Understand risks in software systems

Nobody cares

about code quality

R
R
R
I
I
S
S
K
K

Software engineering

is about managing

risks

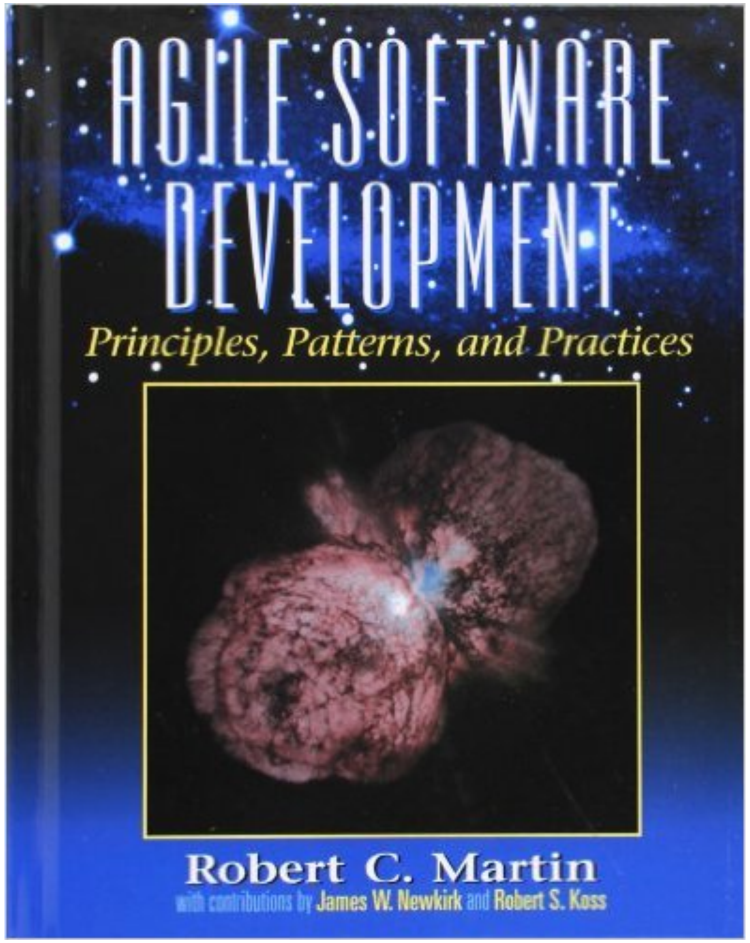
Software design

is about managing

risks

Design principles are about managing

risks of change

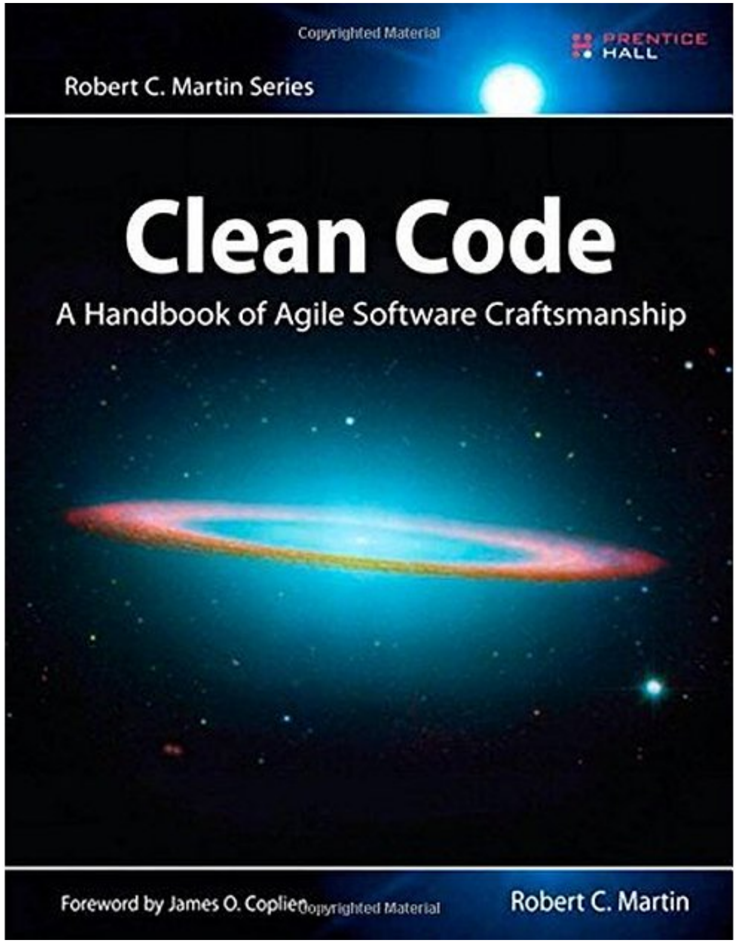


AGILE SOFTWARE DEVELOPMENT

Principles, Patterns, and Practices

Robert C. Martin

with contributions by James W. Newkirk and Robert S. Koss



PRENTICE HALL

Robert C. Martin Series

Clean Code

A Handbook of Agile Software Craftsmanship

Foreword by James O. Coplien

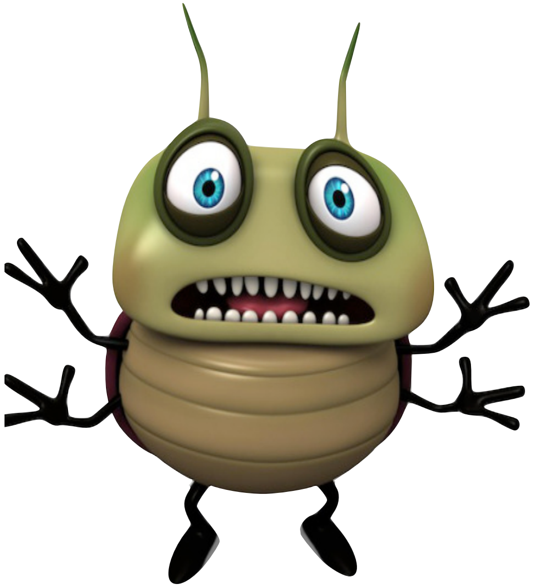
Robert C. Martin

SOFTWARE ARCHITECTURE

is about managing

risks of change

Time

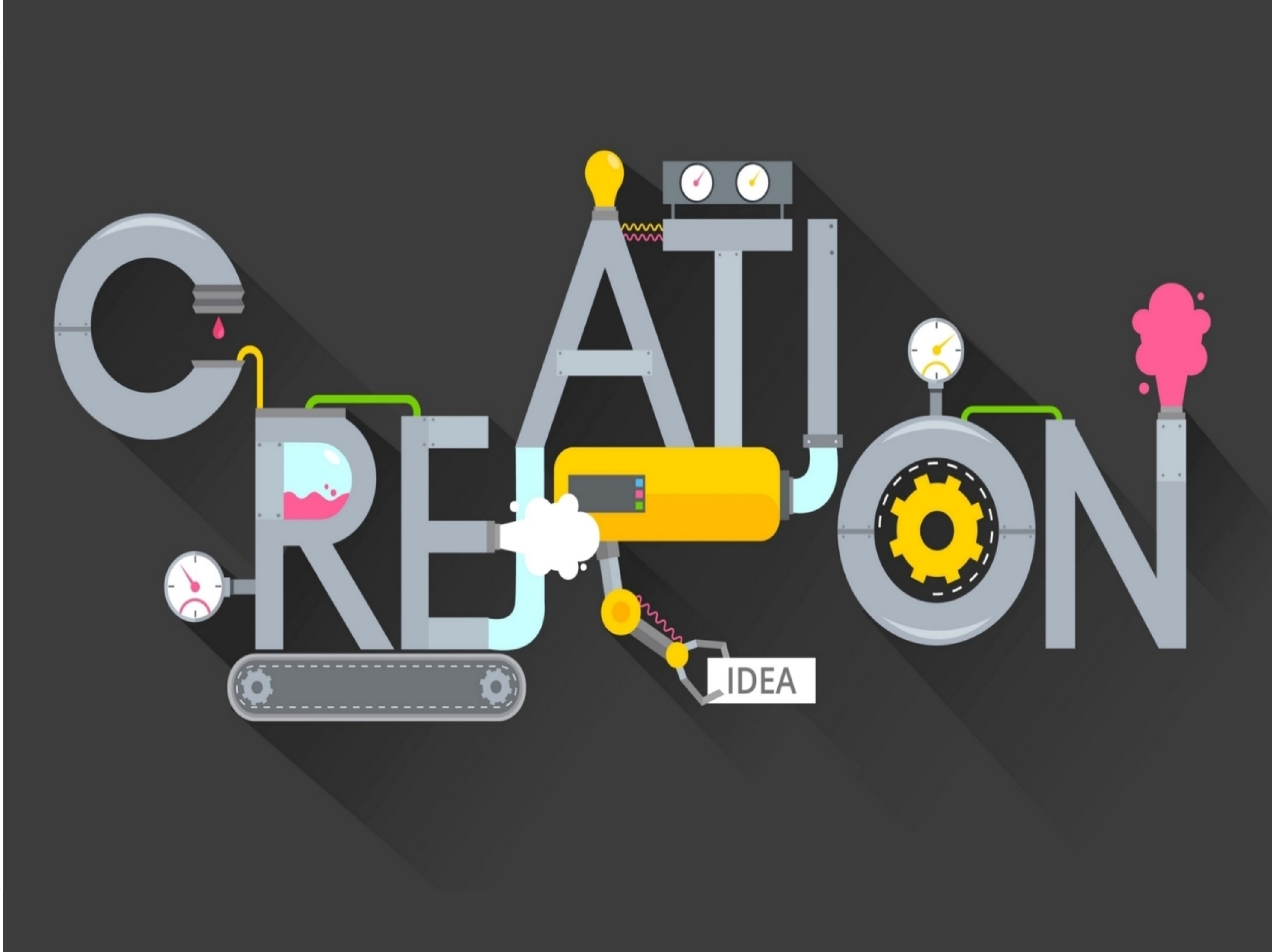


Defects

Risks

come less from

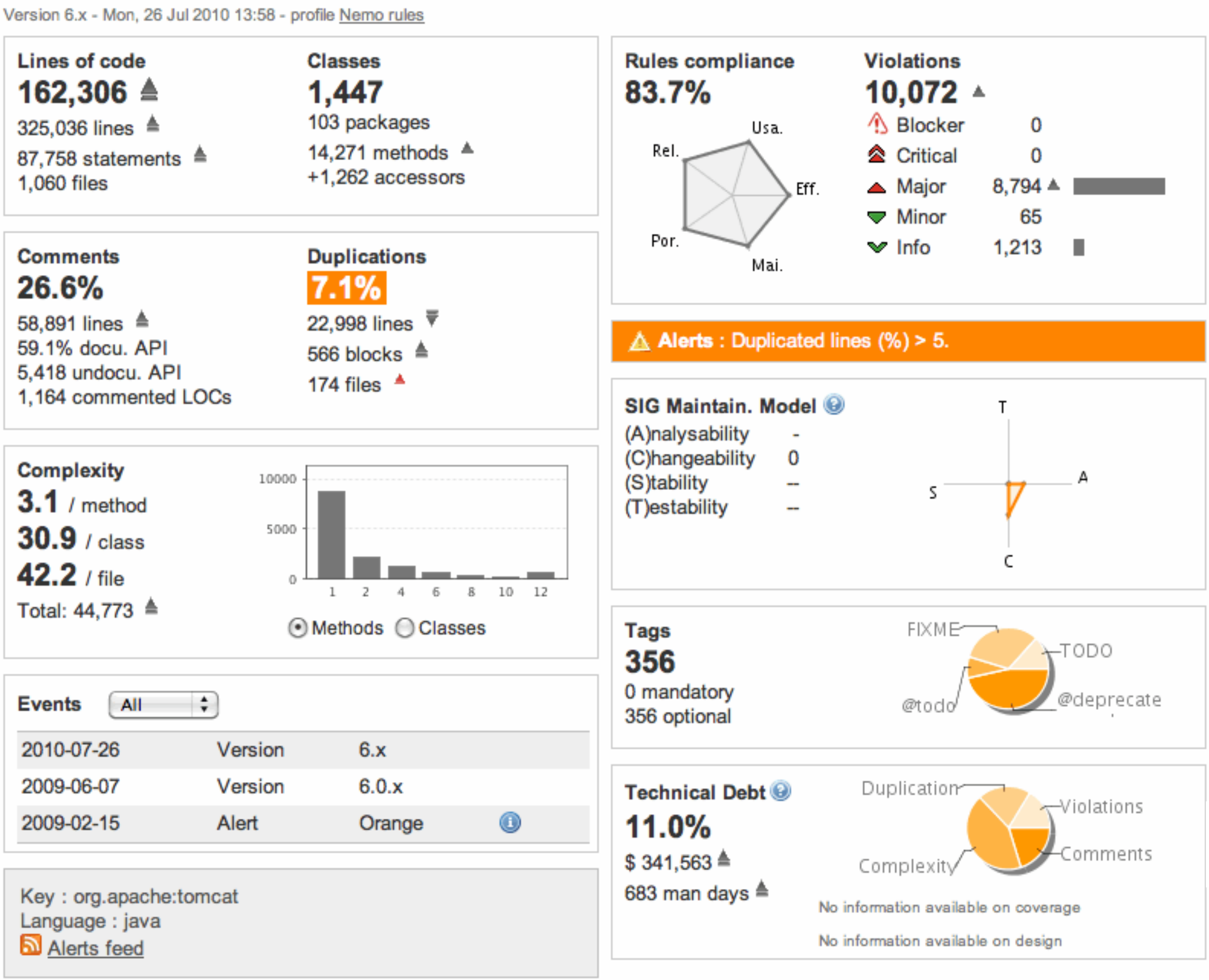
the code itself...


CREATION
IDEA

CHANGE

Rigidity
Fragility
CHANGE
Immobility
Viscosity

Quality analysis tools…



Version 6.x - Mon, 26 Jul 2010 13:58 - profile Nemo rules

Lines of code
162,306 ▲
325,036 lines ▲
87,758 statements ▲
1,060 files

Classes
1,447
103 packages
14,271 methods ▲
+1,262 accessors

Comments
26.6%
58,891 lines ▲
59.1% docu. API
5,418 undocu. API
1,164 commented LOCs

Duplications
7.1%
22,998 lines ▼
566 blocks ▲
174 files ▲

Complexity
3.1 / method
30.9 / class
42.2 / file
Total: 44,773 ▲

◉ Methods ○ Classes

Events All

2010-07-26	Version	6.x	
2009-06-07	Version	6.0.x	
2009-02-15	Alert	Orange	ⓘ

Key : org.apache:tomcat
Language : java
Alerts feed

Rules compliance
83.7%

Violations
10,072 ▲

Blocker	0	
Critical	0	
Major	8,794 ▲	▬
Minor	65	
Info	1,213	▮

⚠ Alerts : Duplicated lines (%) > 5.

SIG Maintain. Model
(A)nalysability -
(C)hangeability 0
(S)tability --
(T)estability --

Tags
356
0 mandatory
356 optional

Technical Debt
11.0%
$ 341,563 ▲
683 man days ▲

No information available on coverage
No information available on design

sonar qube

pmd

CAST

checkstyle

… provide

narrow view of quality

…as they analyse

only CODE

and

only NOW

…and therefore

miss the

cause of problems

CHANGE

Timelines are better that single numbers
but are still missing the point:

they capture the trends of the effect,
not change itself

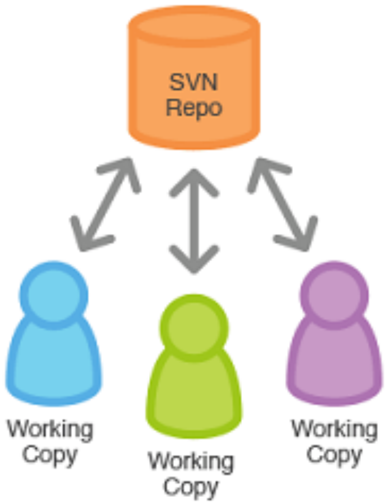
Code & Team
Change

code & team dynamics are **ignored in tools**

because information

is not available in the code

It is available!



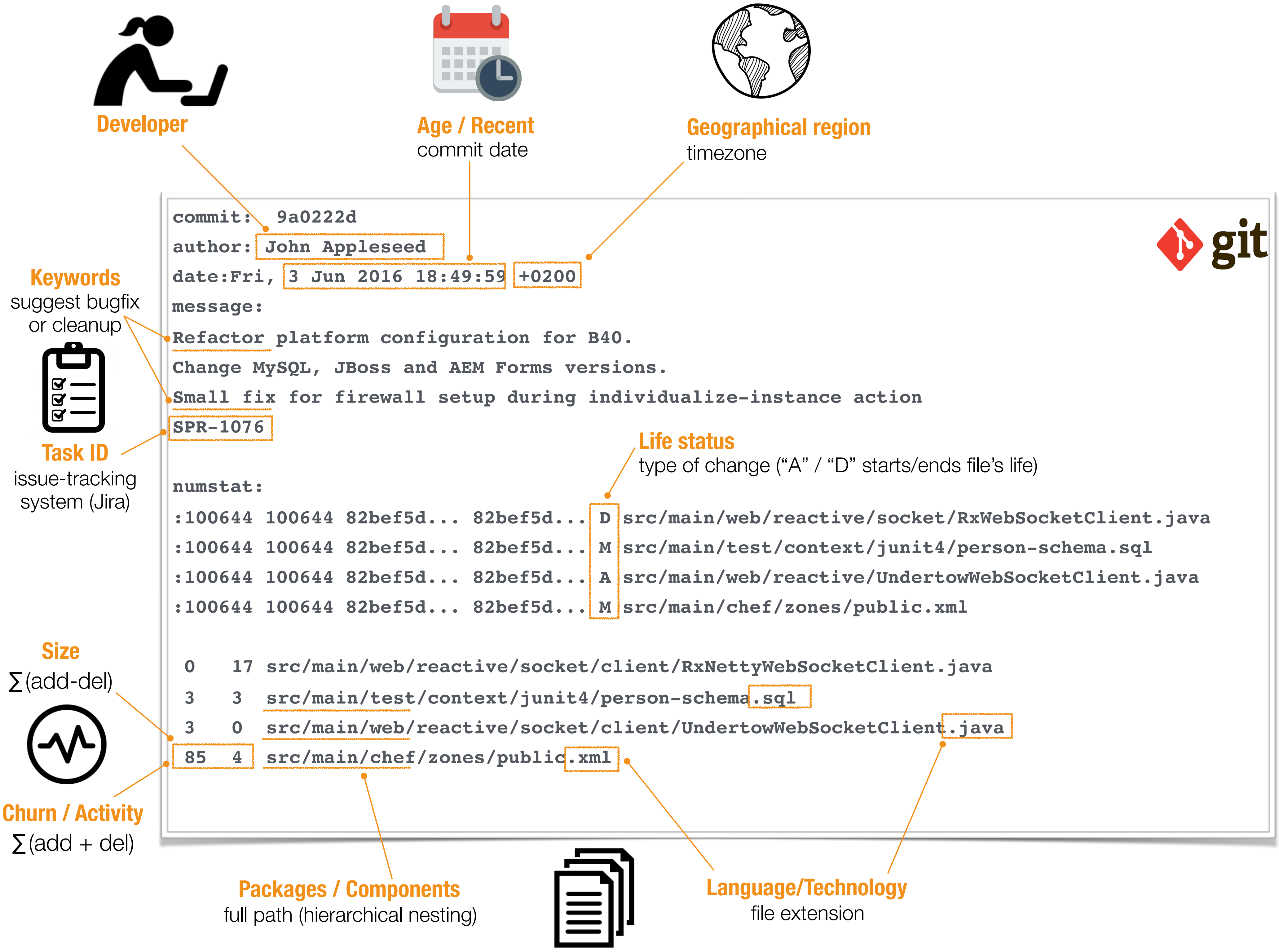
Source-control systems

git

SUBVERSION

IBM Rational ClearCase

Visual Studio Team Foundation Server



```
commit:  9a0222d
author: John Appleseed
date:Fri, 3 Jun 2016 18:49:59 +0200
message:
Refactor platform configuration for B40.
Change MySQL, JBoss and AEM Forms versions.
Small fix for firewall setup during individualize-instance action
SPR-1076

numstat:
:100644 100644 82bef5d... 82bef5d... D src/main/web/reactive/socket/RxWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/test/context/junit4/person-schema.sql
:100644 100644 82bef5d... 82bef5d... A src/main/web/reactive/UndertowWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/chef/zones/public.xml

 0   17  src/main/web/reactive/socket/client/RxNettyWebSocketClient.java
 3    3  src/main/test/context/junit4/person-schema.sql
 3    0  src/main/web/reactive/socket/client/UndertowWebSocketClient.java
 85   4  src/main/chef/zones/public.xml
```

Developer

Age / Recent
commit date

Geographical region
timezone

Keywords
suggest bugfix
or cleanup

Task ID
issue-tracking
system (Jira)

Life status
type of change (“A” / “D” starts/ends file’s life)

Size
∑(add-del)

Churn / Activity
∑(add + del)

Packages / Components
full path (hierarchical nesting)

Language/Technology
file extension

CHRONOS Dx

are software analysis tools that

discover risks

caused by a project's

Code Team

The answers to the

key questions

Code

Team

no code access

language independent

Code

A

Code
Dynamics

Code Evolution: Key Questions

Growth

How did it **grow** over time?

Are there any **size anomalies**?

Is the **growth "organic" or abrupt**?

Are there signs of **systematic refactoring**?

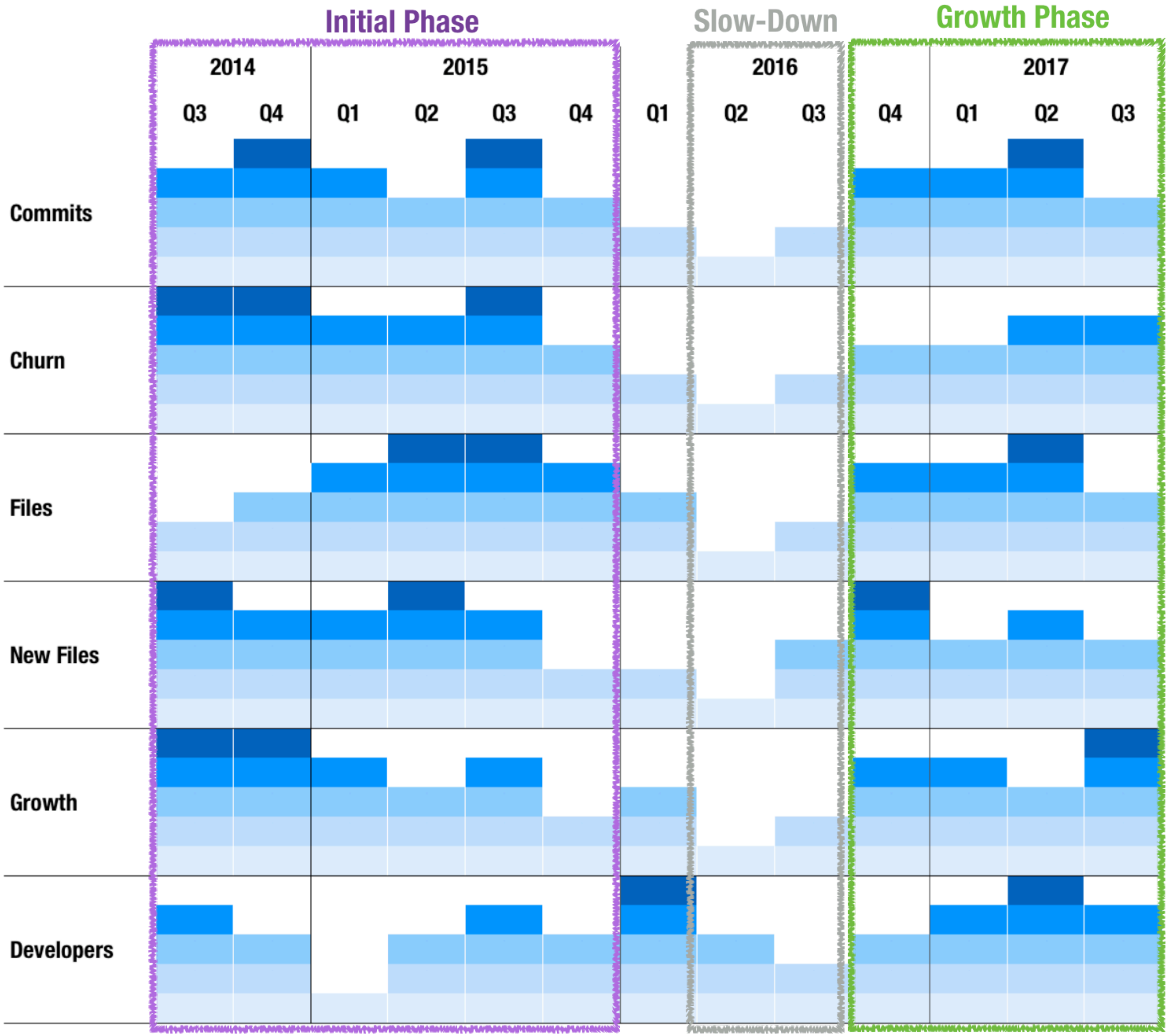
Stability

Which parts **change most** frequently?

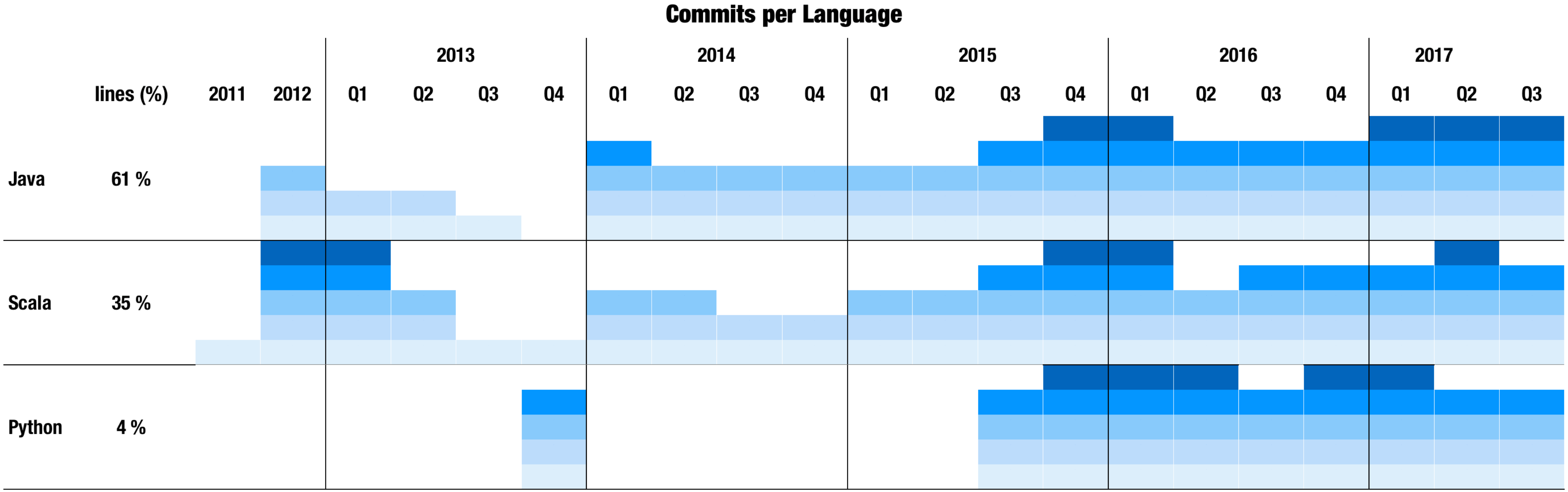
Which parts have been **recently changed**?

Are there files that change for **(too) many different reasons**?

How **old** is the system? How did it **grow**?



