Understanding Software Delivery Delay

Daniel Alencar da Costa

Supervisor: Uirá Kulesza

Co-supervisor: Ahmed E. Hassan

Agenda

Agenda

Agenda

Agenda

Agenda

Introduction

An software issue can represent…


BUG

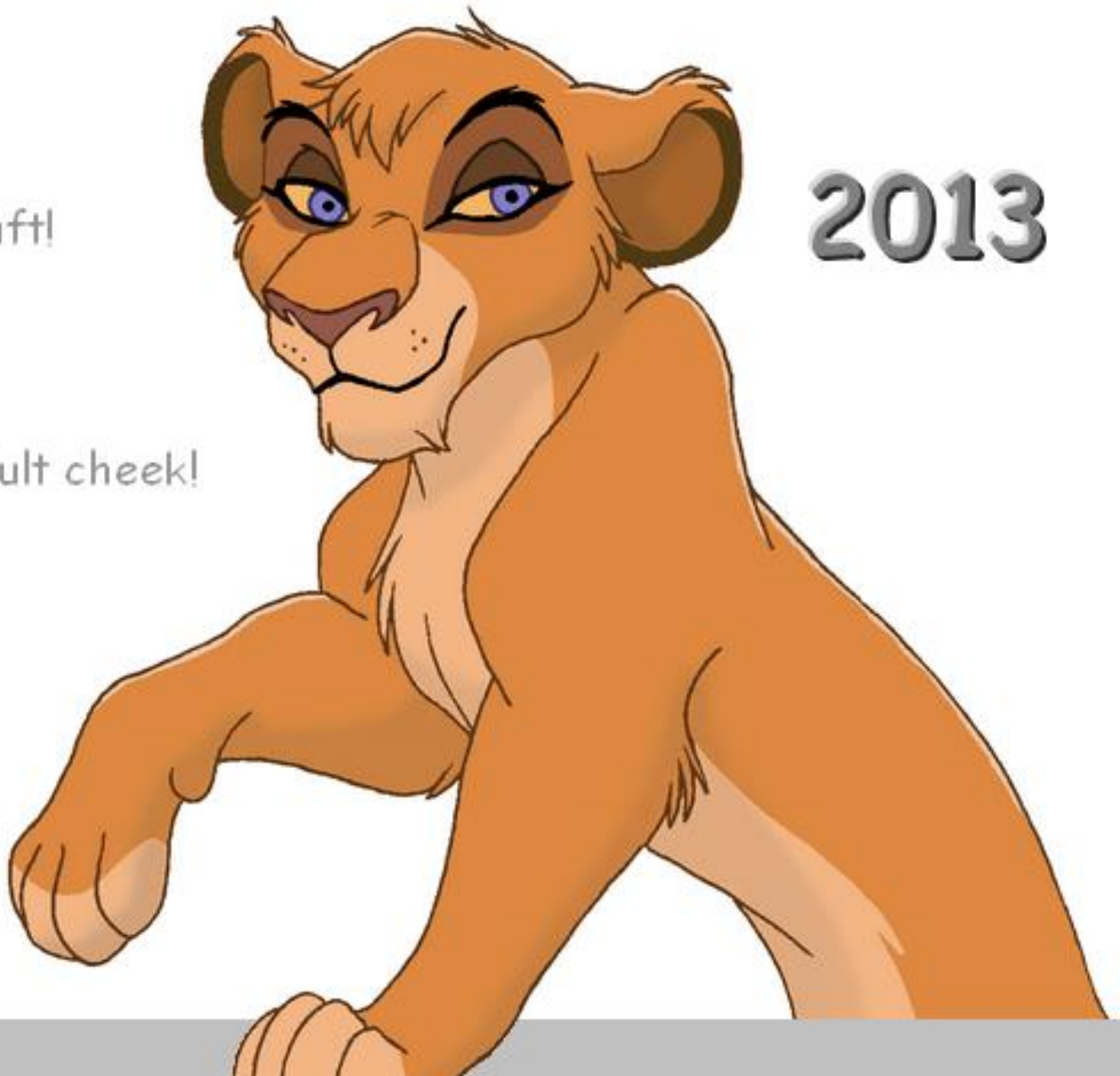
1999

No head tuft!

Cub cheek!

Adult cheek!

2013

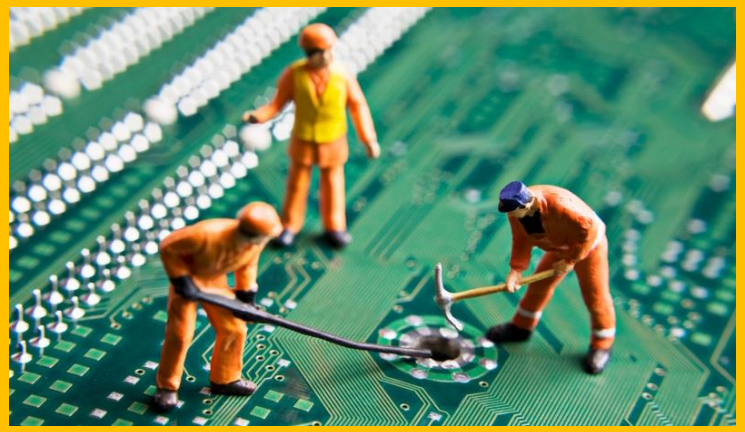


An issue can represent a bug, an enhancement, or a new feature

After an issue is addressed it may still suffer delay to be delivered

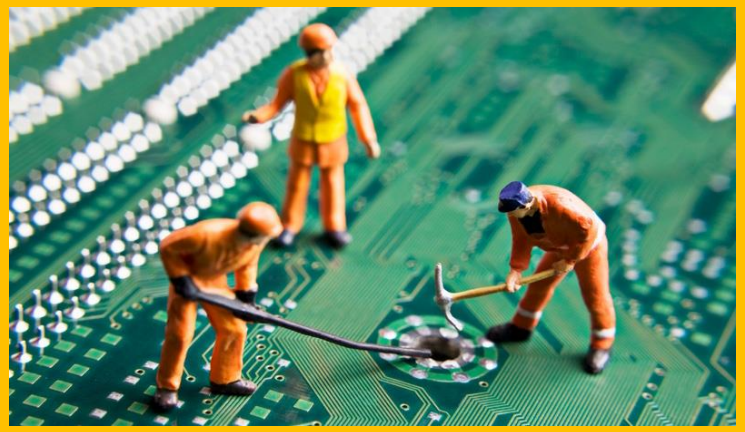
There is a lack of empirical studies to understand delivery delay

There is a lack of empirical studies to understand delivery delay



Fixing

There is a lack of empirical studies to understand delivery delay



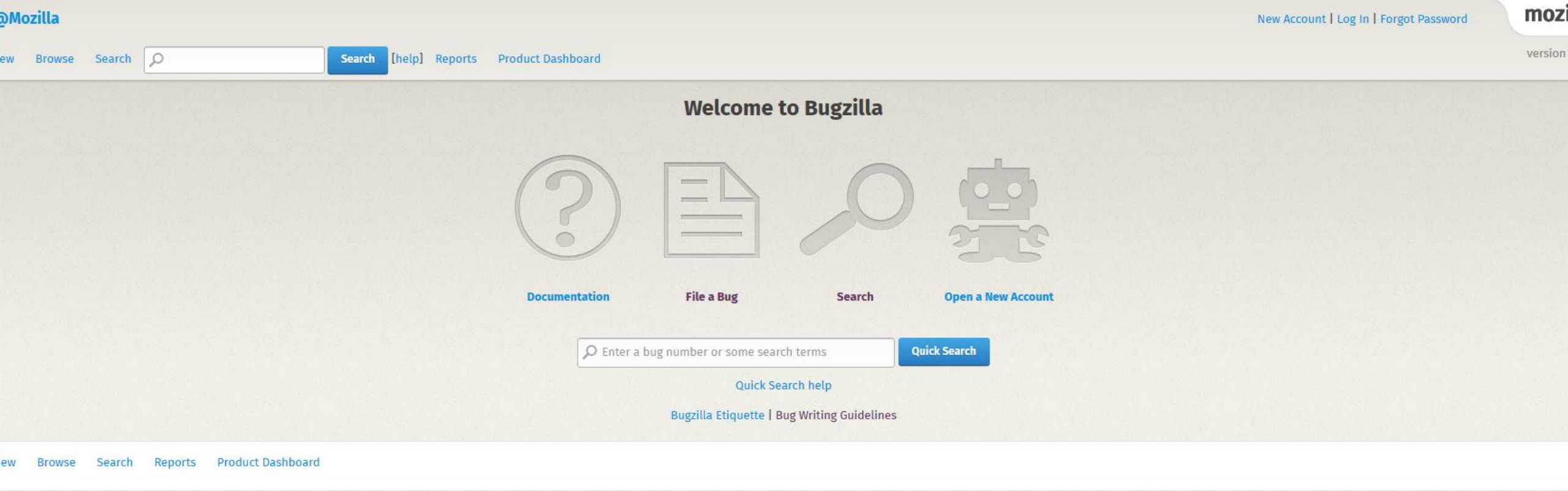
Fixing

Triaging

The understanding of which factors impact on delivery delay remains as a open challenge

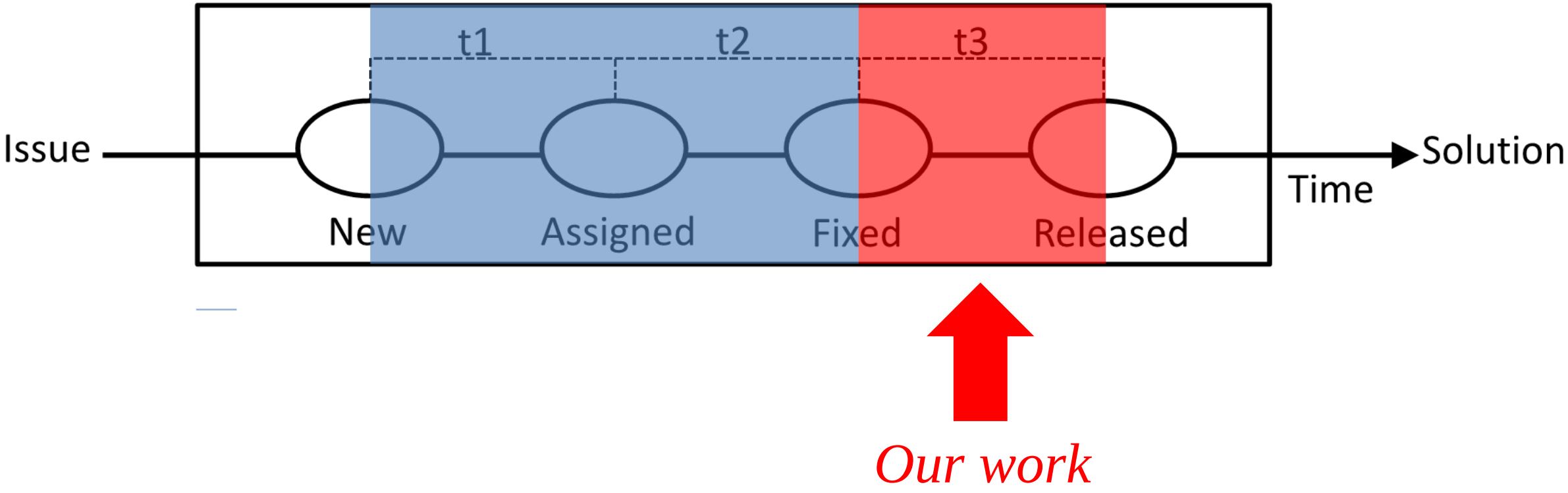
Once issues are addressed why do they still suffer delivery delay?

Background



*We use the term **issue** to broadly refer to bugs, enhancements, and new features reports*

We use the term delivery delay to refer to the time that it takes to deliver an addressed issue



A release cycle is the time that it takes to develop and ship a new version of the software

1.0

2.0

Time

Rapid releases have shorter release cycles

1.0

2.0

Time - traditional

1.0

2.0

Time - rapid

Empirical Studies

Understanding Delivery Delay (Study 1)

Earlier versions of this work are published in the proceedings of ICSME'14 and submitted to EMSE

We may think that to fix a bug is enough to satisfy an interested user

Prediction models and bug detection approaches were invented to help developers

Users may still wait for a long time to see the fixed bug reflected in the software system

We set out to empirically study the delivery delay of three open source systems

We set out to empirically study the delivery delay of three open source systems

14,530 issues

3,344 issues

3,121 issues

We set out to empirically study the delivery delay of three open source systems

14,530 issues

15 releases

3,344 issues

11 releases

3,121 issues

17 releases

We set out to empirically study the delivery delay of three open source systems

14,530 issues

15 releases

Rapid

3,344 issues

11 releases

Traditional

3,121 issues

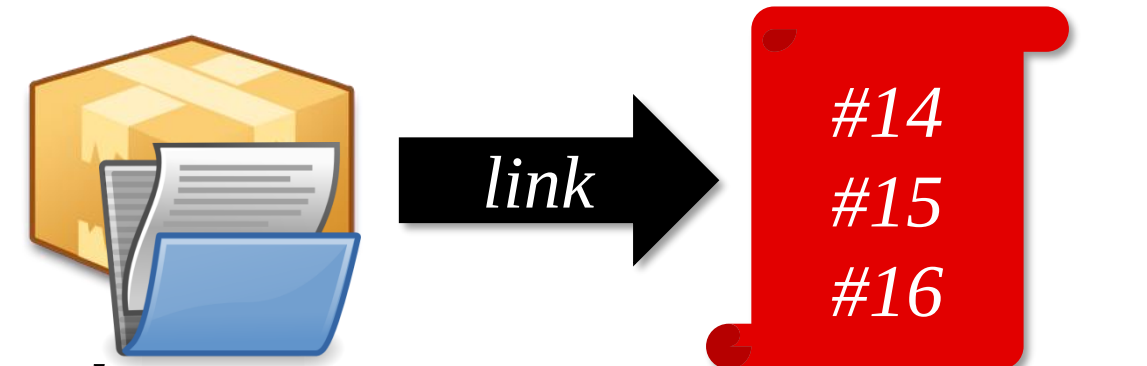
17 releases

Traditional

We collect data based on release notes information

Release Notes

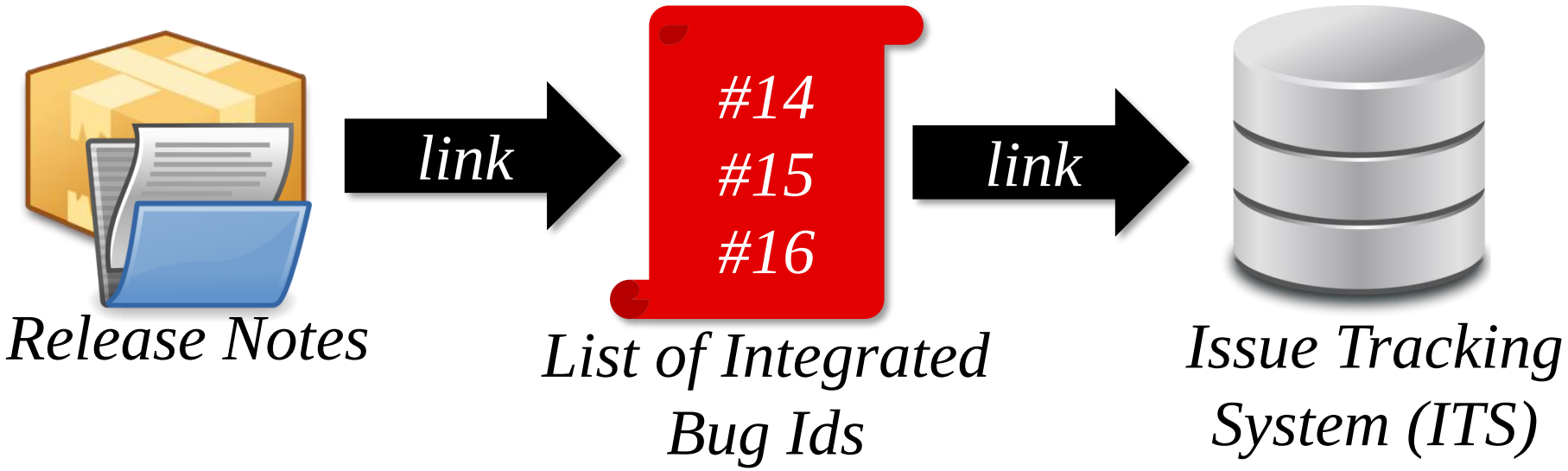
We collect data based on release notes information



