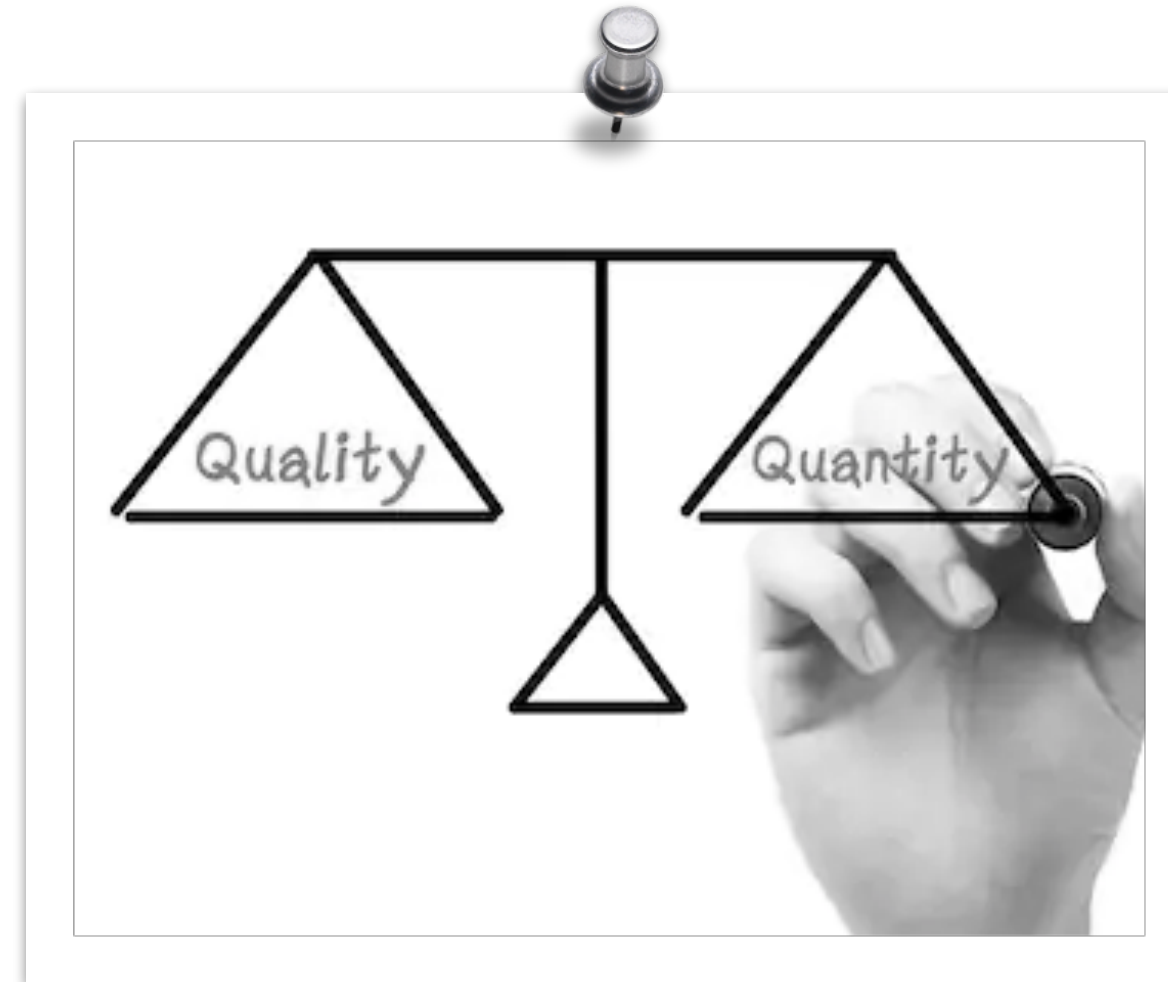


RQ3: How fine-grained data should be used?



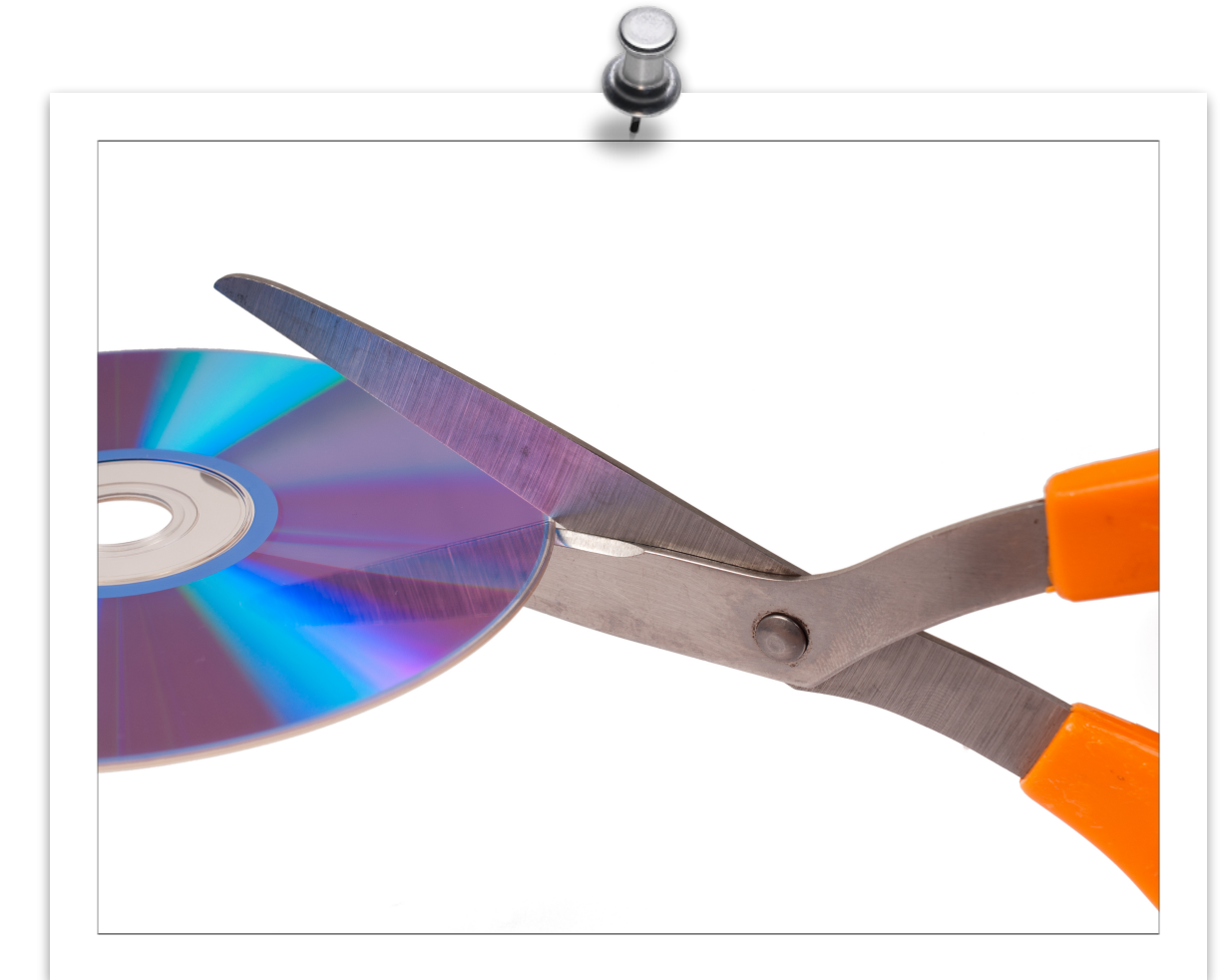
RQ1: Do bug-fixing activities vary across within release cycle?

Yes, fixing activities are different across the release cycle (especially, bug-fixing, priority change)



RQ2: Do Release-aware models for priority change outperform Global model?