How did **language usage** evolve in time

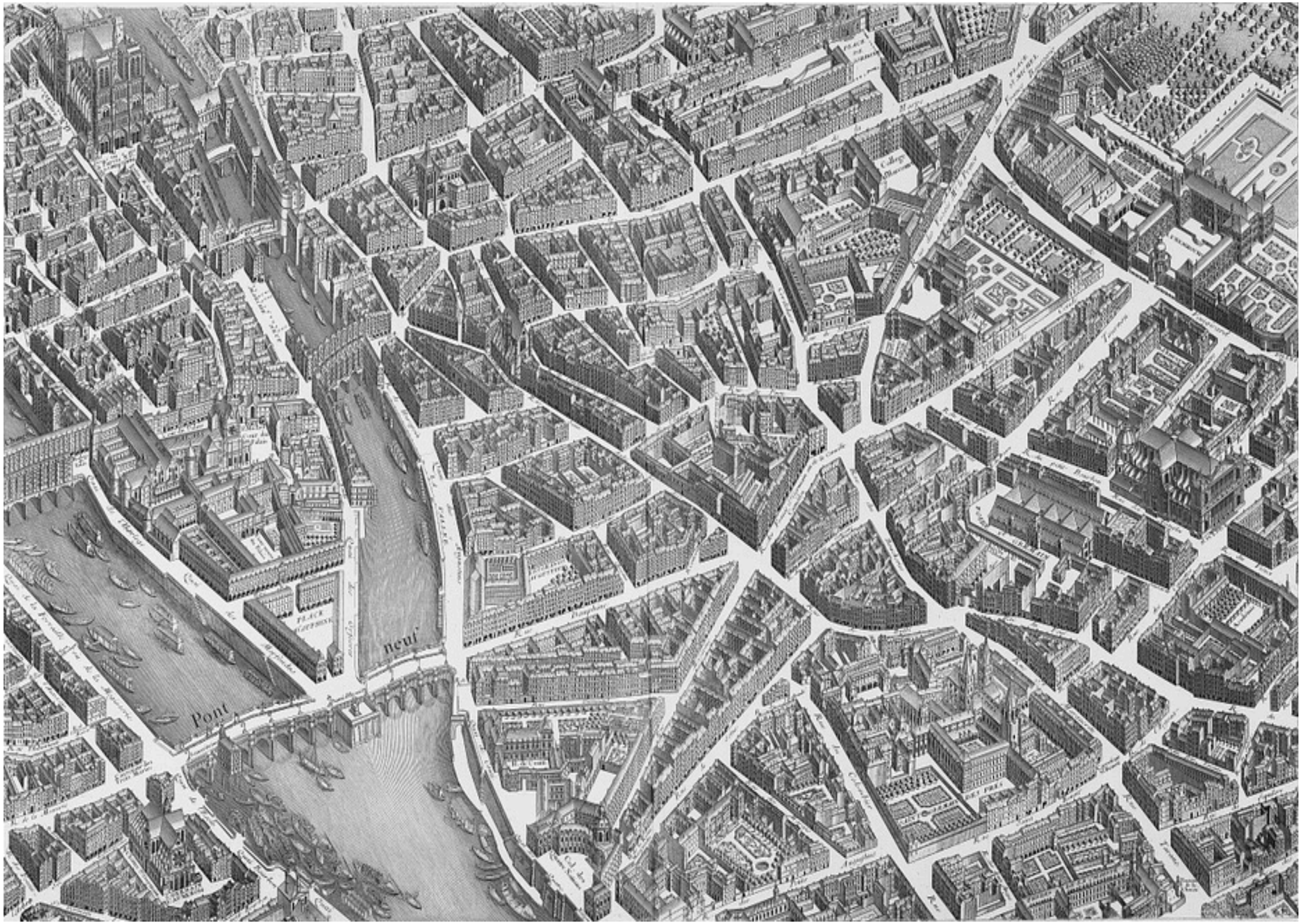

Commits per Language

Seeing

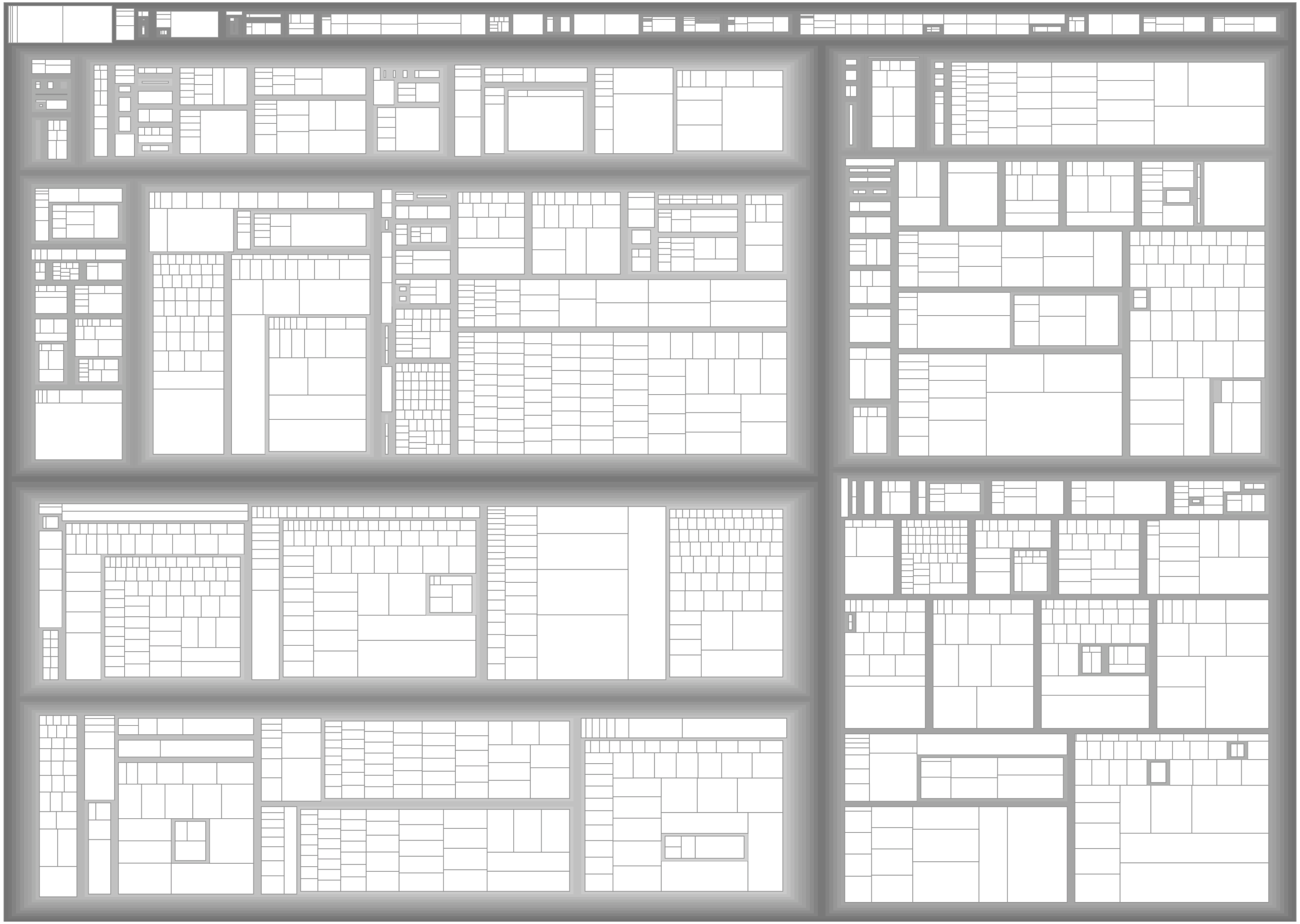
is better than

reading code

Technique: System Map

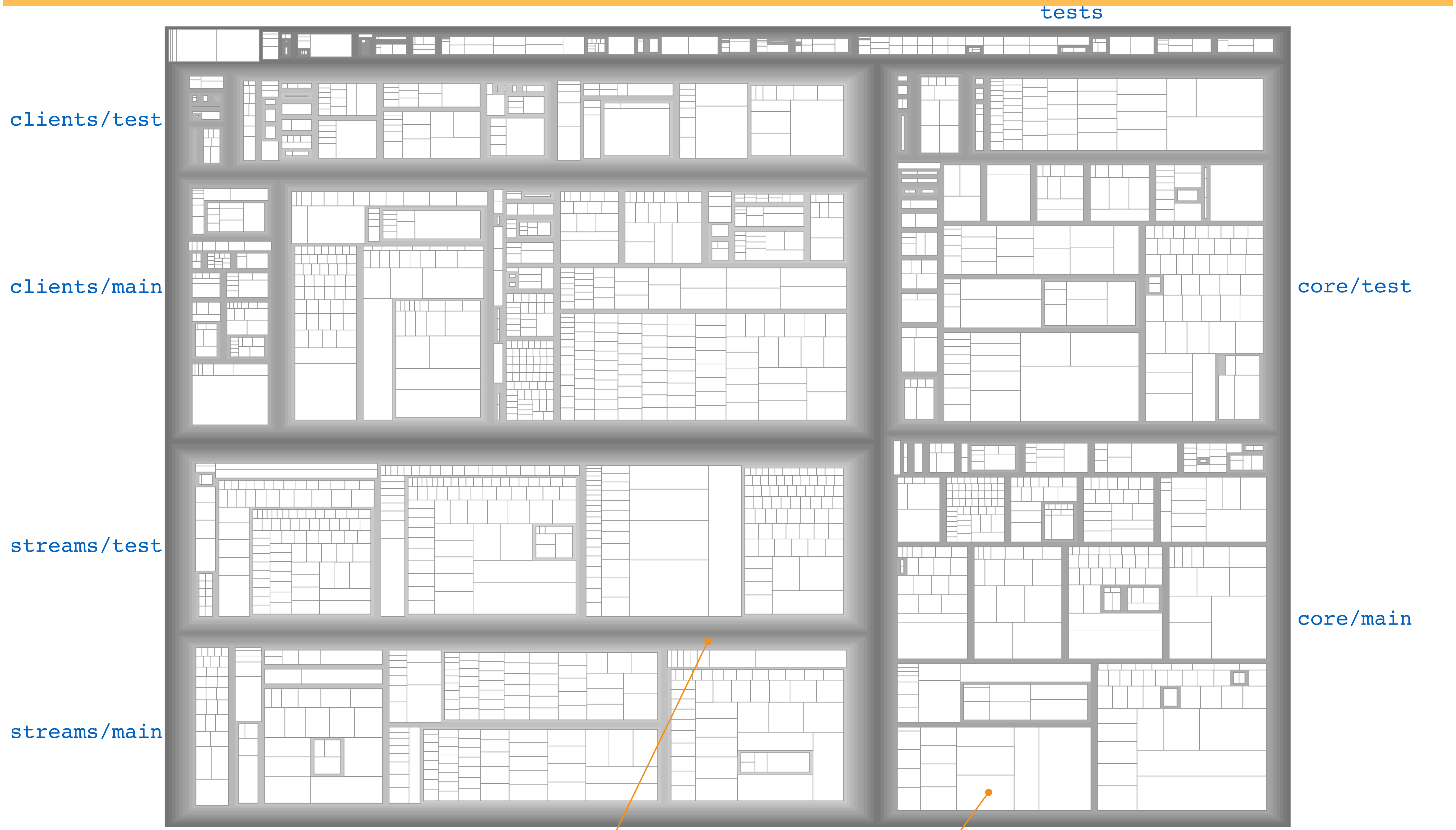


Technique: System Map



APACHE kafka®
A distributed streaming platform

Technique: System Map

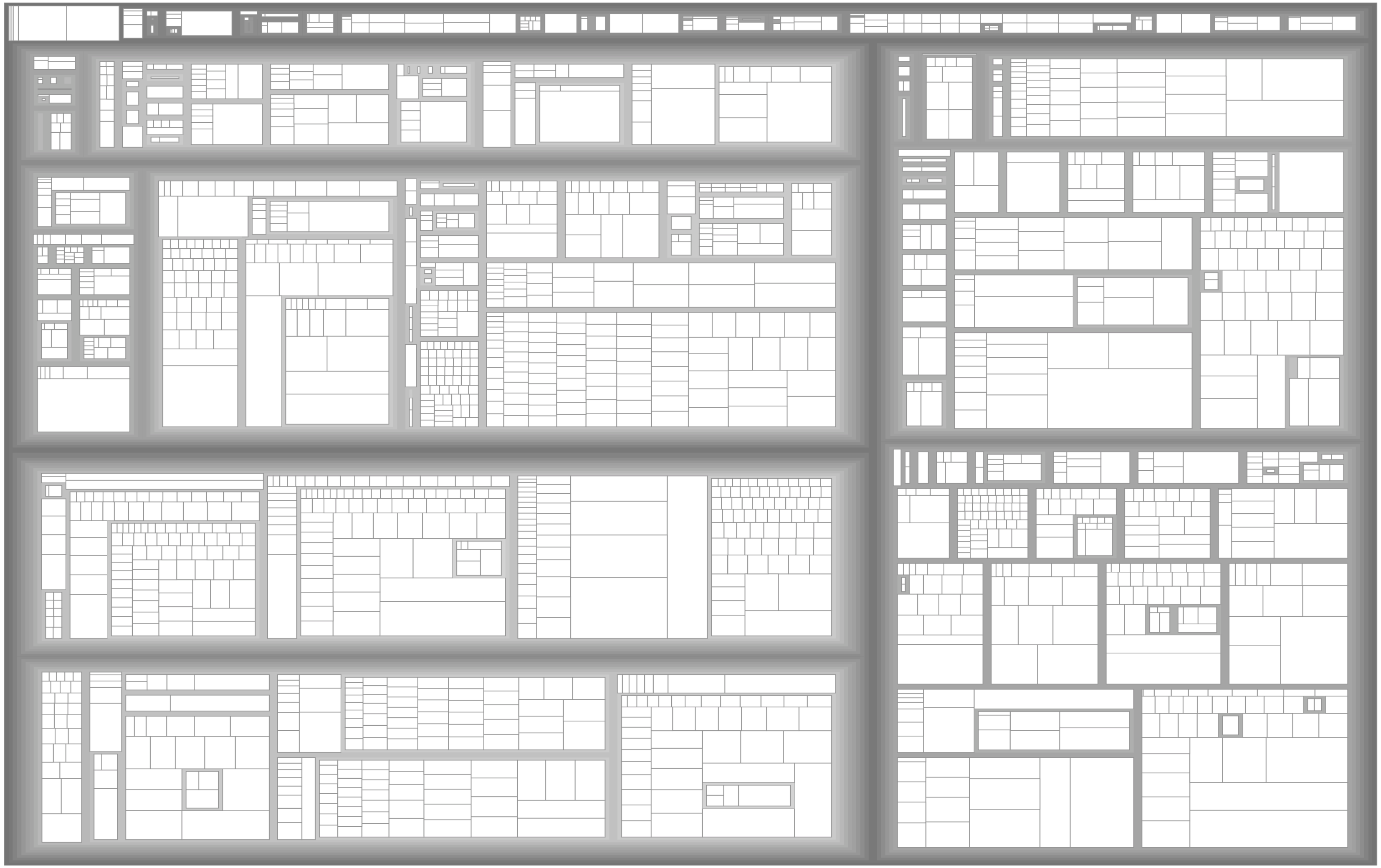


Folders
gray borders
hierarchical

Files
rectangles
size proportional

APACHE kafka®
A distributed streaming platform

Which parts were **recently active**



🟩 **active**

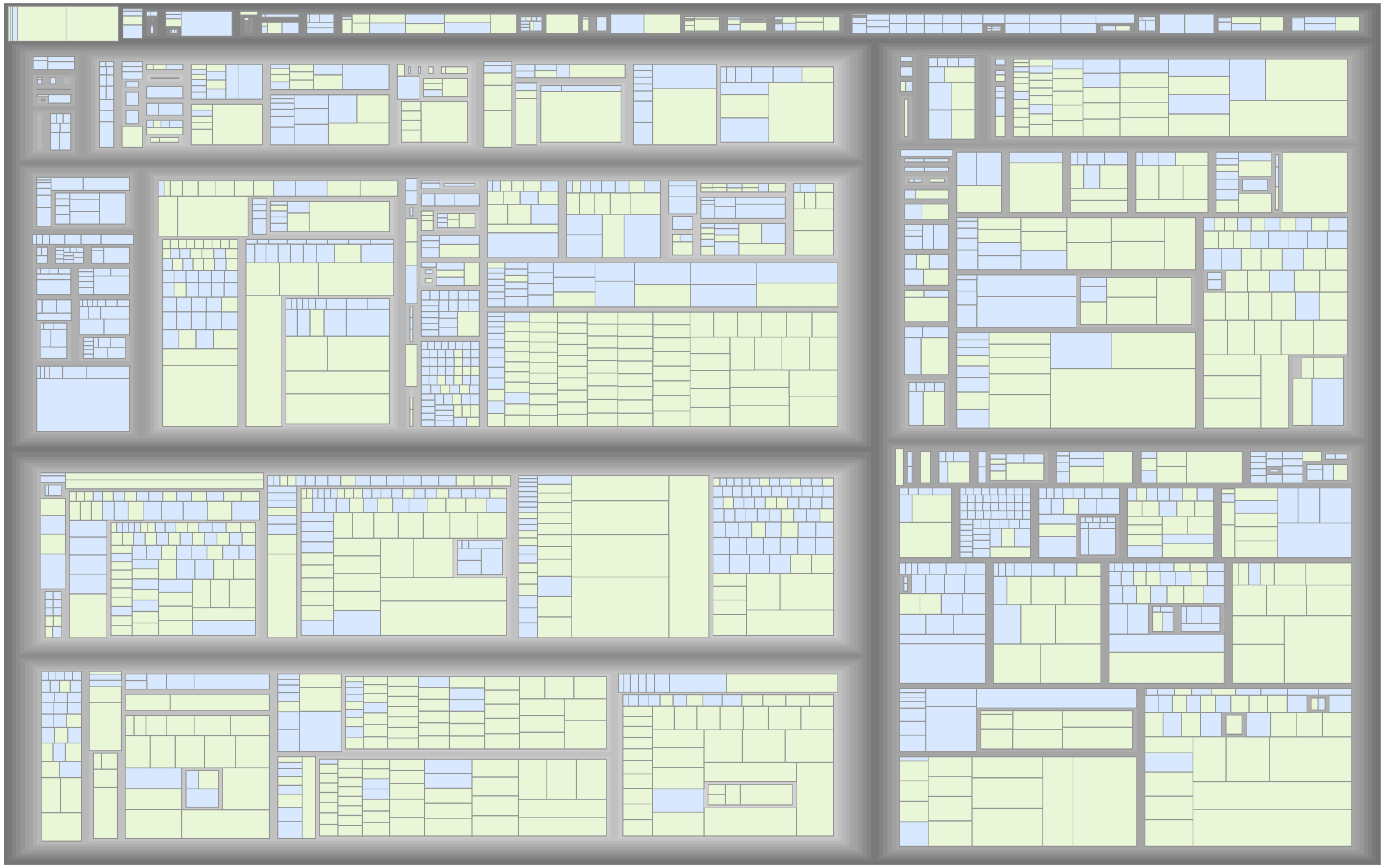
changed over the
latest 3 months

🟦 **idle**

no activity in
latest 3 months

Recently active means that the file has been changed at least once over the last 12 weeks (cca. 3 months)

Which parts were **recently active**

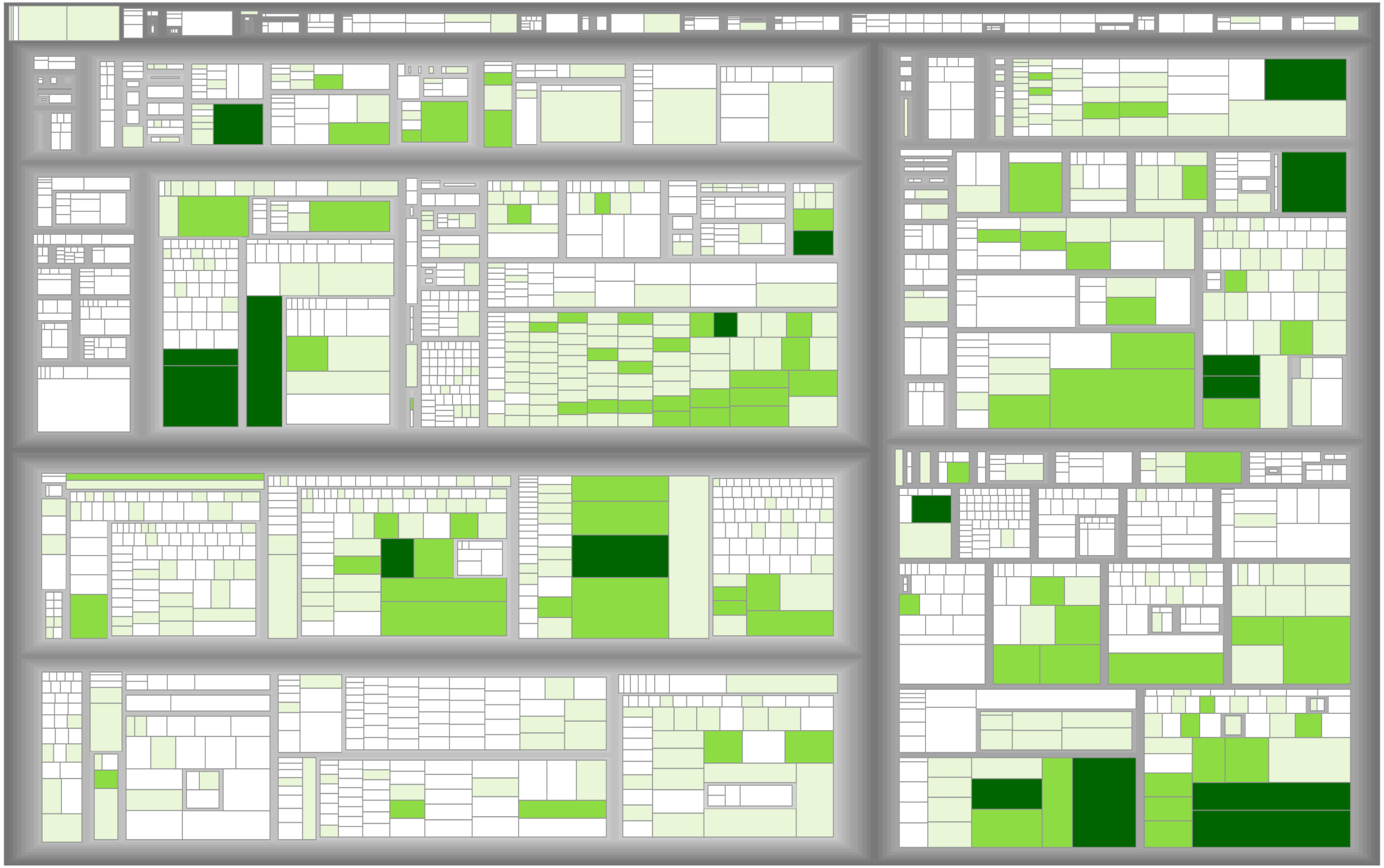


Recently active means that the file has been changed at least once over the last 12 weeks (cca. 3 months)

🟩 **active file**

🟦 **idle file**

Which parts are **recently changed heavily**

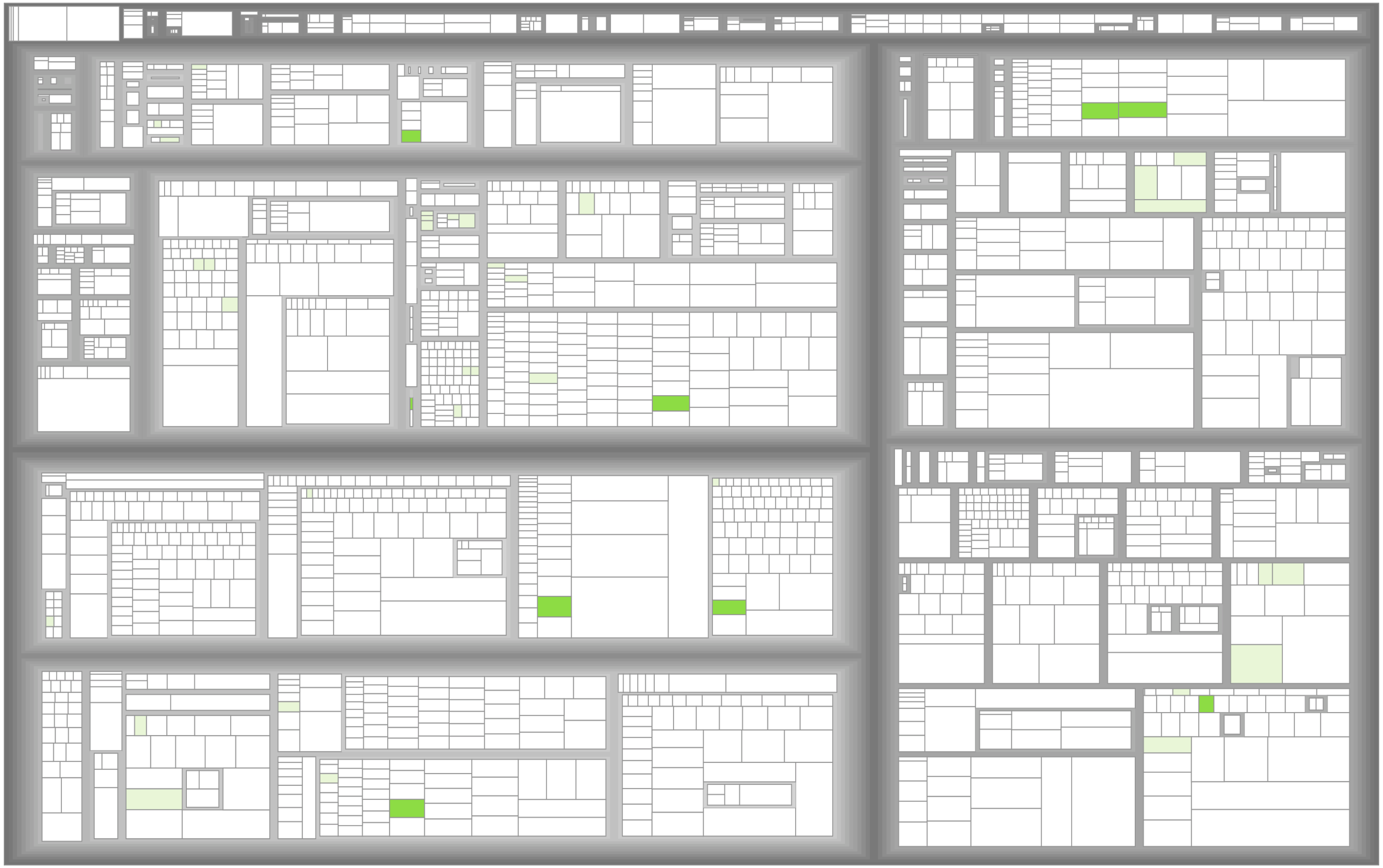


recent changes

1-10	11-20	>20
🟩	🟩	🟩

Recently active means that the file has been changed at least once over the last 12 weeks (cca. 3 months)