Release Notes

List of Integrated Bug Ids

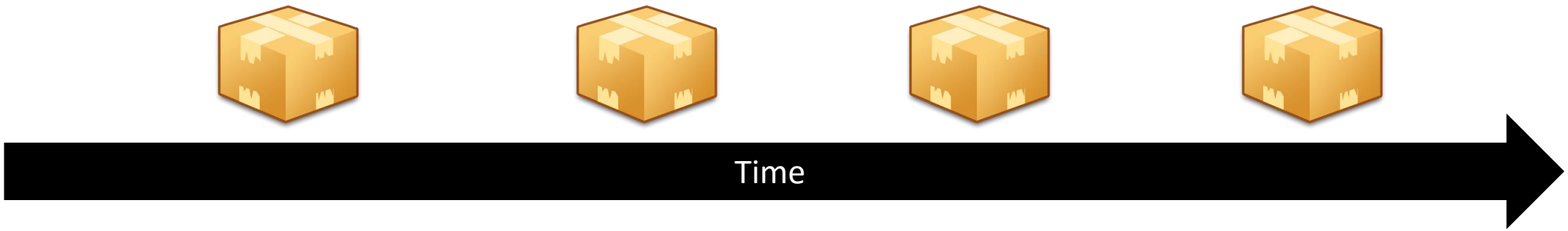
We collect data based on release notes information



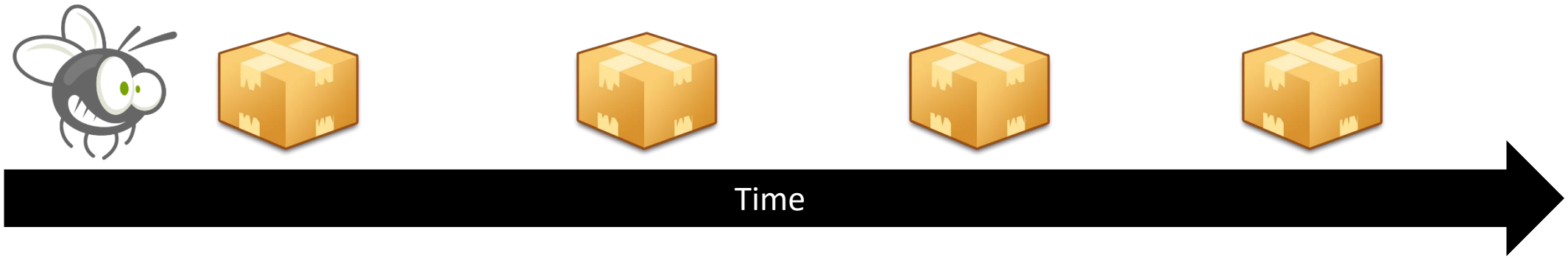
We measure delivery delay in two ways:
release delay *and* ***abnormal delay***

Release Delay

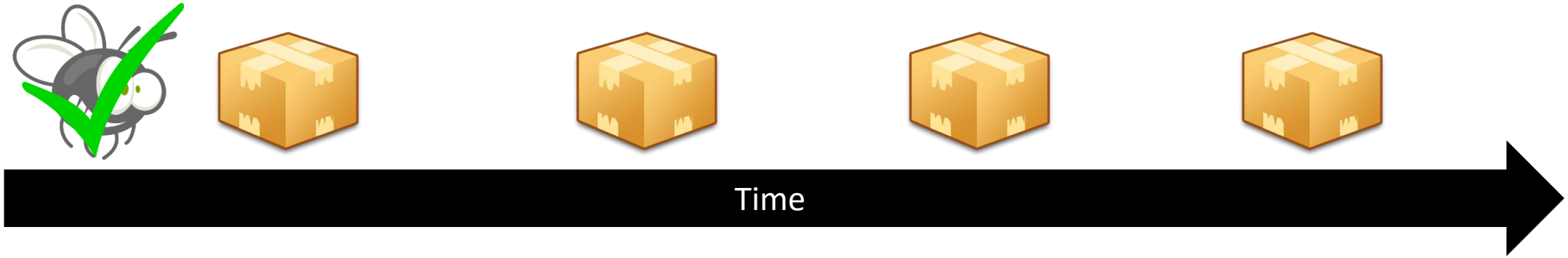
*We measure release delay by counting **how many releases** an addressed issue missed before being released*



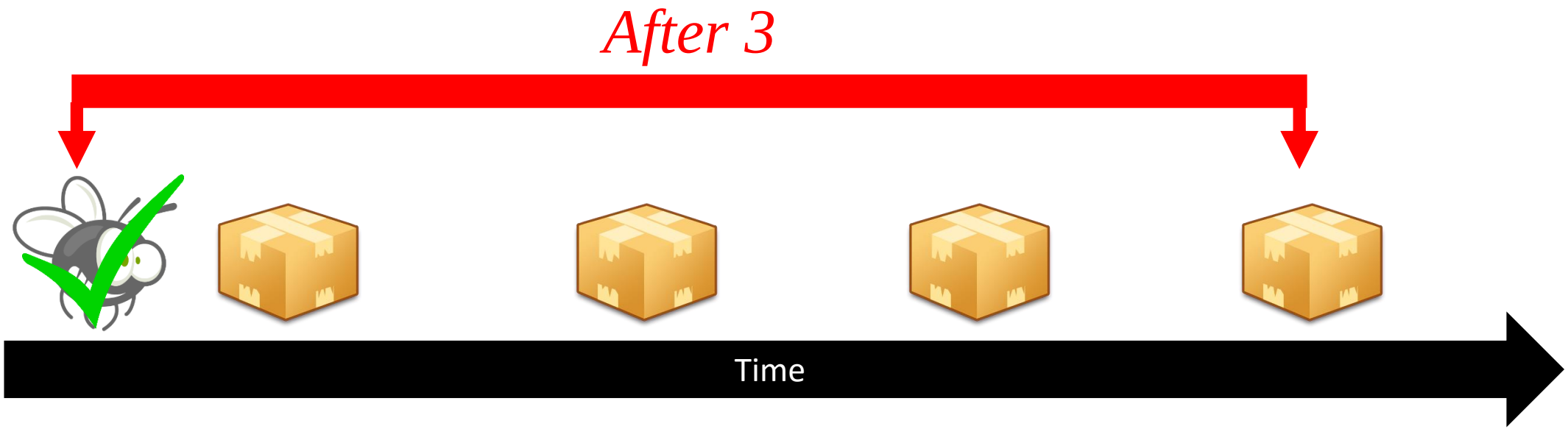
*We measure release delay by counting **how many releases** an addressed issue missed before being released*



*We measure release delay by counting **how many releases** an addressed issue missed before being released*



We measure release delay by counting ***how many releases*** *an addressed issue missed before being released*



Are issues often delayed after being addressed?

Are issues often delayed after being addressed?

Can we accurately explain how many releases an addressed issue will be delayed?

Are issues often delayed after being addressed?

Can we accurately explain how many releases an addressed issue will be delayed?

What are the most influential attributes for estimating release integration delay?

Are issues often delayed after being addressed?

Measuring Release Delay

Can we accurately explain how many releases an addressed issue will be delayed?

What are the most influential attributes for estimating release integration delay?

Are issues often delayed after being addressed?

Measuring Release Delay

Can we accurately explain how many releases an addressed issue will be delayed?

Building Explanatory Models

What are the most influential attributes for estimating release integration delay?

Are issues often delayed after being addressed?

Measuring Release Delay

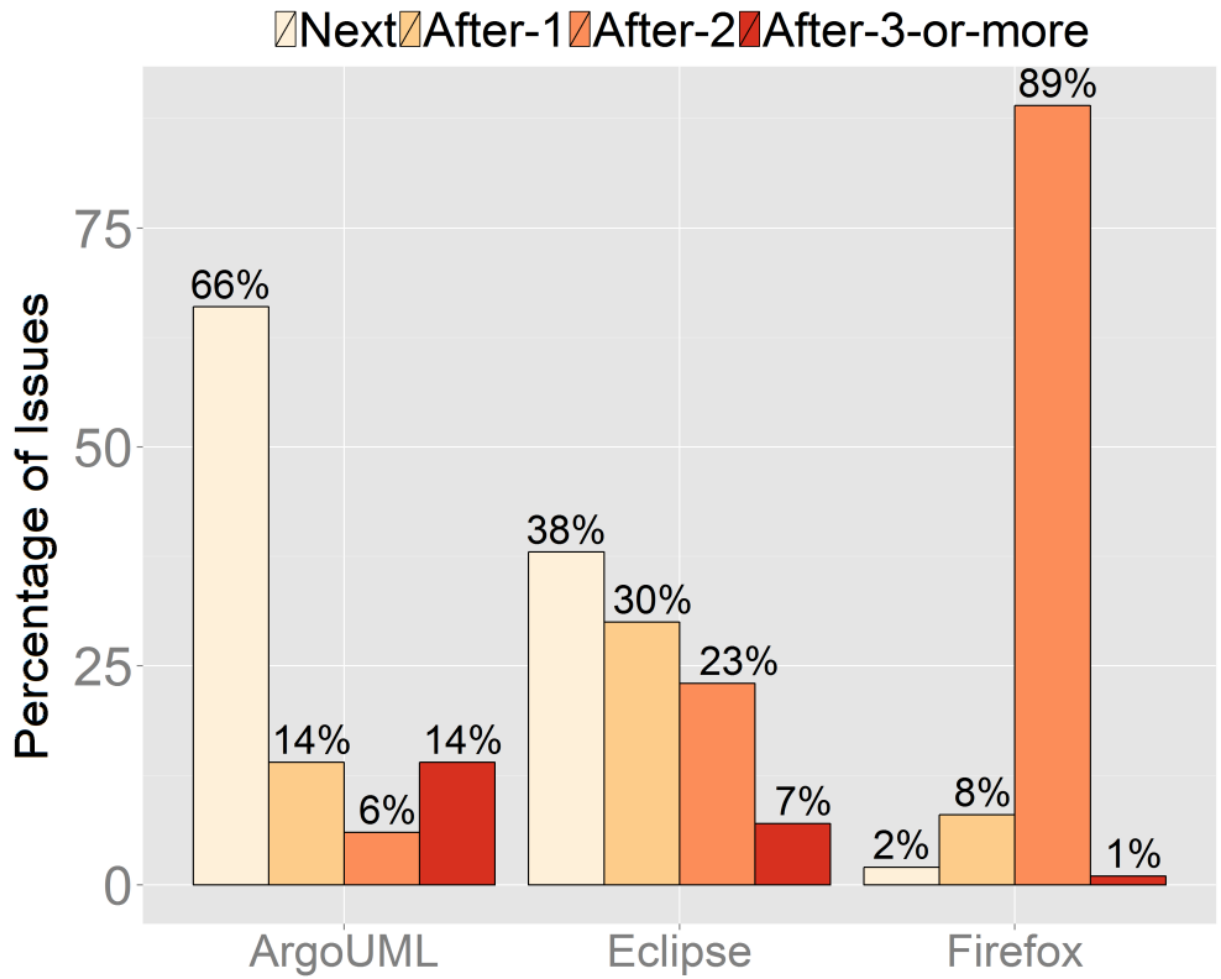
Can we accurately explain how many releases an addressed issue will be delayed?

Building Explanatory Models

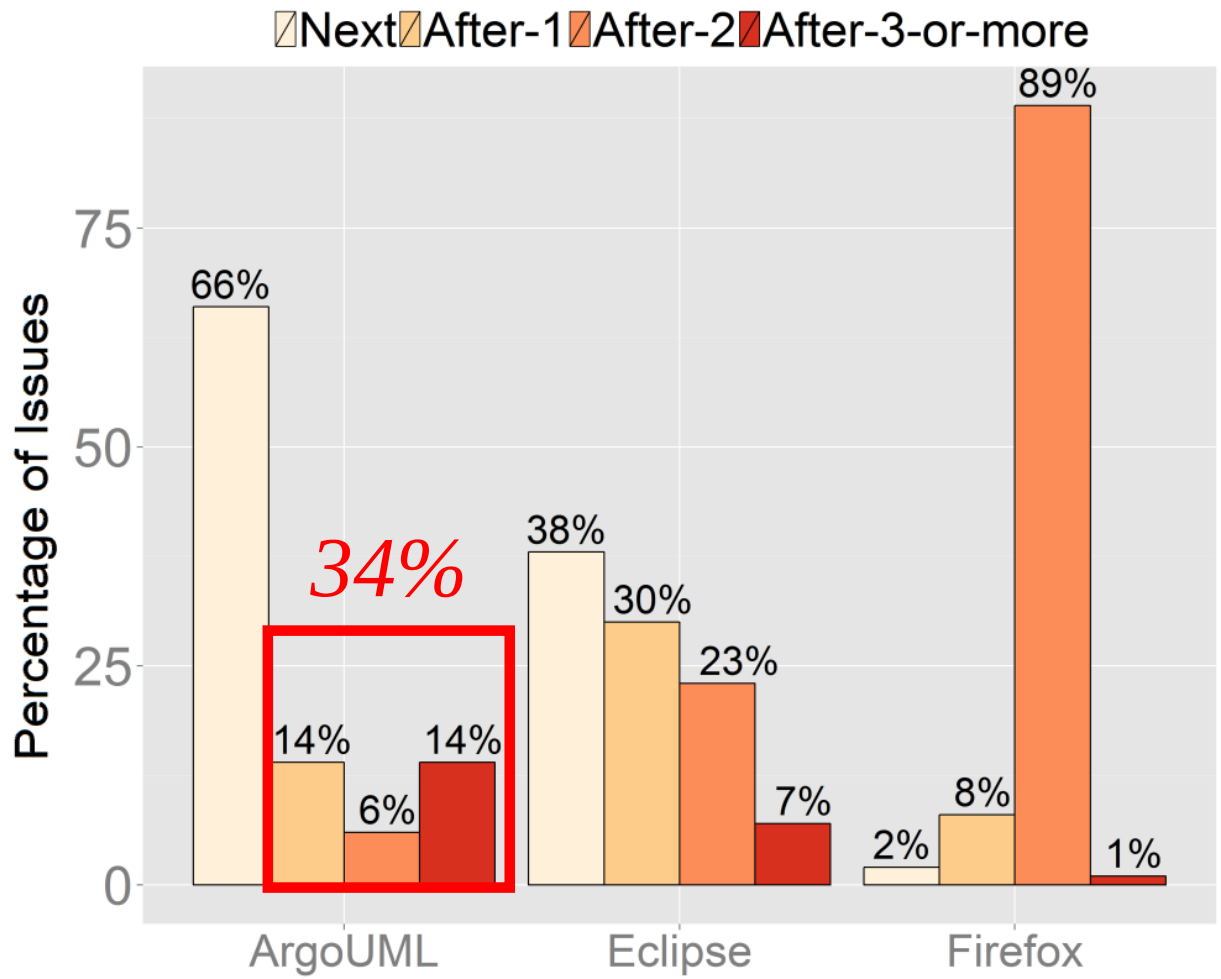
What are the most influential attributes for estimating release integration delay?