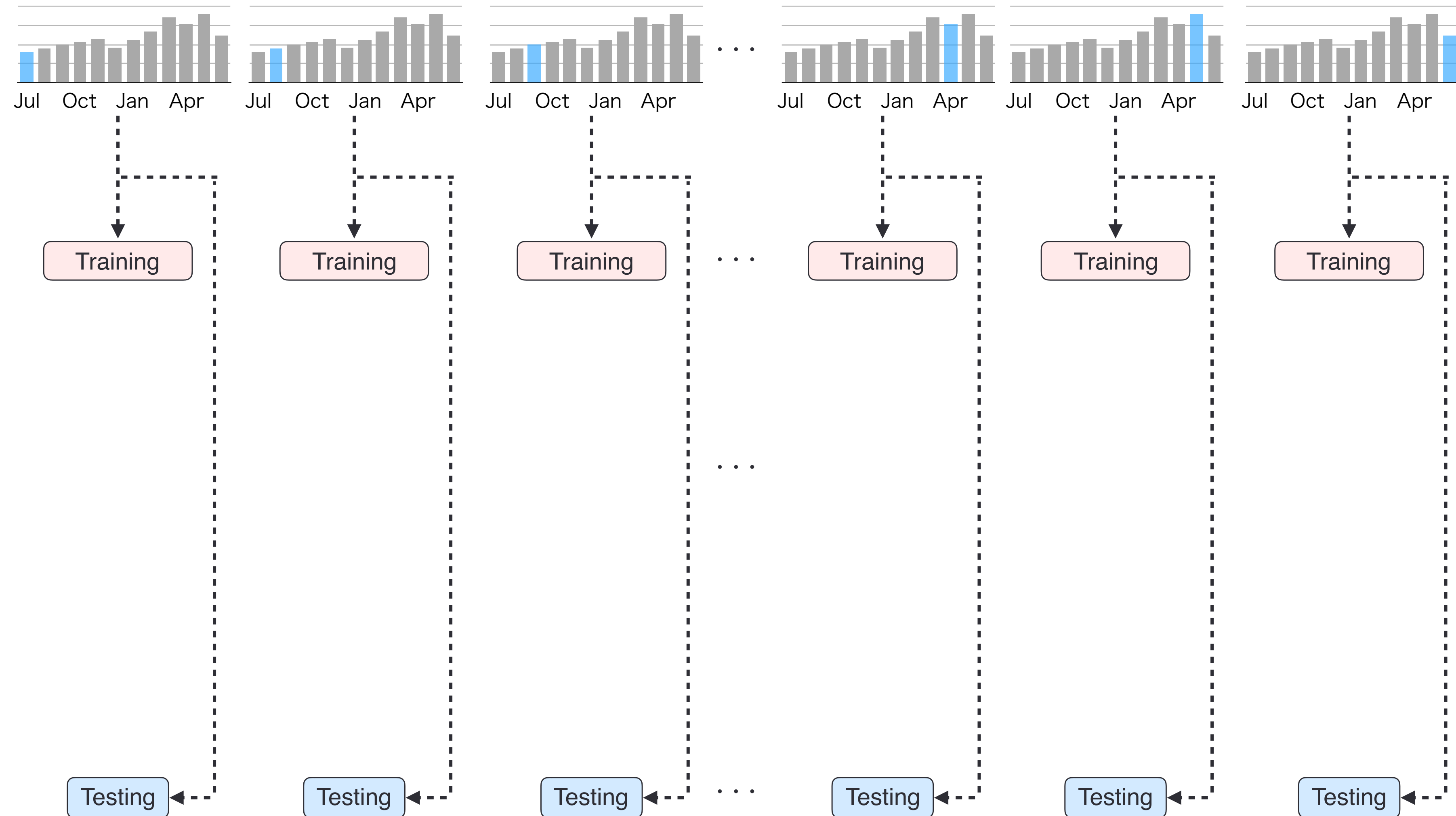
Yes, release aware models leverage the characteristic of the release process



RQ3: How fine-grained data should be used?