Which are the **new files**

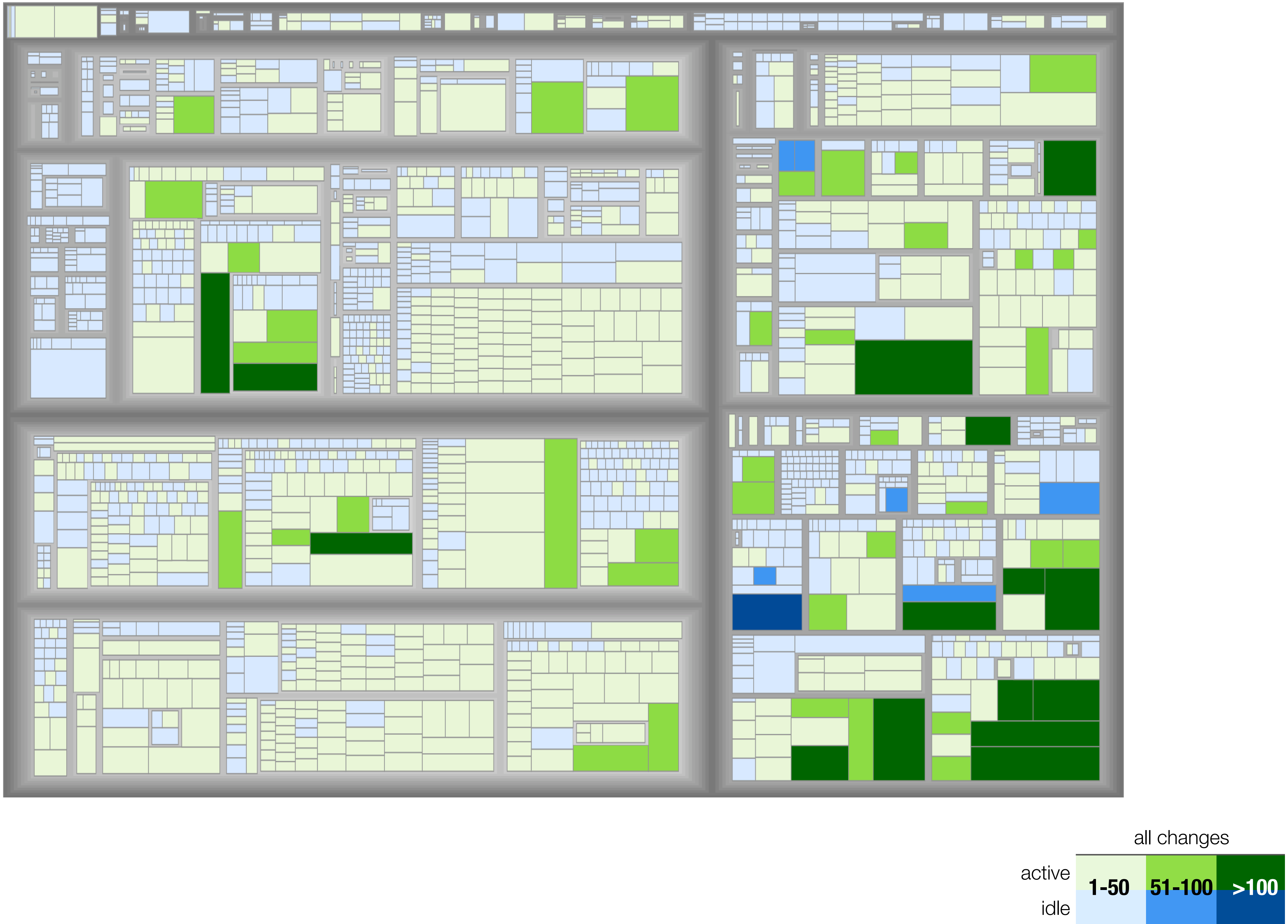


recent changes

1-10	11-20	>20

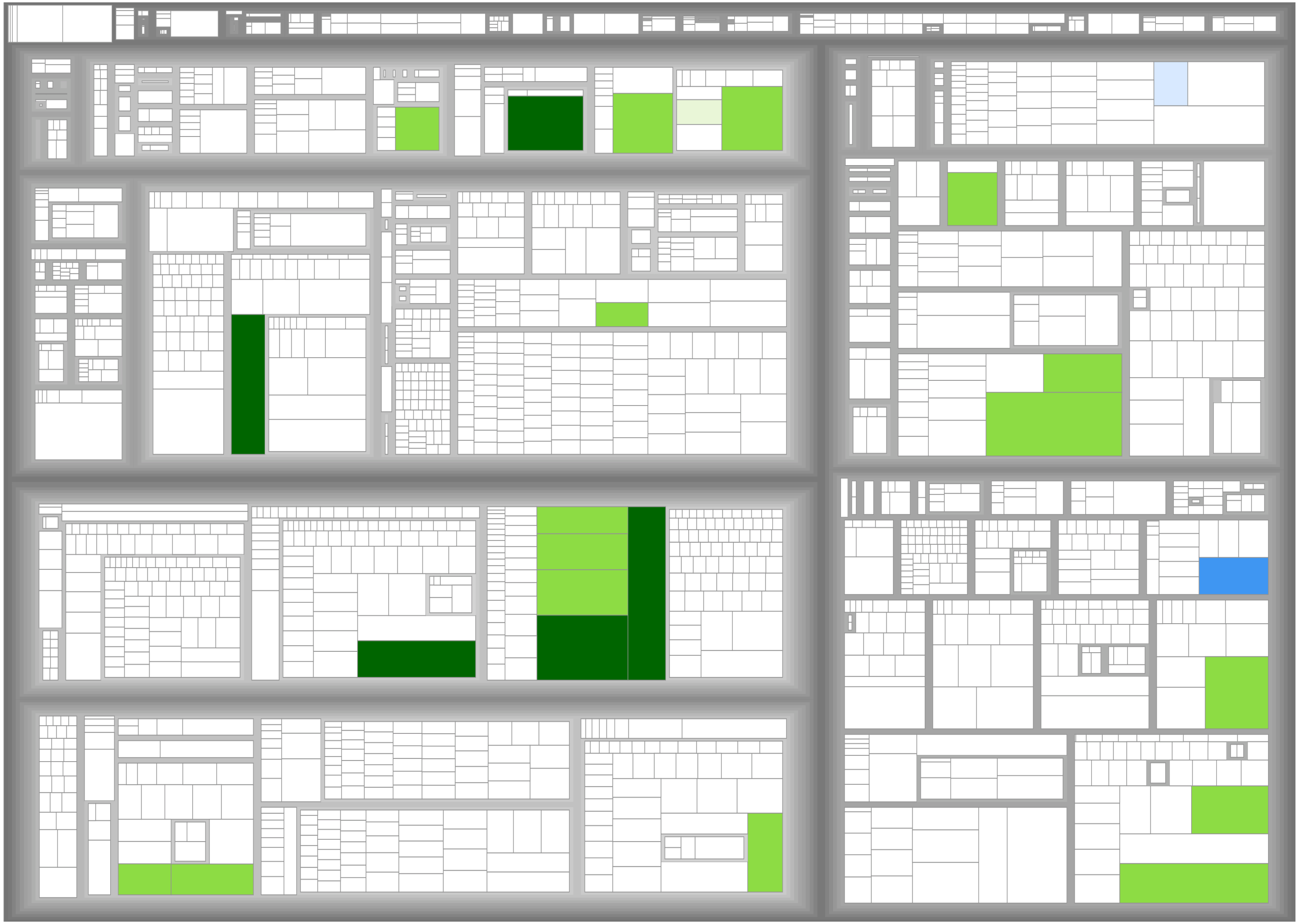
New files are those that have been created in the least 6 weeks

Which parts are **most frequently changed**



Files that **grow abruptly (Supernova)**

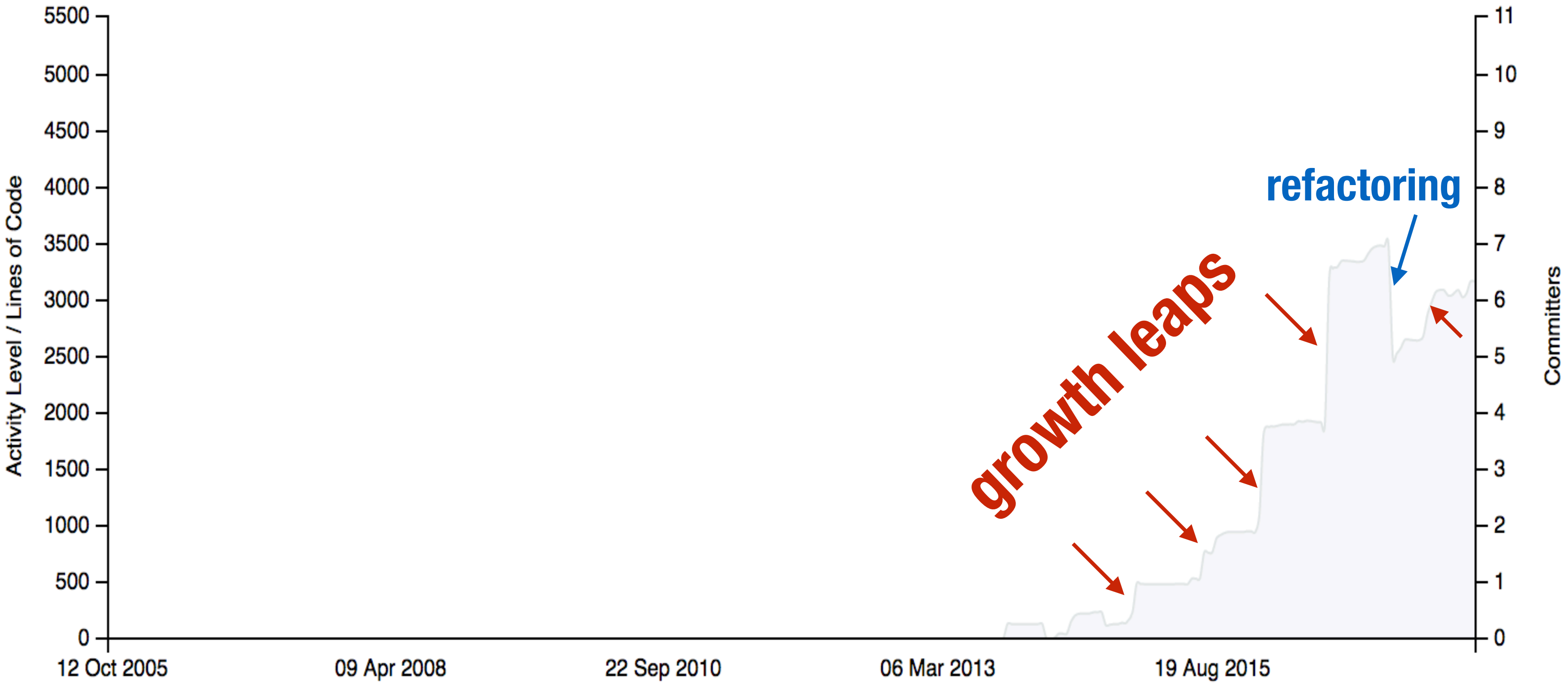
Error-Proneness



	severity		
active	1-4	5-7	8-10
idle			

Supernova Example: AgreementResource.java

Error-Proneness



- Huge file (> 3.000 lines of code)
- Heavy changed (> 350 times)
- Many developers (> 50 contributors)
- Concurrent development (> 5 developers)

Code
Dependencies

Code Dependencies: Key Questions

Dependencies

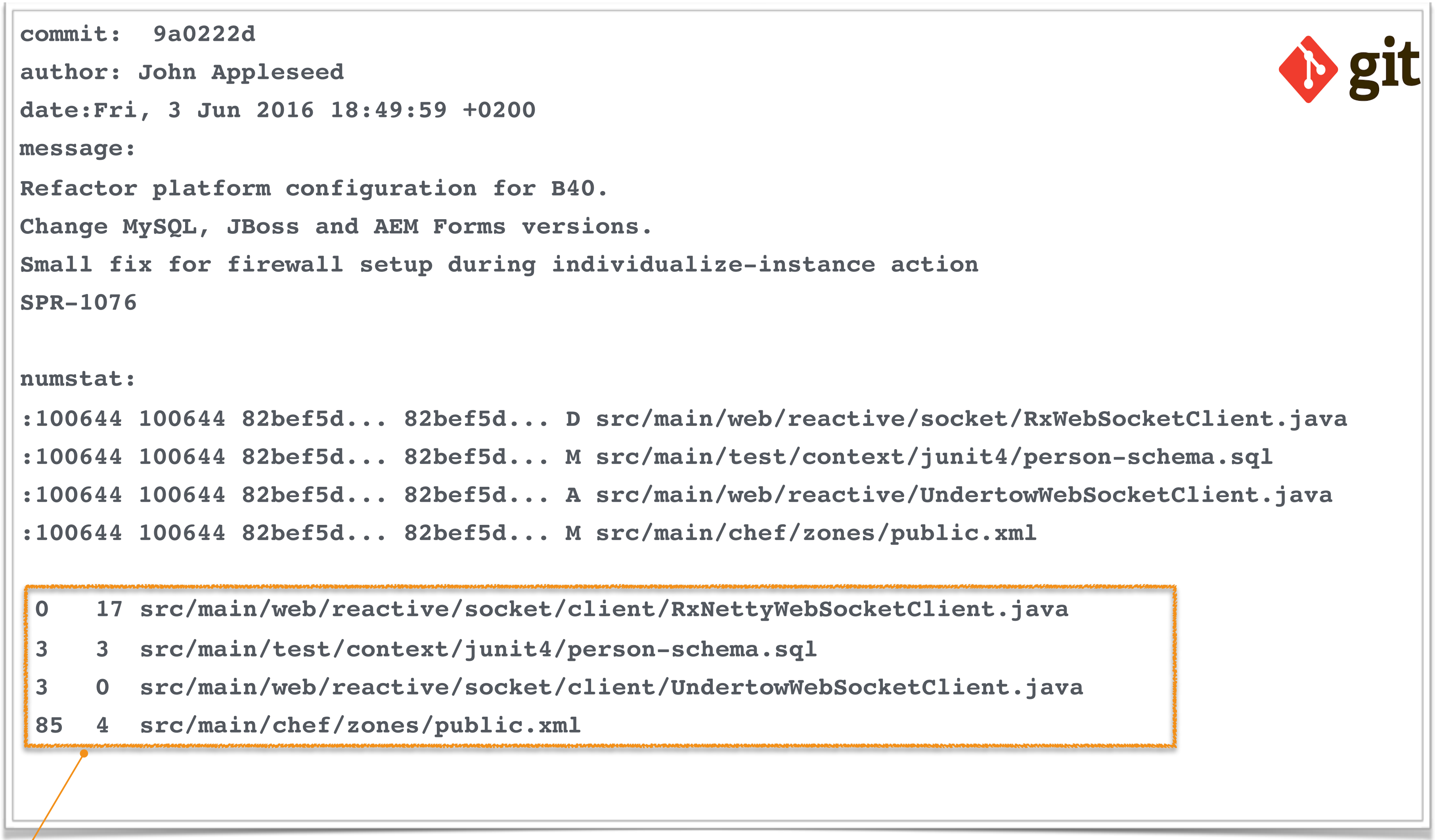
Which files/components **co-change frequently with many others**?

Are there groups of files/components that **frequently co-change**?

Are there co-change **dependencies that cross component boundaries**?

Which tasks that imply changes to **many files scattered over many components**?

Technique: Temporal Coupling

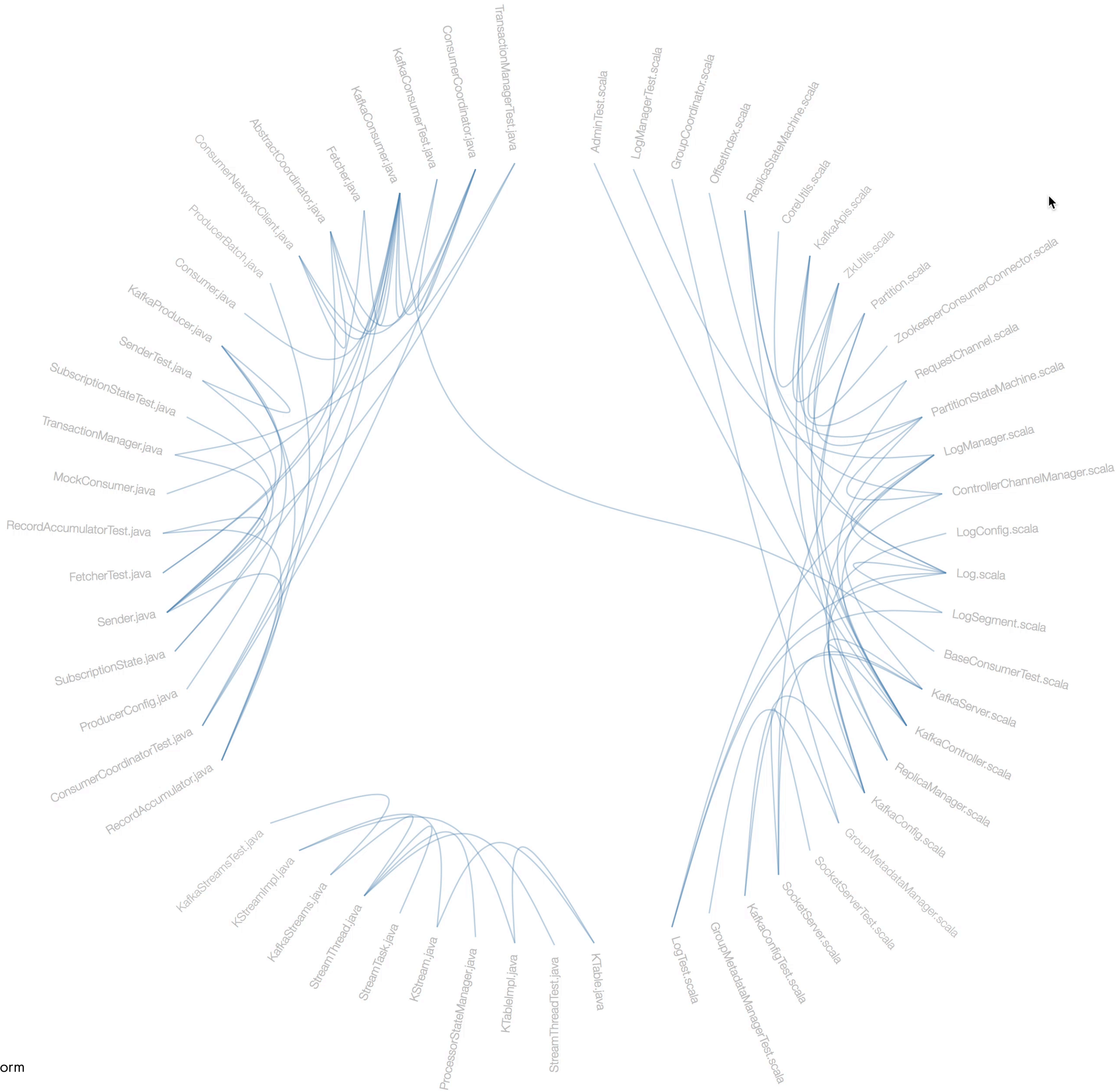


```
commit:  9a0222d
author: John Appleseed
date:Fri, 3 Jun 2016 18:49:59 +0200
message:
Refactor platform configuration for B40.
Change MySQL, JBoss and AEM Forms versions.
Small fix for firewall setup during individualize-instance action
SPR-1076

numstat:
:100644 100644 82bef5d... 82bef5d... D src/main/web/reactive/socket/RxWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/test/context/junit4/person-schema.sql
:100644 100644 82bef5d... 82bef5d... A src/main/web/reactive/UndertowWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/chef/zones/public.xml

0   17 src/main/web/reactive/socket/client/RxNettyWebSocketClient.java
3   3  src/main/test/context/junit4/person-schema.sql
3   0  src/main/web/reactive/socket/client/UndertowWebSocketClient.java
85  4  src/main/chef/zones/public.xml
```

Shared Commits
files part of the
same changeset

Temporal Coupling: pairs of files changed together (*co-changed*) repeatedly in **many commits**

Which files are **temporally coupled**

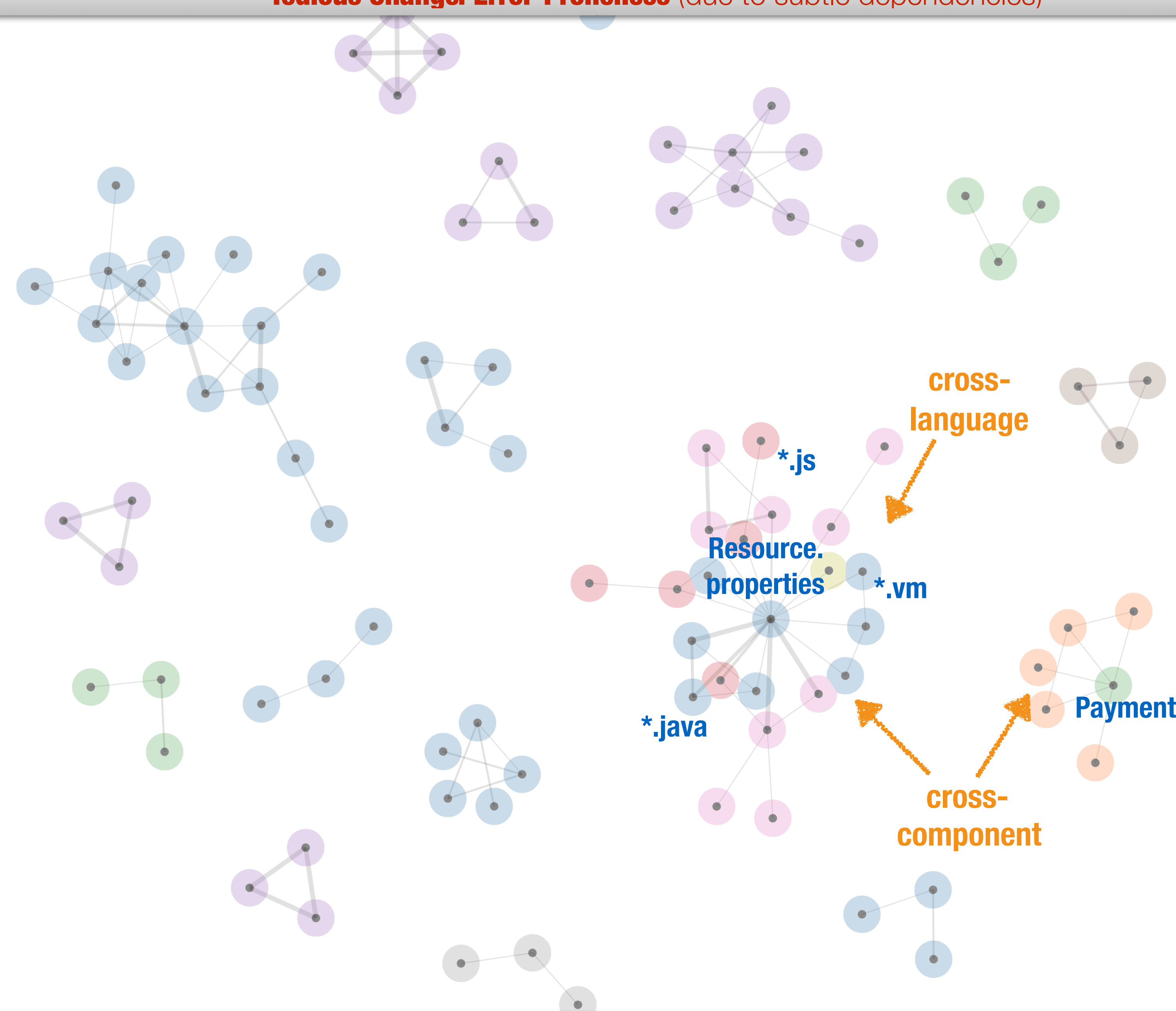


APACHE kafka®
A distributed streaming platform

Temporal Coupling: pairs of files changed together (*co-changed*) repeatedly in **at least 10 commits**

Cross-language and cross-component coupling !

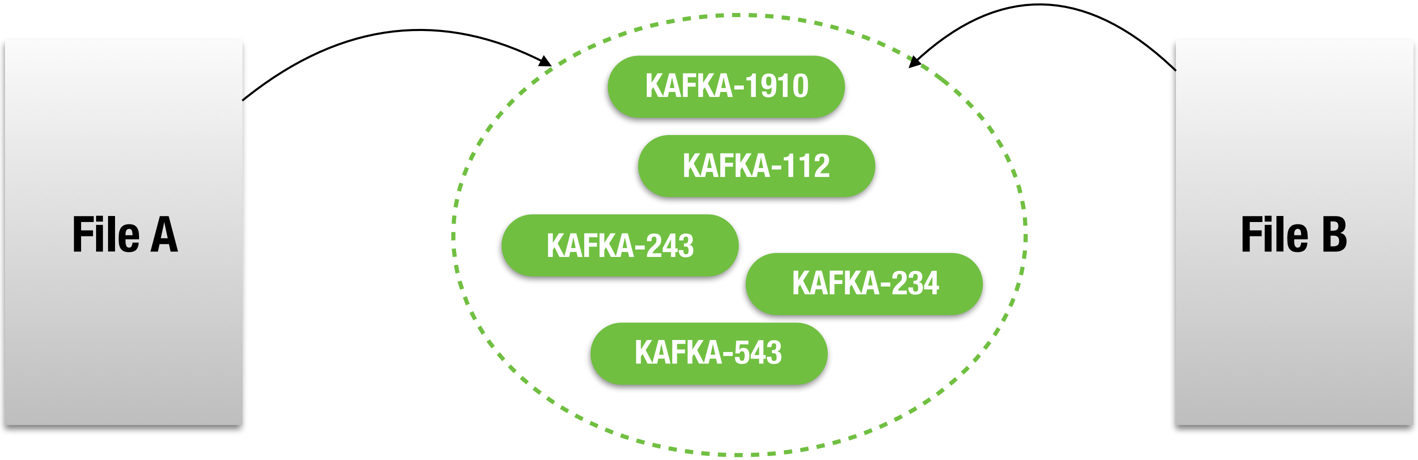
Tedious Change. Error-Proneness (due to subtle dependencies)



sample project

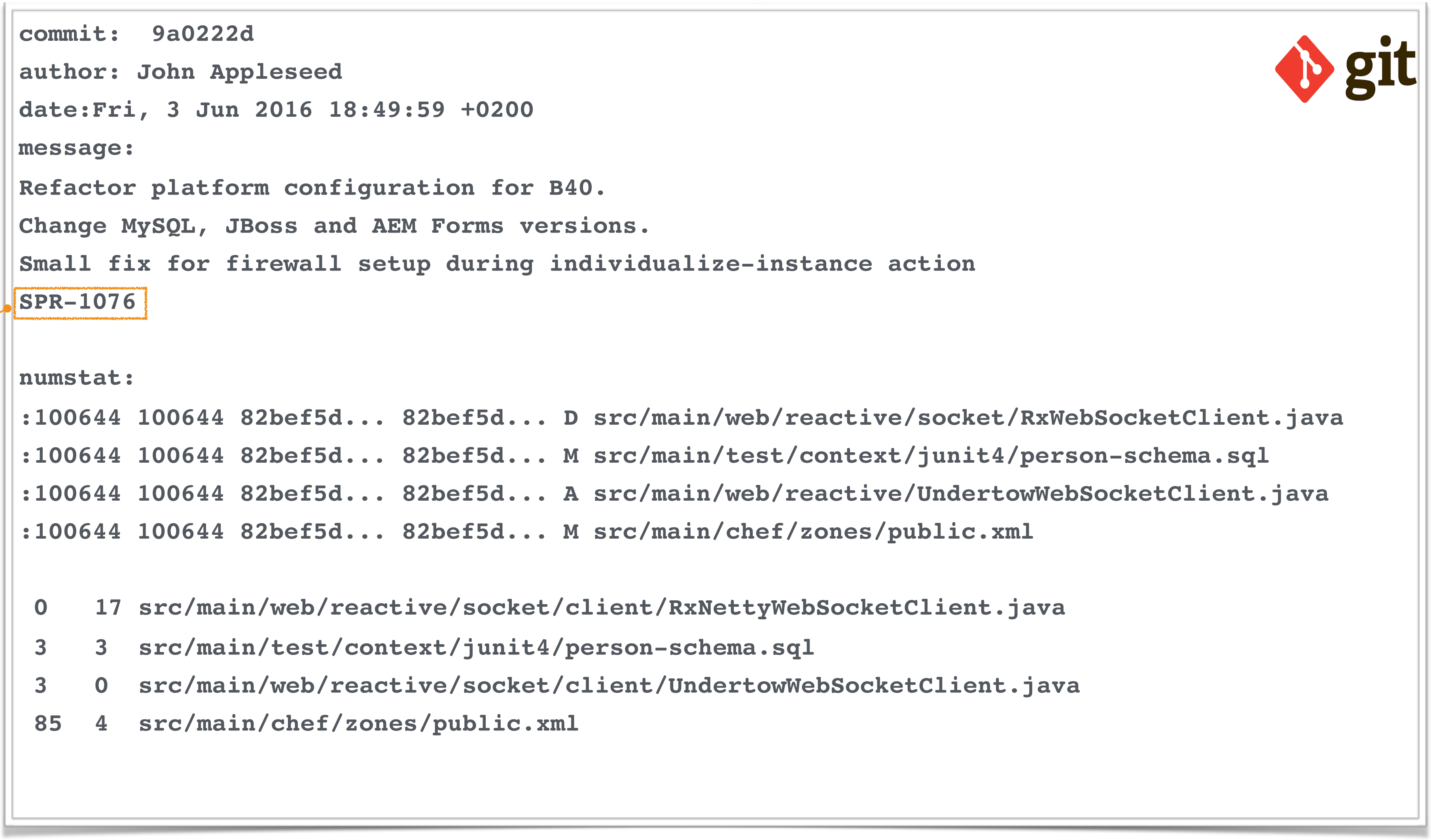
Tasks cause
subtle dependencies

Technique: Shared Tasks Dependencies



Shared Tasks Coupling: pairs of files that are frequently changed in the context of the same task IDs

Technique: Smart commits (traceability links)