Identifying Top Influential Factors

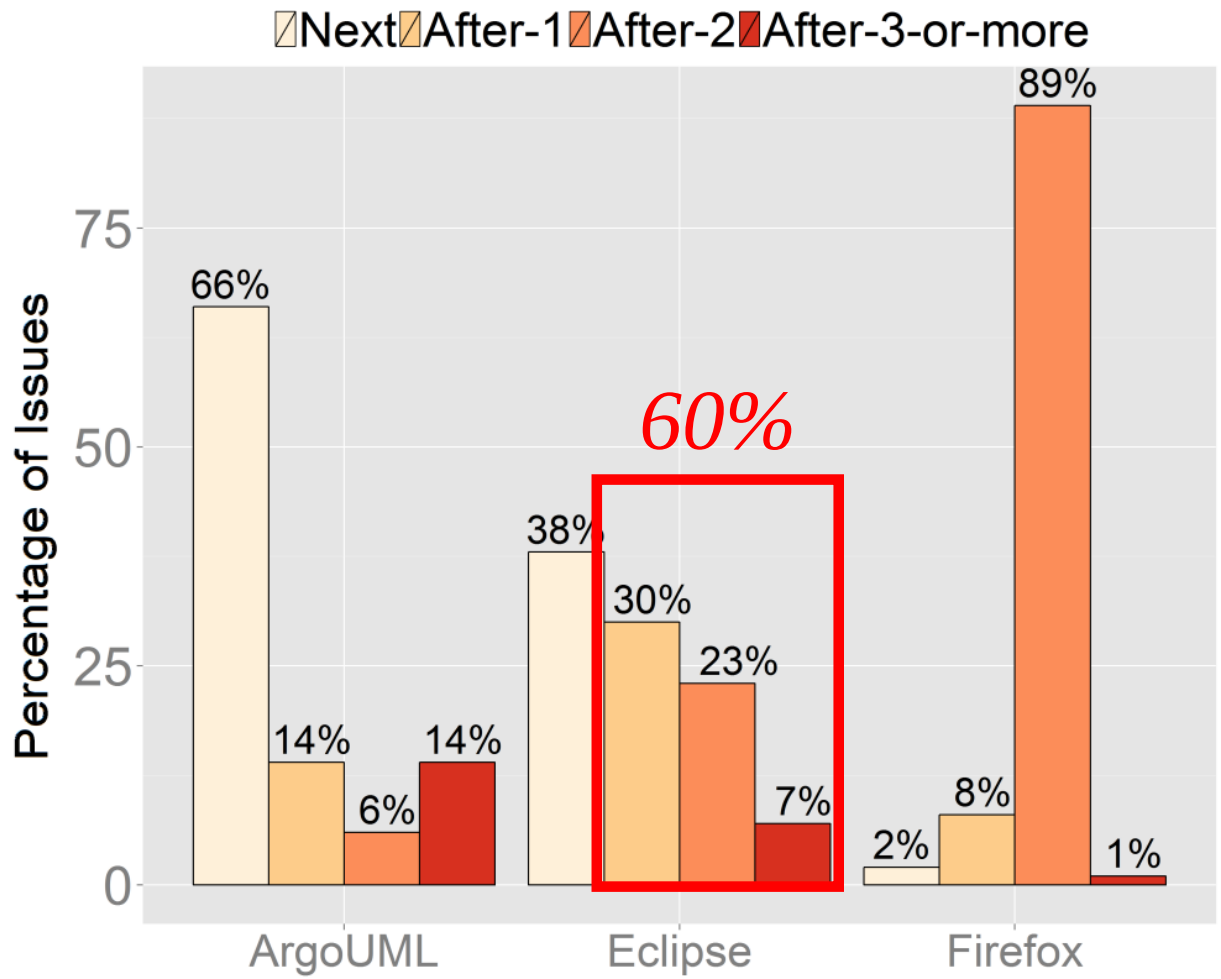
Issues are usually delayed in the rapid release cycle system



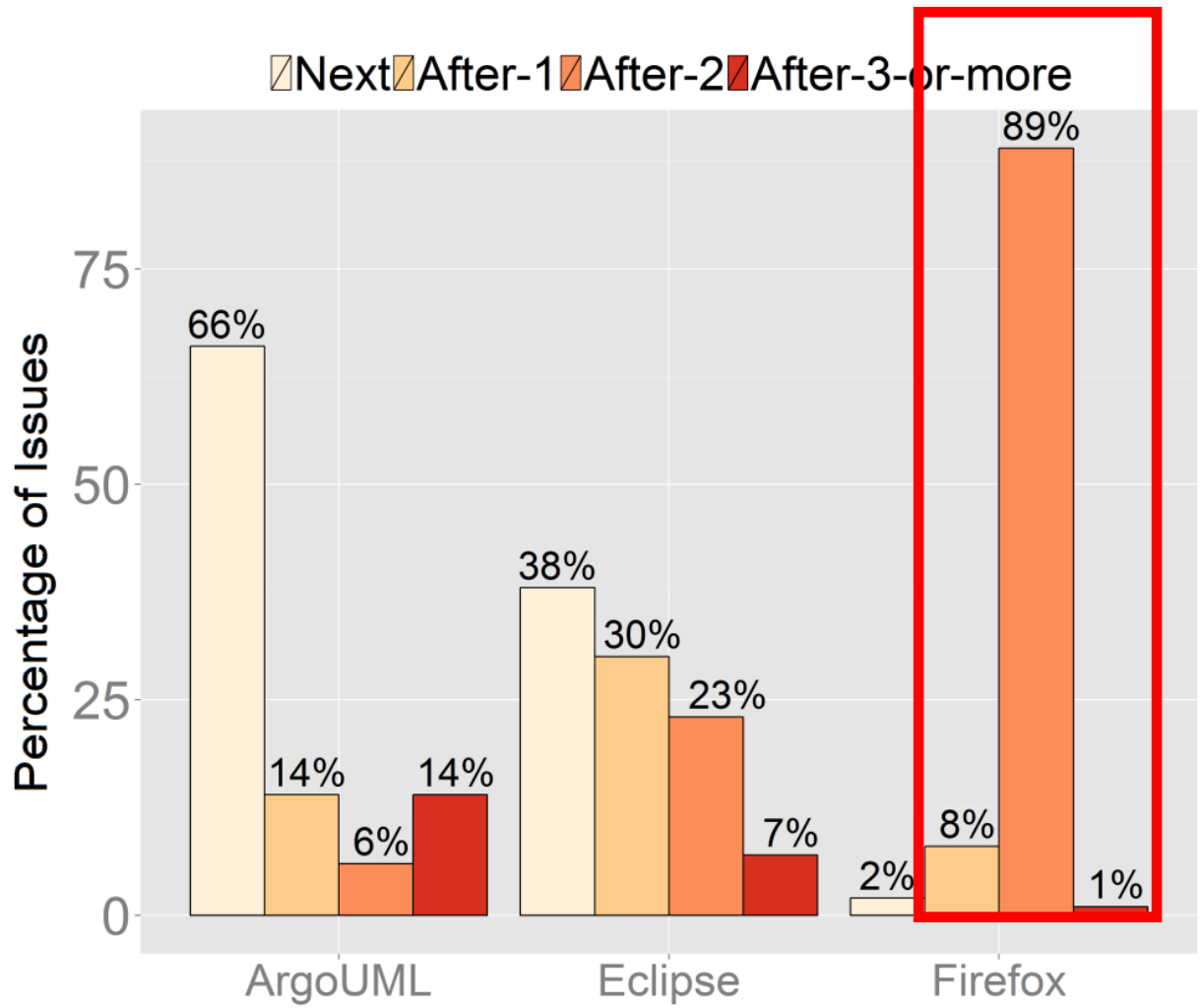
Issues are usually delayed in the rapid release cycle system



Issues are usually delayed in the rapid release cycle system



Issues are usually delayed in the rapid release cycle system



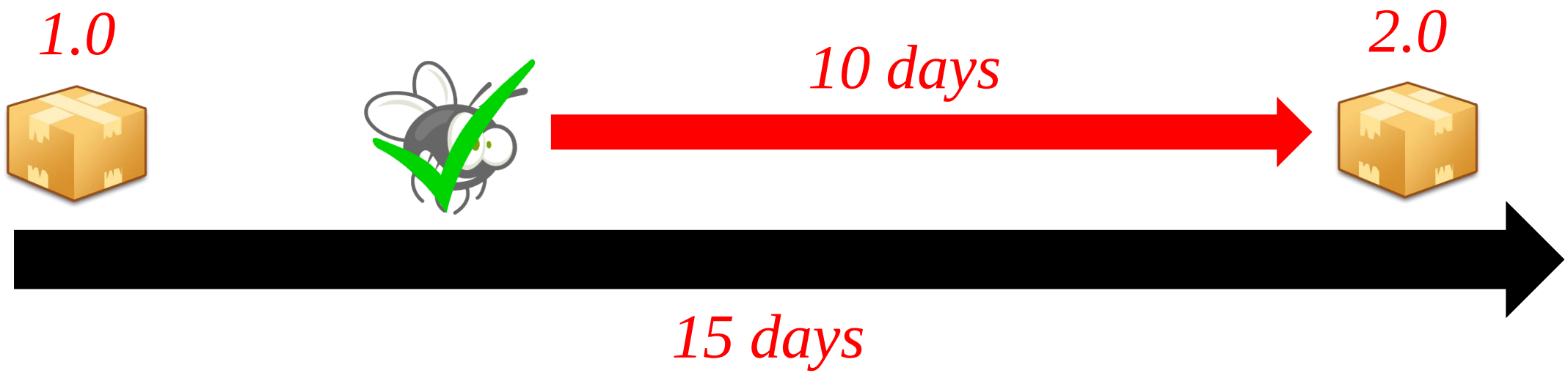
Issues are unlikely to be delayed solely because they were addressed close to an upcoming release

Issues are unlikely to be delayed solely because they were addressed close to an upcoming release

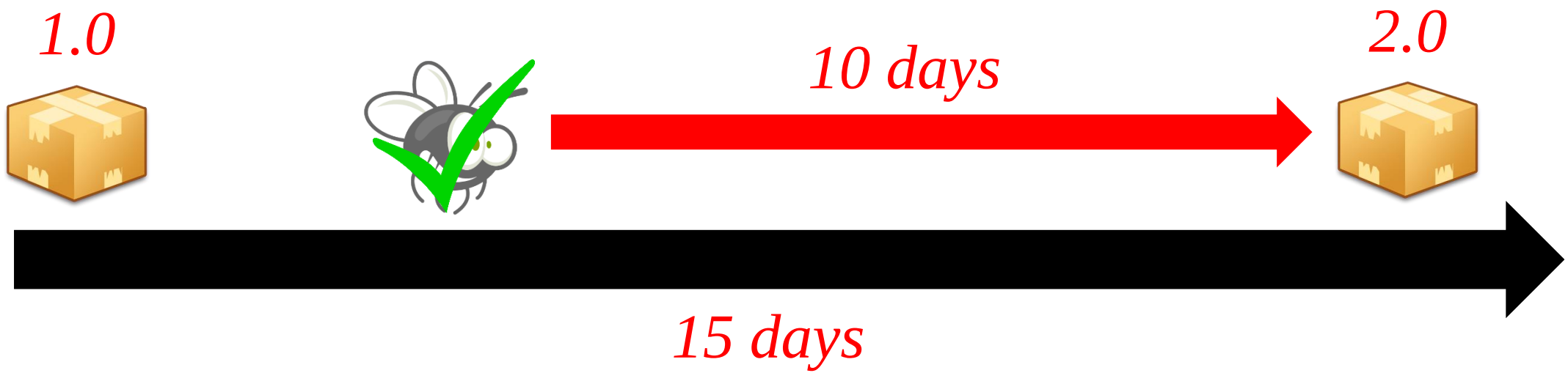
Issues are unlikely to be delayed solely because they were addressed close to an upcoming release

Issues are unlikely to be delayed solely because they were addressed close to an upcoming release

Issues are unlikely to be delayed solely because they were addressed close to an upcoming release

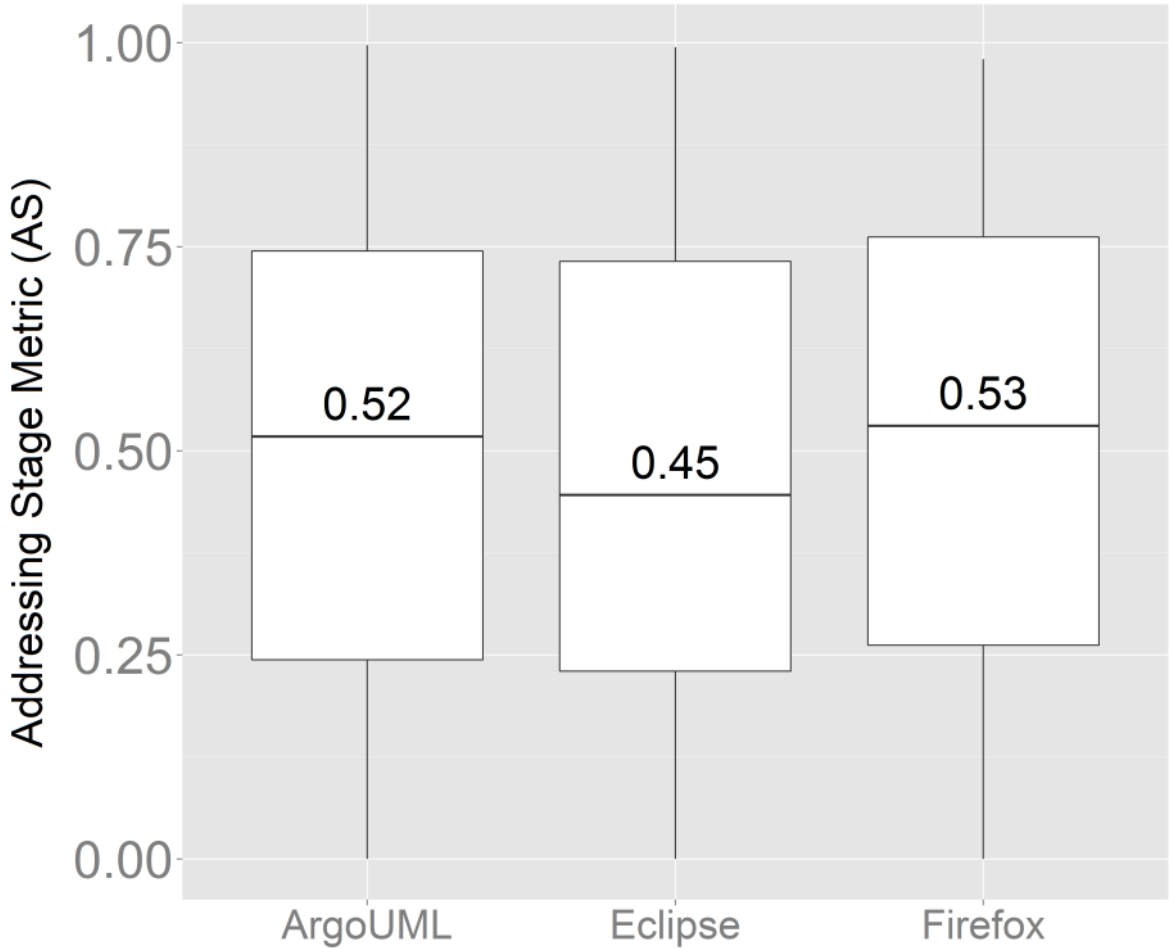


Issues are unlikely to be delayed solely because they were addressed close to an upcoming release



10/15 = 0.66

Issues are unlikely to be delayed solely because they were addressed close to an upcoming release