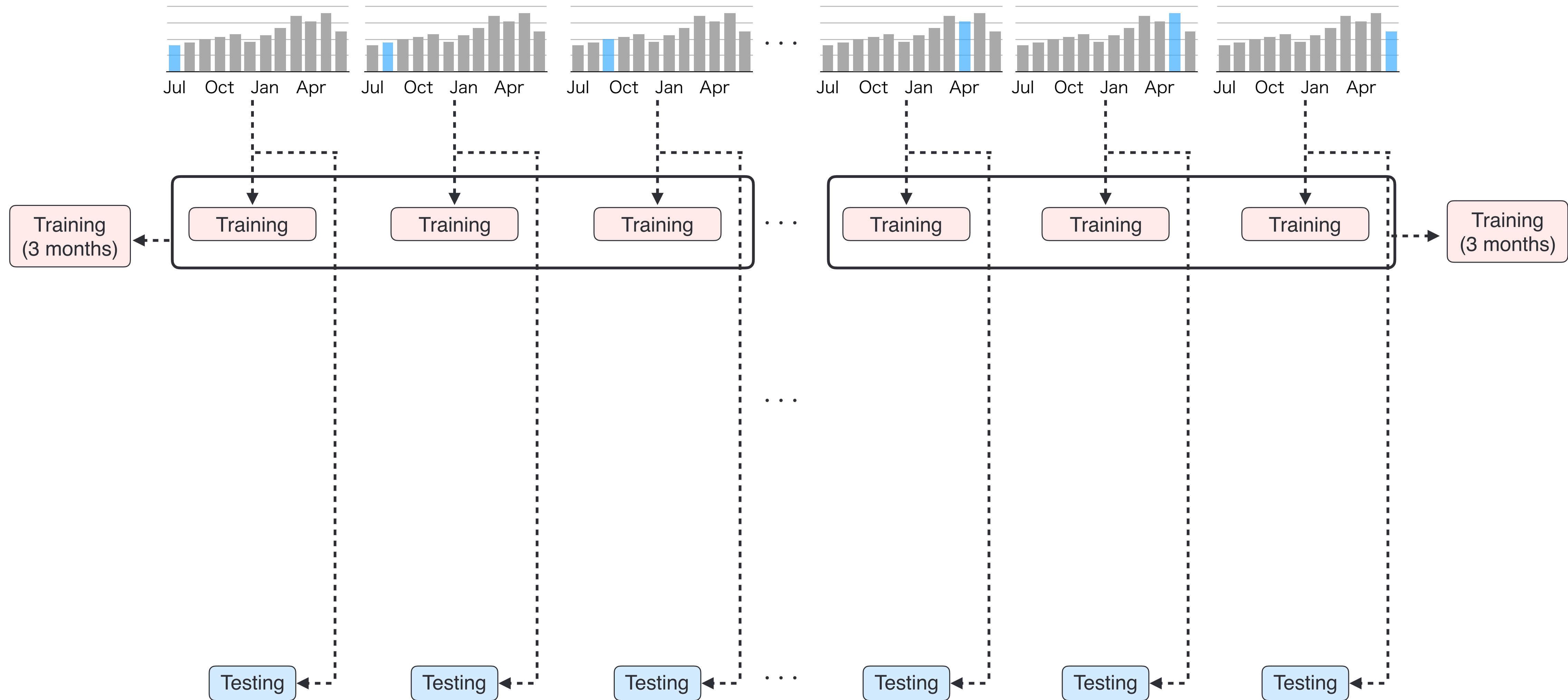


An evaluation method for 1/4/12-divided models



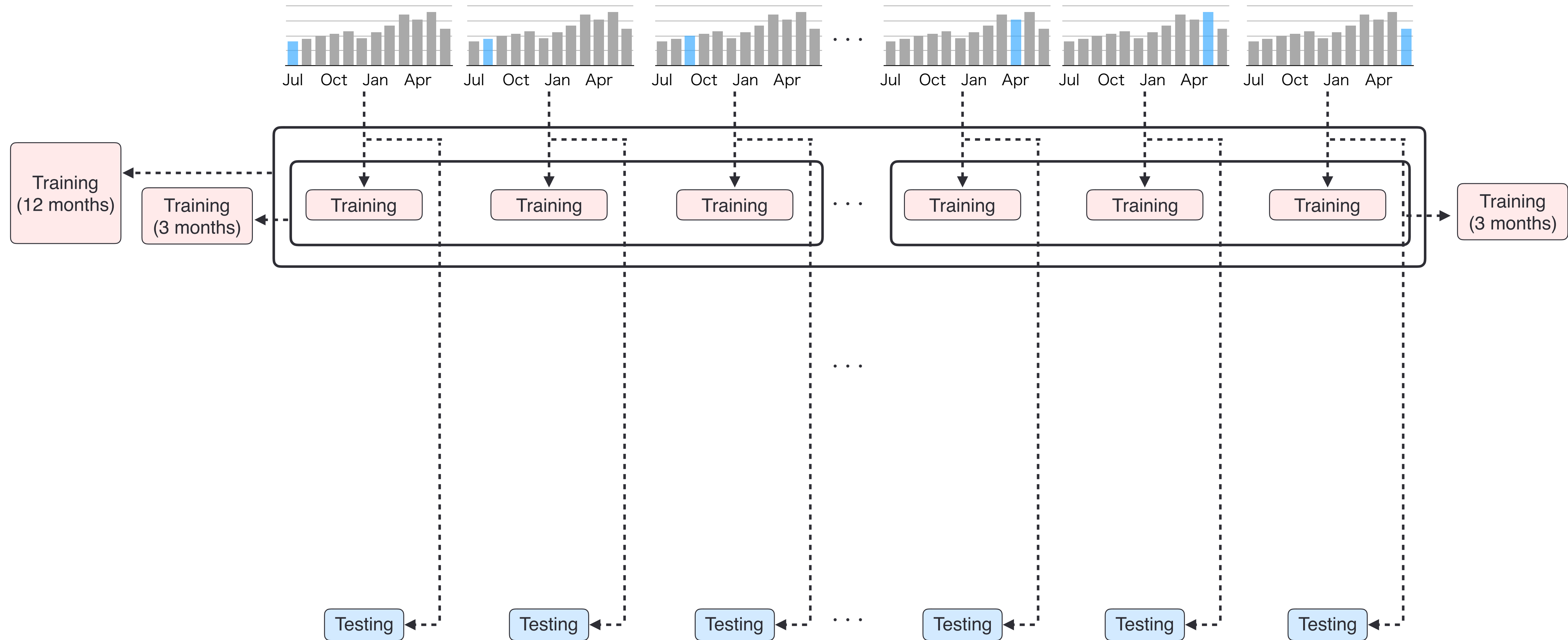


An evaluation method for 1/4/12-divided models



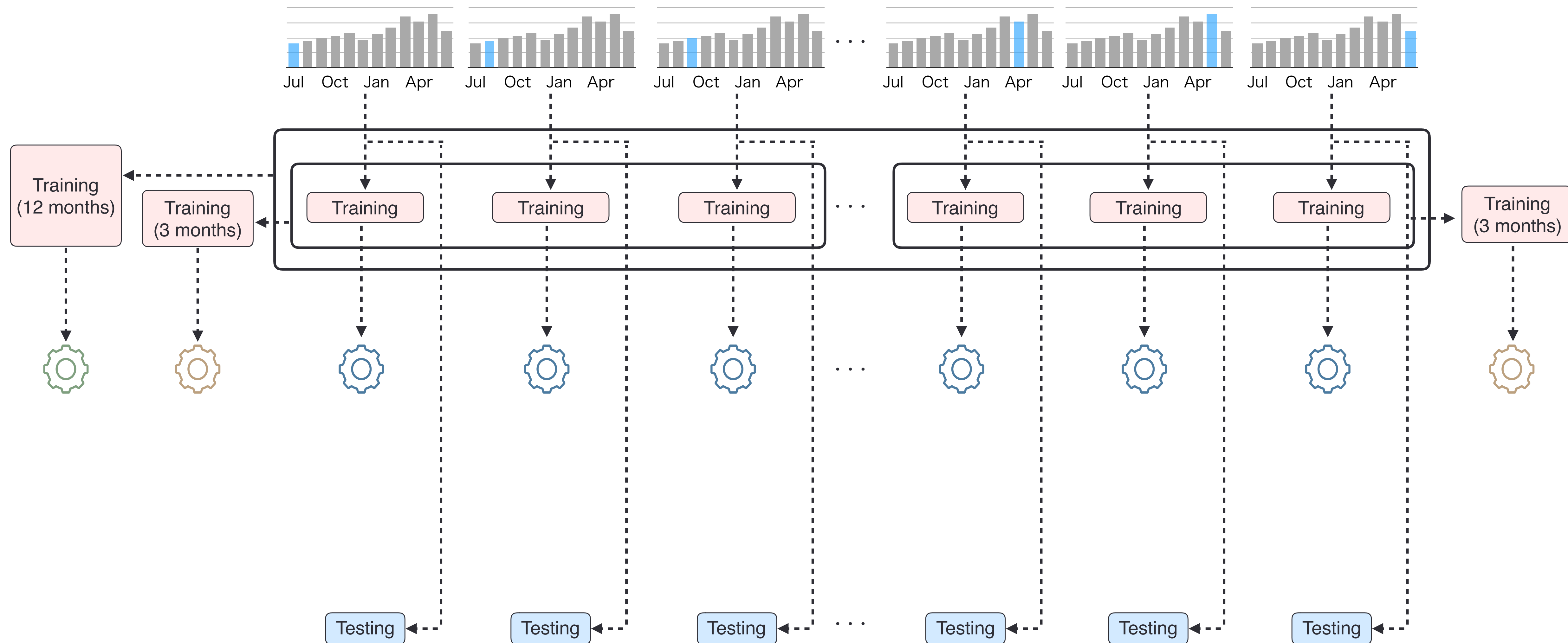


An evaluation method for 1/4/12-divided models