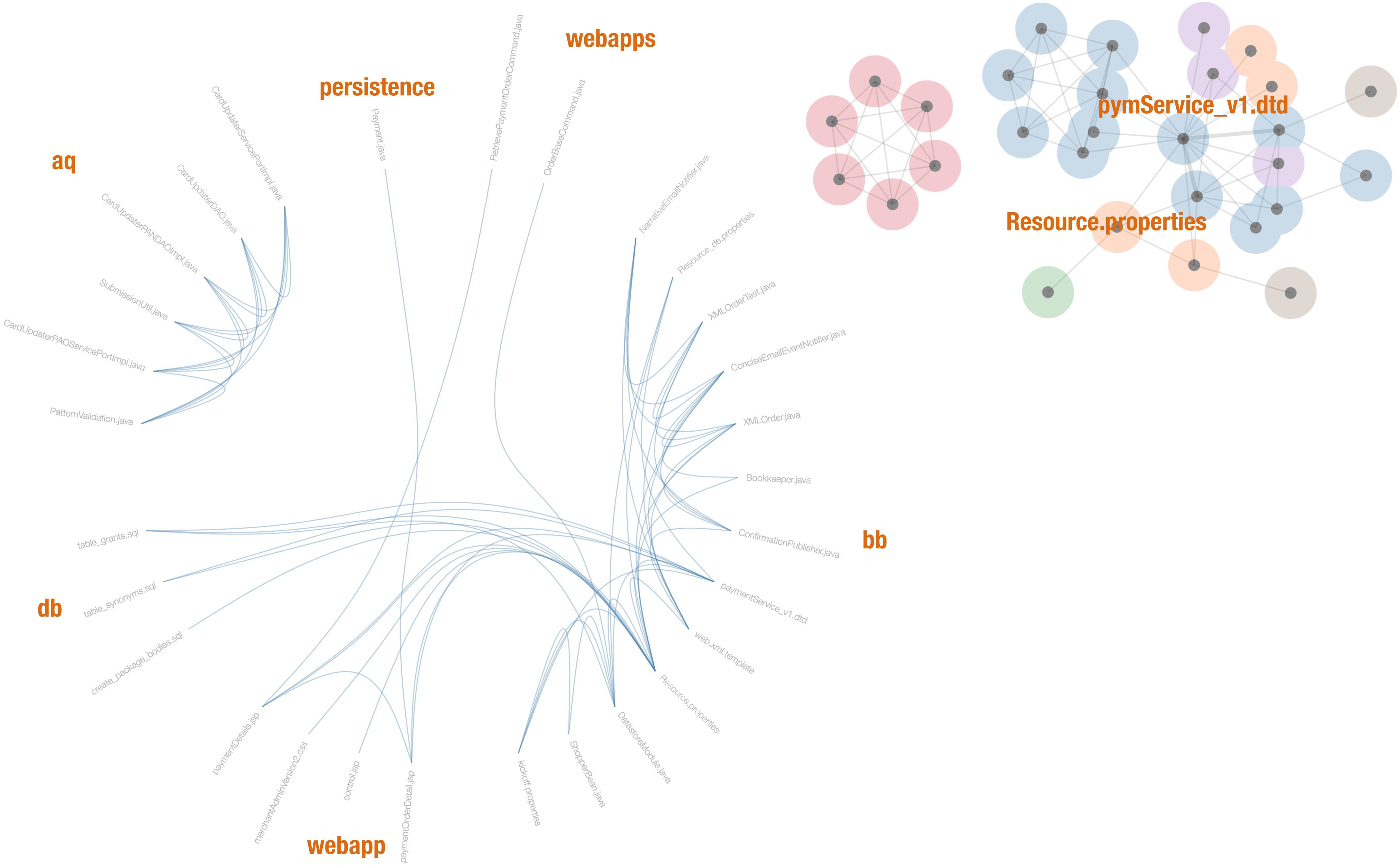


```
commit:  9a0222d
author: John Appleseed
date:Fri, 3 Jun 2016 18:49:59 +0200
message:
Refactor platform configuration for B40.
Change MySQL, JBoss and AEM Forms versions.
Small fix for firewall setup during individualize-instance action
SPR-1076

numstat:
:100644 100644 82bef5d... 82bef5d... D src/main/web/reactive/socket/RxWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/test/context/junit4/person-schema.sql
:100644 100644 82bef5d... 82bef5d... A src/main/web/reactive/UndertowWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/chef/zones/public.xml

 0   17  src/main/web/reactive/socket/client/RxNettyWebSocketClient.java
 3   3   src/main/test/context/junit4/person-schema.sql
 3   0   src/main/web/reactive/socket/client/UndertowWebSocketClient.java
 85  4   src/main/chef/zones/public.xml
```

Task ID
issue-tracking
system (Jira)

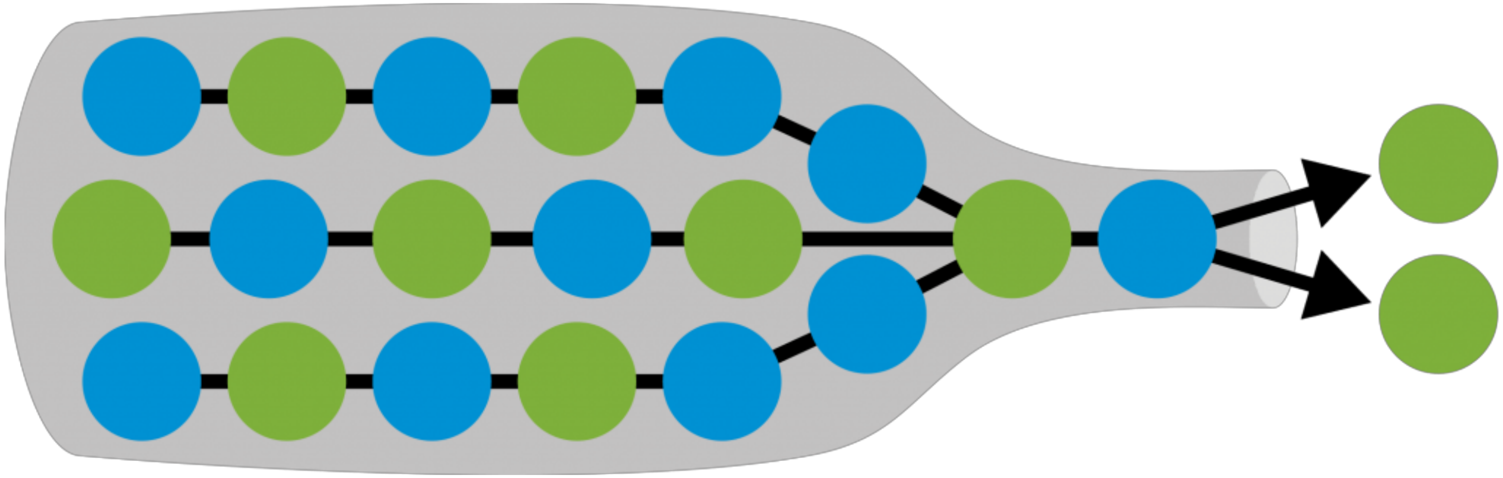
`(task_prefix\s*)(-|:|#){0,1}\s*)([0-9]+)`

Which files are changed by the **same tasks**



Technique: Files changed by many tasks

✓ **central**

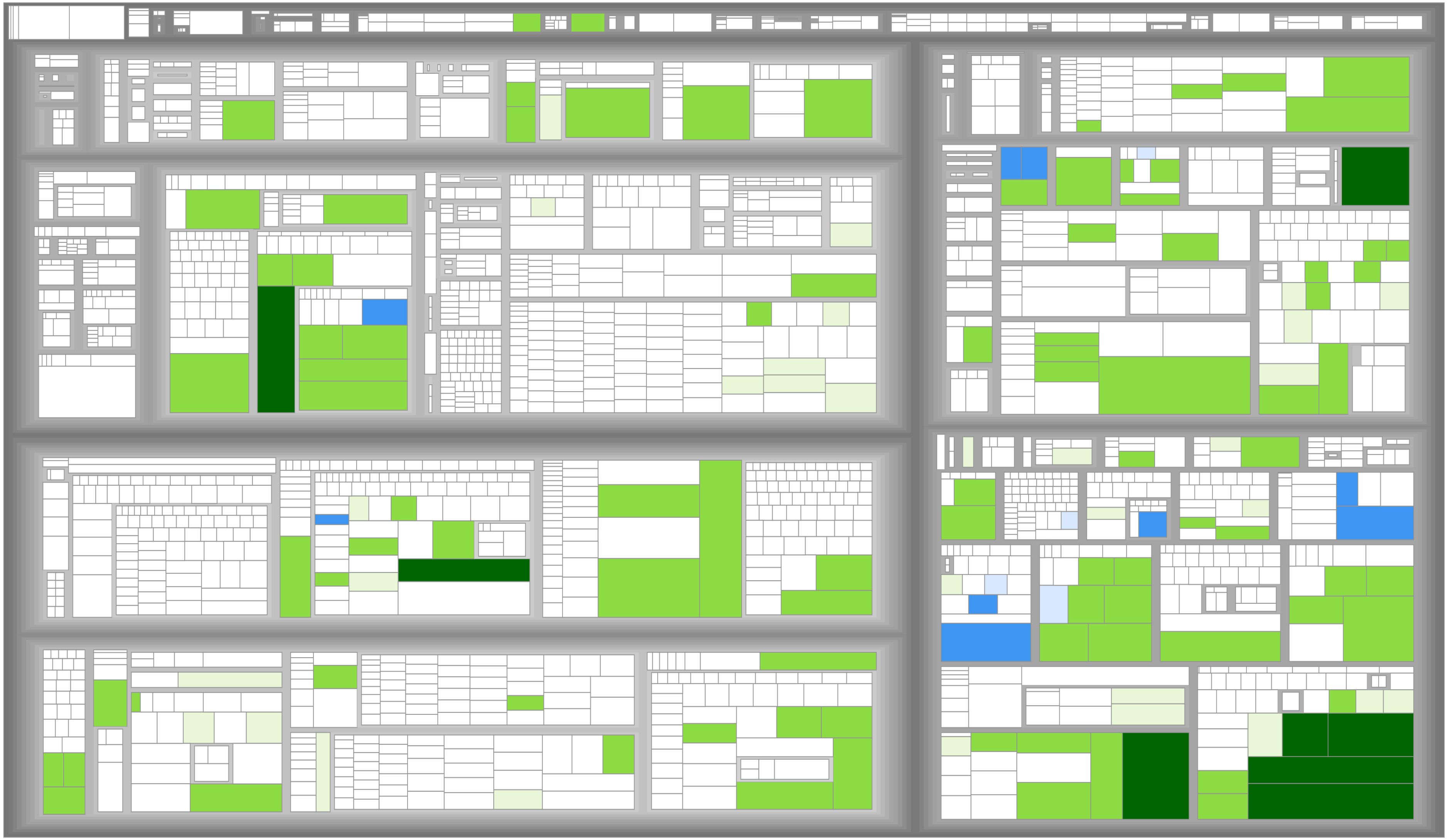


✗ **too many responsibilities**

Bottleneck File: a file that is changes in connection with many different tasks (**at least 20** in this analysis). Such a file is playing a central role in the system, but is also suspect of having multiple responsibilities, if chaged in the context of unrelated tasks.

Bottlenecks: files changed by many tasks

Hard to maintain. Error-Prone.



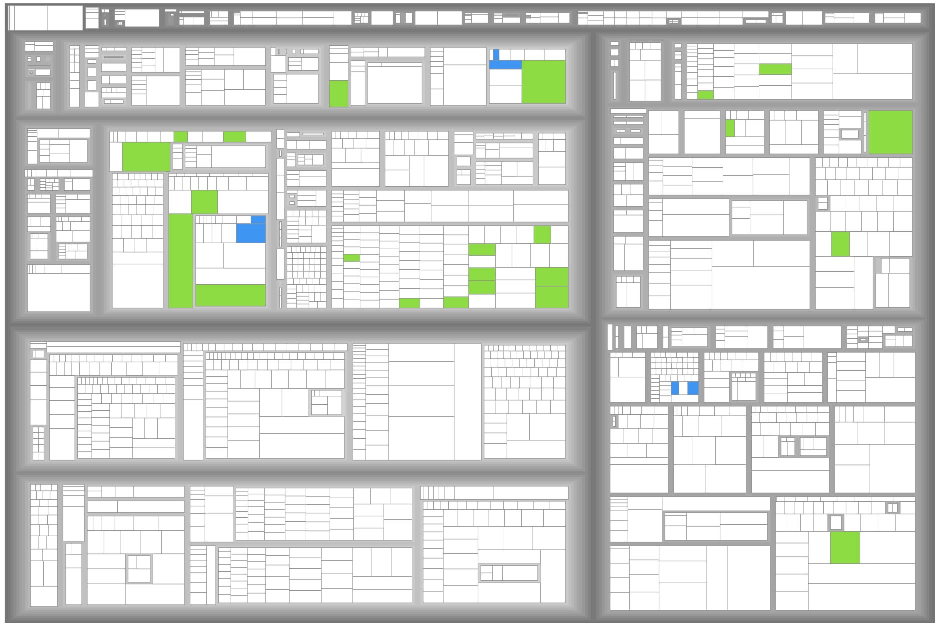
# unique tasks

	1-80	81-150	151-
active			
idle			

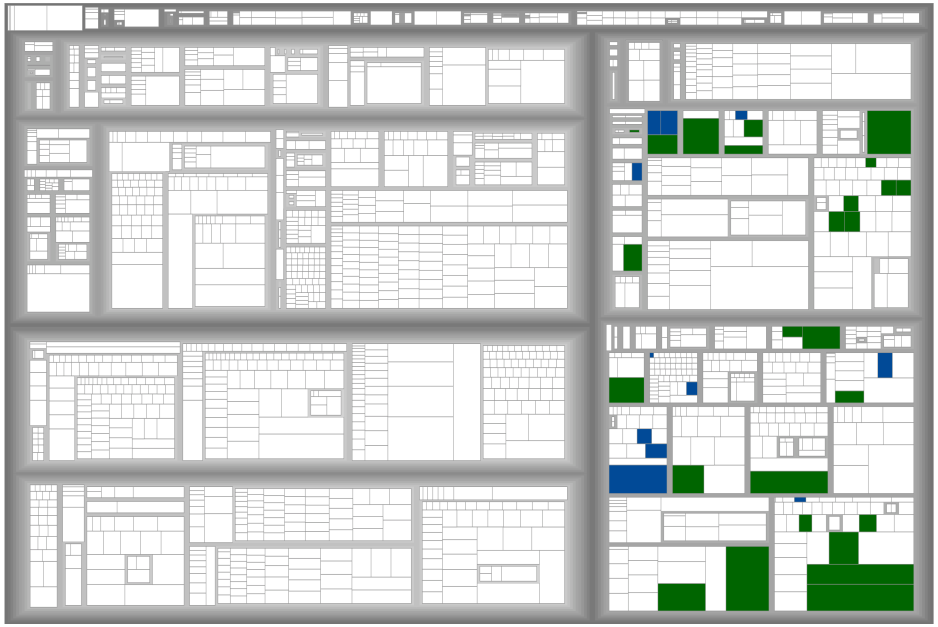
A distributed streaming platform

High Impact Tasks: tasks causing scattered changes

High Cost of (Re-)Testing. Tedious Changes



KAFKA-1910



KAFKA-343

churn / activity

	1-2000	2001-4000	4001-
active	🟩	🟩	🟩
idle	🟦	🟦	🟦

APACHE kafka®
A distributed streaming platform

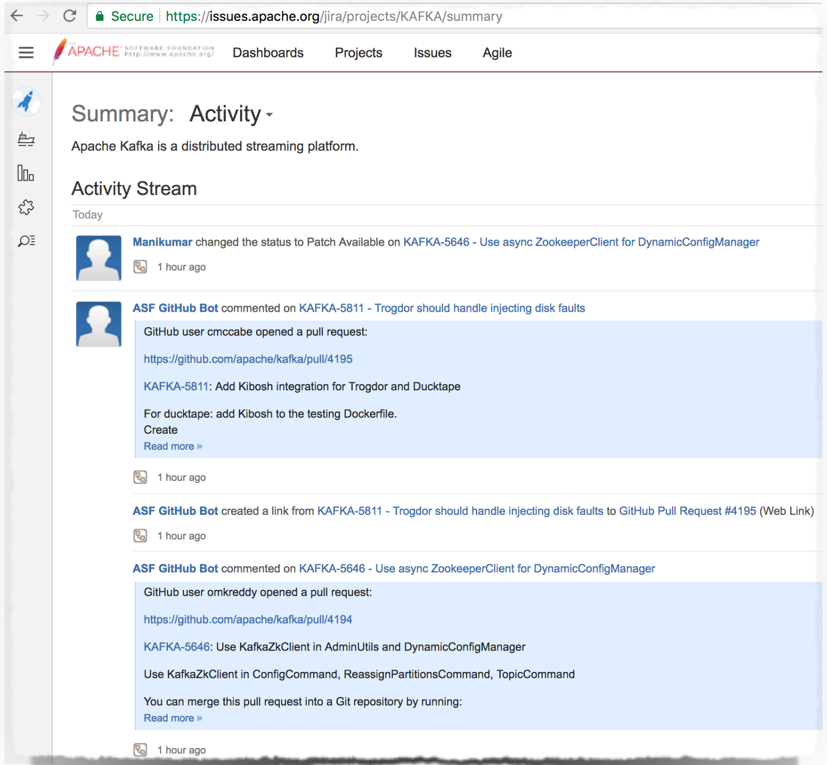
Tasks also reveal defects distribution

Defects

How much **effort is spent on bug-fixing**?

Are there code areas that concentrate many **bug-fixing changes**?

Technique: Bug-fixing commits (Jira tasks)


Secure https://issues.apache.org/jira/projects/KAFKA/summary
APACHE SOFTWARE FOUNDATION http://www.apache.org/
Dashboards
Projects
Issues
Agile

Summary: Activity

Apache Kafka is a distributed streaming platform.

Activity Stream

Today

Manikumar changed the status to Patch Available on KAFKA-5646 - Use async ZookeeperClient for DynamicConfigManager

1 hour ago

ASF GitHub Bot commented on KAFKA-5811 - Trogdor should handle injecting disk faults

> GitHub user cmccabe opened a pull request:
>
> https://github.com/apache/kafka/pull/4195
>
> KAFKA-5811: Add Kibosh integration for Trogdor and Ducktape
>
> For ducktape: add Kibosh to the testing Dockerfile.
> Create
> Read more »

1 hour ago

ASF GitHub Bot created a link from KAFKA-5811 - Trogdor should handle injecting disk faults to GitHub Pull Request #4195 (Web Link)

1 hour ago

ASF GitHub Bot commented on KAFKA-5646 - Use async ZookeeperClient for DynamicConfigManager

> GitHub user omkreddy opened a pull request:
>
> https://github.com/apache/kafka/pull/4194
>
> KAFKA-5646: Use KafkaZkClient in AdminUtils and DynamicConfigManager
>
> Use KafkaZkClient in ConfigCommand, ReassignPartitionsCommand, TopicCommand
>
> You can merge this pull request into a Git repository by running:
> Read more »

1 hour ago

https://issues.apache.org/jira/projects/KAFKA

export from JIRA

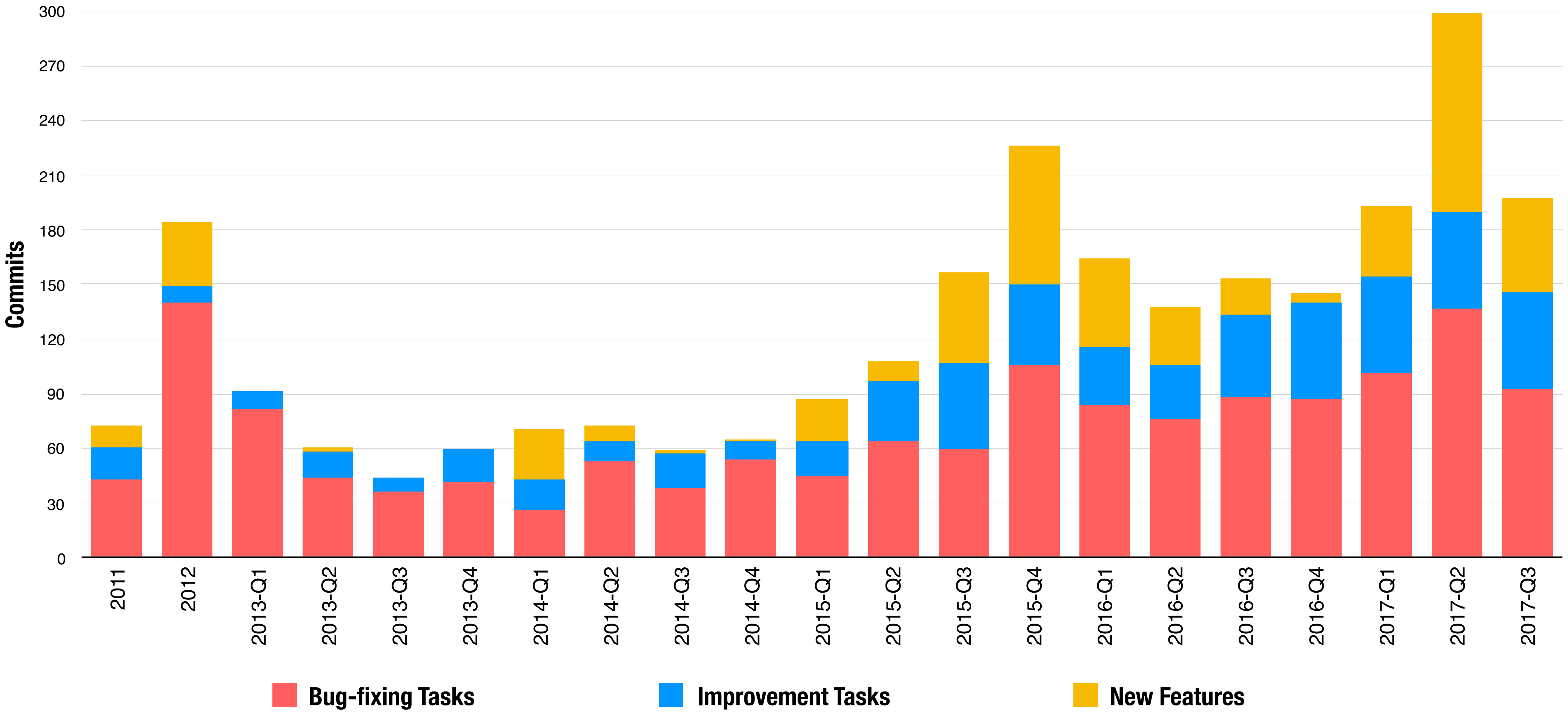
```
commit:bfa82fc0b63a5ad0121da6e2190a587eb672a1d5
author:Vahid Hashemian
date:Thu, 12 Oct 2017 10:09:16 +0100
message:
KAFKA-6055; Fix typo in KAFKA_JVM_PERFORMAN
```

```
{
    "key": "KAFKA-6055",
    "parent-key": "",
    "issuetype": "Bug",
    "status": "Resolved",
    "start-date": "2017-10-11T22:51:06.000+0000",
    "end-date": "2017-10-12T09:15:37.000+0000"
},
{
    "key": “KAFKA-3663",
    "parent-key": "",
    "issuetype": "Improvement",
    "status": "Patch Available",
    "start-date": "2017-10-12T15:26:37.000+0000",
    "end-date": ""
},
```

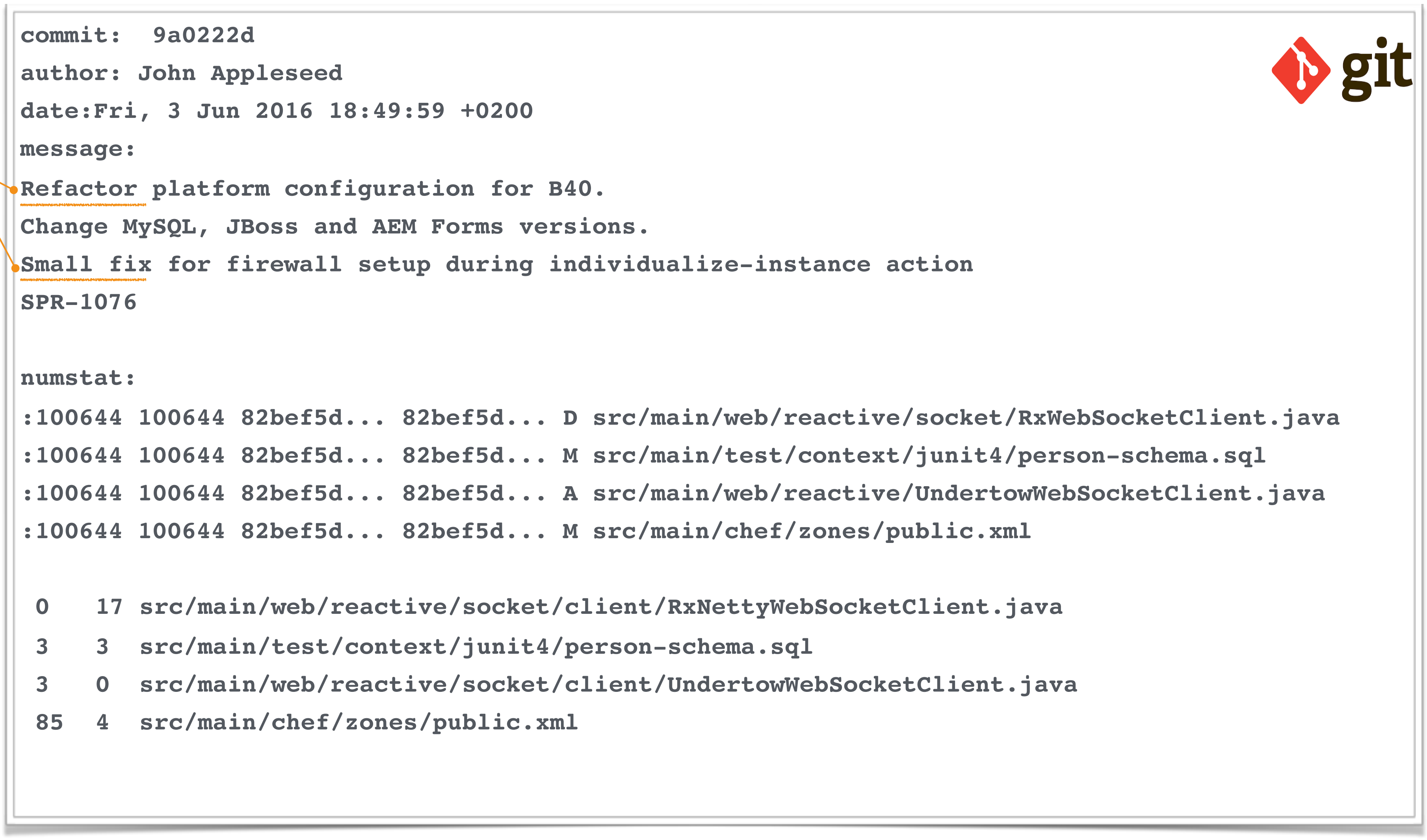
```
commit:c0f7a7705851eec4a77d3e42cf6bf2546c07ffa8
author:Andrey Dyachkov
date:Mon, 14 Aug 2017 18:24:43 +0100
message:
KAFKA-3663; Improve test coverage
```

https://github.com/seb-luke/Data-Mining-in-Issue-Tracking-Systems

Which are the **trends of (Jira) task types**



Technique: Bug-fixing commits (keywords)

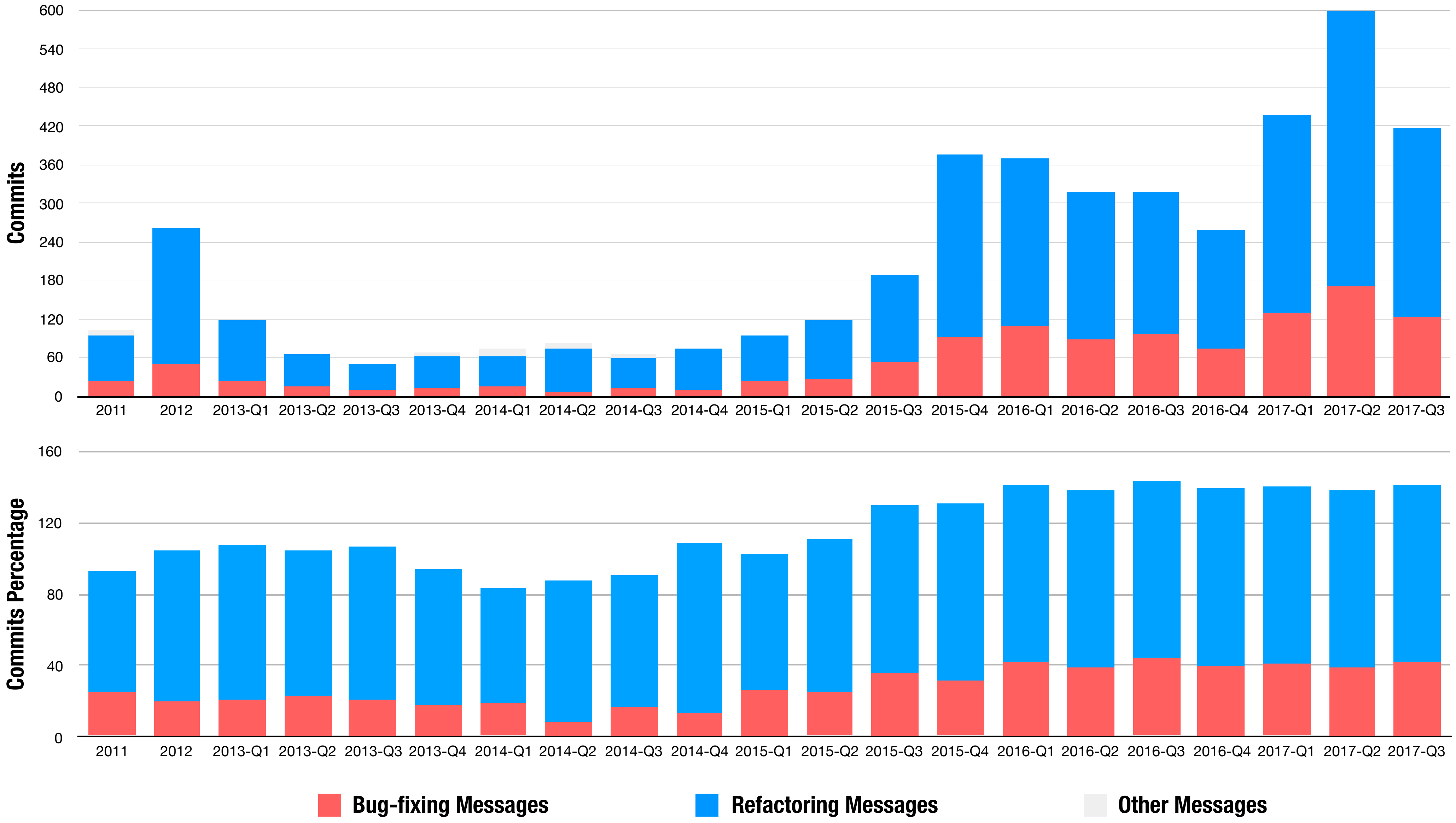


```
commit:  9a0222d
author: John Appleseed
date:Fri, 3 Jun 2016 18:49:59 +0200
message:
Refactor platform configuration for B40.
Change MySQL, JBoss and AEM Forms versions.
Small fix for firewall setup during individualize-instance action
SPR-1076

numstat:
:100644 100644 82bef5d... 82bef5d... D src/main/web/reactive/socket/RxWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/test/context/junit4/person-schema.sql
:100644 100644 82bef5d... 82bef5d... A src/main/web/reactive/UndertowWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/chef/zones/public.xml

 0   17 src/main/web/reactive/socket/client/RxNettyWebSocketClient.java
 3   3  src/main/test/context/junit4/person-schema.sql
 3   0  src/main/web/reactive/socket/client/UndertowWebSocketClient.java
 85  4  src/main/chef/zones/public.xml
```

Keywords
suggest bugfix
or cleanup

Bugfixing and Refactoring Messages: Commits are labeled as “bug-fixing” and resp. “refactoring” by mining commit messages for keywords suggesting bugfixing changes; for “**bug fixing**” the message must contain one of the strings: *"fix"*, *"bug"*, *"error"*, *"issue"*, *"correct"*, *"workaround"*, *“crash”*; for **refactoring** the strings are: *"refactor"*, *"restructur"*, *"clean"*, *"improve"*, *"rename"*, *"rework"*, *"move"*, *"improved"*, *“reorgani"*.