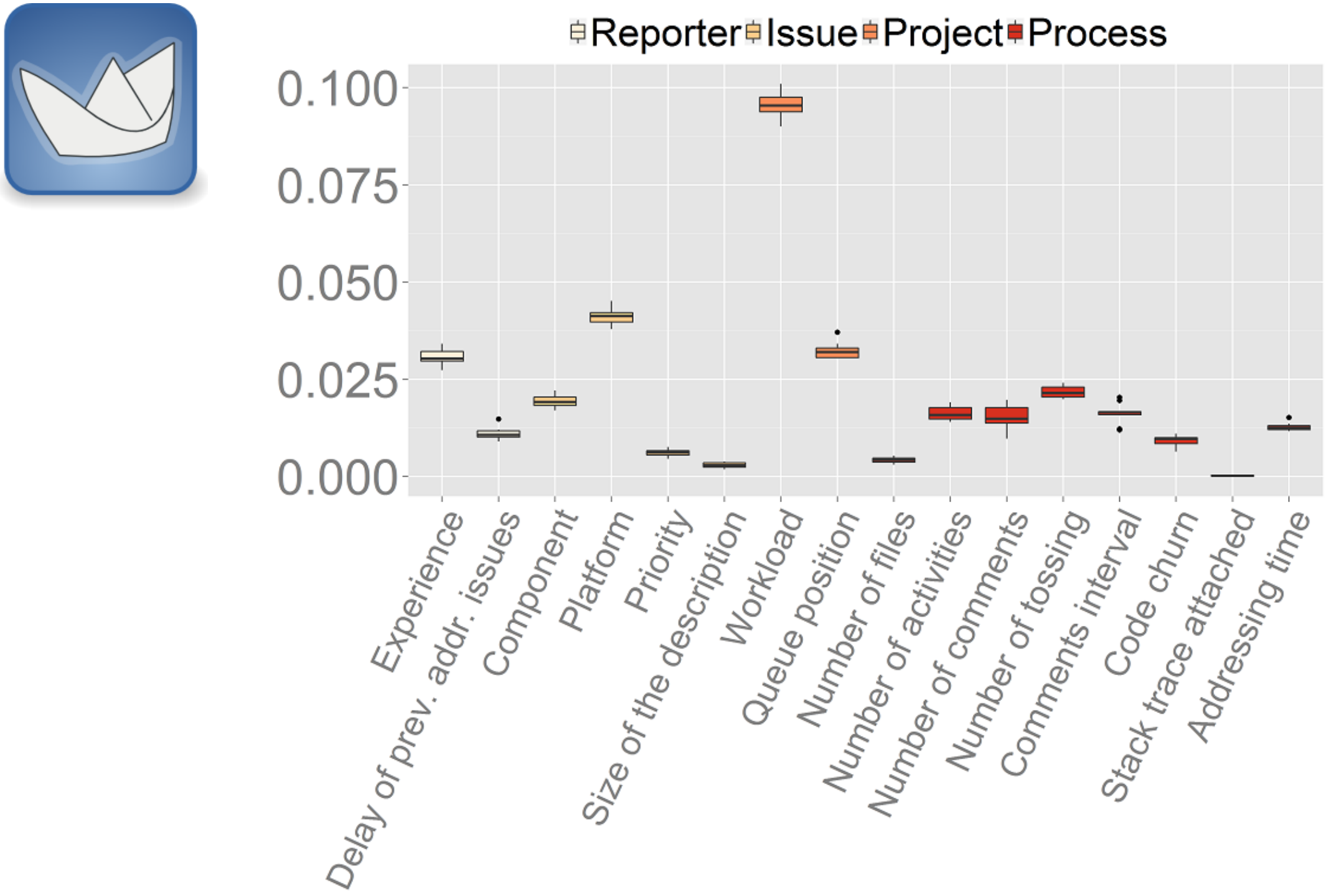


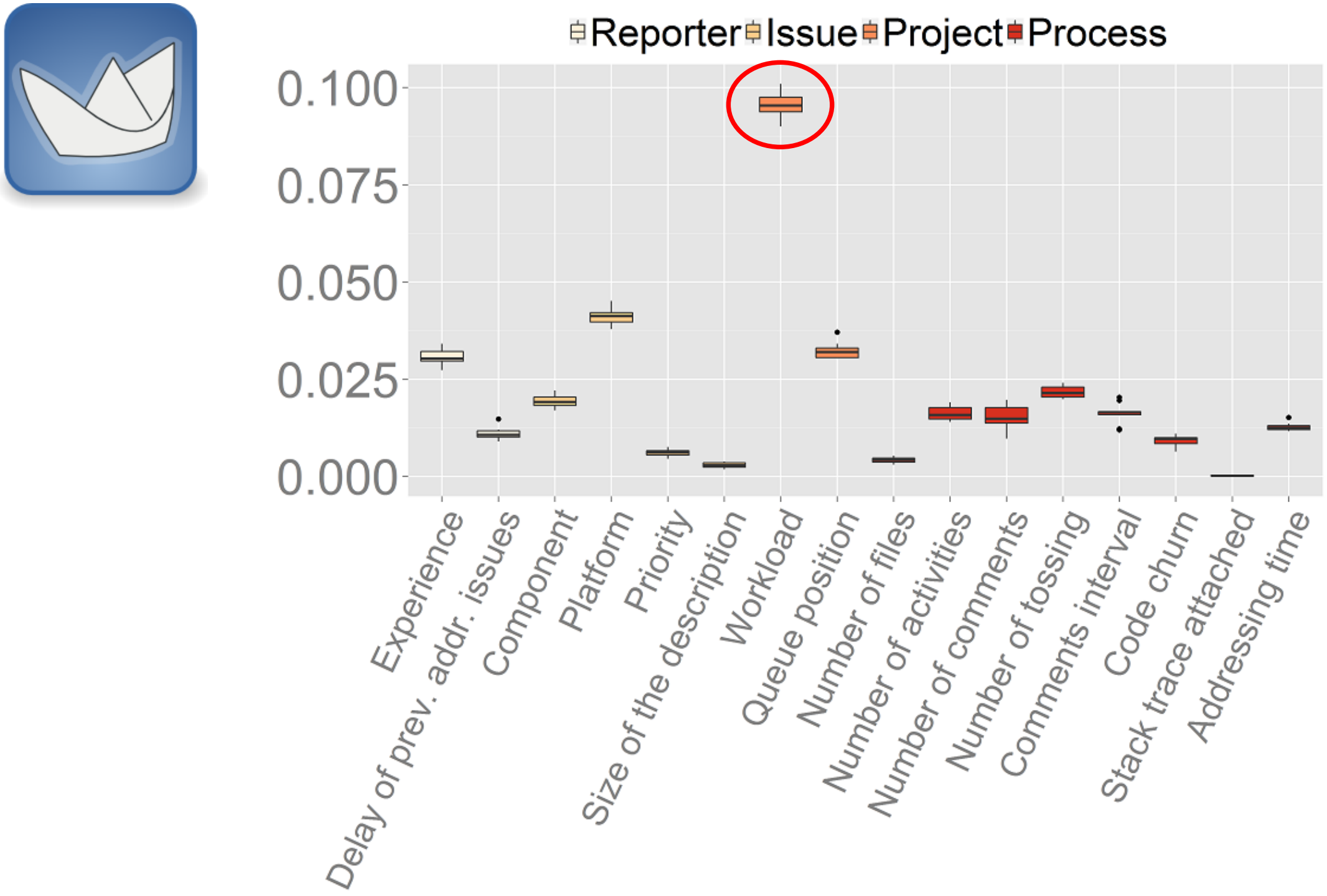
Our explanatory models outperform naïve approaches when estimating how many releases an addressed issue will miss

Our explanatory models outperform naïve approaches when estimating how many releases an addressed issue will miss

ROC areas > 0.74

*We find that the **workload** of integrators is a top most important factor in all of the studied systems*

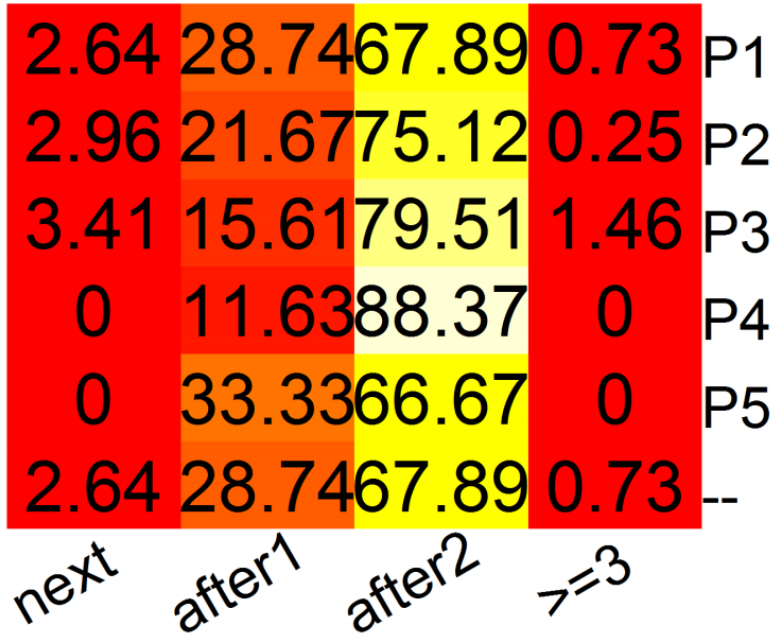

Reporter
Issue
Project
Process
0.100
0.075
0.050
0.025
0.000
Experience
Delay of prev. addr. issues
Component
Platform
Priority
Size of the description
Workload
Queue position
Number of files
Number of activities
Number of comments
Number of tossing
Comments interval
Code churn
Stack trace attached
Addressing time


Reporter
Issue
Project
Process
0.100
0.075
0.050
0.025
0.000
Experience
Delay of prev. addr. issues
Component
Platform
Priority
Size of the description
Workload
Queue position
Number of files
Number of activities
Number of comments
Number of tossing
Comments interval
Code churn
Stack trace attached
Addressing time

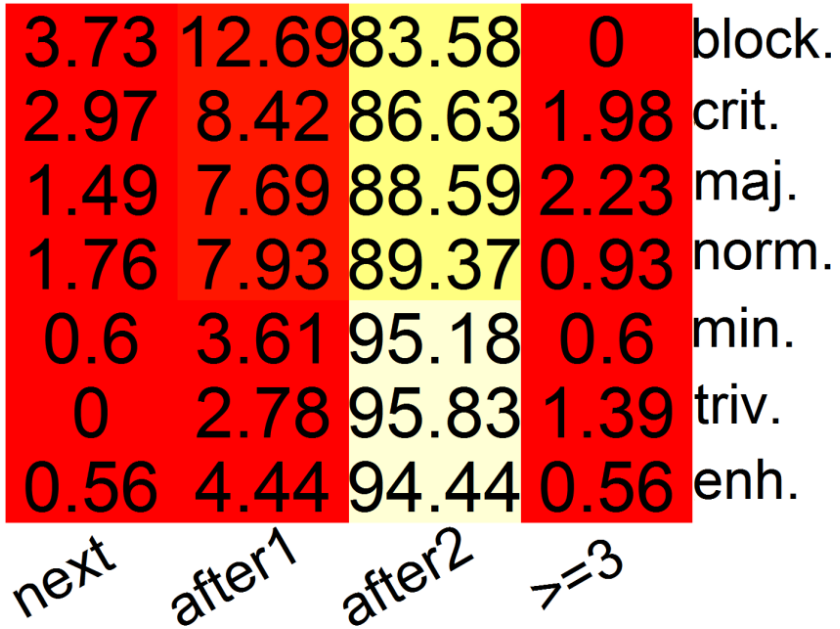
We also find that priority and severity have little impact on release delay

We also find that priority and severity have little impact on release delay

Priority



Severity



Abnormal Delay

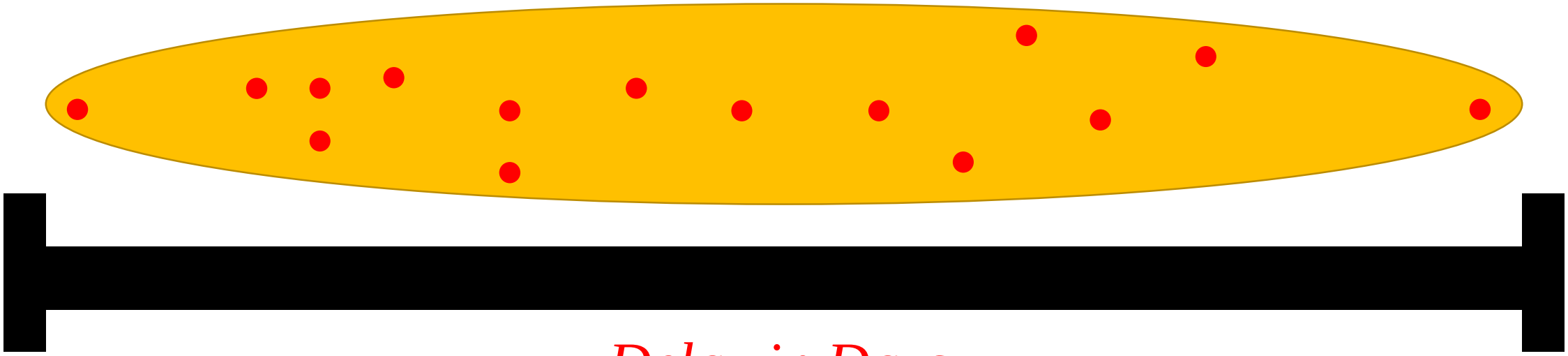
Since release delay may vary along projects, we measure the abnormal delay of each project

We measure abnormal delay ***based on the median*** *of days to release addressed issues in a project*

We measure abnormal delay ***based on the median*** *of days to release addressed issues in a project*

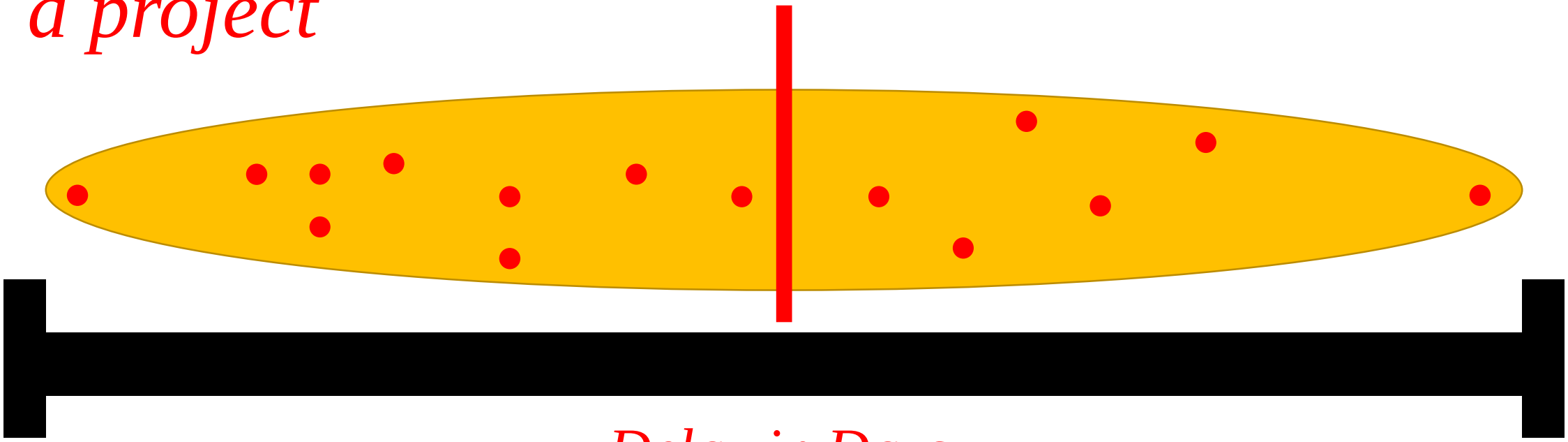
Delay in Days

We measure abnormal delay ***based on the median*** *of days to release addressed issues in a project*



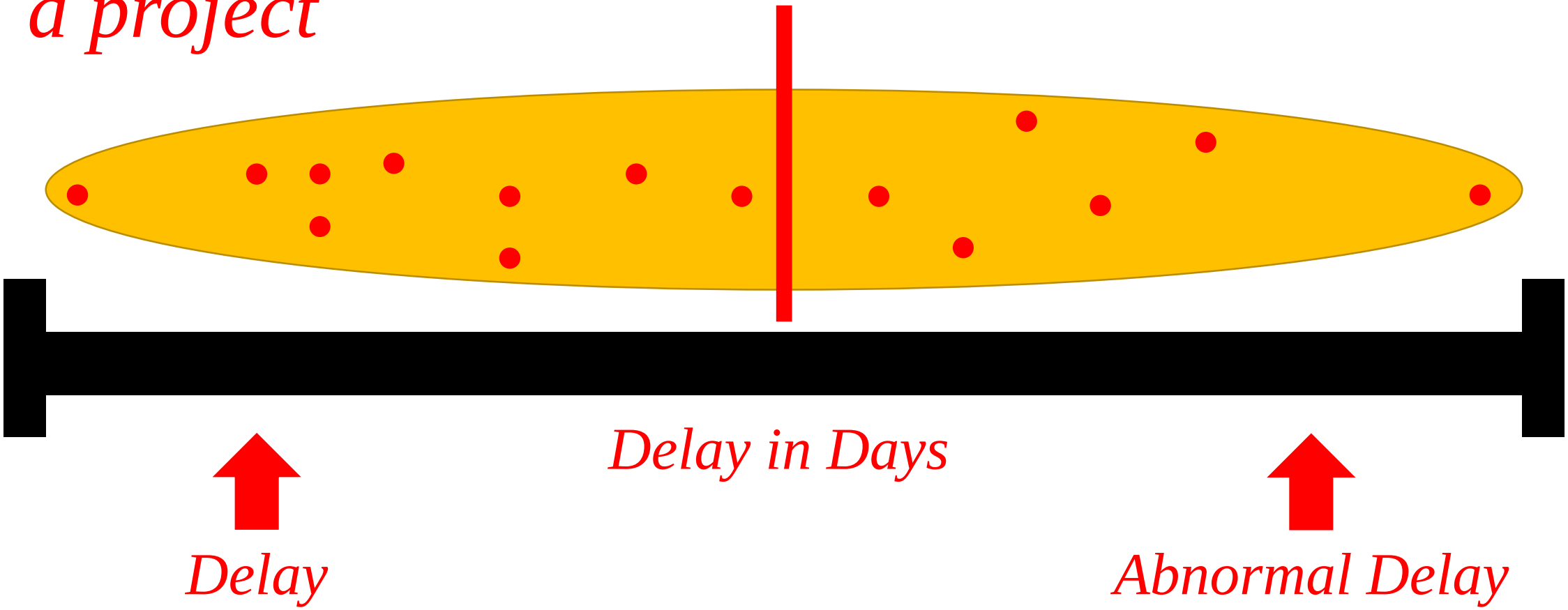
Delay in Days

We measure abnormal delay ***based on the median*** *of days to release addressed issues in a project*



Delay in Days

We measure abnormal delay ***based on the median*** *of days to release addressed issues in a project*



Can we explain which addressed issues will take longer to be integrated than most others?

Can we explain which addressed issues will take longer to be integrated than most others?

What are the most influential attributes for estimating abnormally delayed issues?

Can we explain which addressed issues will take longer to be integrated than most others?

What are the most influential attributes for estimating abnormally delayed issues?

Does the delivery delay of addressed issues relates to the components that they are being modified?

Can we explain which addressed issues will take longer to be integrated than most others?

Building Explanatory Models

What are the most influential attributes for estimating abnormally delayed issues?

Does the delivery delay of addressed issues relates to the components that they are being modified?

Can we explain which addressed issues will take longer to be integrated than most others?

Building Explanatory Models

What are the most influential attributes for estimating abnormally delayed issues?

Identifying Top Influential Factors

Does the delivery delay of addressed issues relates to the components that they are being modified?

Can we explain which addressed issues will take longer to be integrated than most others?

Building Explanatory Models

What are the most influential attributes for estimating abnormally delayed issues?