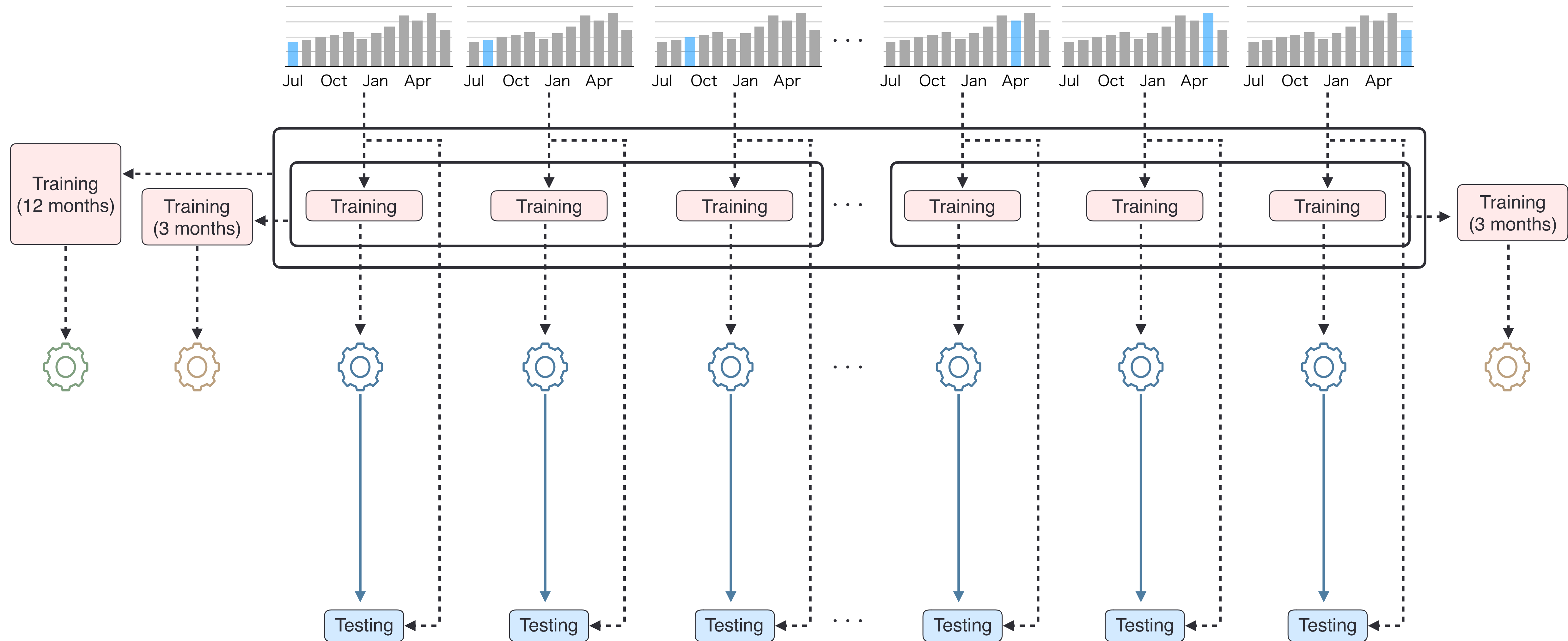


An evaluation method for 1/4/12-divided models



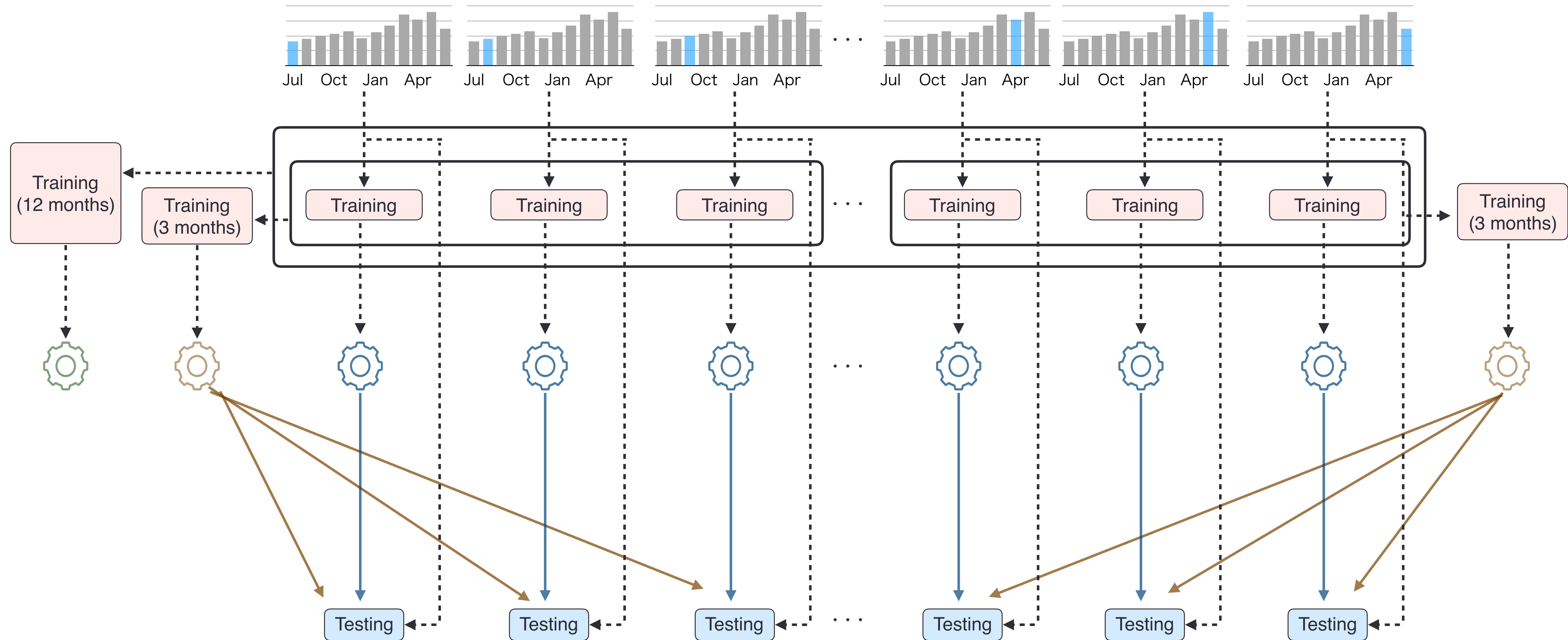


An evaluation method for 1/4/12-divided models



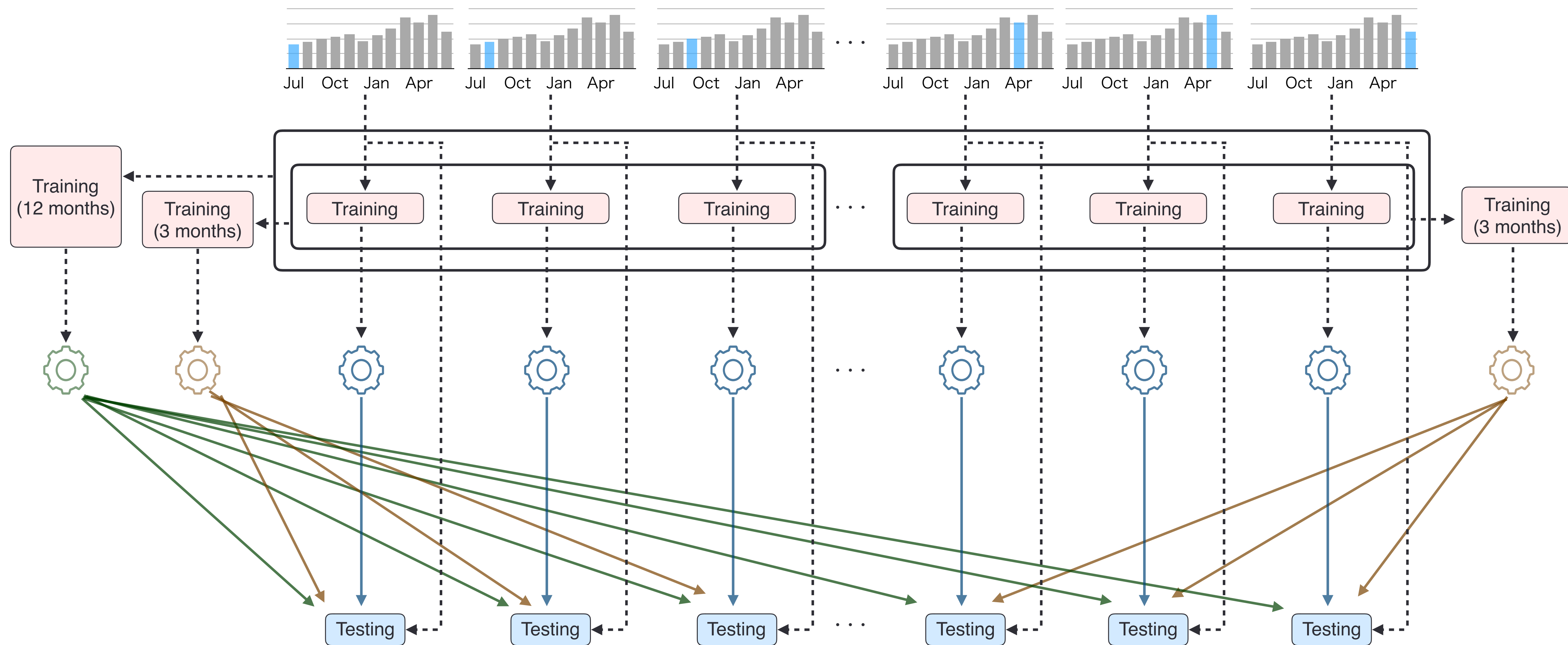


An evaluation method for 1/4/12-divided models