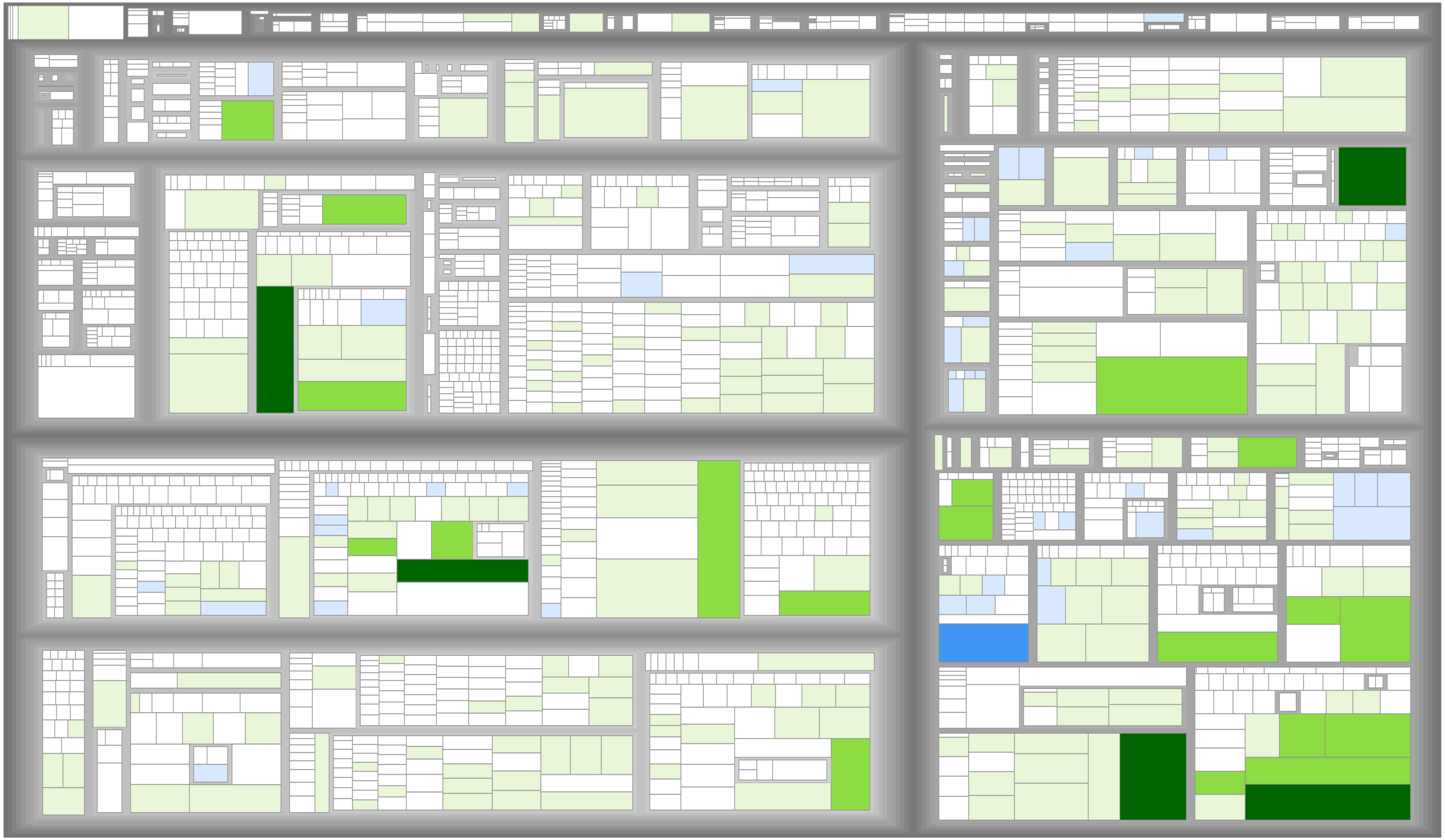
Which are the **bugfixing/refactoring trends**



Bugfixing and Refactoring Messages: Commits are labeled as “bug-fixing” and resp. “refactoring” by mining commit messages for keywords suggesting bugfixing changes; for “**bug fixing**” the message must contain one of the strings: "*fix*", "*bug*", "*error*", "*issue*", "*correct*", "*workaround*", *“crash”*; for **refactoring** the strings are: "*refactor*", "*restructur*", "*clean*", "*improve*", "*rename*", "*rework*", "*move*", "*improved*", *“reorgani*".

Bug Magnets (based on keywords)

Error-Prone Code. Insufficient / Superficial Testing



commits w. bugfix message

	1-20	21-40	41-
active	🟩	🟩	🟩
idle	🟦	🟦	🟦

Bug Magnets: files with many commits containing bugfixing messages

Team

B

Team
Knowledge

Knowledge

Does the **current team know enough** about the system?

Is the **current team large enough** for the current development stage?

For which code areas does the **current team lack knowledge**?

Which developers have **most system knowledge** ?

Which developers have most **knowledge about each code area**?

How many developers **cover each language**?

Is **knowledge shared** or is it **polarised**?

Knowledge: two key questions

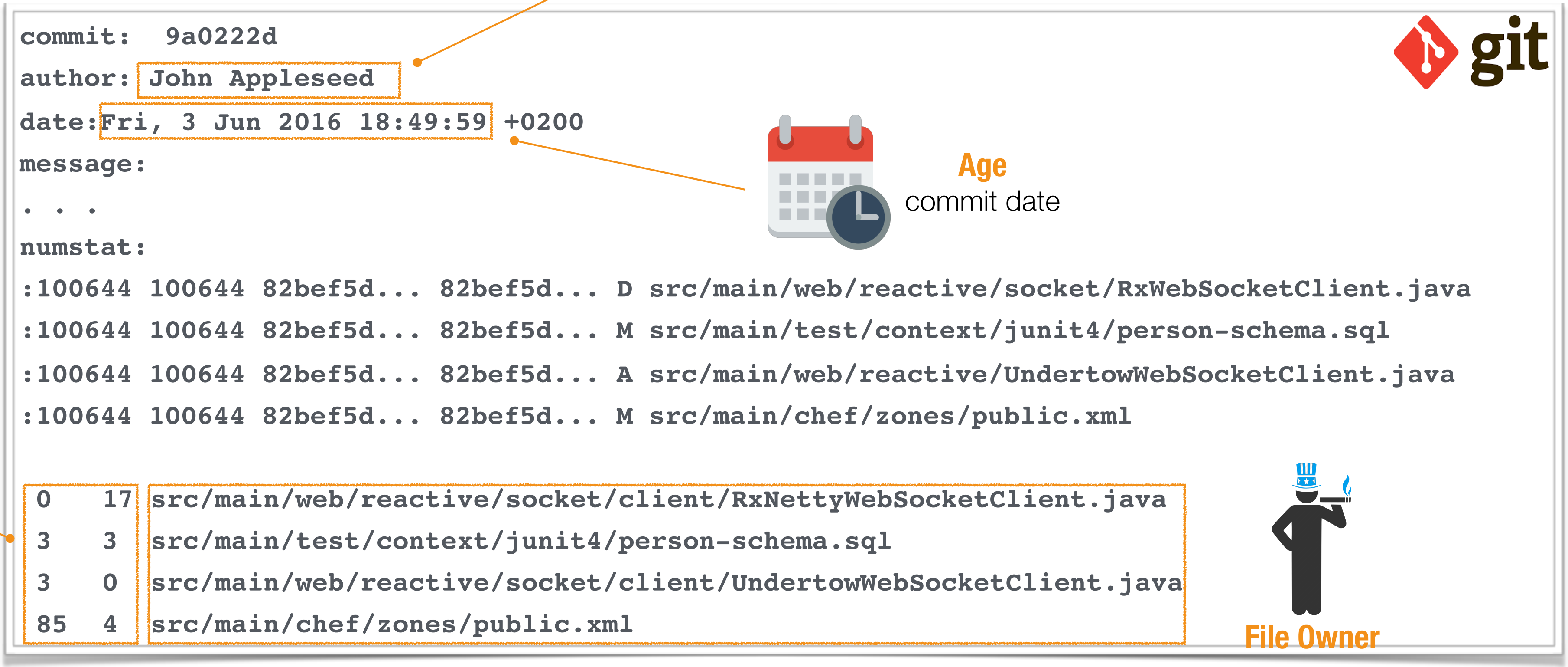
1

What **knowledge does she hold?**

Developer

2

Is she **still in the project?**



```
commit:  9a0222d
author: John Appleseed
date:Fri, 3 Jun 2016 18:49:59 +0200
message:
. . .
numstat:
:100644 100644 82bef5d... 82bef5d... D src/main/web/reactive/socket/RxWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/test/context/junit4/person-schema.sql
:100644 100644 82bef5d... 82bef5d... A src/main/web/reactive/UndertowWebSocketClient.java
:100644 100644 82bef5d... 82bef5d... M src/main/chef/zones/public.xml

0    17  src/main/web/reactive/socket/client/RxNettyWebSocketClient.java
3    3   src/main/test/context/junit4/person-schema.sql
3    0   src/main/web/reactive/socket/client/UndertowWebSocketClient.java
85   4   src/main/chef/zones/public.xml
```

Age
commit date

Churn / Activity
Σ(add + del)

File Owner

active	**passive**	**retired**
commits over the latest 6 weeks	commits over the previous 6 weeks	no commits in latest 12 weeks

The Developers: Merging redundat user IDs

rnpridgeon

Ryan P

Ryan Pridgeon

↓

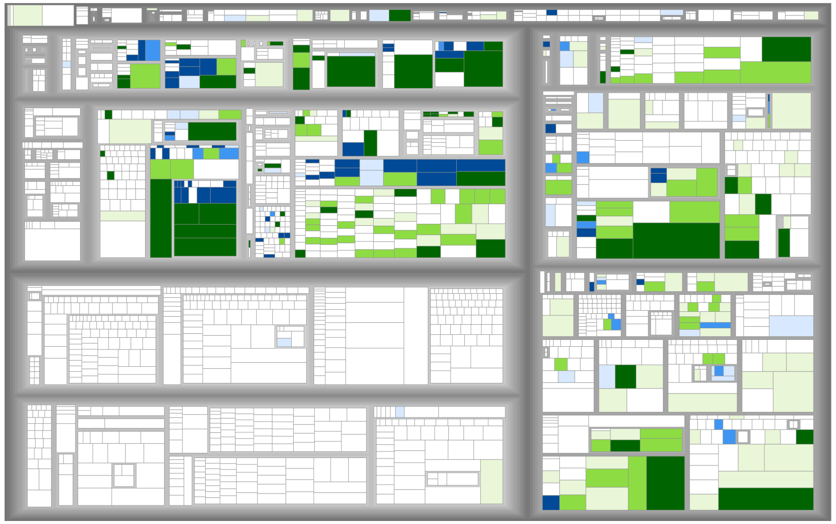
Ryan Pridgeon

326

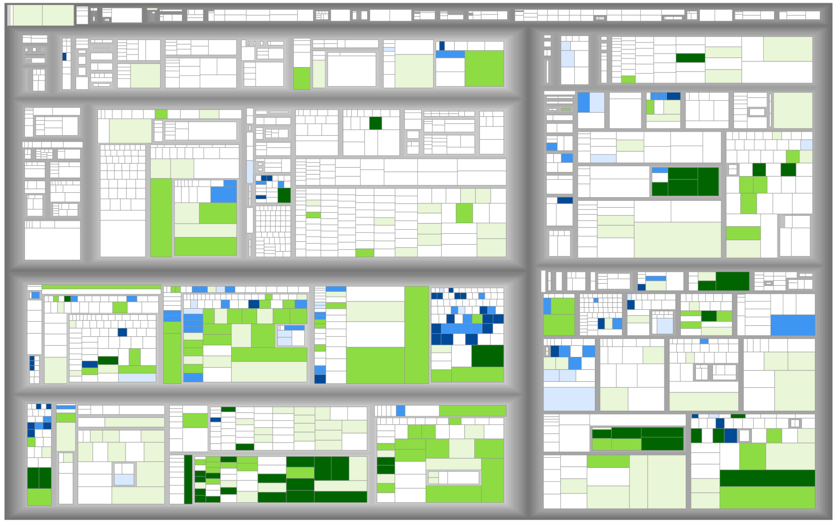
↓

290

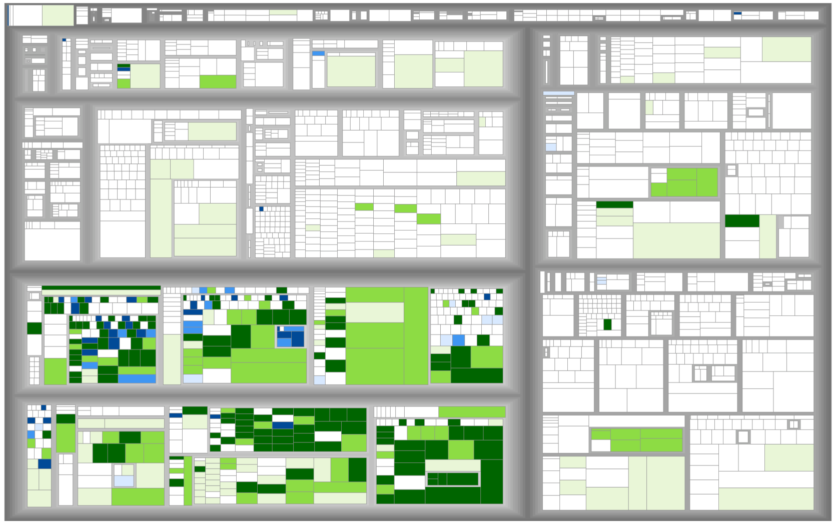
Who works on what?



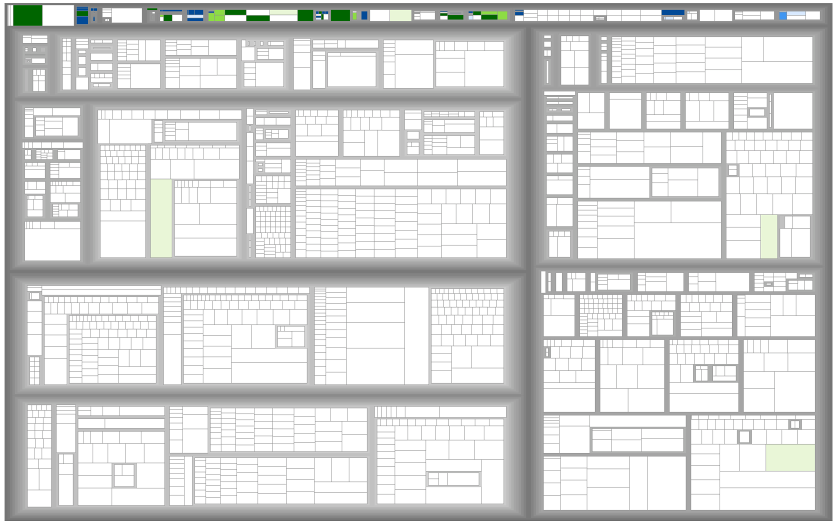
Jason Gustafson



Guozhang Wang



Damian Guy



Geoff Anderson

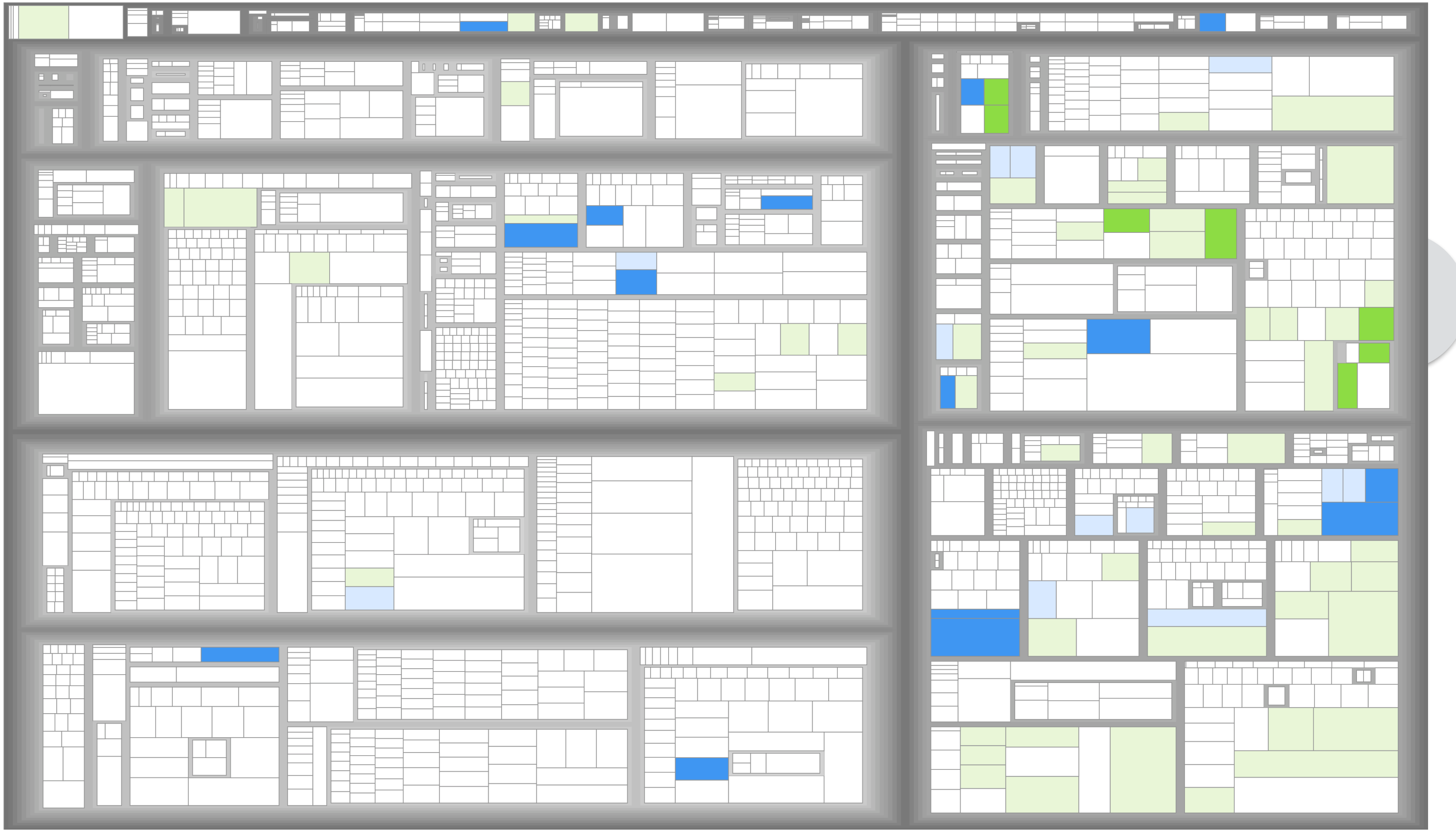
APACHE kafka®
A distributed streaming platform

activity level

active	minor	major	owner
idle			

Orphans: files with "**retired**" **file owner**

Slower and Riskier Development (caused by lack of knowledge)



all	active	% active	java	scala	py
88	60	68 %	19 %	74 %	6 %

APACHE kafka
A distributed streaming platform

Who causes most Orphans?

	Files		Lines of Code	
Retired file owner	**all**	**active**	**all**	**active**
Edward Jay Kreps	17	13	9,919	7,668
Neha Narkhede	16	9	8,392	5,232
Jun Rao	15	12	9,225	8,184
Ben Stopford	7	7	2,618	2,618

Technique: Project-level Knowledge

	%	votes	clients	core	streams	tests
	100	963	323	321	287	32

Project Knowledge Score (PKS) is computed as follows: each component, gets a number of “votes” based on its size (1 vote / KLOC). Developers split these votes per component, based on their activity on the component’s files. The PKS is the percentage of “votes” that an developer got from all components

Technique: Project-level Knowledge

	%	votes	clients	core	streams	tests
	100	963	323	321	287	32
Jason Gustafson						
Ismael Juma						
Guozhang Wang						
Damian Guy						
Matthias J. Sax						
Rajini Sivaram						
Edward Jay Kreps						
Yasuhiro Matsuda						
Colin P. Mccabe						
Eno Thereska						
Onur Karaman						
Apurva Mehta						
Jun Rao						
Neha Narkhede						
Jiangjie Qin						
Ben Stopford						
Vahid Hashemian						
Geoff Anderson						
Bill Bejeck						
Dong Lin						
Ewen Cheslack-Postava						
Ashish Singh						

Project Knowledge Score (PKS) is computed as follows: each component, gets a number of “votes” based on its size (1 vote / KLOC). Developers split these votes per component, based on their activity on the component’s files. The PKS is the percentage of “votes” that an developer got from all components

Technique: Project-level Knowledge

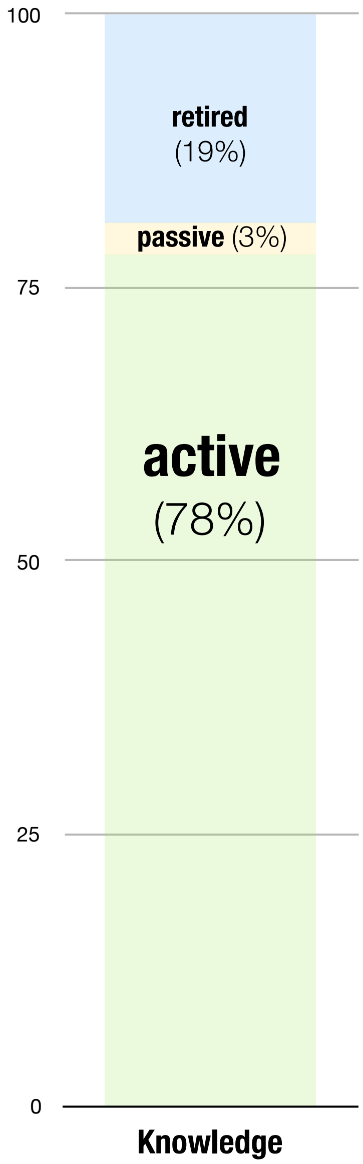
	%	votes	clients	core	streams	tests
	100	**963**	**323**	**321**	**287**	**32**
Jason Gustafson	17.4	168	109	56		3
Ismael Juma	12.0	116	46	69		1
Guozhang Wang	11.2	108	19	43	46	
Damian Guy	9.3	90			90	
Matthias J. Sax	7.9	76			76	
Rajini Sivaram	7.1	68	46	19		3
Edward Jay Kreps	5.0	48	29	19		
Yasuhiro Matsuda	3.6	35			35	
Colin P. Mccabe	3.3	32	29			3
Eno Thereska	3.1	30			29	1
Onur Karaman	2.7	26		26		
Apurva Mehta	2.7	26	23			3
Jun Rao	2.6	25	6	19		
Neha Narkhede	2.4	23		23		
Jiangjie Qin	2.0	19		19		
Ben Stopford	2.0	19		19		
Vahid Hashemian	1.7	16	16			
Geoff Anderson	1.2	12				12
Bill Bejeck	1.1	11			11	
Dong Lin	0.9	9		9		
Ewen Cheslack-Postava	0.5	5				5
Ashish Singh	0.1	1				1

Project Knowledge Score (PKS) is computed as follows: each component, gets a number of “votes” based on its size (1 vote / KLOC). Developers split these votes per component, based on their activity on the component’s files. The PKS is the percentage of “votes” that an developer got from all components

Which are the **key developers**

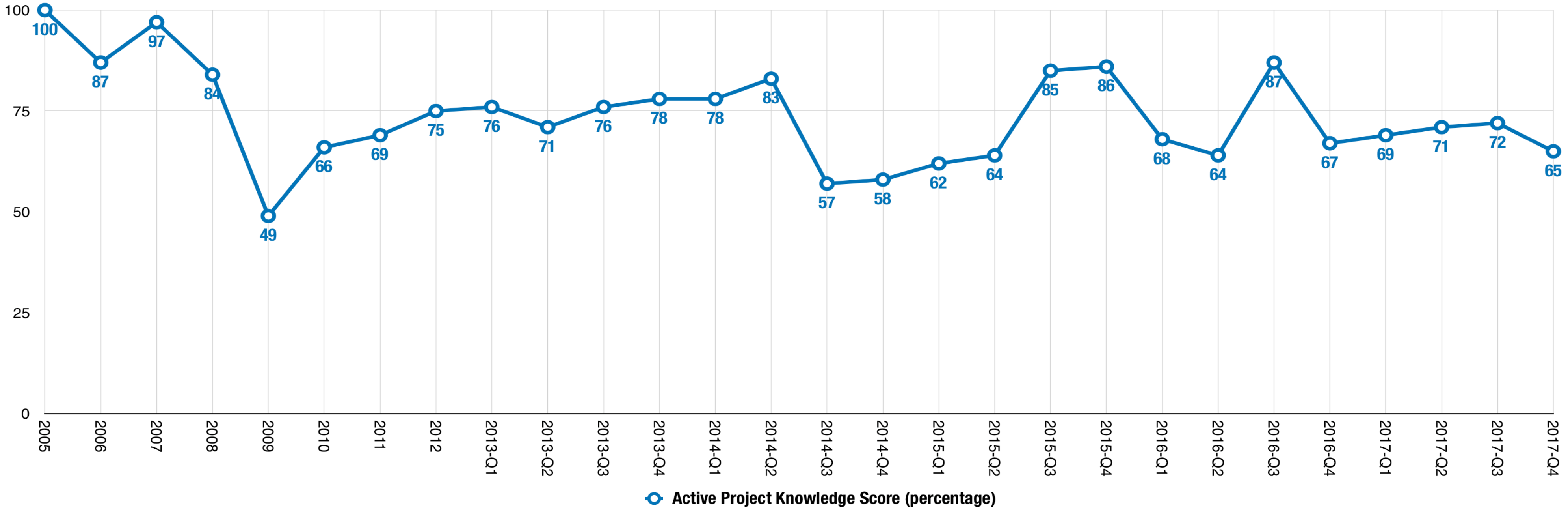
	%	*votes*	clients	core	streams	tests
	100	**963**	**323**	**321**	**287**	**32**
Jason Gustafson	17.4	168	109	56		3
Ismael Juma	12.0	116	46	69		1
Guozhang Wang	11.2	108	19	43	46	
Damian Guy	9.3	90			90	
Matthias J. Sax	7.9	76			76	
Rajini Sivaram	7.1	68	46	19		3
Edward Jay Kreps	5.0	48	29	19		
Yasuhiro Matsuda	3.6	35			35	
Colin P. Mccabe	3.3	32	29			3
Eno Thereska	3.1	30			29	1
Onur Karaman	2.7	26		26		
Apurva Mehta	2.7	26	23			3
Jun Rao	2.6	25	6	19		
Neha Narkhede	2.4	23		23		
Jiangjie Qin	2.0	19		19		
Ben Stopford	2.0	19		19		
Vahid Hashemian	1.7	16	16			
Geoff Anderson	1.2	12				12
Bill Bejeck	1.1	11			11	
Dong Lin	0.9	9		9		
Ewen Cheslack-Postava	0.5	5				5
Ashish Singh	0.1	1				1