Identifying Top Influential Factors

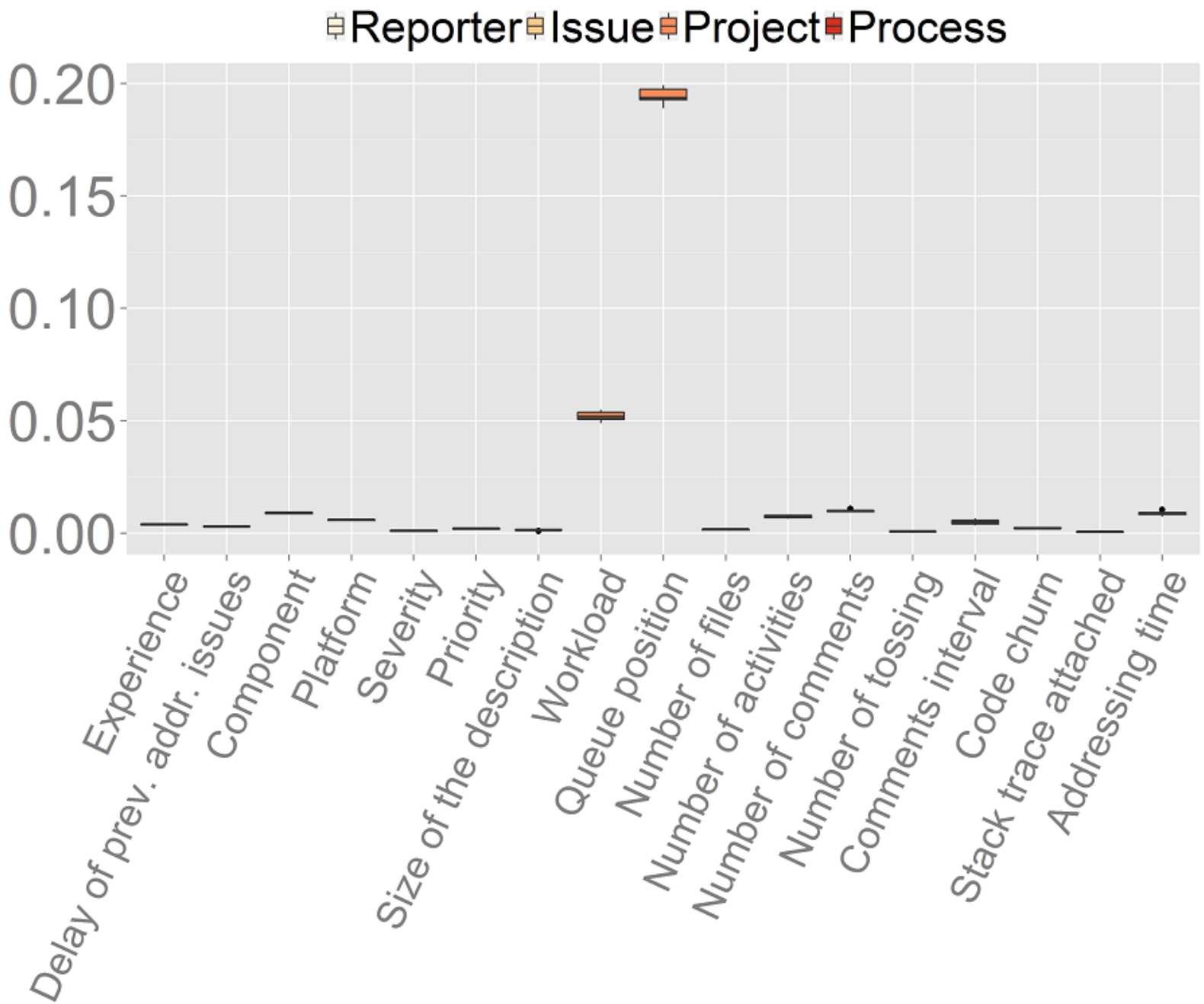
Does the delivery delay of addressed issues relates to the components that they are being modified?

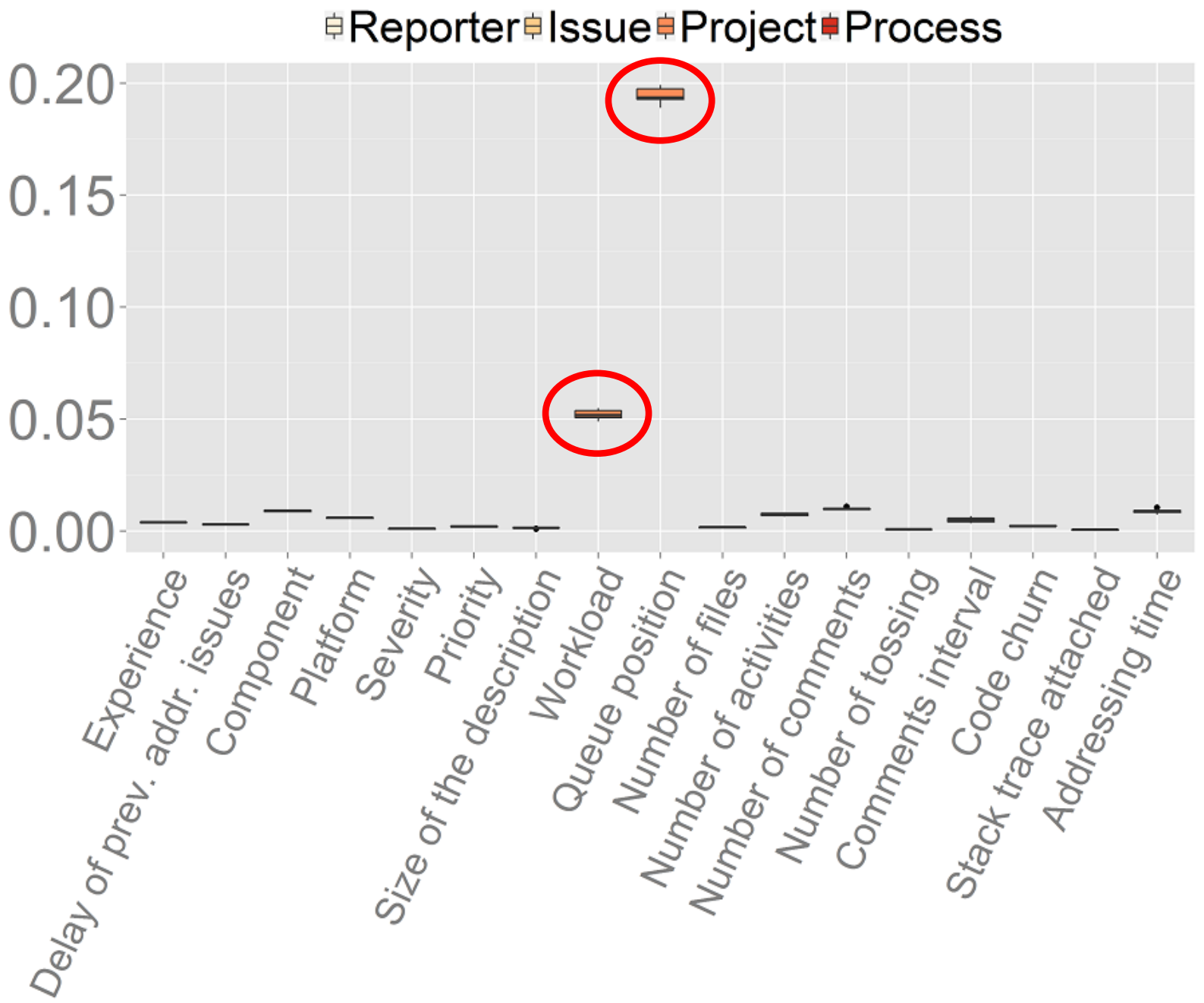
Measuring Delivery Delay per Components

Our models can accurately estimate if an addressed issue will be abnormally delayed

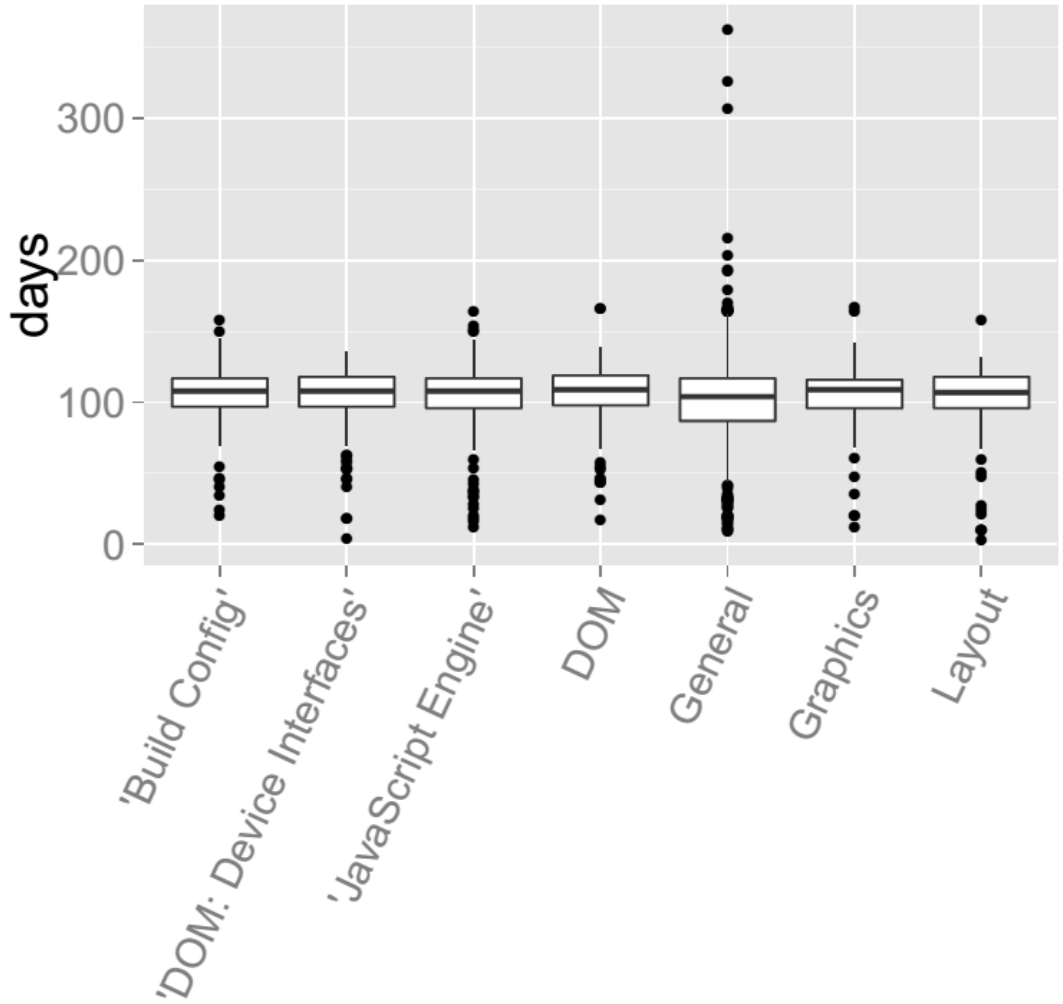
ROC areas > 0.87

*We find that **workload** and the **timing (queue position)** to address issues are the most important factors to explain abnormally delayed issues*


Reporter
Issue
Project
Process
0.20
0.15
0.10
0.05
0.00
Experience
Delay of prev. addr. issues
Component
Platform
Severity
Priority
Size of the description
Workload
Queue position
Number of files
Number of activities
Number of comments
Number of tossing
Comments interval
Code churn
Stack trace attached
Addressing time

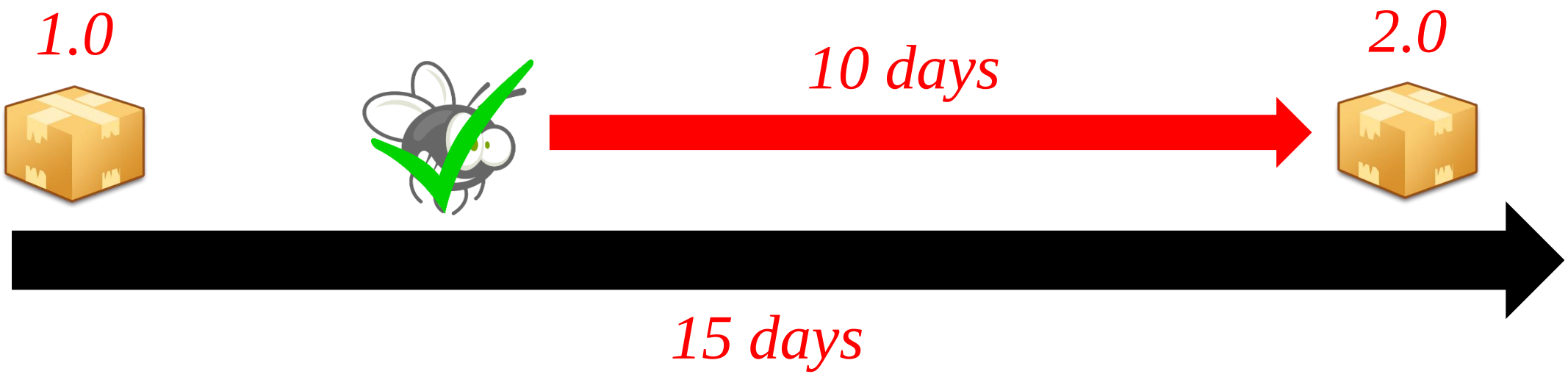

Reporter
Issue
Project
Process
0.20
0.15
0.10
0.05
0.00
Experience
Delay of prev. addr. issues
Component
Platform
Severity
Priority
Size of the description
Workload
Queue position
Number of files
Number of activities
Number of comments
Number of tossing
Comments interval
Code churn
Stack trace attached
Addressing time

We find that components are unlikely to be related to delivery delay measured in days

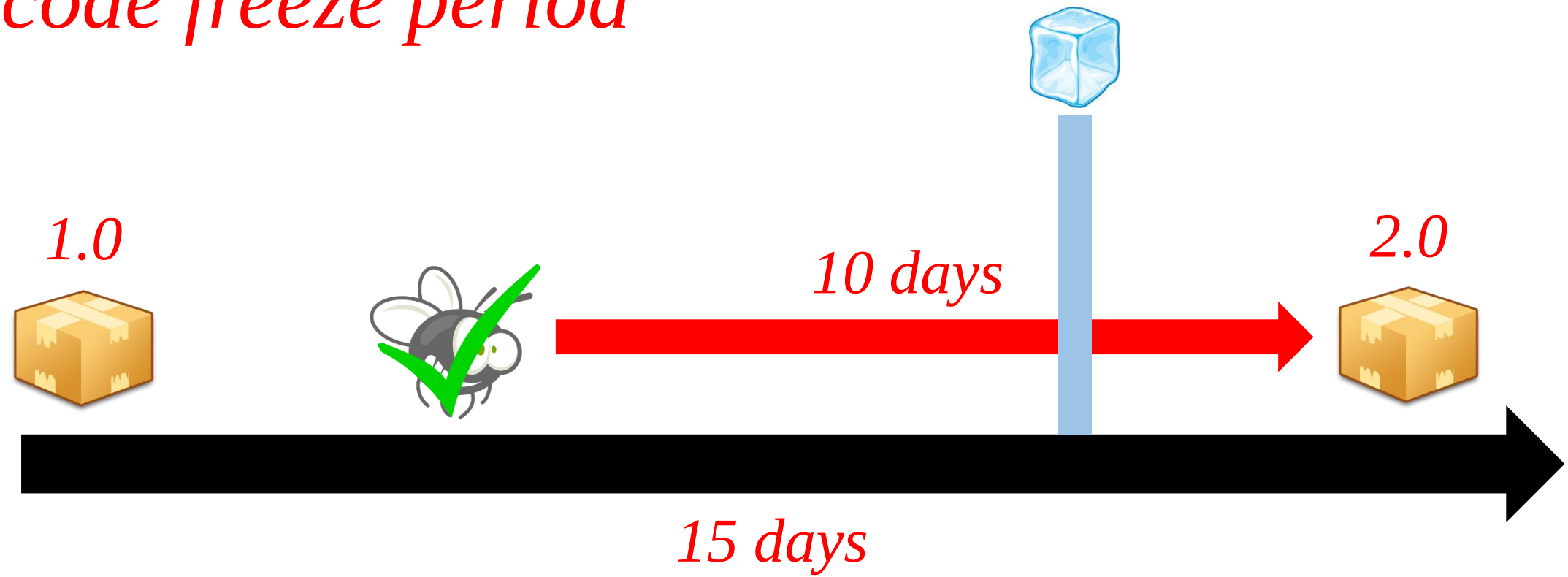


We find that issues are unlikely to be delayed solely because they were addressed close to a code freeze period