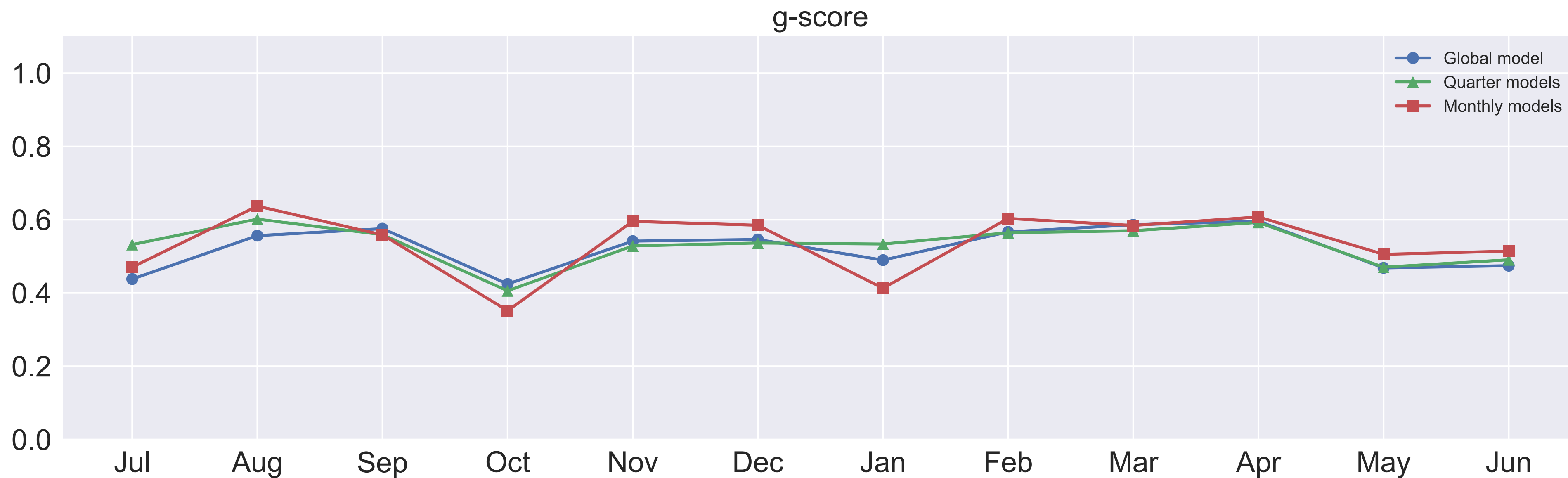
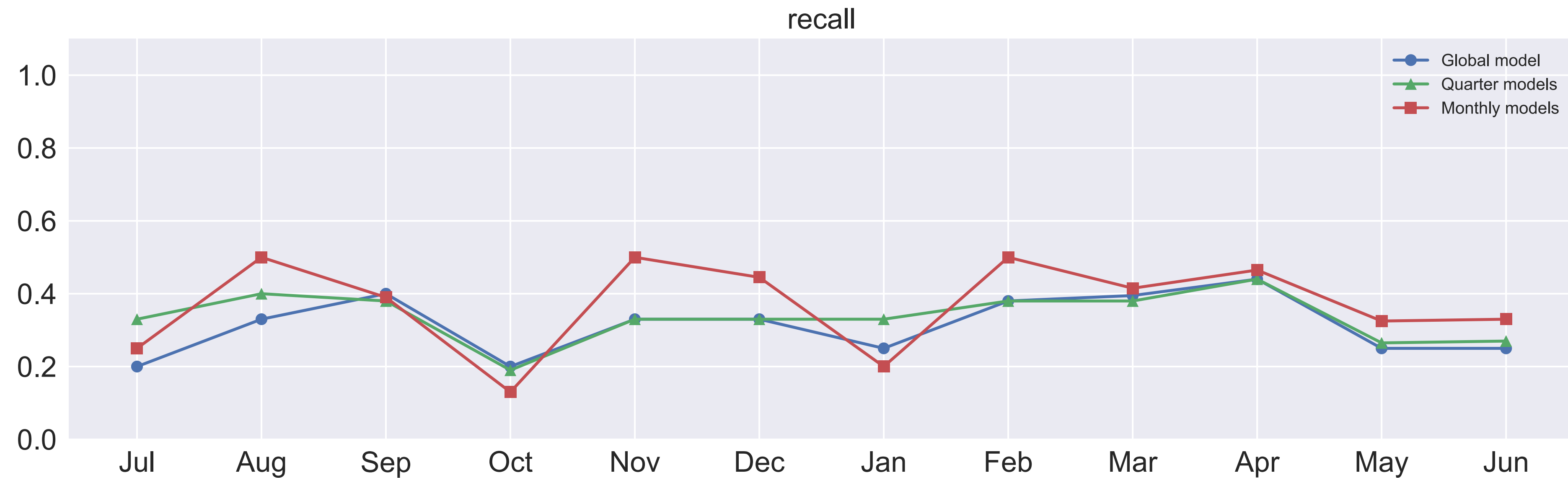


An evaluation method for 1/4/12-divided models





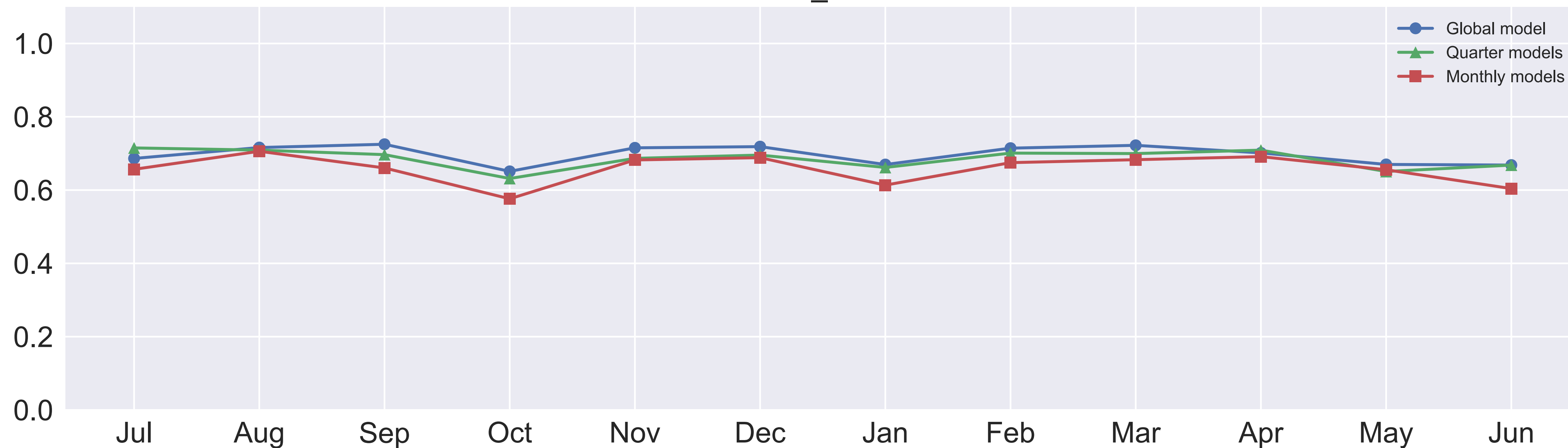
More fine-grained models tend to outperform less fine-grained models but be more unstable