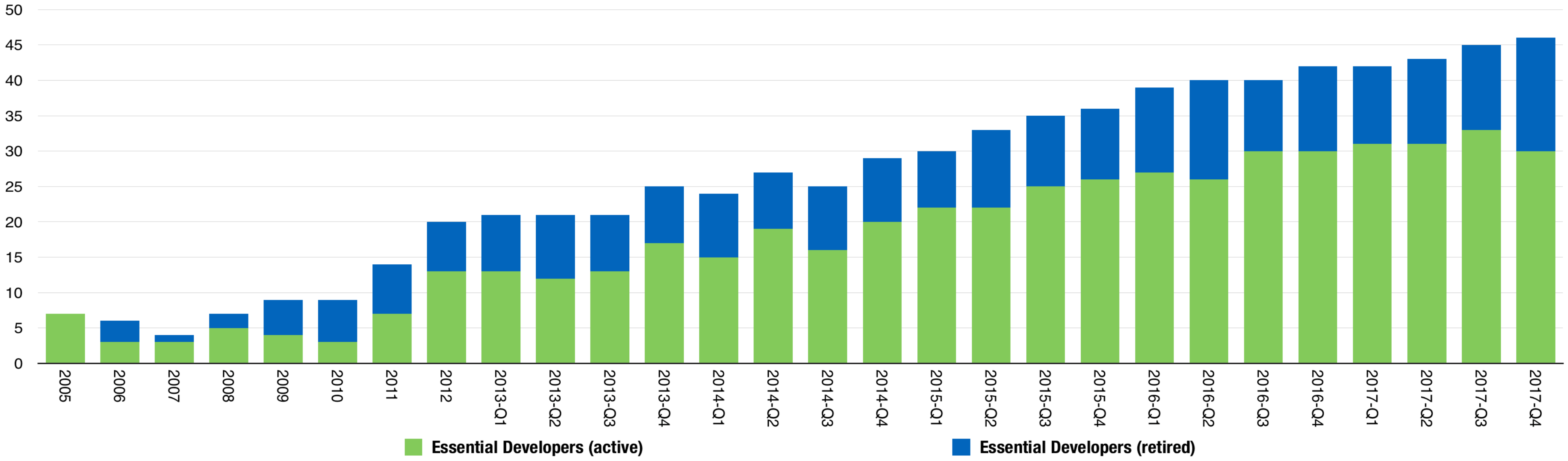
Knowledge held by **active developers**



	%	votes	clients	core	streams	tests
	100	**963**	**323**	**321**	**287**	**32**
Jason Gustafson	17.4	168	109	56		3
Ismael Juma	12.0	116	46	69		1
Guozhang Wang	11.2	108	19	43	46	
Damian Guy	9.3	90			90	
Matthias J. Sax	7.9	76			76	
Rajini Sivaram	7.1	68	46	19		3
Edward Jay Kreps	5.0	48	29	19		
Yasuhiro Matsuda	3.6	35			35	
Colin P. Mccabe	3.3	32	29			3
Eno Thereska	3.1	30			29	1
Onur Karaman	2.7	26		26		
Apurva Mehta	2.7	26	23			3
Jun Rao	2.6	25	6	19		
Neha Narkhede	2.4	23		23		
Jiangjie Qin	2.0	19		19		
Ben Stopford	2.0	19		19		
Vahid Hashemian	1.7	16	16			
Geoff Anderson	1.2	12				12
Bill Bejeck	1.1	11			11	
Dong Lin	0.9	9		9		
Ewen Cheslack-Postava	0.5	5				5
Ashish Singh	0.1	1				1

Evolution of knowledge held by **active developers**





What **knowledge have active developers**

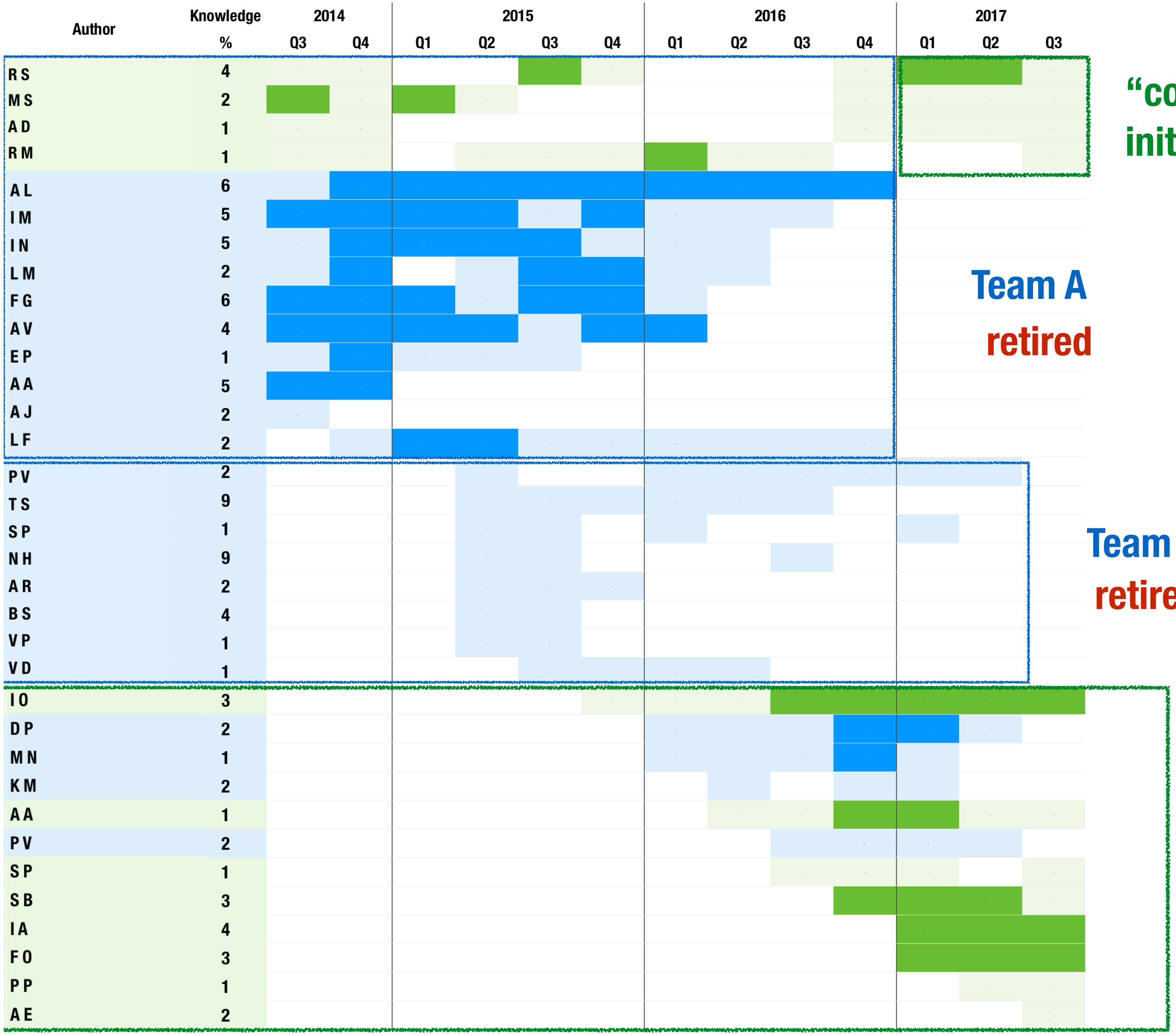
		Developer		Radu S.	Ionut A.	Iulia O.	Simona B.	Flavius O.	Mirela S.	Andrei E.	Andrei A.	Petru P.	Alina D.	Silviu P.	Radu Z.	Laura B.
		Knowledge (%)		4	4	3	3	3	2	2	1	1	1	1		
Component	**Size (%)**	**Active (%)**	**Active Devs**	7	3	4	2	1	1	1	2	1	2	2	2	1
portal-app	27	47	7	5		8	8	11	8		4					
portal	14	26	3	6	15								5			
dao	10	0	0													
security	9	5	1	5												
services	8	0	0													
portal-admin-app	6	14	2	3			11									
utils	4	0	0													
persistence	4	0	0													
coreapi	3	0	0													
serviceweb	2	0	0													
notification-service-app	2	100	2							78				22		
securityweb-app	2	16	1								16					
selfservemanager	2	26	2	5		21										
service	2	0	0													
ds	2	100	4		63								13	6	19	
campaigns	1	100	3	8	62	31										
campaign	1	100	3			11						56				33
filteroption	1	100	2	83											17	

Narrow Focus

Weak Knowledge

Polarised Ownership

A story of **team changes…**



"come back" of several initial developers

Team A retired

Team B retired

new team gradually replaces the initial one

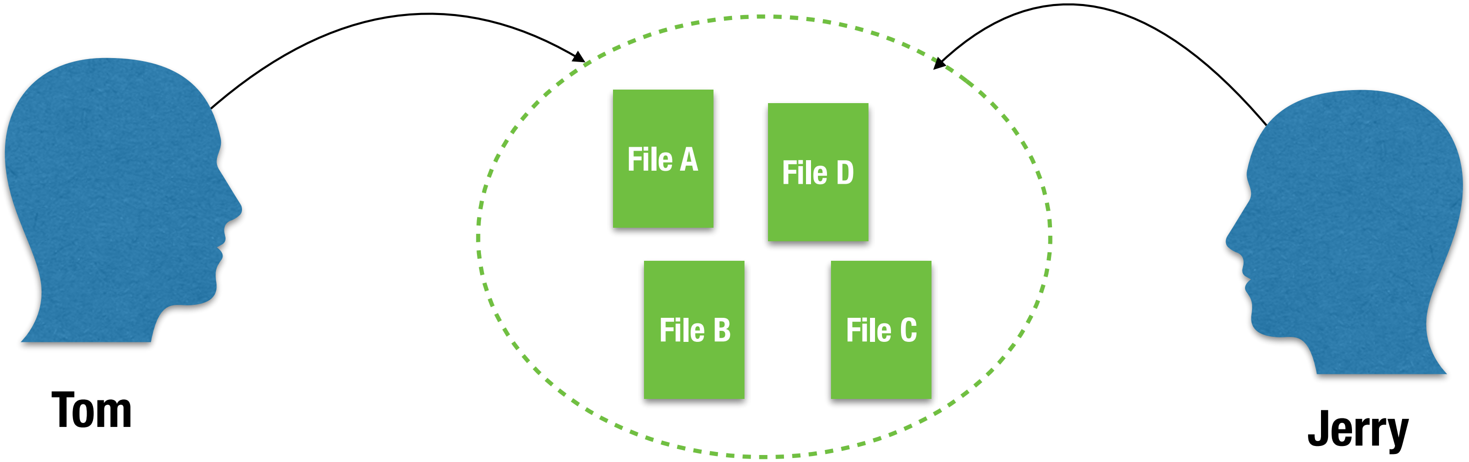
developer activity state

changes	**active**	**retired**
many changes	🟩	🟦

Technique: Developers sharing knowledge

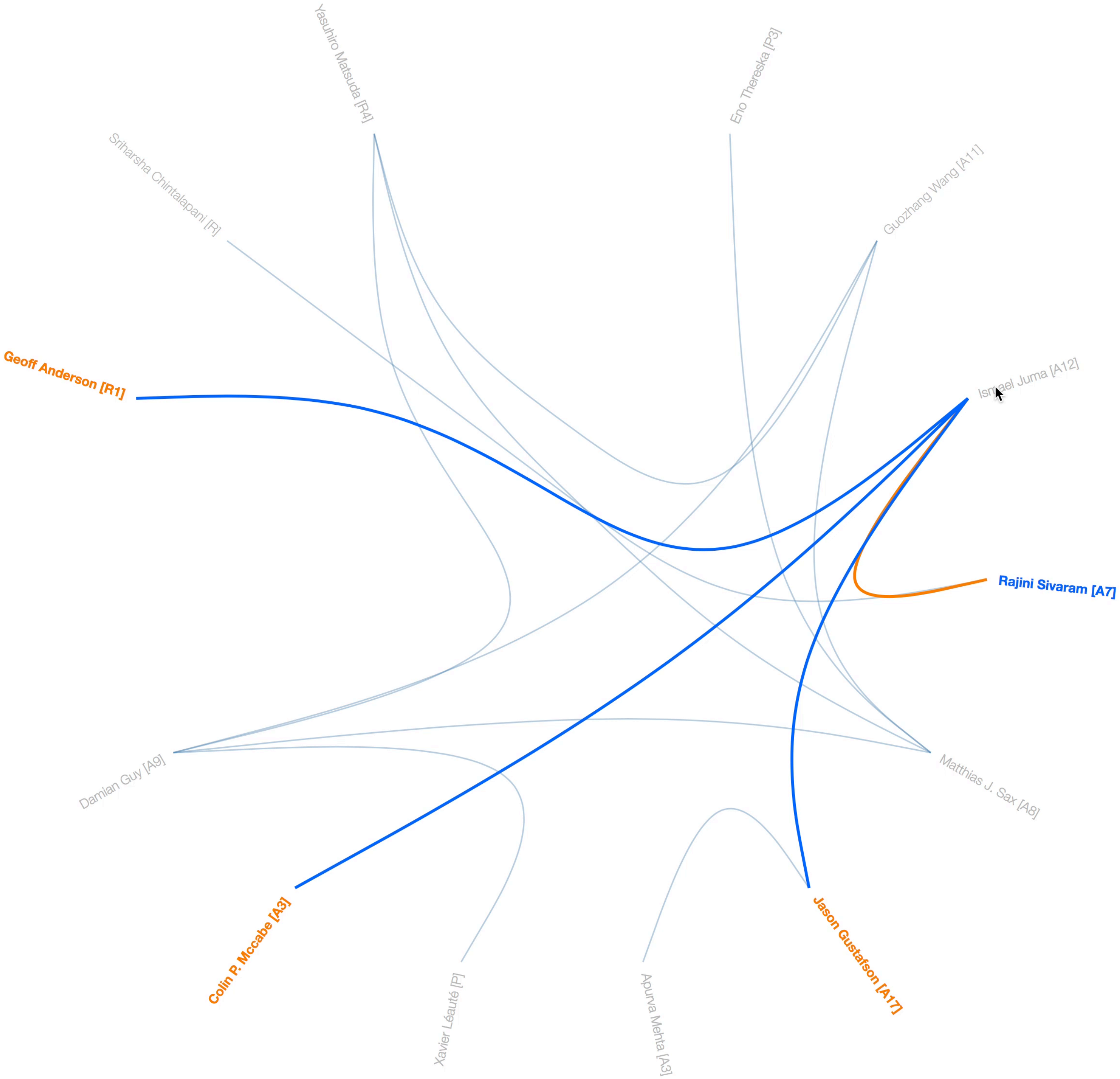
	work on same **Tasks**	work on same **Code Areas**	work during same **Period**
Developer	Developers with Shared Projects	Developers with Shared Files	Developers with Shared Time (timeframe)
	Developers with Shared Tasks	Developers with Shared Components	

Technique: Shared knowledge (files)

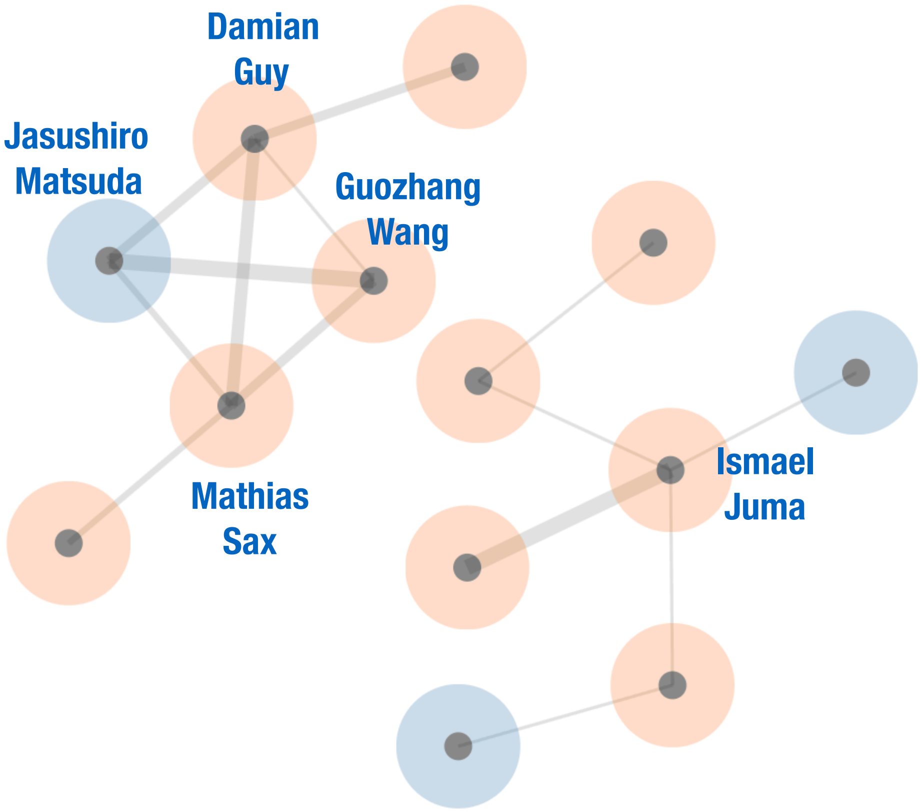


Shared Tasks Coupling: pairs of developers that are frequently committing using the same task IDs

Who is **co-changing the same files**



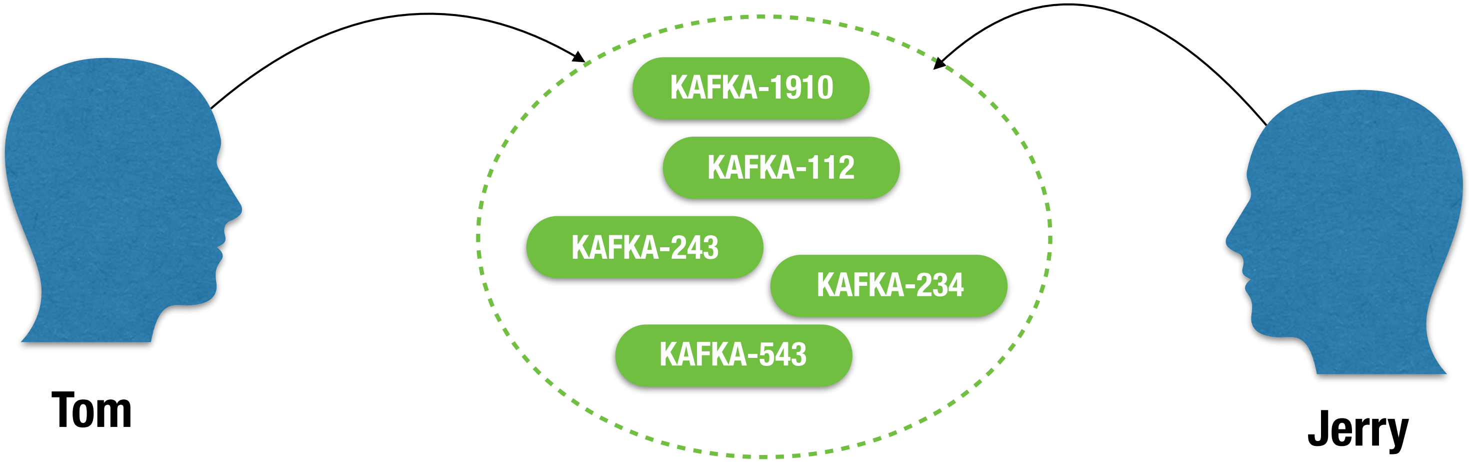
Who is **co-changing the same files**



developer activity state

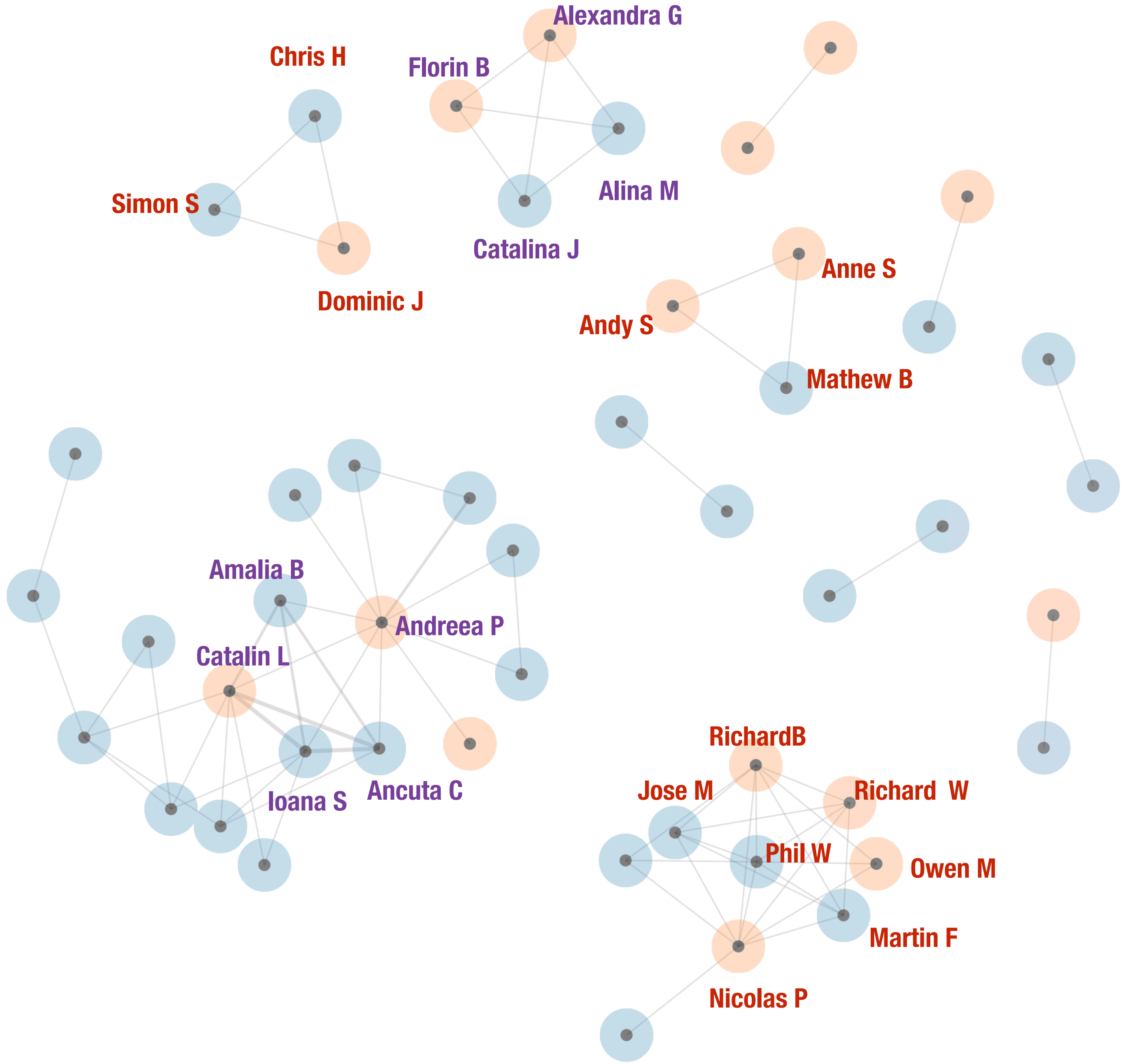
active	retired

Technique: Shared knowledge (tasks)



Shared Tasks Coupling: pairs of developers that are frequently committing using the same task IDs

Who is **co-working on the same tasks**



Pairs of developers that have worked significantly (at least **10% churn**) on the same at least **10 tasks**

developer activity state	
active	**retired**

Team
Stability

Team Stability: Key Questions

Stability

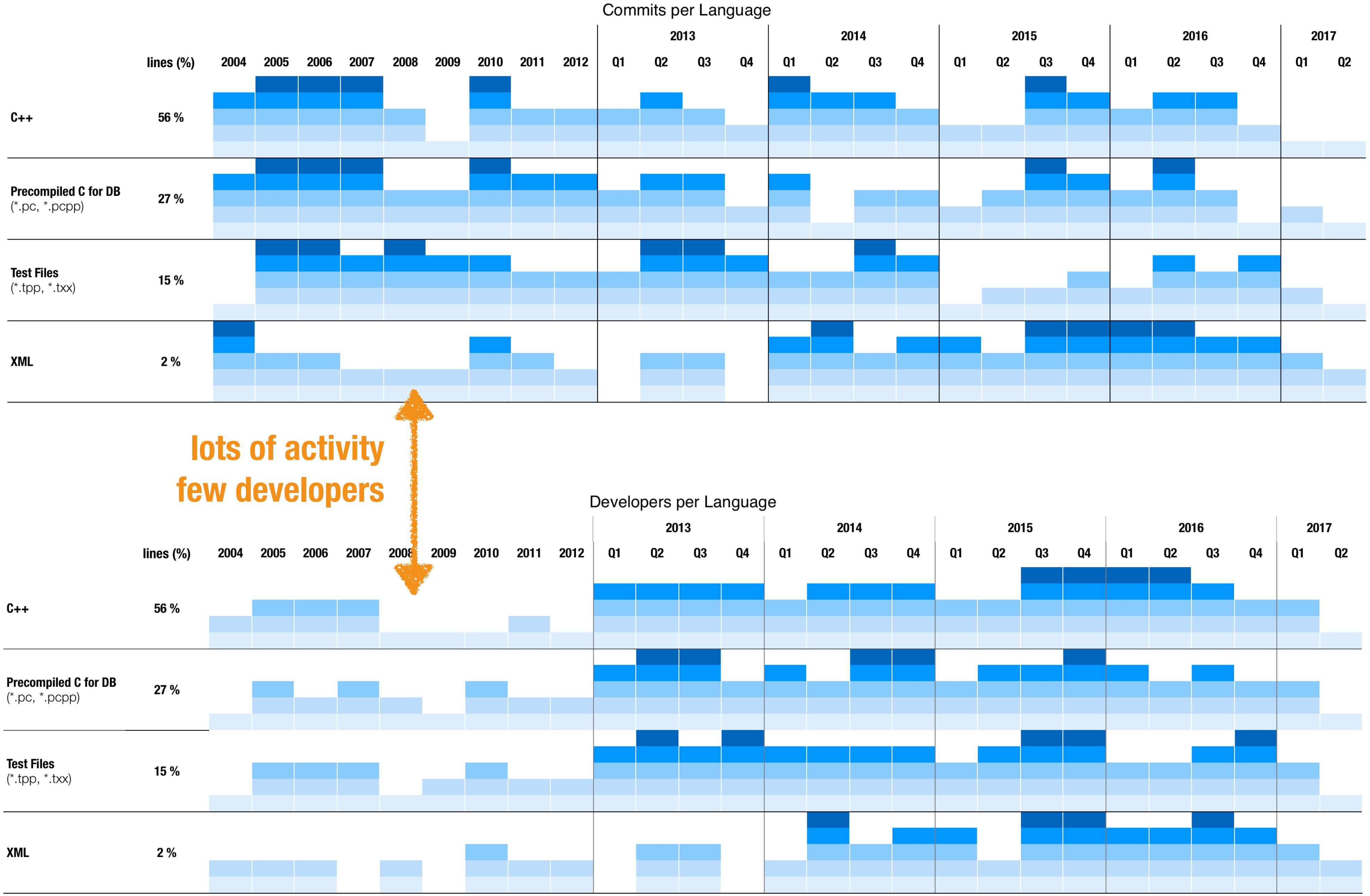
How **many developers** worked on the system?

How did the **team size evolve over time**?

When and for how long did each developer work?

Are there code areas that **had many file owners**?