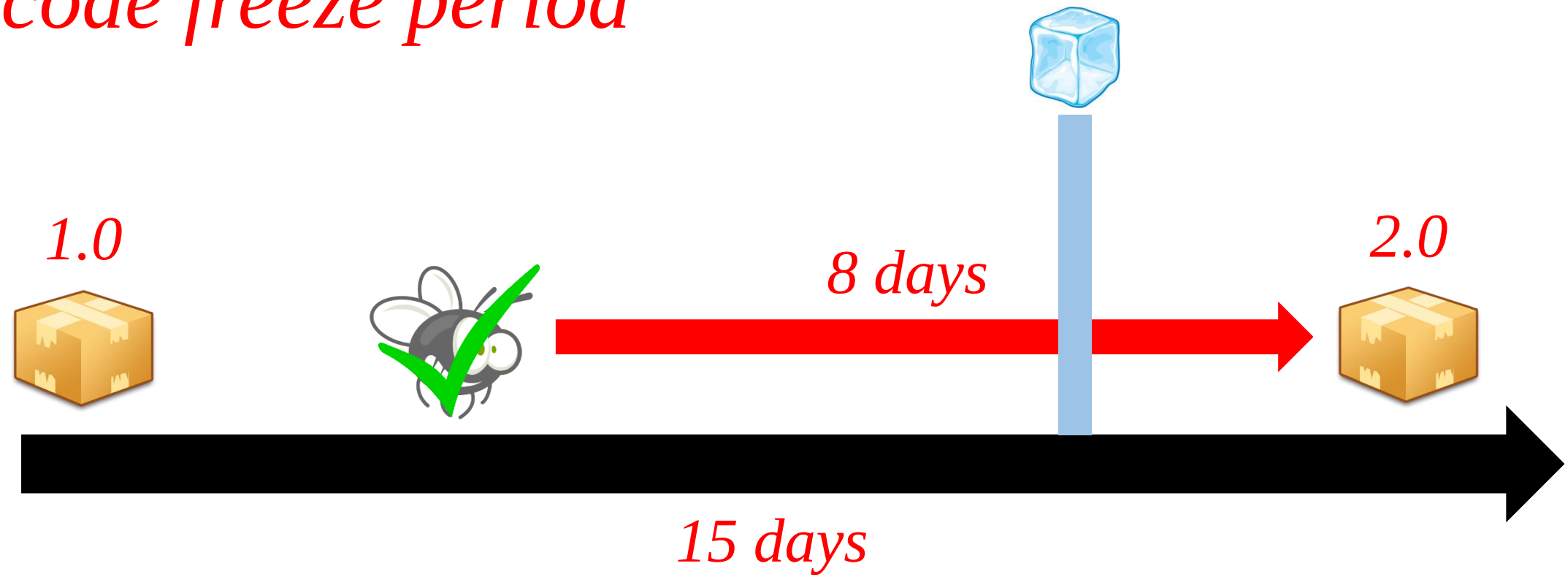
We find that issues are unlikely to be delayed solely because they were addressed close to a code freeze period



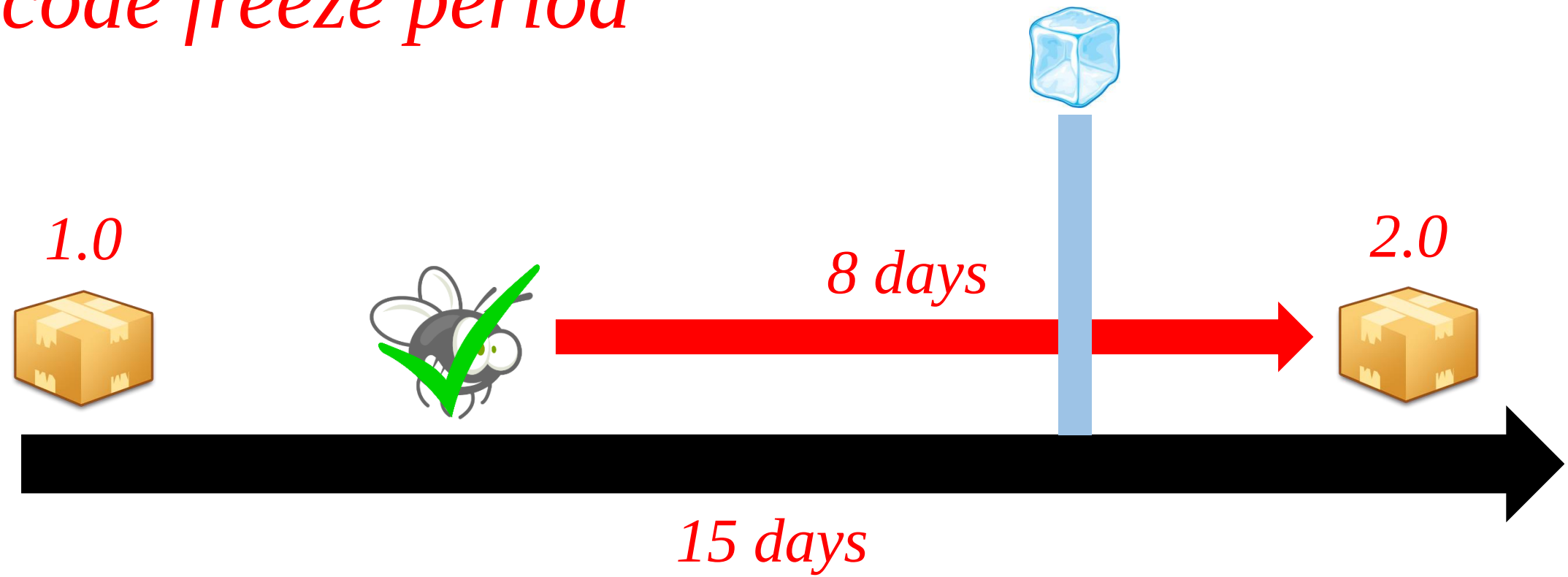
We find that issues are unlikely to be delayed solely because they were addressed close to a code freeze period



We find that issues are unlikely to be delayed solely because they were addressed close to a code freeze period

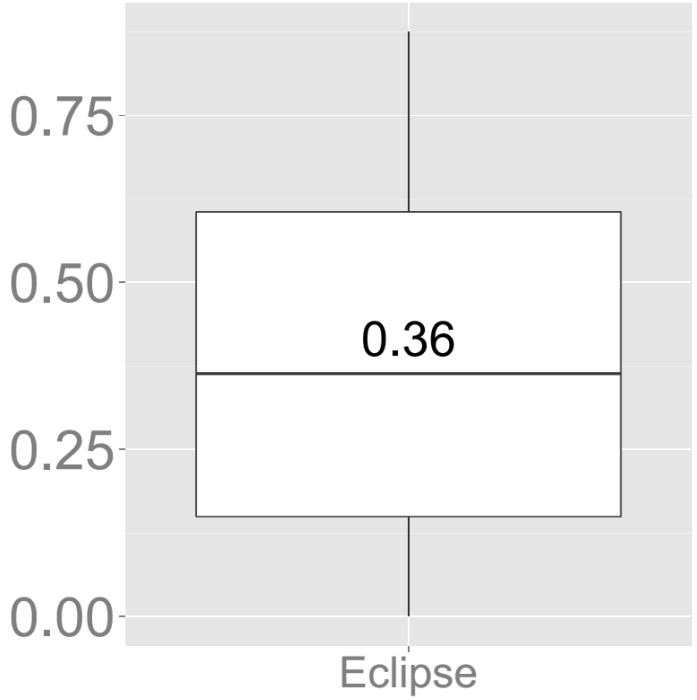


We find that issues are unlikely to be delayed solely because they were addressed close to a code freeze period



8/15 = 0.53

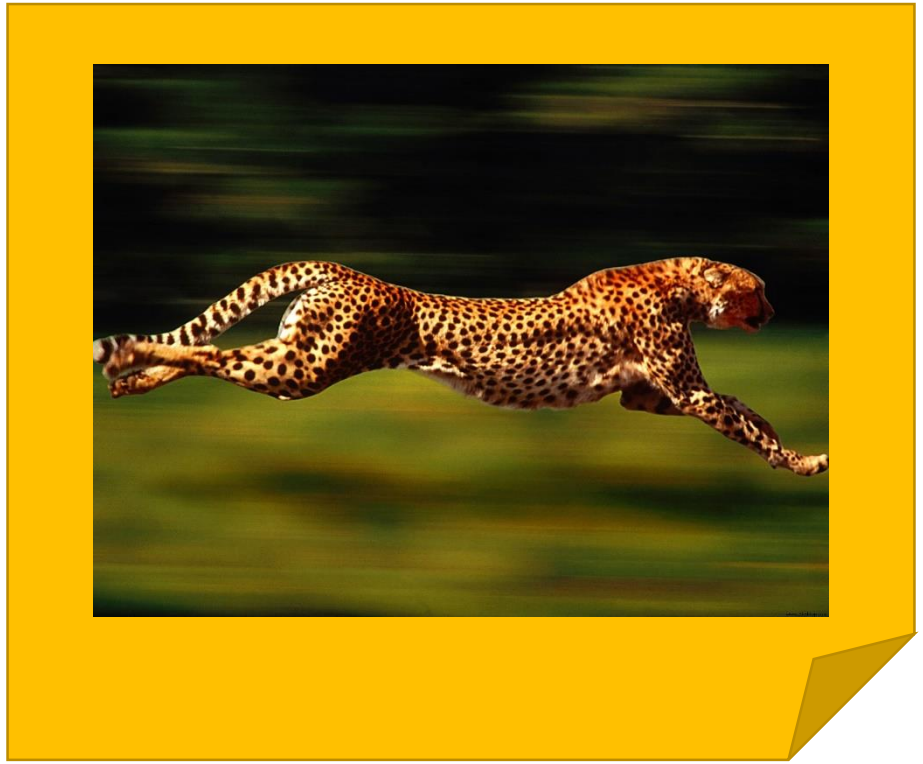
We find that issues are unlikely to be delayed solely because they were addressed close to a code freeze period



Impact of Rapid Release Cycle on Delivery Delay (Study 2)

An alternative version of this work was submitted to MSR'16

Short release cycles are usually thought to be associated with faster delivery of software



There is a lack of empirical studies to check if rapid releases lead to faster delivery of addressed issues

Previous research has invested in studying how rapid releases impact on the speed to fix issues

We set out to compare how rapid releases of Firefox impact on the speed to deliver addressed issues of that system

We set out to compare how rapid releases of Firefox impact on the speed to deliver addressed issues of that system

Traditional

VS

Rapid

We set out to compare how rapid releases of Firefox impact on the speed to deliver addressed issues of that system

Traditional

111 releases

VS

Rapid

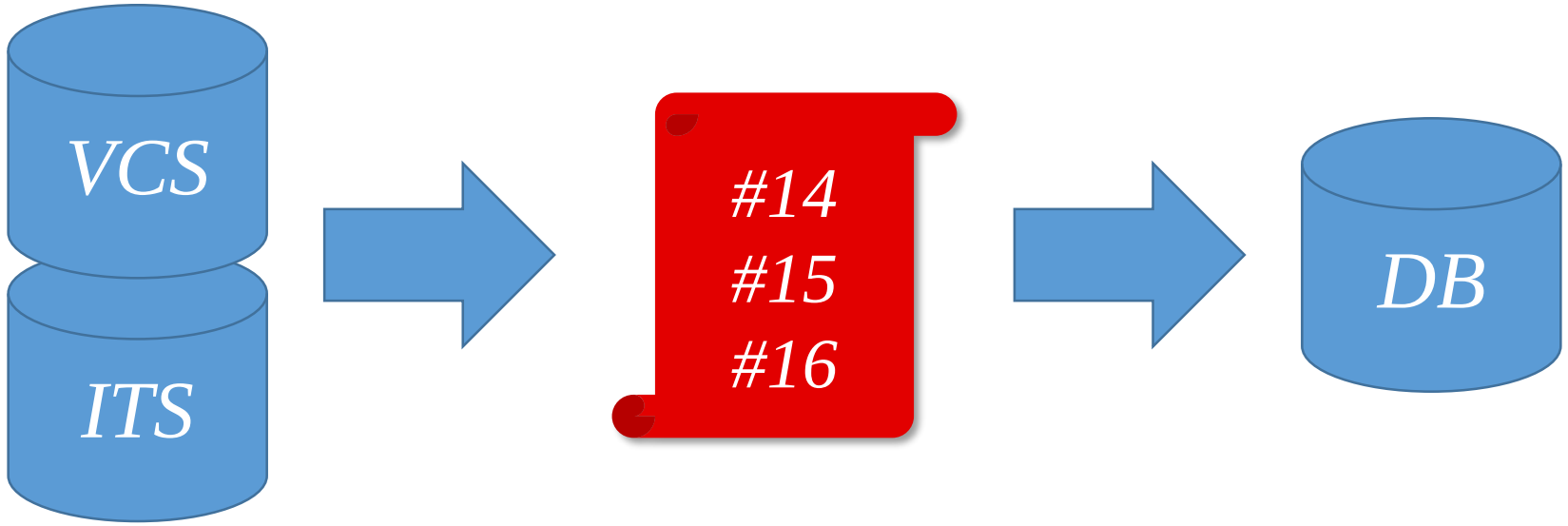
73 releases

We set out to compare how rapid releases of Firefox impact on the speed to deliver addressed issues of that system

VS

Traditional	Rapid
111 releases	73 releases
34,673 issues	37,441 issues

We collect data from code and issue repositories to perform our analyses



We measure delivery delay by counting the number of days to release an addressed issue

Are addressed issues delivered more quickly in rapid releases?

Are addressed issues delivered more quickly in rapid releases?

Why can traditional releases integrate addressed issues more quickly?

Are addressed issues delivered more quickly in rapid releases?

Why can traditional releases integrate addressed issues more quickly?

Does the change in the release strategy have an impact on the characteristics of delayed issues?

Are addressed issues delivered more quickly in rapid releases?

Comparing Traditional and Rapid Releases

Why can traditional releases integrate addressed issues more quickly?

Does the change in the release strategy have an impact on the characteristics of delayed issues?

Are addressed issues delivered more quickly in rapid releases?

Comparing Traditional and Rapid Releases

Why can traditional releases integrate addressed issues more quickly?

Studying major/minor releases

Does the change in the release strategy have an impact on the characteristics of delayed issues?

Are addressed issues delivered more quickly in rapid releases?

Comparing Traditional and Rapid Releases

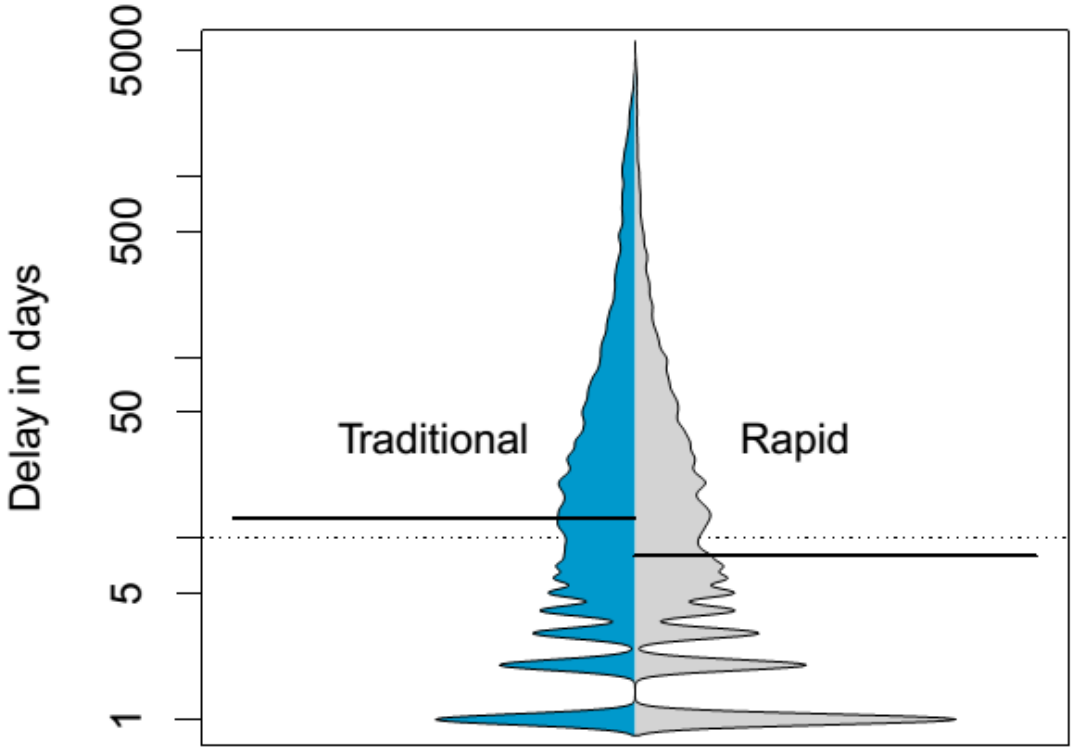
Why can traditional releases integrate addressed issues more quickly?