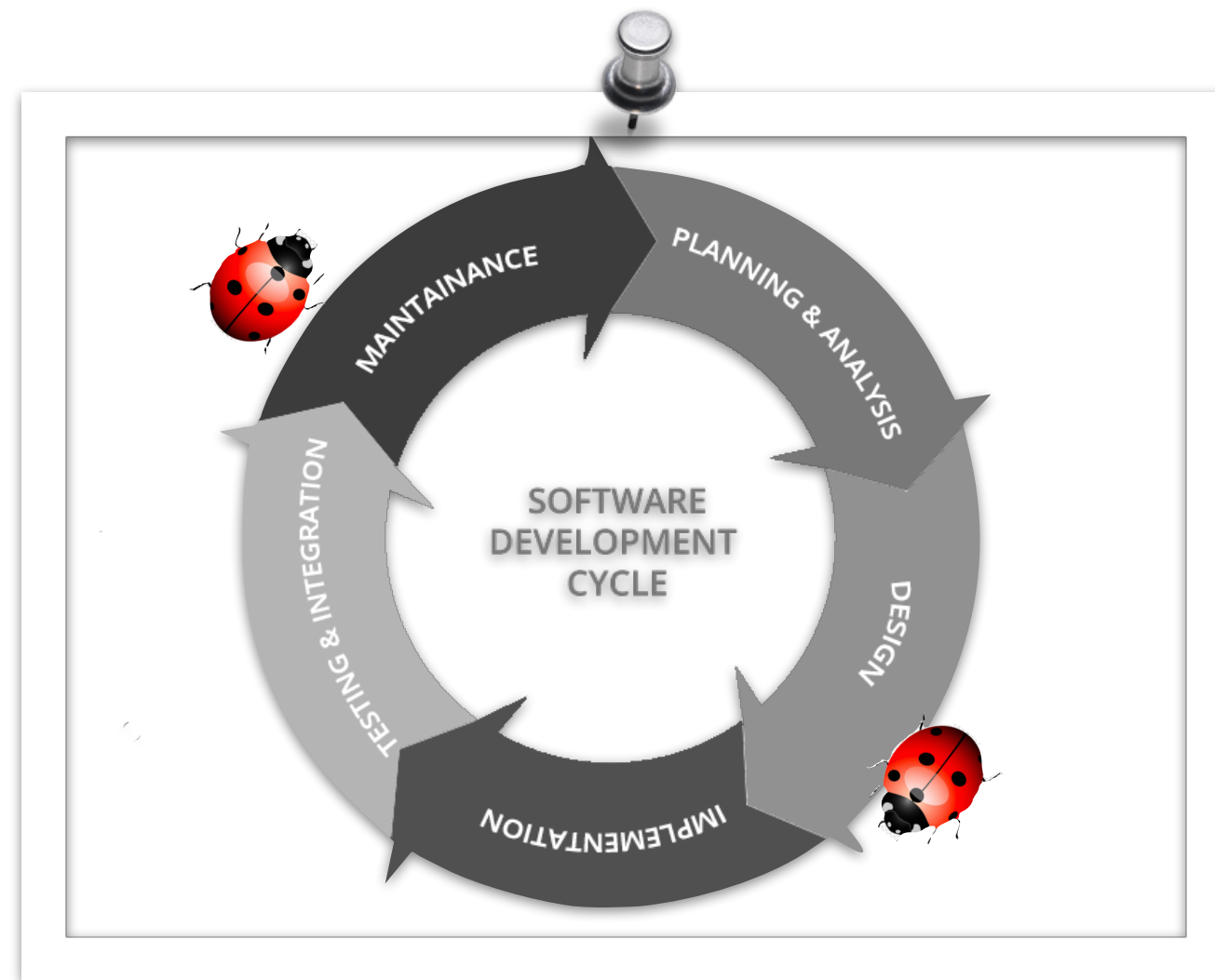




The ROC-AUC of monthly models are constantly lower than others

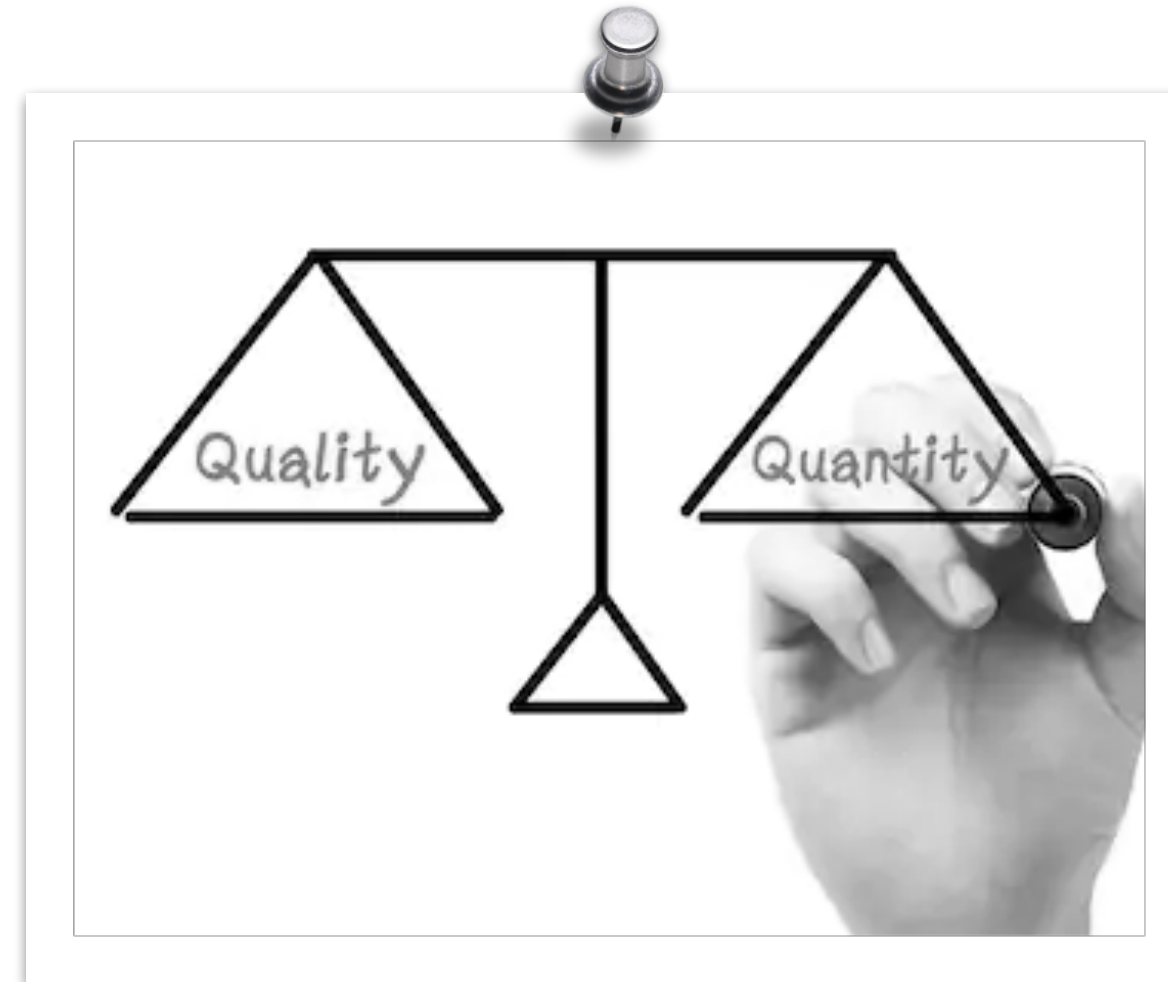
roc_auc





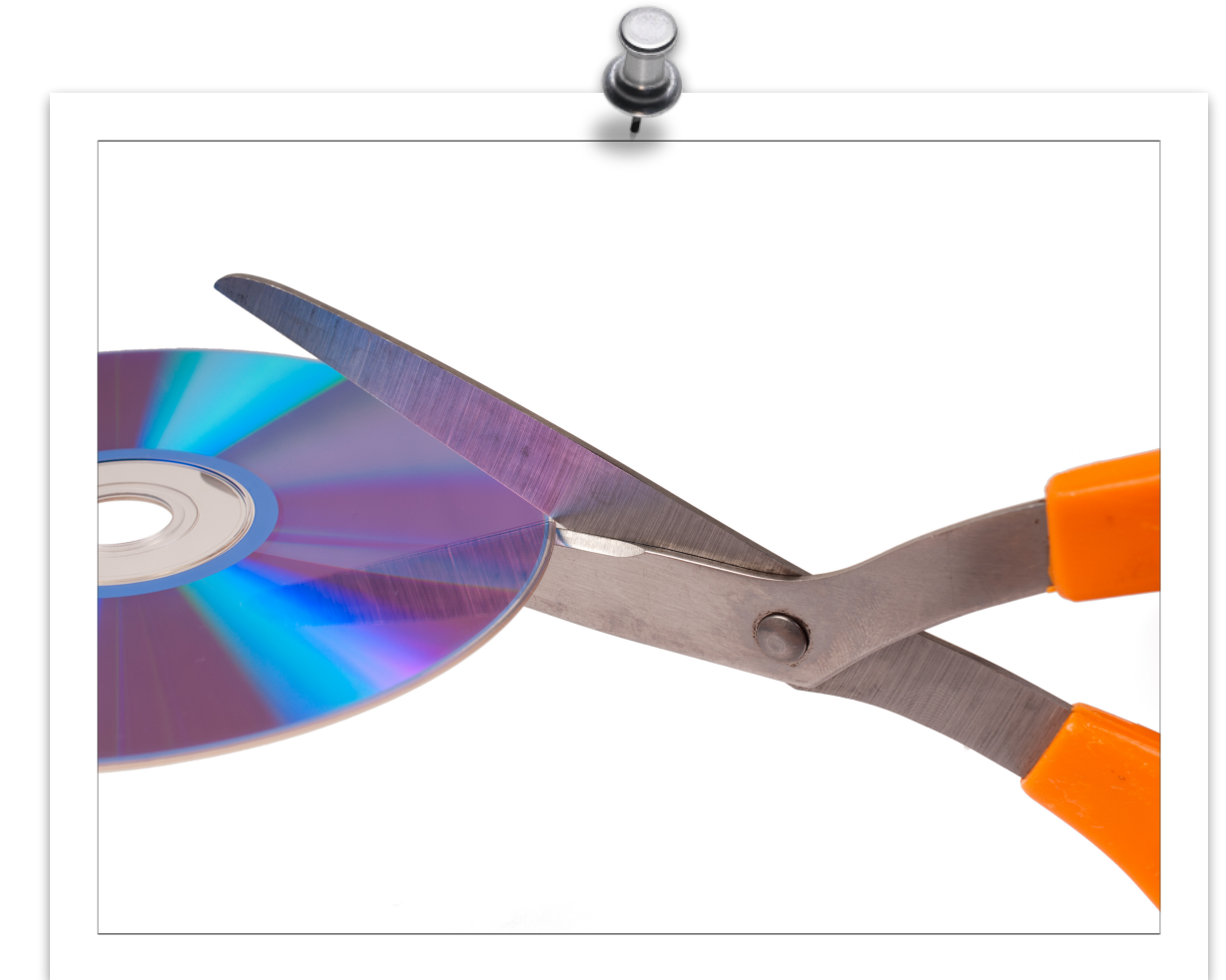
RQ1: Do bug-fixing activities vary across release cycle?