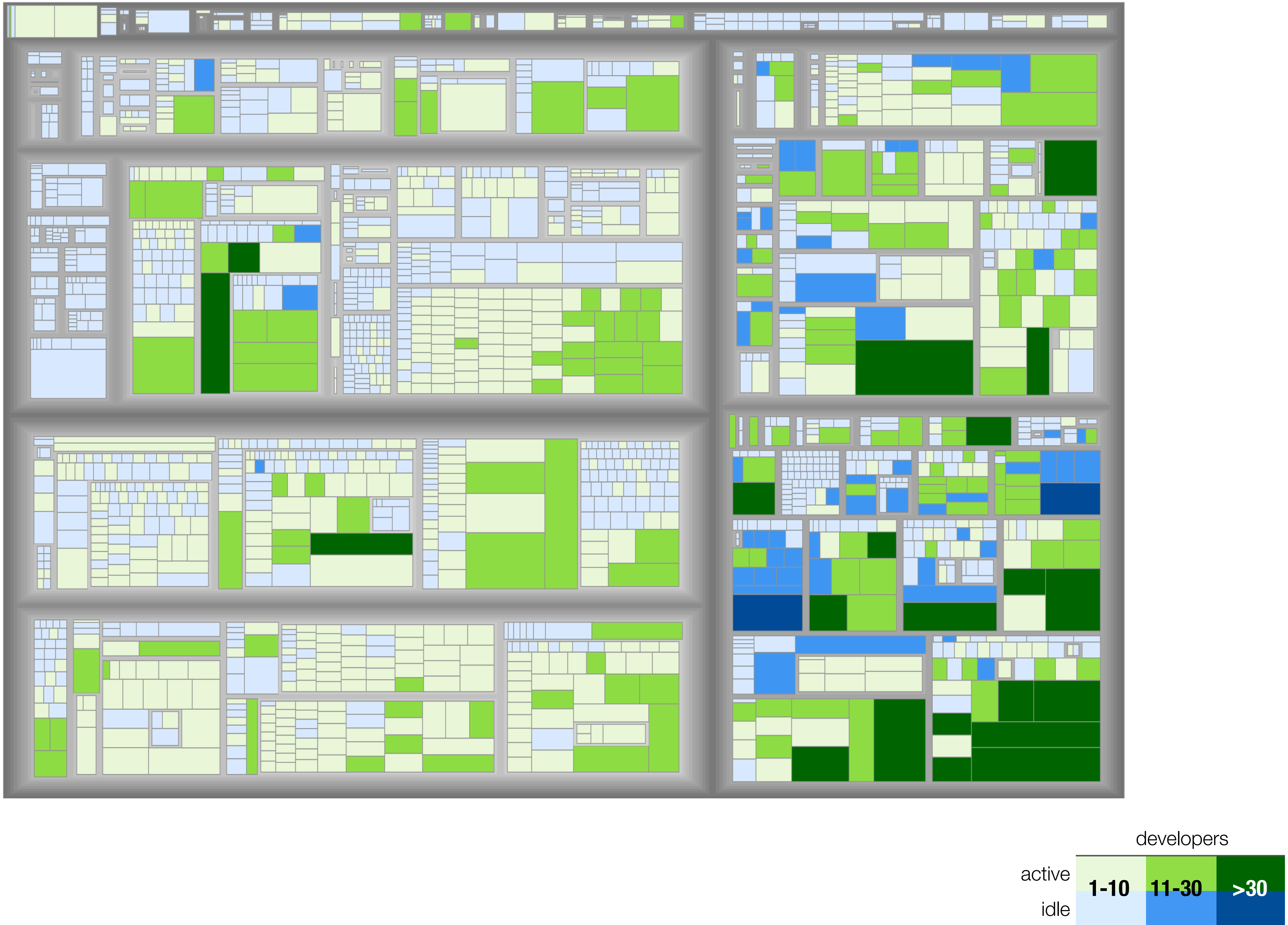
How did the **team size evolve**



lots of activity
few developers

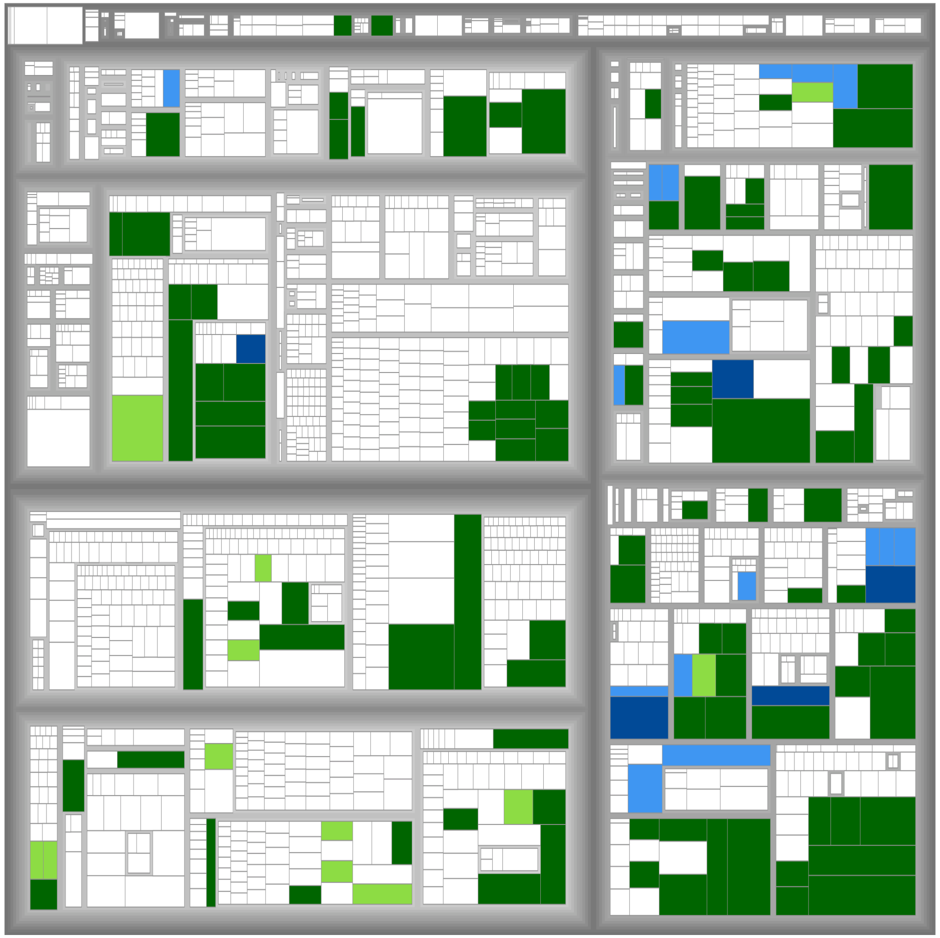
sample project

Which parts are **changed by many developers**

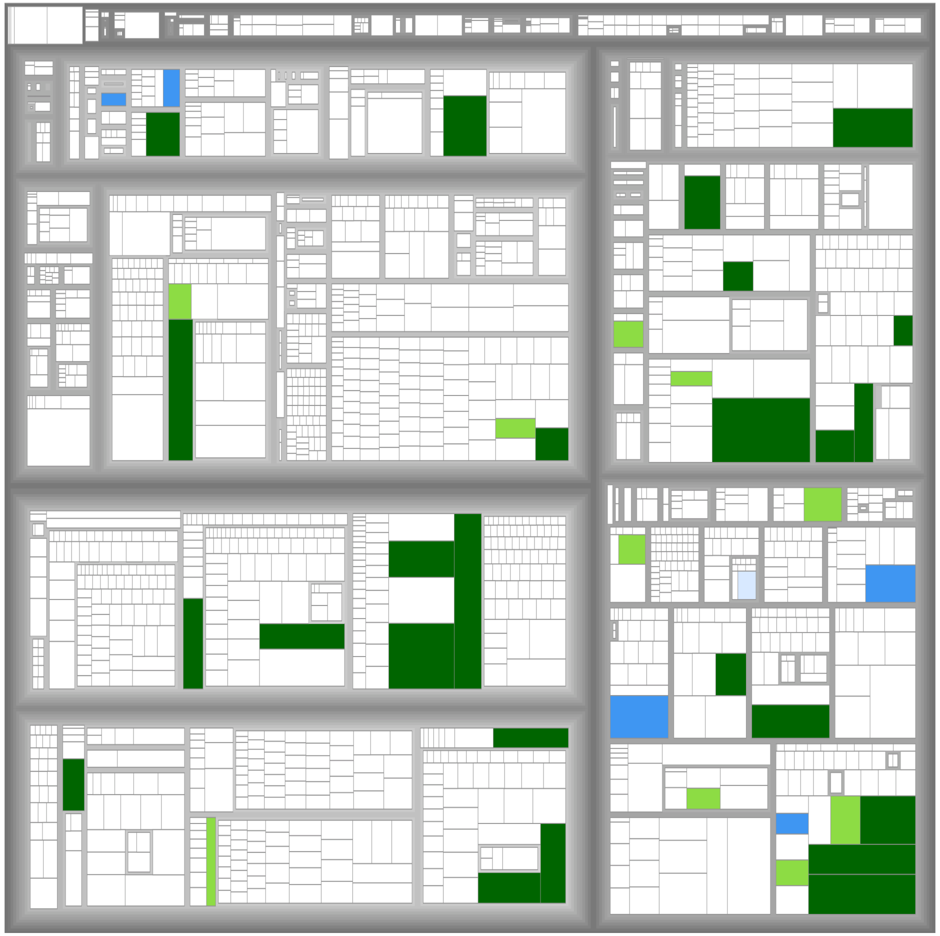


Files that are **changed by too many**

Error-Prone Code (caused by inconsistency). **Higher Maintenance Costs**



Team Churn



Owner Churn

Team Churn: file has been changed by many developers over time, some having limited knowledge about the file

Owner Churn: file for which the developer with the most activity on the file (file owner) switches many times

severity

active	1-4	5-7	8-10
idle			

Team Coordination

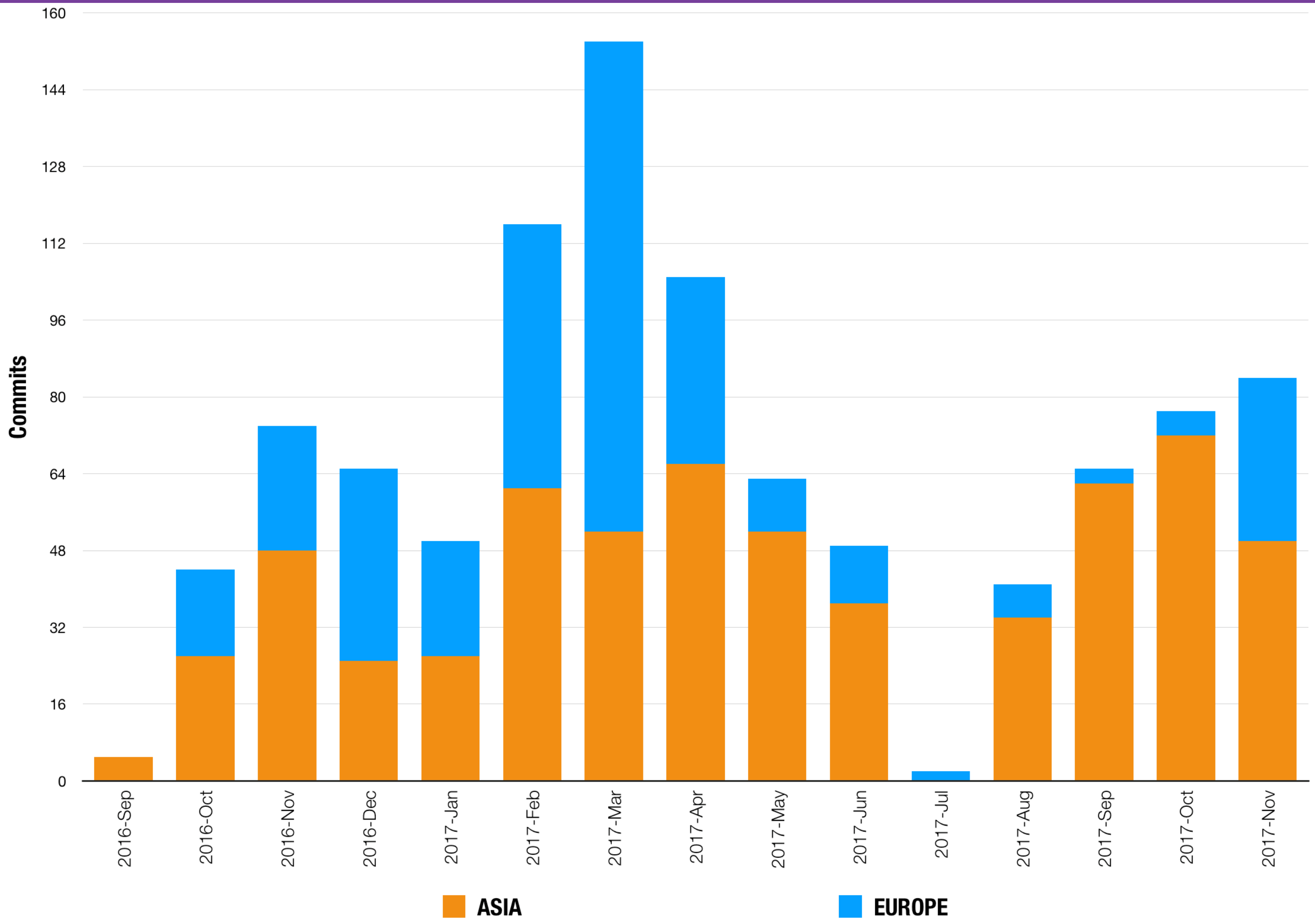
Team Coordination: Key Questions

Coordination

Which parts are **changed by (too) many developers**?

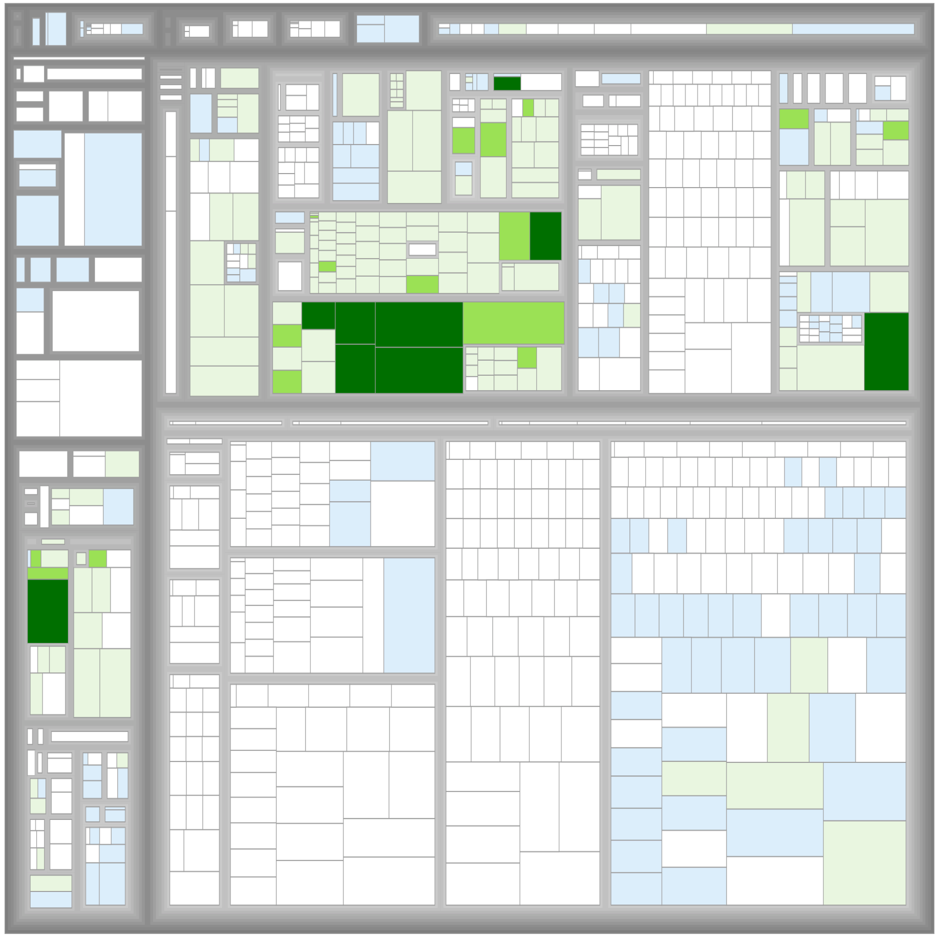
Are development teams working on **different timezones**?

In which **world regions** was it developed ?

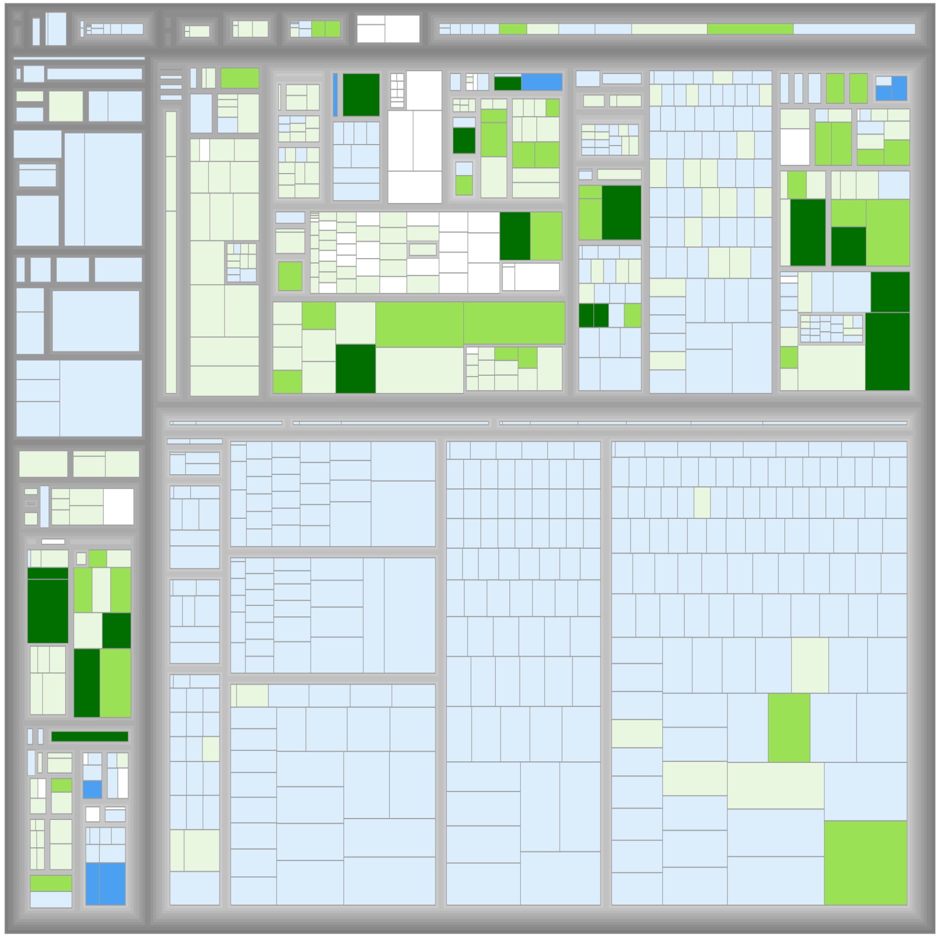


sample project

In which **world regions** was it developed



EUROPE

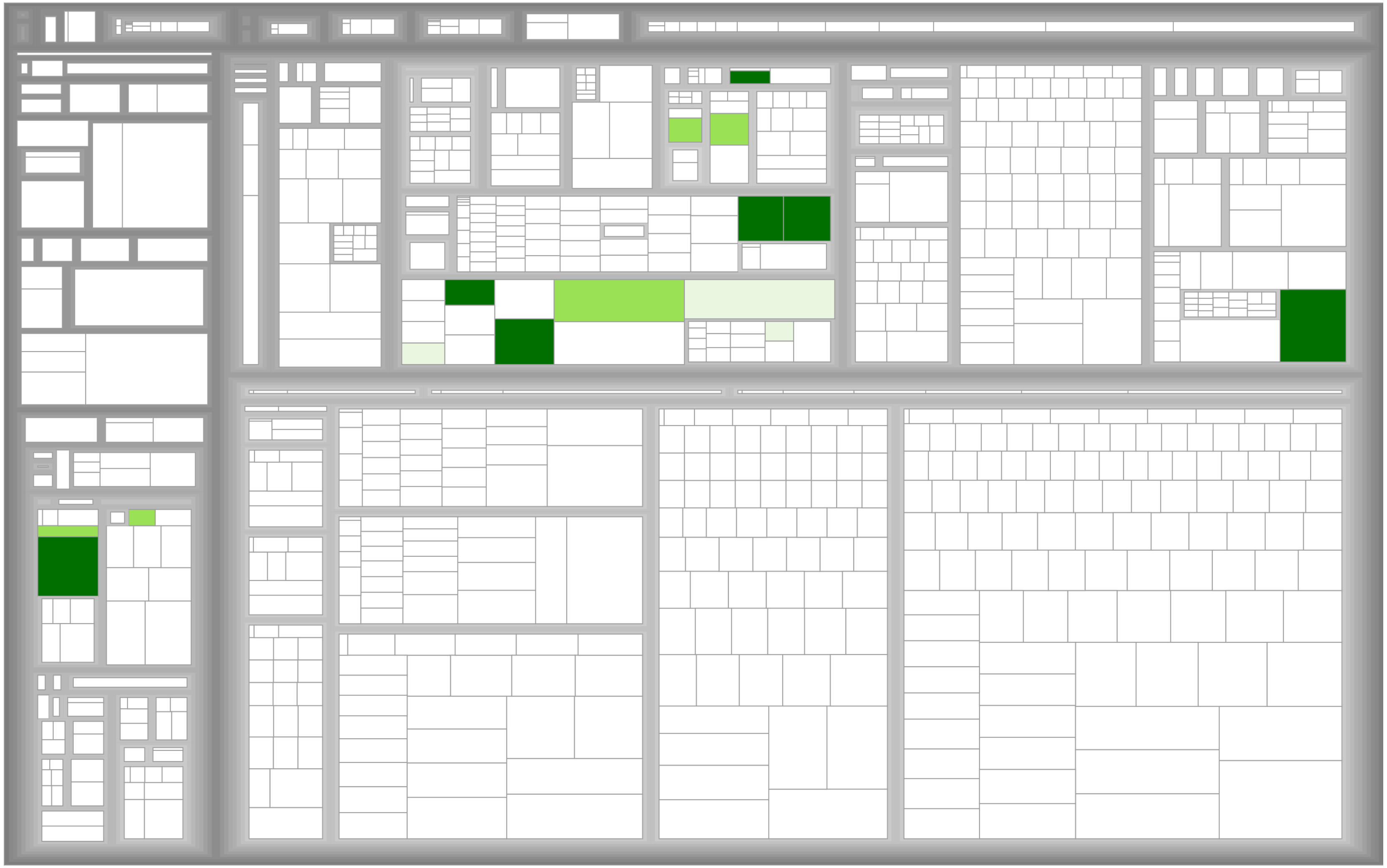


ASIA

	all changes		
active	1-21	21-50	>50
idle			

Files with many **cross-region changes**

Error-Prone Code (due to potential inconsistencies). **Slower Development** (need of coordination)



all changes

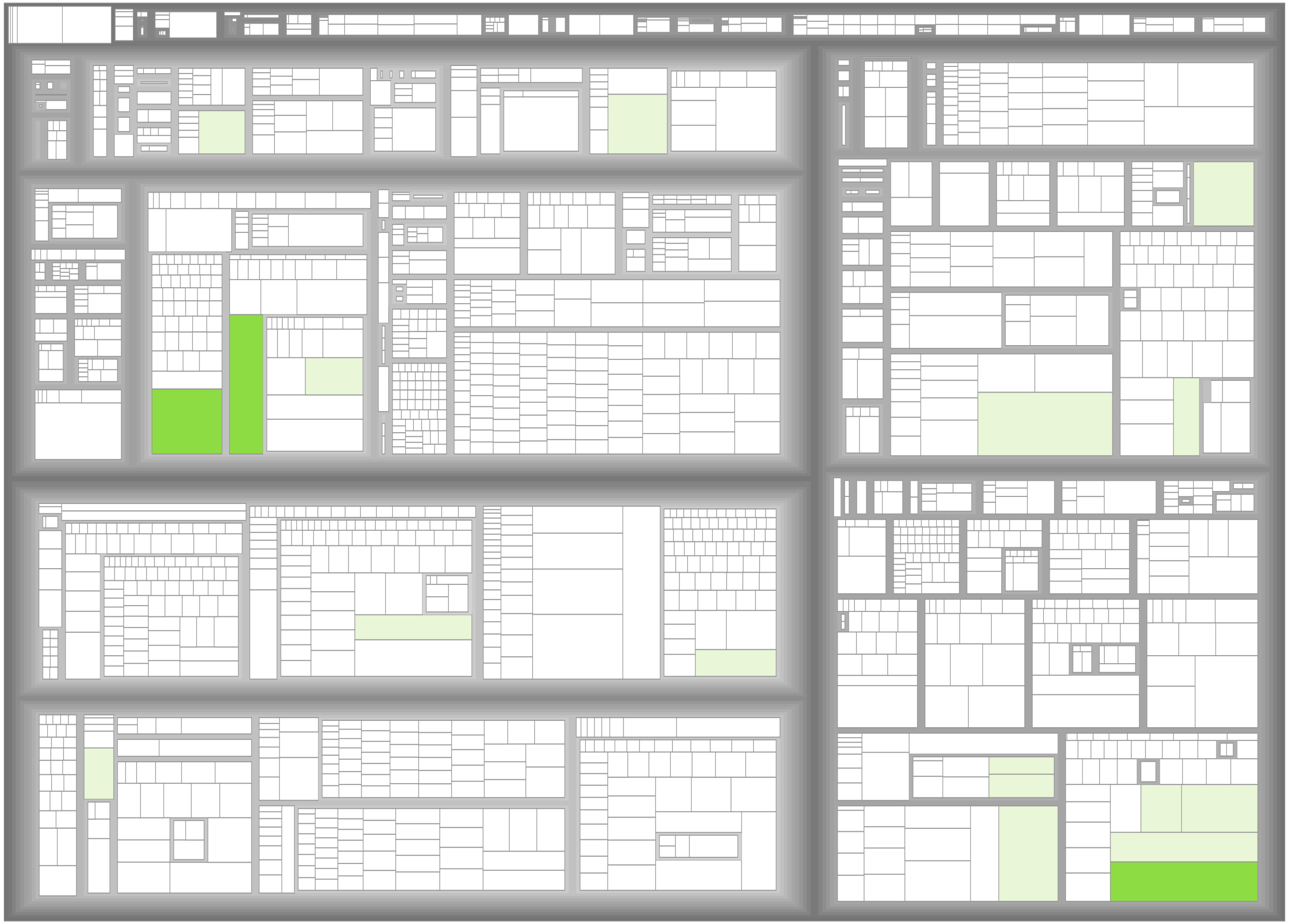
	1-4	5-9	10-
active	🟩	🟩	🟩
idle	🟦	🟦	🟦

Zone Crossroads are files that have been frequently change both from Asia and from Europe

sample project

Bazaar: Files with many **concurrent changes**

Error-Prone Code (due to potential inconsistencies). **Harder to Merge**



severity

	1-4	5-7	8-10
active			
idle			

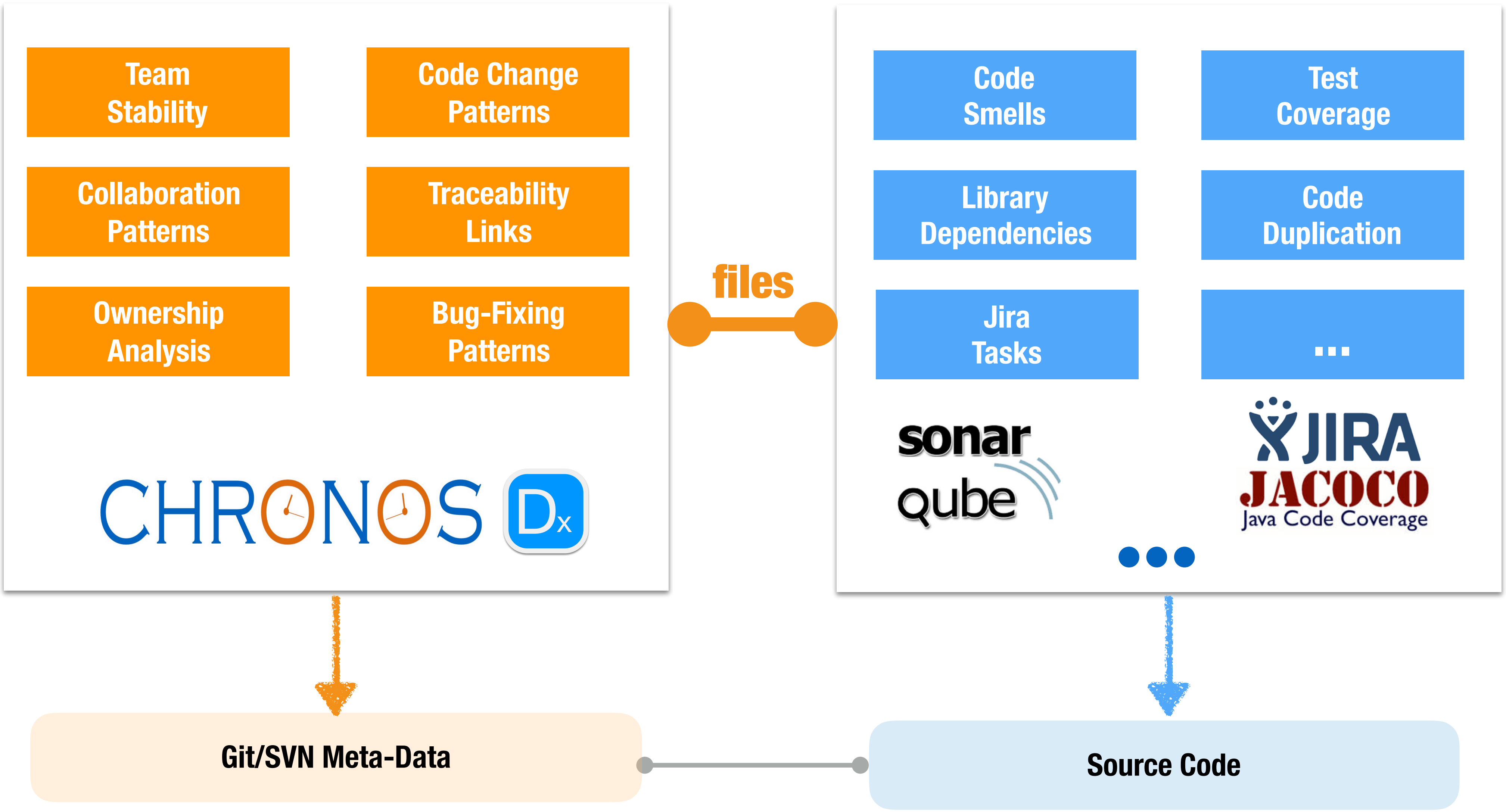
Bazaar: file that is changed by many developers over a short period of time (e.g. a sprint)