Studying major/minor releases

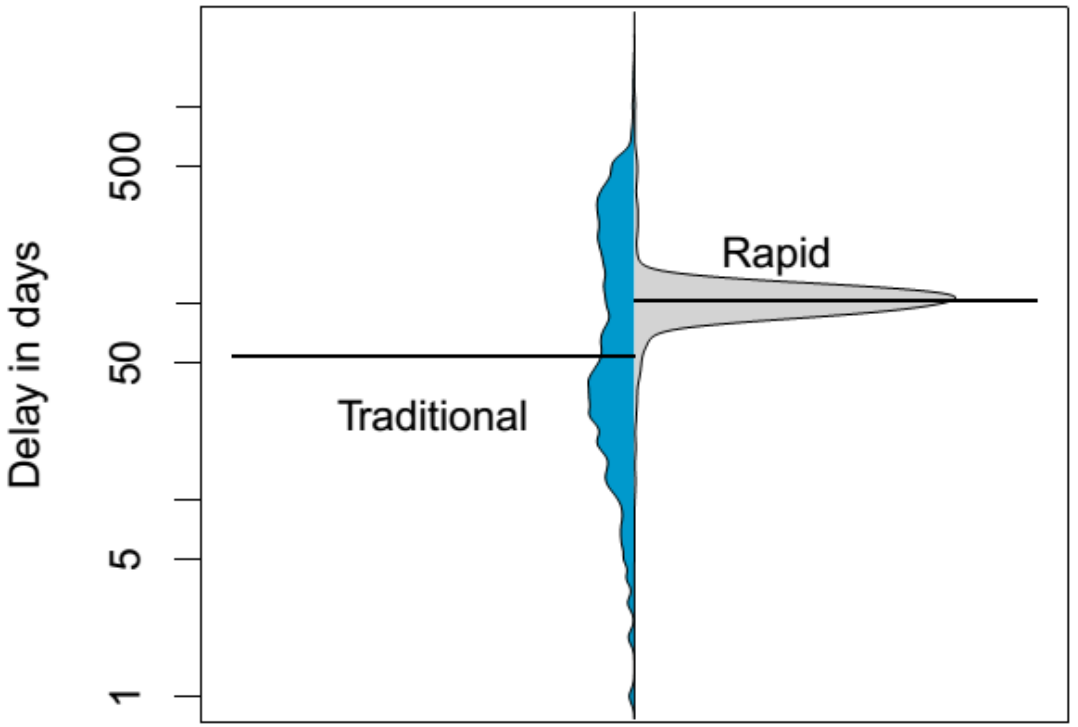
Does the change in the release strategy have an impact on the characteristics of delayed issues?

Building explanatory models

Although issues are triaged and fixed faster in rapid releases, they wait a longer time to be delivered

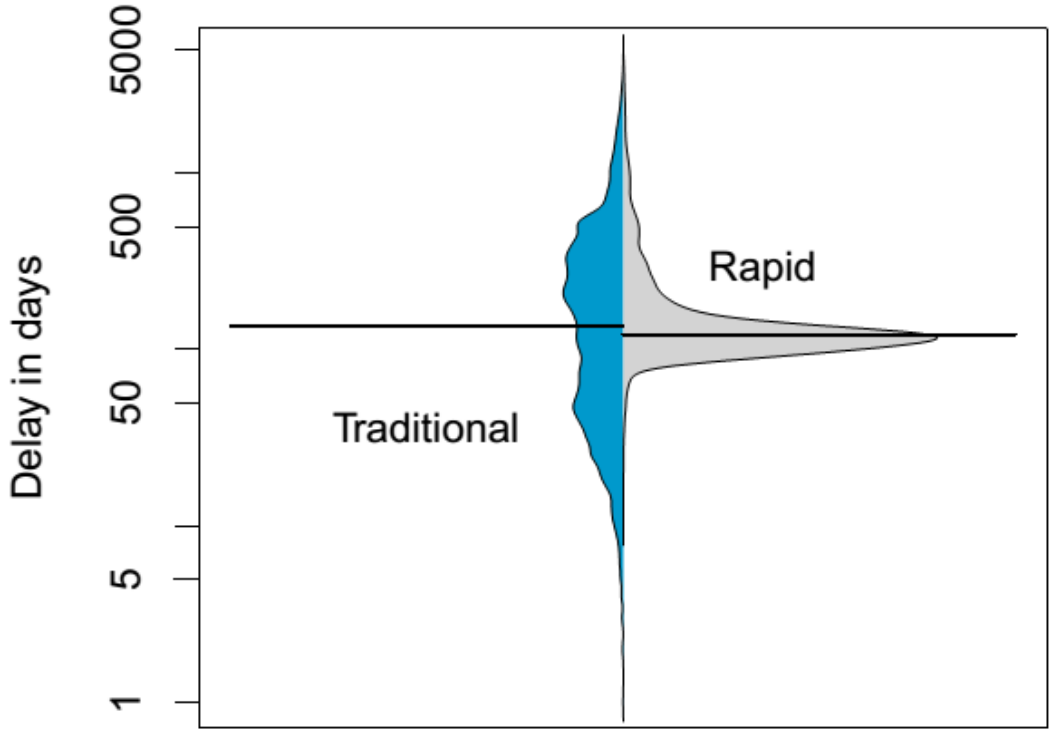


(c) Fixing phase

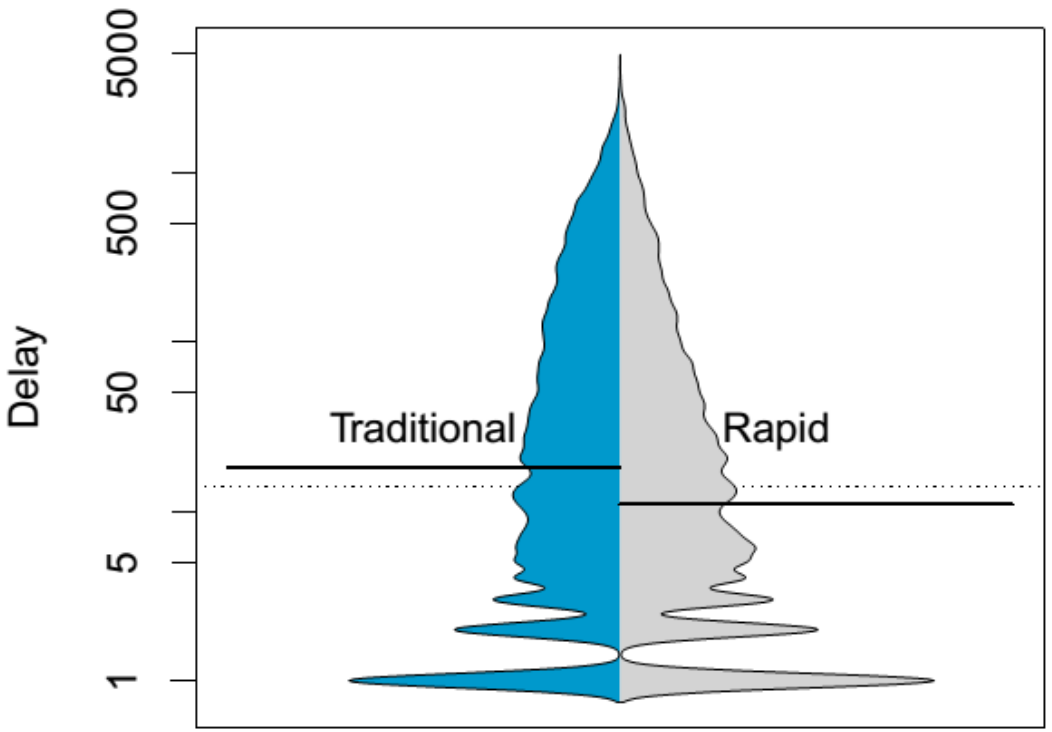


(d) Integration phase

Although issues are triaged and fixed faster in rapid releases, they wait a longer time to be delivered

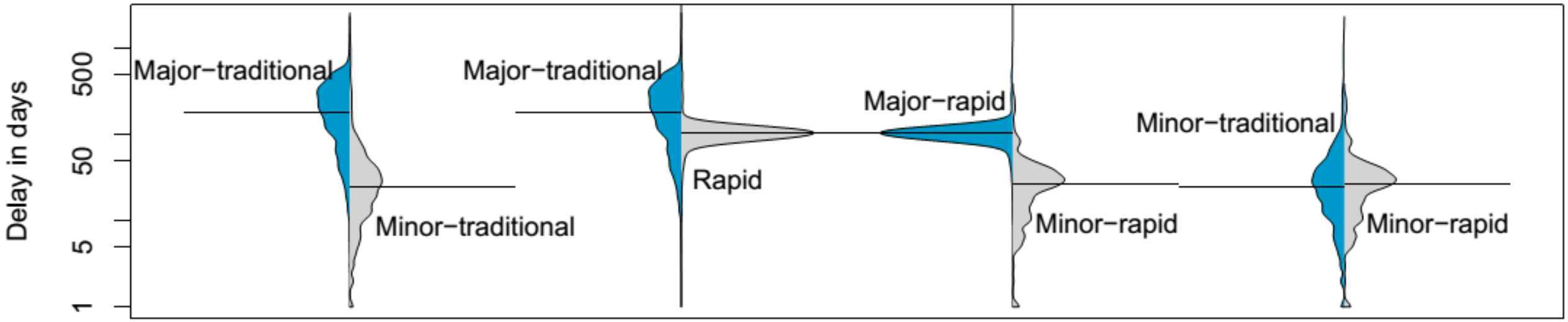


(a) Lifetime



(b) Triaging phase

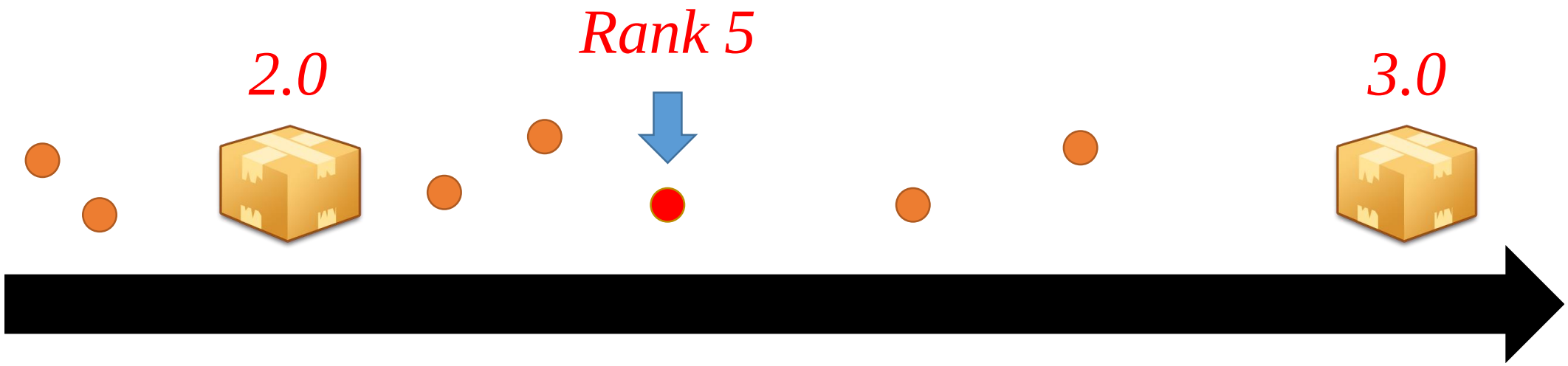
Minor-traditional releases are a key reason why traditional releases can deliver issues more quickly



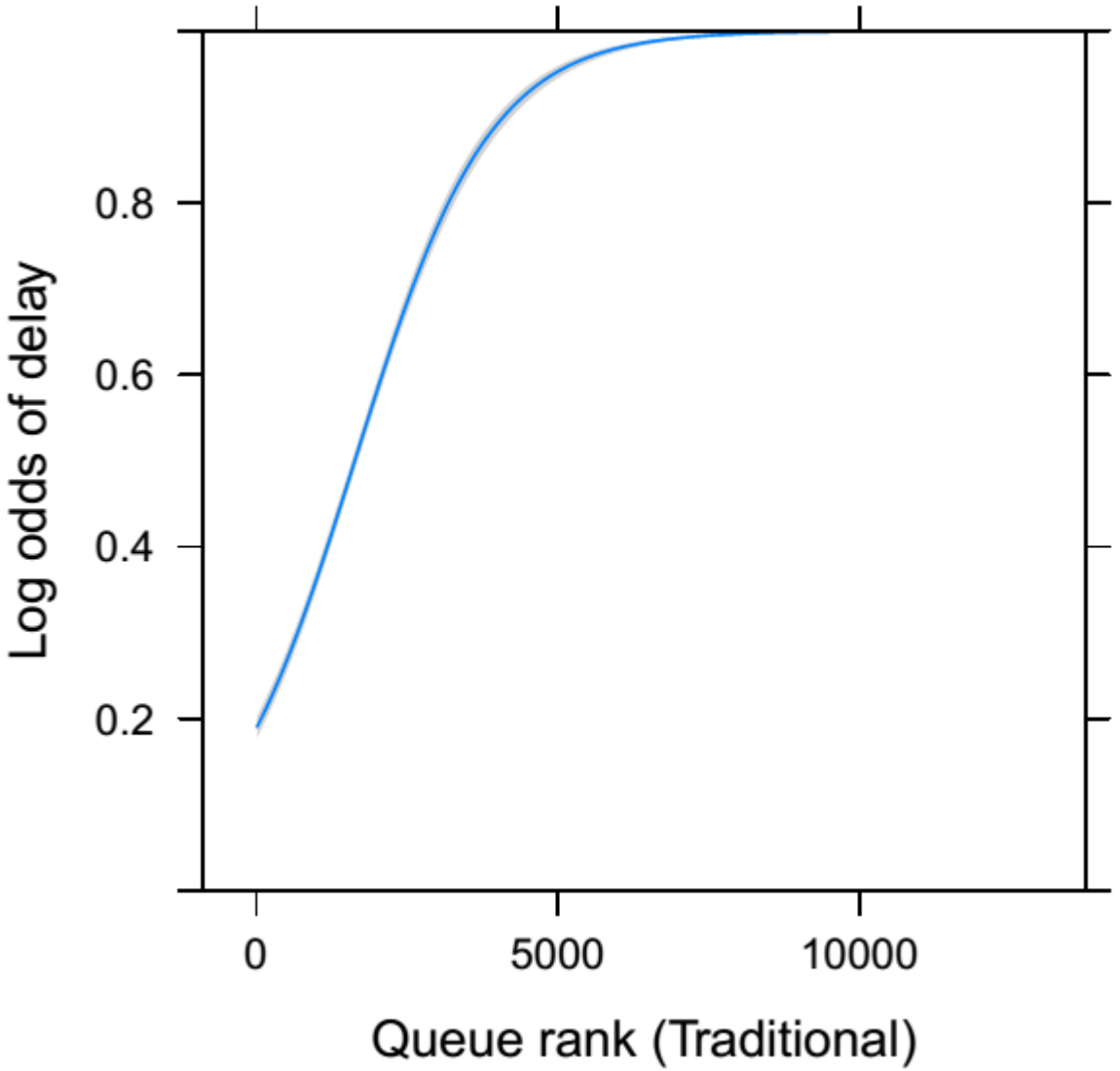
Traditional Releases prioritize backlog issues

Traditional Releases prioritize backlog issues

Traditional Releases prioritize backlog issues



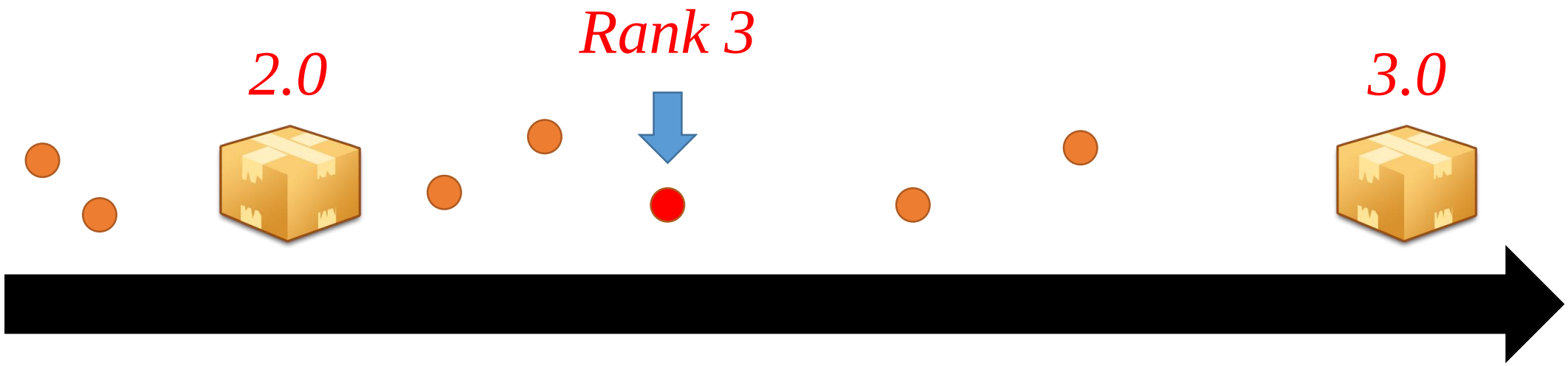
Traditional Releases prioritize backlog issues