Yes, fixing activities are different across release cycle (especially, bug-fixing, priority change)



RQ2: Do Release-aware models for priority change outperform Global model?

Yes, release aware models leverage the characteristics of release process



RQ3: How fine-grained data should be used?

More fine-grained models tend to outperform less fine-grained models but be more unstable



Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca



Predicting Bug-Fixing Time: An Empirical Study of
Commercial Software Projects

Hongyu Zhang¹, Liang Gong¹, Steve Verstee²
¹University of Melbourne, Australia
h.zhang@unimelb.edu.au, l.gong@unimelb.edu.au, s.versteeg@unimelb.edu.au

bug report data
for 6 years

1 prediction model

2013 IEEE International Conference on Software Maintenance

DRONE: Predicting Priority of Reported Bugs by
Multi-Factor Analysis

Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore

Is using one prediction model enough to predict accurately?



Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca

Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects

Hongyu Zhang¹, Liang Gong¹, Steve Verstee²
¹CA Technologies
Melbourne, Australia
steve.verstee@ca.com

2013 IEEE International Conference on Software Maintenance

DRONE: Predicting Priority of Reported Bugs by Multi-Factor Analysis

Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore

bug report data
for 6 years

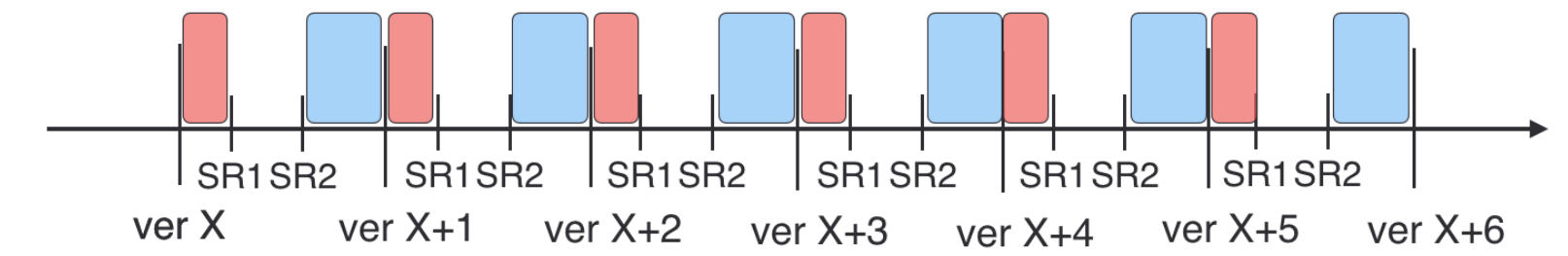
1 prediction model

Is using one prediction model enough to predict accurately?

Metrics might be affect by release cycles



Dataset



Our goal:
Understand the impact of the release cycles on bug fixing process

Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca

Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects

bug report data
for 6 years

1 prediction model

Is using one prediction model enough to predict accurately?

Hongyu Zhang¹, Liang Gong¹, Steve Verstee²
¹CA Technologies
Melbourne, Australia
steve.verstee@ca.com

2013 IEEE International Conference on Software Maintenance

DRONE: Predicting Priority of Reported Bugs by Multi-Factor Analysis

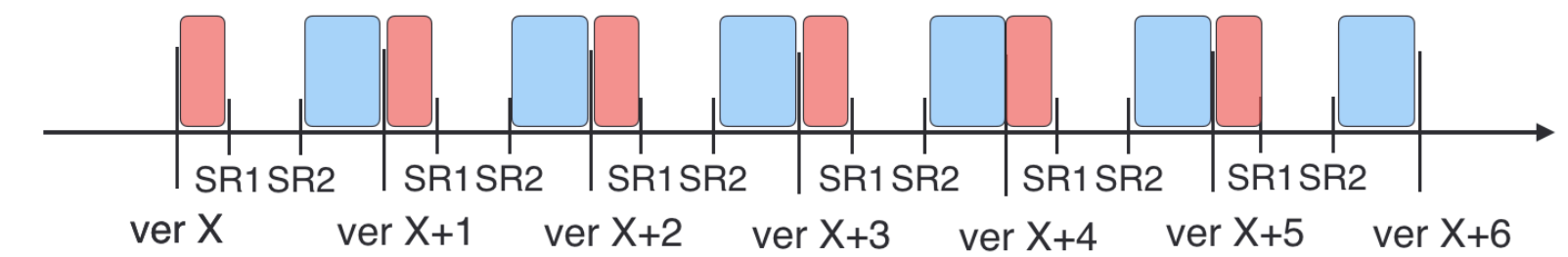
Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore



Metrics might be affect by release cycles

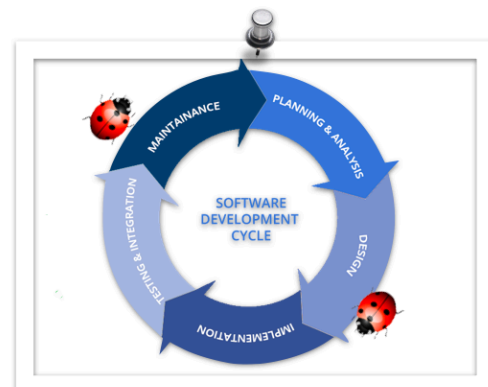


Dataset



6 years

Our goal:
Understand the impact of the release cycles on bug fixing process



RQ1: Do bug-fixing activities vary across release cycle?

Yes, fixing activities are effected by release cycle (especially, bug reporting, bug-fixing, priority change).



RQ2: Do Release-aware models for priority change outperform Global model?



RQ3: How fine-grained data should be used?

Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca

Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects

bug report data
for 6 years

1 prediction model

Is using one prediction model enough to predict accurately?