Integration with code analysis tools

The power of synergies

(CHRONOS meets other tools)

Risk Model



Team Stability

Code Change Patterns

Collaboration Patterns

Traceability Links

Ownership Analysis

Bug-Fixing Patterns

CHRONOS Dx

files

Code Smells

Test Coverage

Library Dependencies

Code Duplication

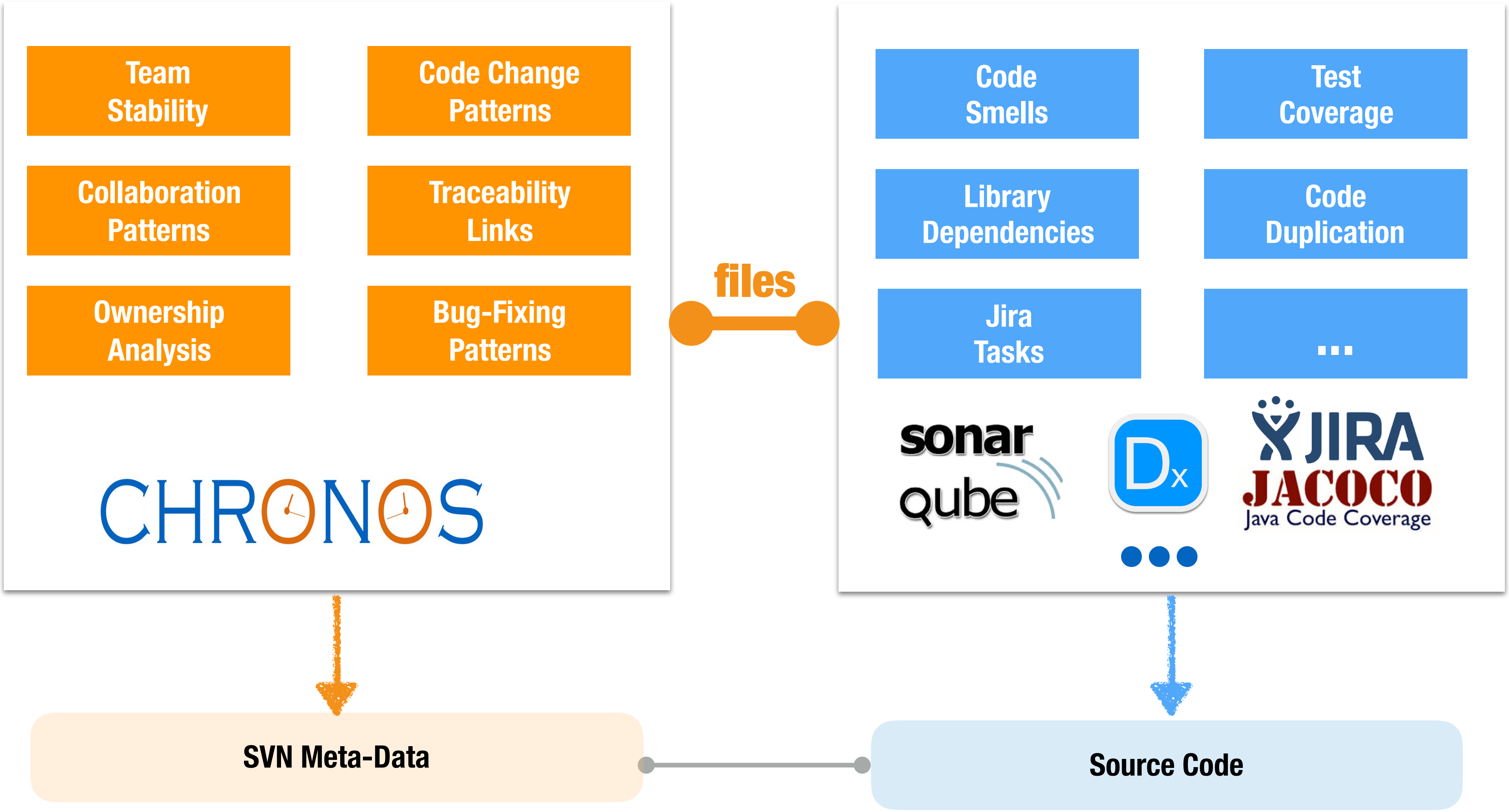
Jira Tasks

...

Git/SVN Meta-Data

Source Code

Risk & Quality Model



Team Stability

Code Change Patterns

Collaboration Patterns

Traceability Links

Ownership Analysis

Bug-Fixing Patterns

files

Code Smells

Test Coverage

Library Dependencies

Code Duplication

Jira Tasks

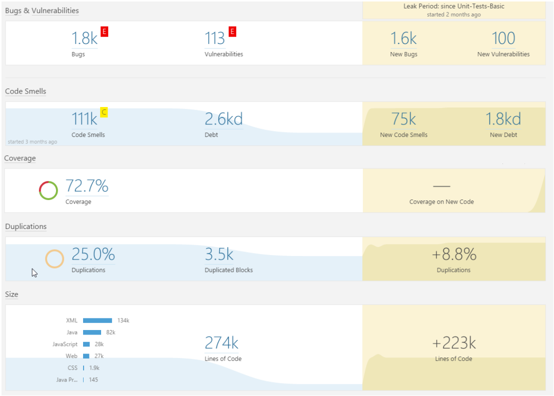
...

SVN Meta-Data

Source Code

Synergies…

sonarqube



Bugs & Vulnerabilities

Leak Period: since Unit-Tests-Basic
started 2 months ago

1.8k E Bugs

113 E Vulnerabilities

1.6k New Bugs

100 New Vulnerabilities

Code Smells

111k C Code Smells

2.6kd Debt

started 3 months ago

75k New Code Smells

1.8kd New Debt

Coverage

72.7% Coverage

— Coverage on New Code

Duplications

25.0% Duplications

3.5k Duplicated Blocks

+8.8% Duplications

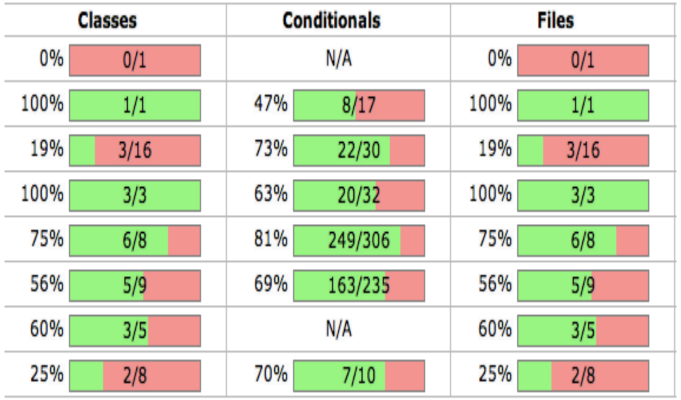
Size

XML 134k
Java 82k
JavaScript 28k
Web 27k
CSS 1.9k
Java Pr... 145

274k Lines of Code

+223k Lines of Code

Code Smells

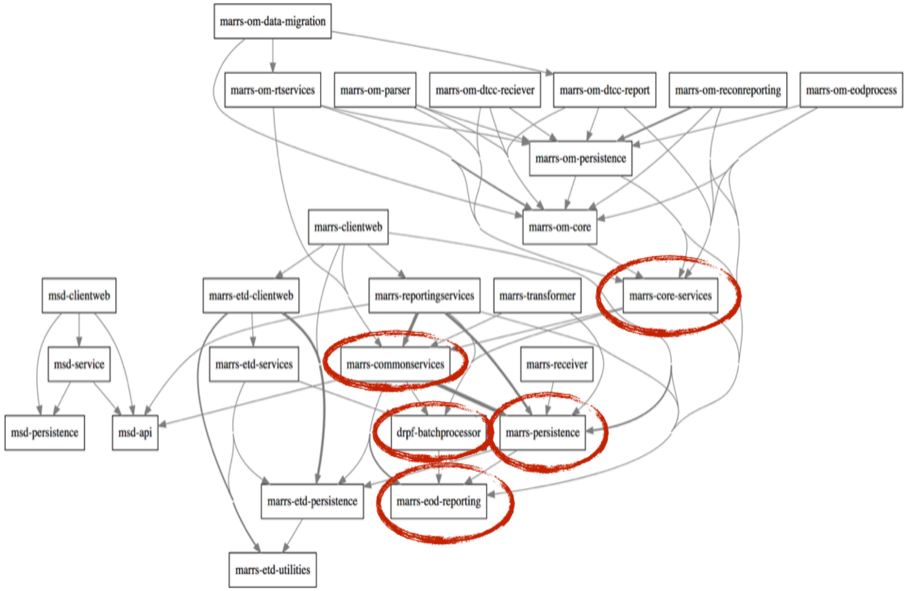


Classes		Conditionals		Files	
0%	0/1	N/A		0%	0/1
100%	1/1	47%	8/17	100%	1/1
19%	3/16	73%	22/30	19%	3/16
100%	3/3	63%	20/32	100%	3/3
75%	6/8	81%	249/306	75%	6/8
56%	5/9	69%	163/235	56%	5/9
60%	3/5	N/A		60%	3/5
25%	2/8	70%	7/10	25%	2/8

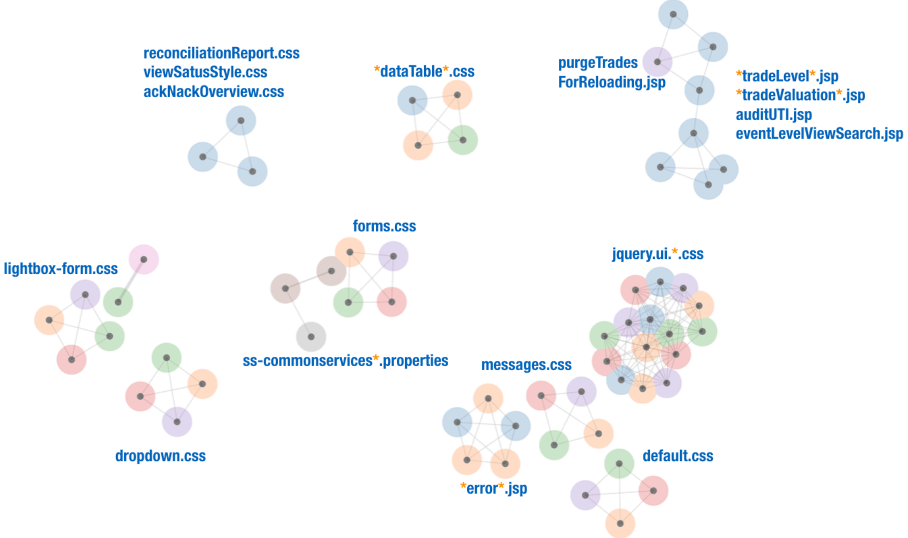
Test Coverage

BLACKDUCK

Library Dependencies



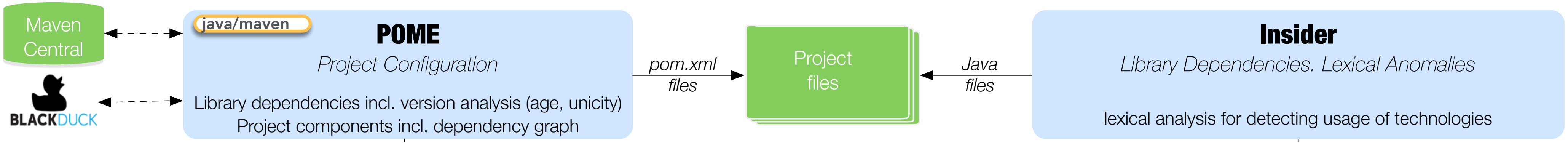
Architectural Issues



Code Duplication

The analysis of library dependencies

Discover Technology Fingerprints



Maven Central

BLACKDUCK

java/maven

POME

Project Configuration

Library dependencies incl. version analysis (age, unicity)
Project components incl. dependency graph

pom.xml files

Project files

Java files

Insider

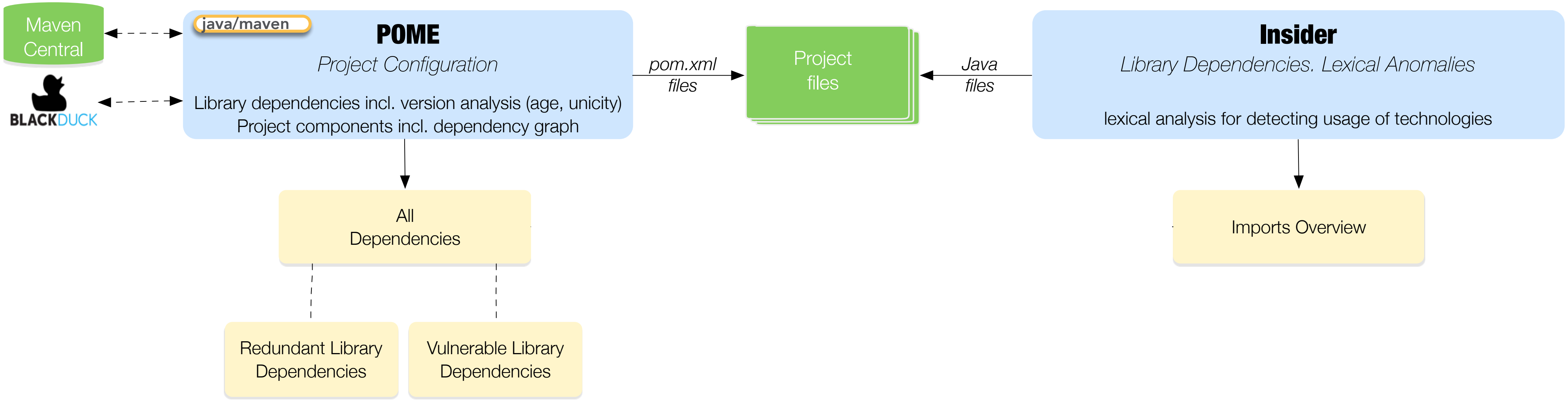
Library Dependencies. Lexical Anomalies

lexical analysis for detecting usage of technologies

Legend

🟩 input | 🟦 **tool** | 🟨 result

Discover Technology Fingerprints



Maven Central

BLACKDUCK

java/maven

POME

Project Configuration

Library dependencies incl. version analysis (age, unicity)
Project components incl. dependency graph

pom.xml files

Project files

Java files

Insider

Library Dependencies. Lexical Anomalies

lexical analysis for detecting usage of technologies

All Dependencies

Redundant Library Dependencies

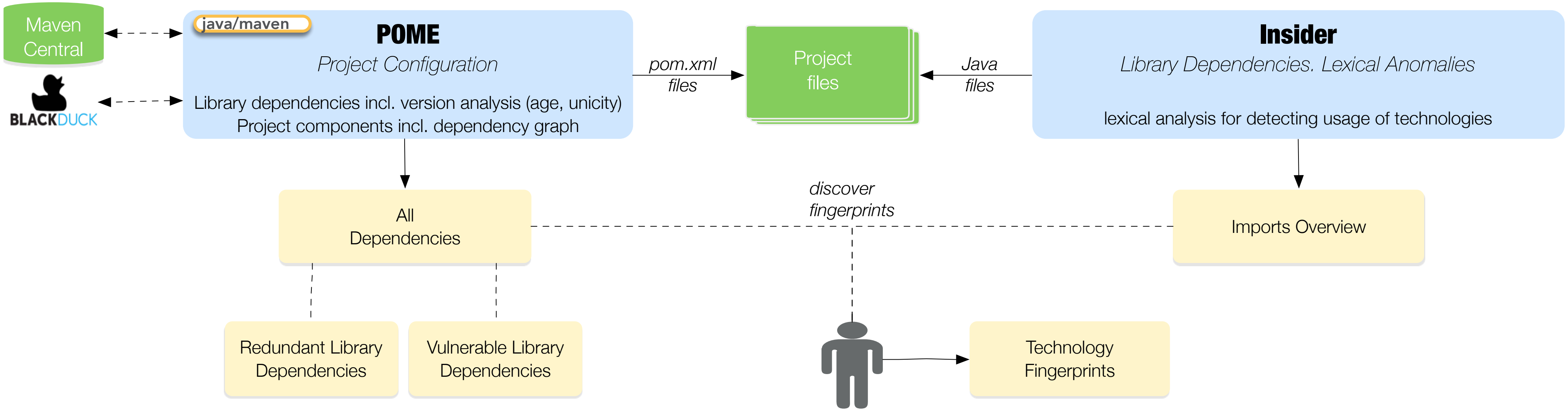
Vulnerable Library Dependencies

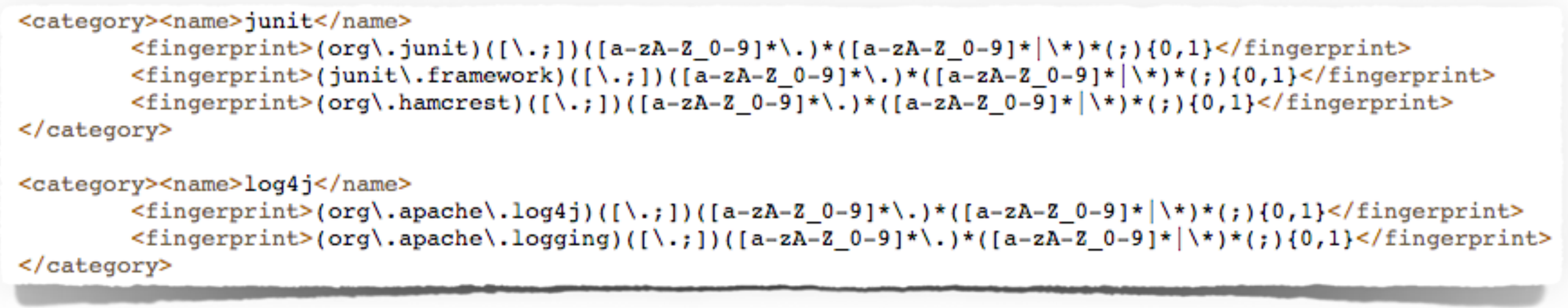
Imports Overview

Legend

🟩 input | **tool** | result

Discover Technology Fingerprints

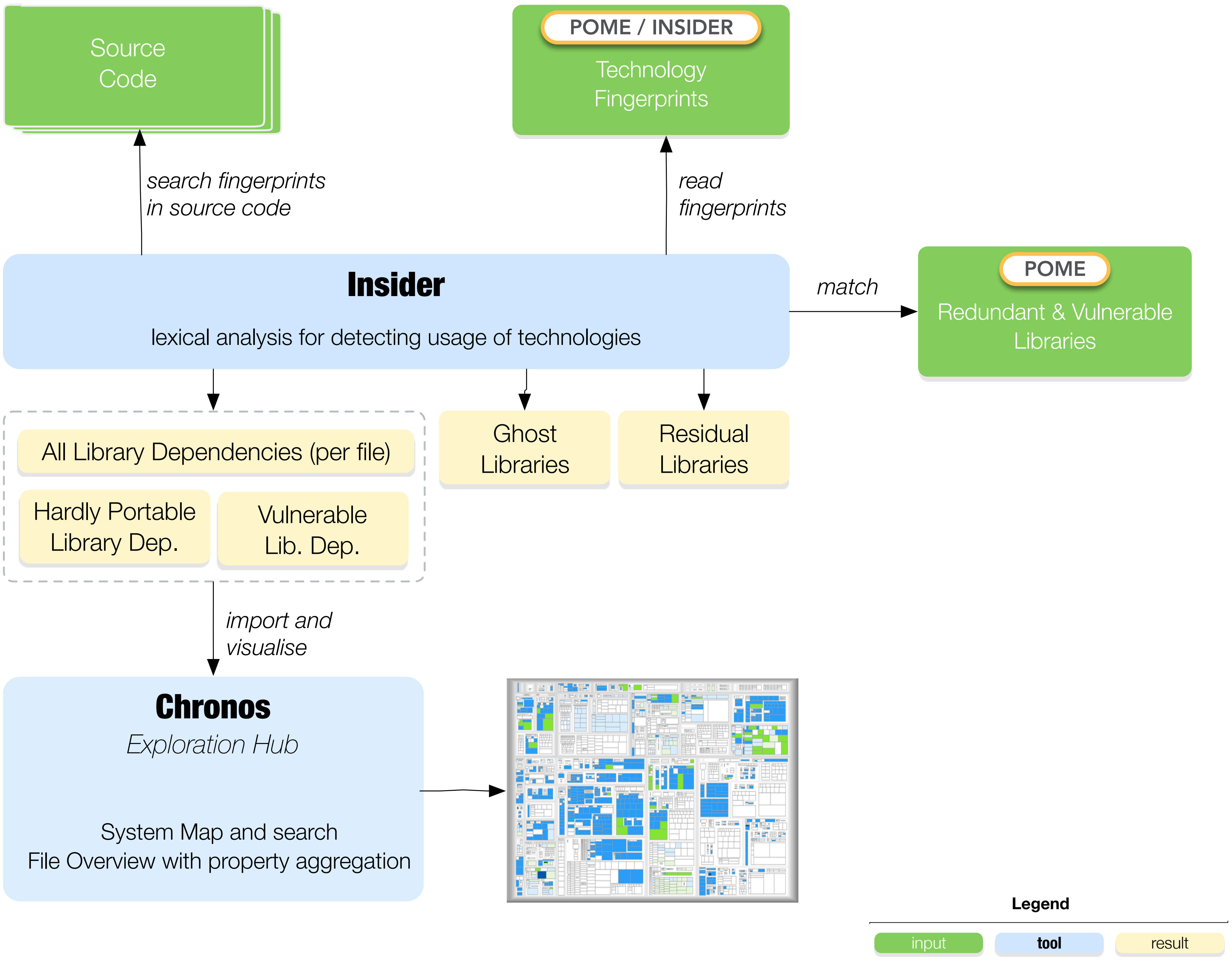

Maven Central
BLACKDUCK
java/maven
POME
Project Configuration
Library dependencies incl. version analysis (age, unicity)
Project components incl. dependency graph
pom.xml files
Project files
Java files
Insider
Library Dependencies. Lexical Anomalies
lexical analysis for detecting usage of technologies
All Dependencies
discover fingerprints
Imports Overview
Redundant Library Dependencies
Vulnerable Library Dependencies
Technology Fingerprints


```
<category><name>junit</name>
        <fingerprint>(org\.junit)([\.;])([a-zA-Z_0-9]*\.)*([a-zA-Z_0-9]*|\*)*(;){0,1}</fingerprint>
        <fingerprint>(junit\.framework)([\.;])([a-zA-Z_0-9]*\.)*([a-zA-Z_0-9]*|\*)*(;){0,1}</fingerprint>
        <fingerprint>(org\.hamcrest)([\.;])([a-zA-Z_0-9]*\.)*([a-zA-Z_0-9]*|\*)*(;){0,1}</fingerprint>
</category>

<category><name>log4j</name>
        <fingerprint>(org\.apache\.log4j)([\.;])([a-zA-Z_0-9]*\.)*([a-zA-Z_0-9]*|\*)*(;){0,1}</fingerprint>
        <fingerprint>(org\.apache\.logging)([\.;])([a-zA-Z_0-9]*\.)*([a-zA-Z_0-9]*|\*)*(;){0,1}</fingerprint>
</category>
```

Legend

input tool result

Discover Code Impact of Libraries



Source Code

POME / INSIDER

Technology Fingerprints

search fingerprints in source code

read fingerprints

Insider

lexical analysis for detecting usage of technologies

match

POME

Redundant & Vulnerable Libraries

All Library Dependencies (per file)

Hardly Portable Library Dep.

Vulnerable Lib. Dep.

Ghost Libraries

Residual Libraries

import and visualise

Chronos

Exploration Hub

System Map and search

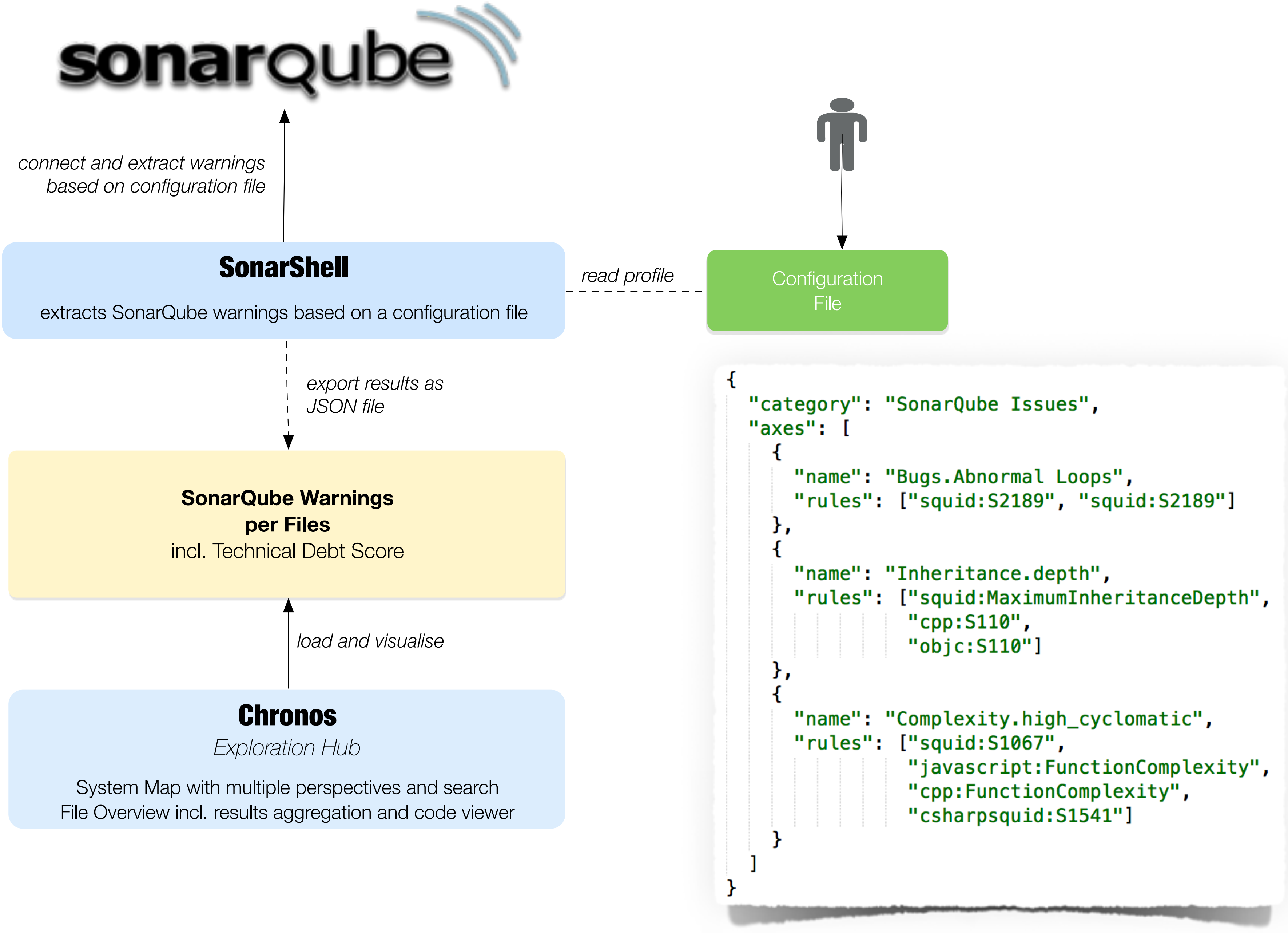
File Overview with property aggregation

Legend

input	tool	result

The analysis of code smells

Code Smells: CHRONOS + sonarqube



sonarqube

connect and extract warnings based on configuration file

SonarShell

extracts SonarQube warnings based on a configuration file

read profile

Configuration File

export results as JSON file

SonarQube Warnings per Files

incl. Technical Debt Score

load and visualise

Chronos

Exploration Hub

System Map with multiple perspectives and search

File Overview incl. results aggregation and code viewer

```
{
  "category": "SonarQube Issues",
  "axes": [
    {
      "name": "Bugs.Abnormal Loops",
      "rules": ["squid:S2189", "squid:S2189"]
    },
    {
      "name": "Inheritance.depth",
      "rules": ["squid:MaximumInheritanceDepth",
                "cpp:S110",
                "objc:S110"]
    },
    {
      "name": "Complexity.high_cyclomatic",
      "rules": ["squid:S1067",
                "javascript:FunctionComplexity",
                "cpp:FunctionComplexity",
                "csharpsquid:S1541"]
    }
  ]
}
```

https://github.com/cezarcoca/sonar-shell

Legend

input	tool	result