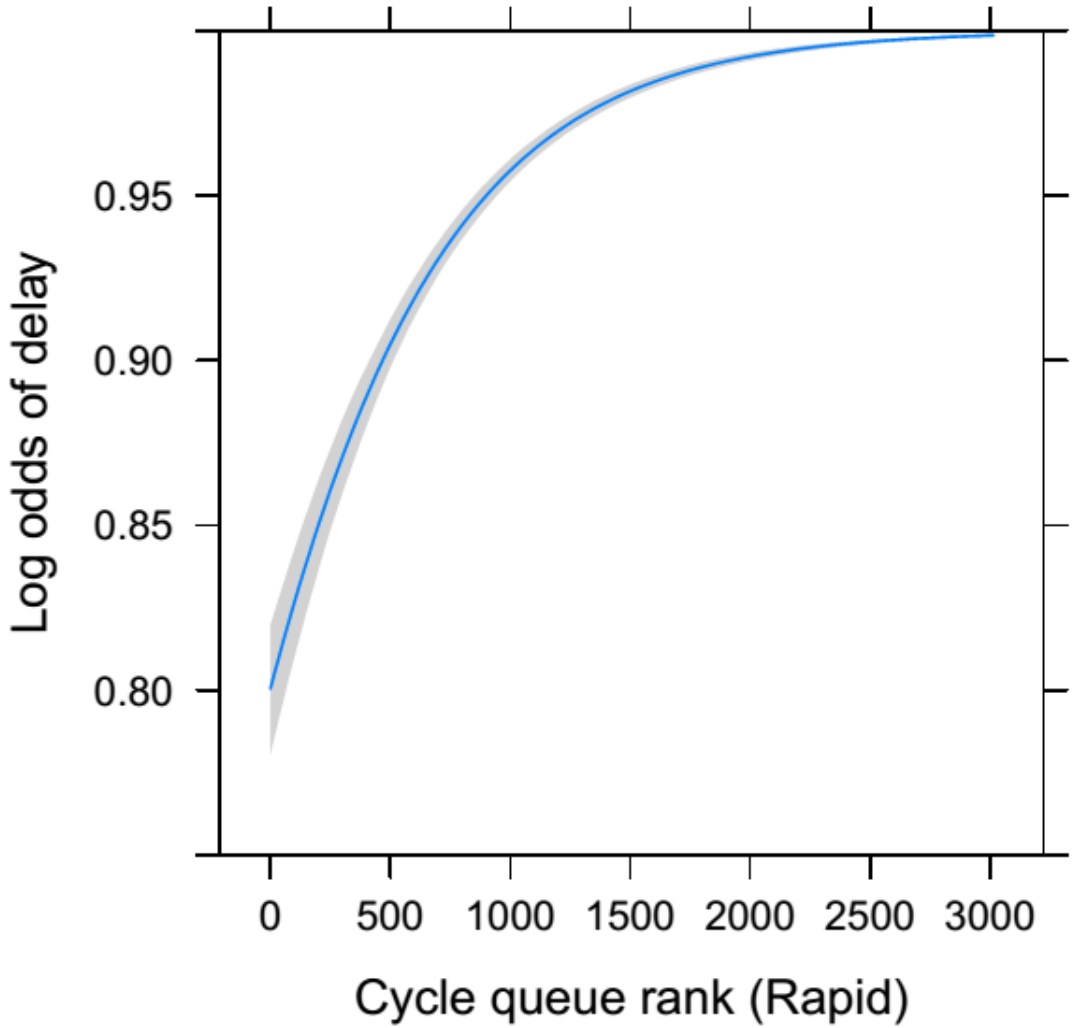
Rapid releases prioritize issues of the current release cycle

Rapid releases prioritize issues of the current release cycle

Rapid releases prioritize issues of the current release cycle

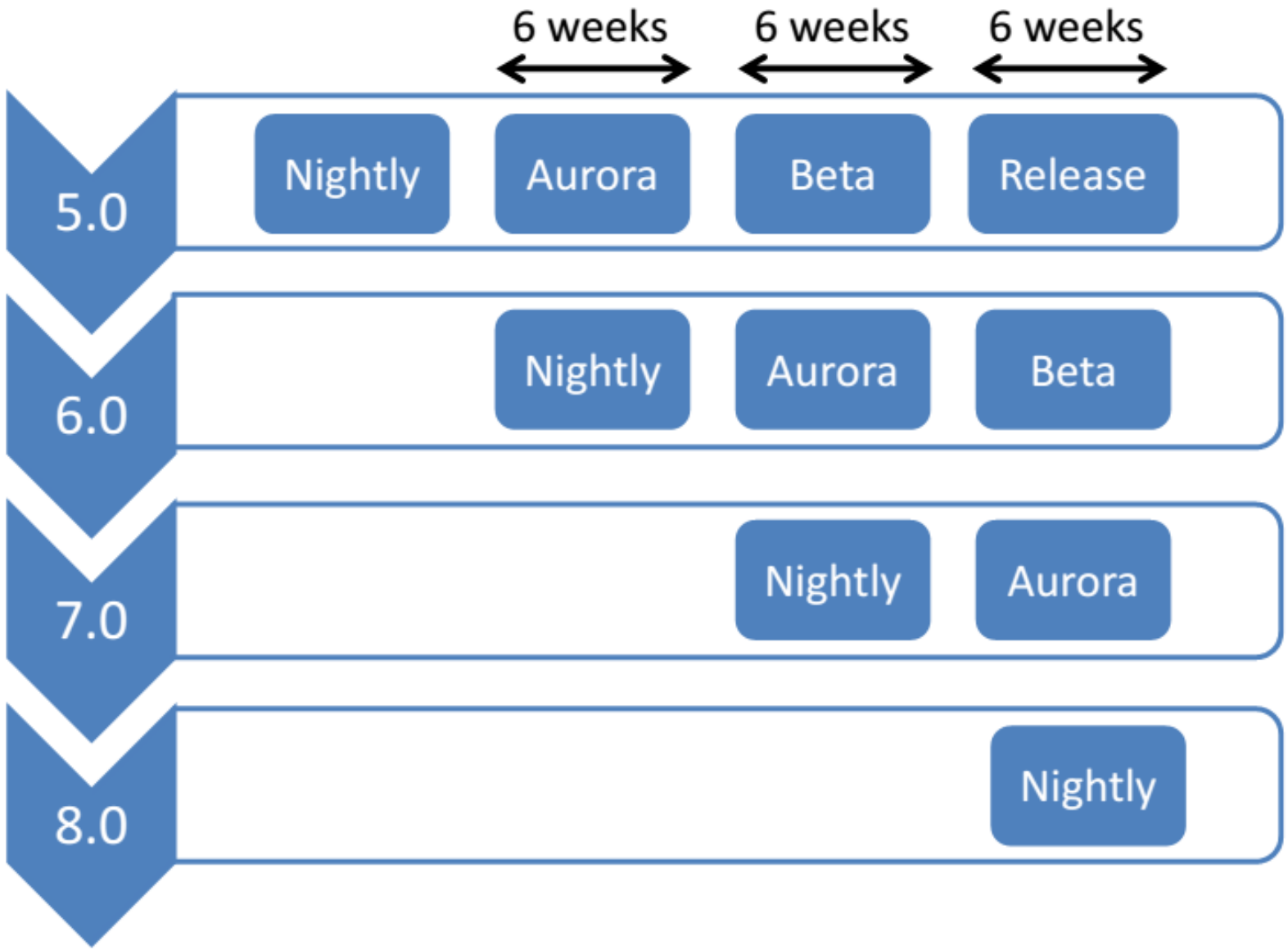


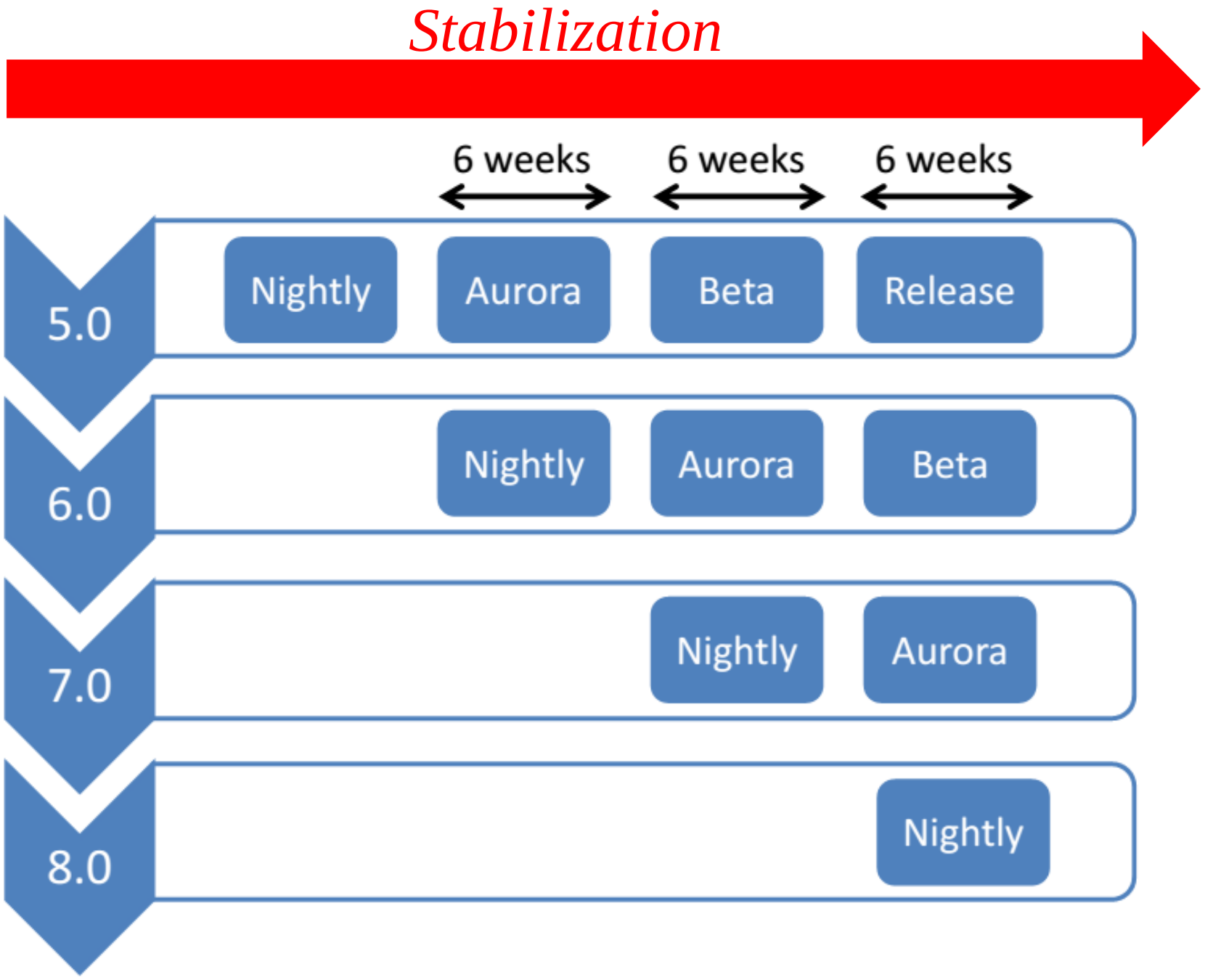
Rapid releases prioritize issues of the current release cycle



Staged Delivery of Addressed Issue (Study 3)

We Intend to study how addressed issues are stabilized in pipelining releases before being delivered to users


6 weeks
6 weeks
6 weeks
5.0
Nightly
Aurora
Beta
Release
6.0
Nightly
Aurora
Beta
7.0
Nightly
Aurora
8.0
Nightly


Stabilization
6 weeks
6 weeks
6 weeks
5.0
Nightly
Aurora
Beta
Release
6.0
Nightly
Aurora
Beta
7.0
Nightly
Aurora
8.0
Nightly

How much time it takes for each stabilization phase?

How much time it takes for each stabilization phase?

Do all of the addressed issues follow the same process to be stabilized?

How much time it takes for each stabilization phase?

Do all of the addressed issues follow the same process to be stabilized?

What are the special characteristics of those issues that are stabilized more quickly?

How much time it takes for each stabilization phase?

Measuring Stabilization Delay

Do all of the addressed issues follow the same process to be stabilized?

What are the special characteristics of those issues that are stabilized more quickly?

How much time it takes for each stabilization phase?

Measuring Stabilization Delay

Do all of the addressed issues follow the same process to be stabilized?

Studying the stabilization workflow of addressed issues

What are the special characteristics of those issues that are stabilized more quickly?

How much time it takes for each stabilization phase?

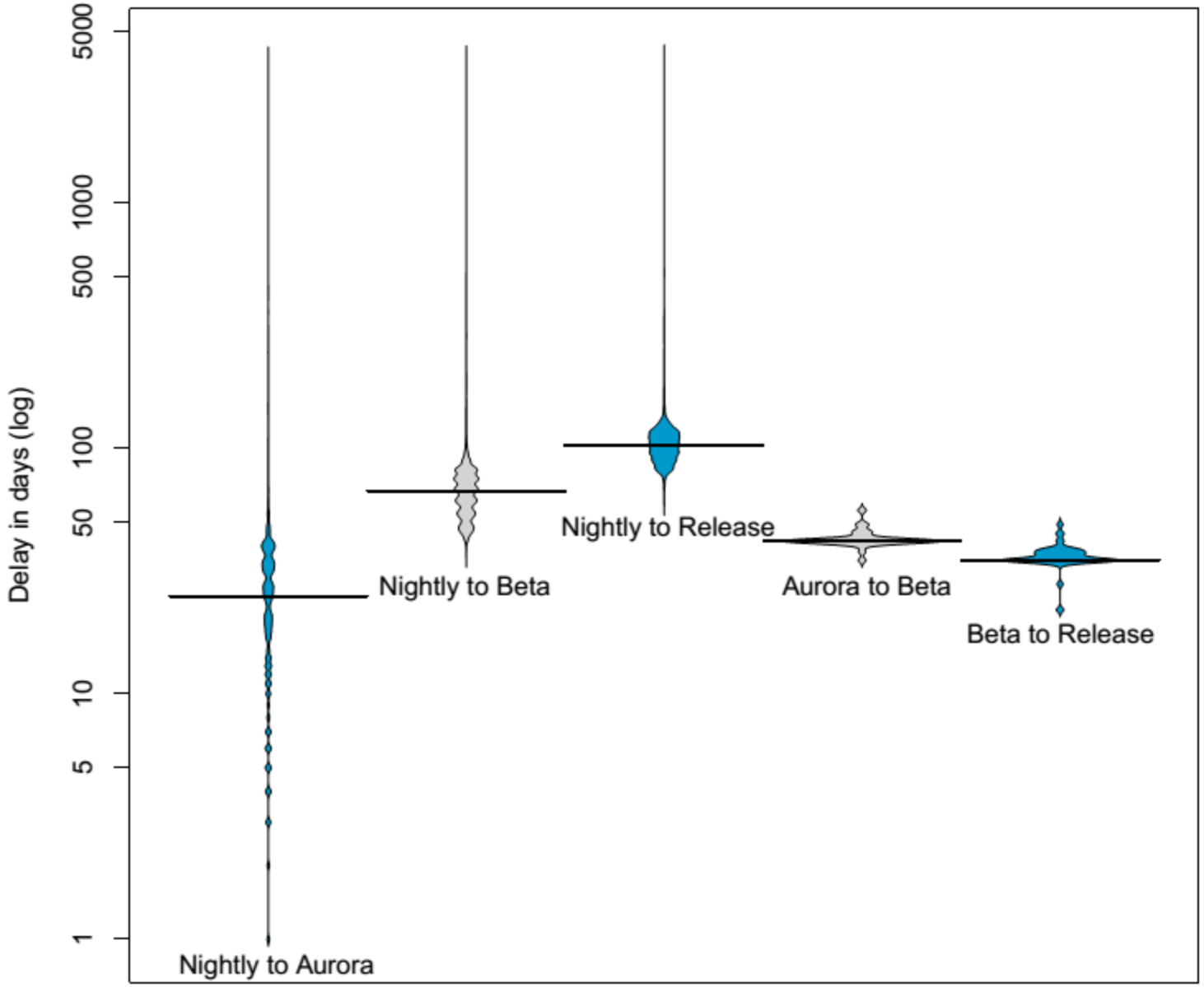
Measuring Stabilization Delay

Do all of the addressed issues follow the same process to be stabilized?

Studying the stabilization workflow of addressed issues

What are the special characteristics of those issues that are stabilized more quickly?

Building explanatory models


Delay in days (log)
5000
1000
500
100
50
10
5
1
Nightly to Aurora
Nightly to Beta
Nightly to Release
Aurora to Beta
Beta to Release

Conclusions

Yes, delivery delay exists and it is not rare. And release strategies have some impact on such delays

We expect that by performing our empirical studies we can possibly provide some guidelines and insights to the practice

Understanding Software Delivery Delay

Daniel Alencar da Costa

Supervisor: Uirá Kulesza

Co-supervisor: Ahmed E. Hassan