Hongyu Zhang¹, Liang Gong¹, Steve Versteeg²
¹CA Technologies
Melbourne, Australia
ctava.versteeg@ca.com

2013 IEEE International Conference on Software Maintenance

DRONE: Predicting Priority of Reported Bugs by Multi-Factor Analysis

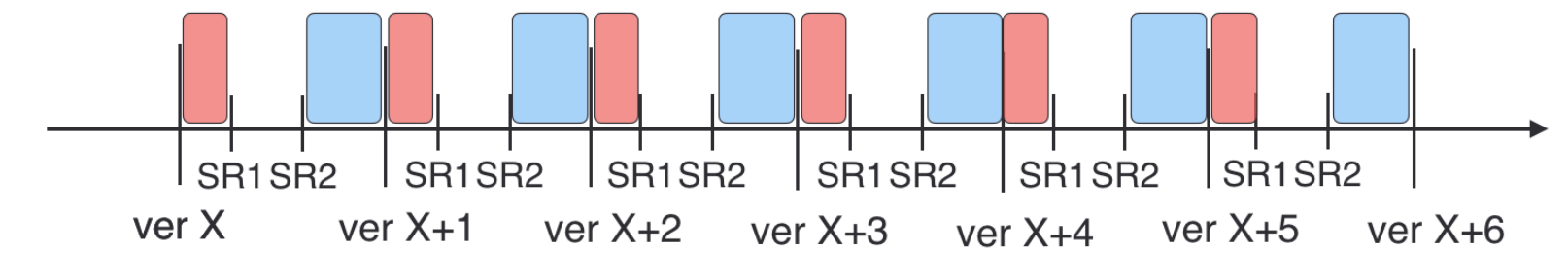
Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore



Metrics might be affect by release cycles

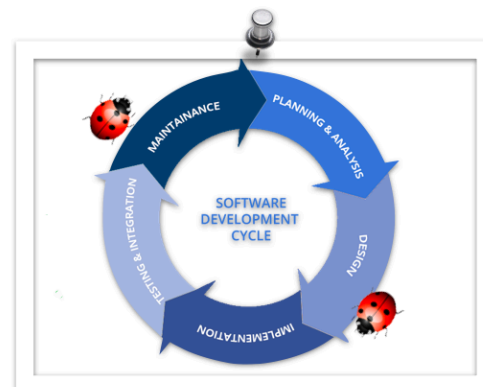


Dataset



6 years

Our goal:
Understand the impact of the release cycles on bug fixing process



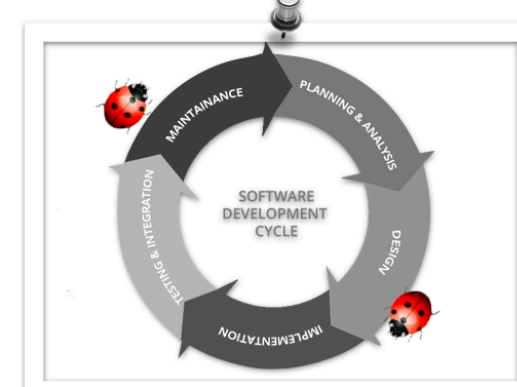
RQ1: Do bug-fixing activities vary across within release cycle?

Yes, fixing activities are effected by release cycle (especially, bug reporting, bug-fixing, priority change).



RQ2: Do Release-aware models for priority change outperform Global model?

RQ3: data



RQ1: Do bug-fixing activities vary across within release cycle?

Yes, fixing activities are e release cycle (especially, bu bug-fixing, priority chang



RQ2: Do Release-aware models for priority change outperform Global model?

Yes, release aware models leverage the characteristic of the release process



RQ3: How fine-grained data should be used?

Predicting Faults Using the Complexity of Code Changes

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca

Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects

bug report data
for 6 years

1 prediction model

Is using one prediction model enough to predict accurately?

Hongyu Zhang¹, Liang Gong¹, Steve Versteeg²
¹CA Technologies
Melbourne, Australia
steve.versteeg@ca.com

2013 IEEE International Conference on Software Maintenance

DRONE: Predicting Priority of Reported Bugs by Multi-Factor Analysis

Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore



Metrics might be affect by release cycles

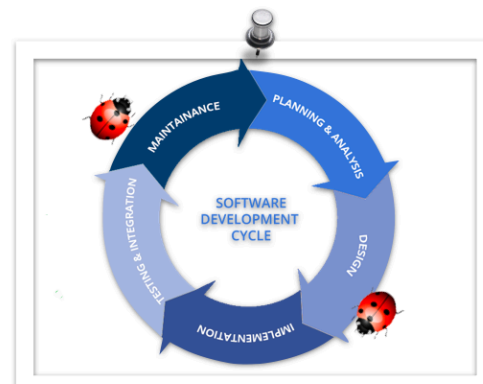


Dataset



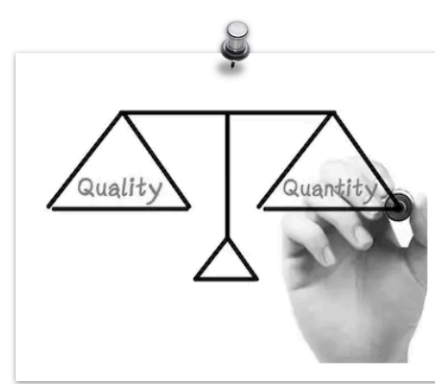
6 years

Our goal:
Understand the impact of the release cycles on bug fixing process



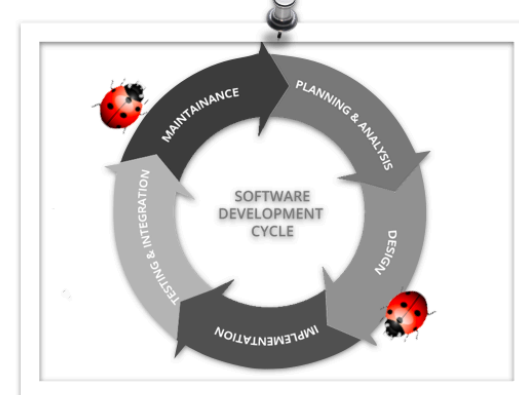
RQ1: Do bug-fixing activities vary across within release cycle?

Yes, fixing activities are effected by release cycle (especially, bug reporting, bug-fixing, priority change).



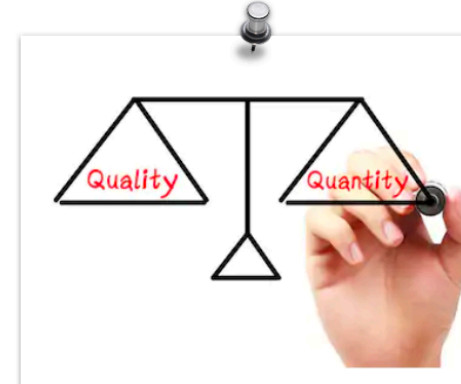
RQ2: Do Release-aware models for priority change outperform Global model?

RQ3: data



RQ1: Do bug-fixing activities vary across within release cycle?

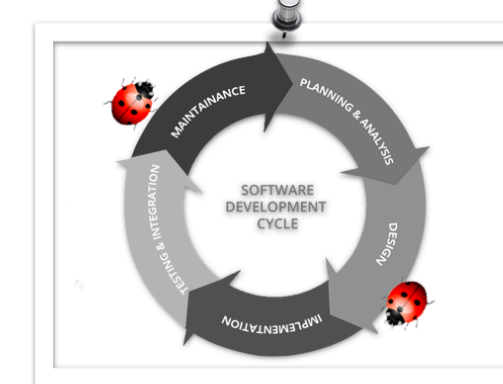
Yes, fixing activities are e release cycle (especially, bu bug-fixing, priority chang



RQ2: Do Release-aware models for priority change outperform Global model?

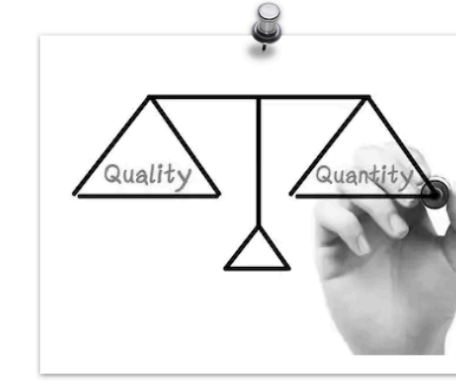
Yes, release aware models leverage the characteristic of the release process

RQ3: H data sh



RQ1: Do bug-fixing activities vary across within release cycle?

Yes, fixing activities are e release cycle (especially, bu bug-fixing, priority chang



RQ2: Do Release-aware models for priority change outperform Global model?

Yes, release awa leverage the charac release pro



RQ3: How fine-grained data should be used?

More fine-grained models tend to outperform less fine-grained models but be more unstable

