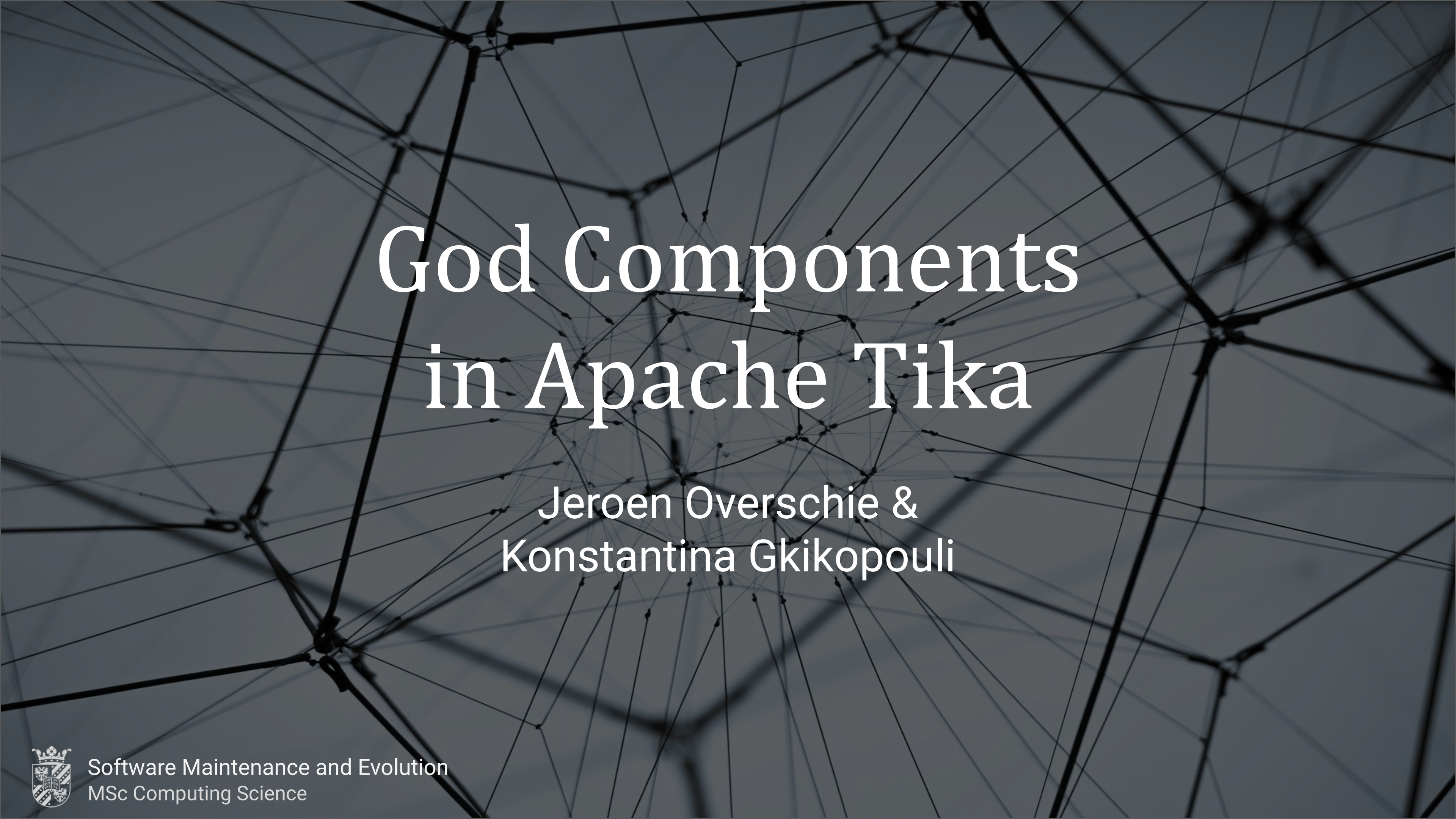


God Components in Apache Tika

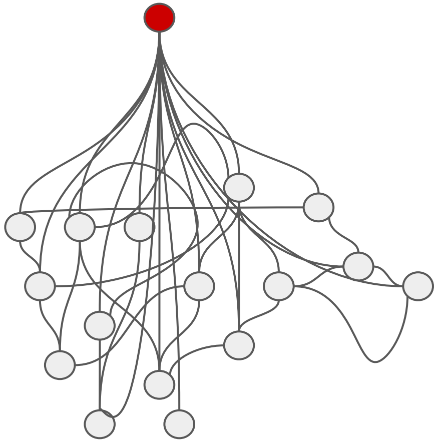
Jeroen Overschie &
Konstantina Gkikopouli

Software Maintenance and Evolution
MSc Computing Science

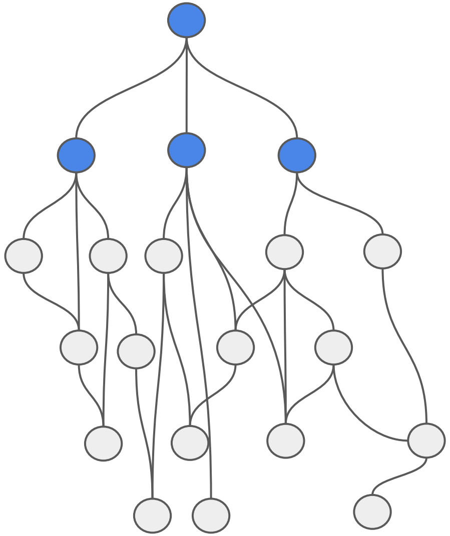
What is a God Component ?

It *knows too much* or *does too much*

→ Software **anti-pattern**

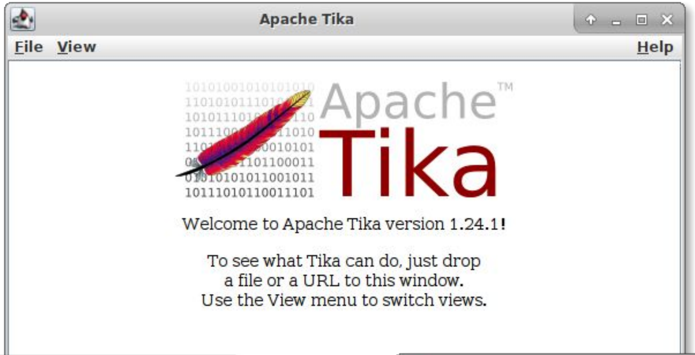


- → Smaller problems
- → Single responsibility
- → Reusable components



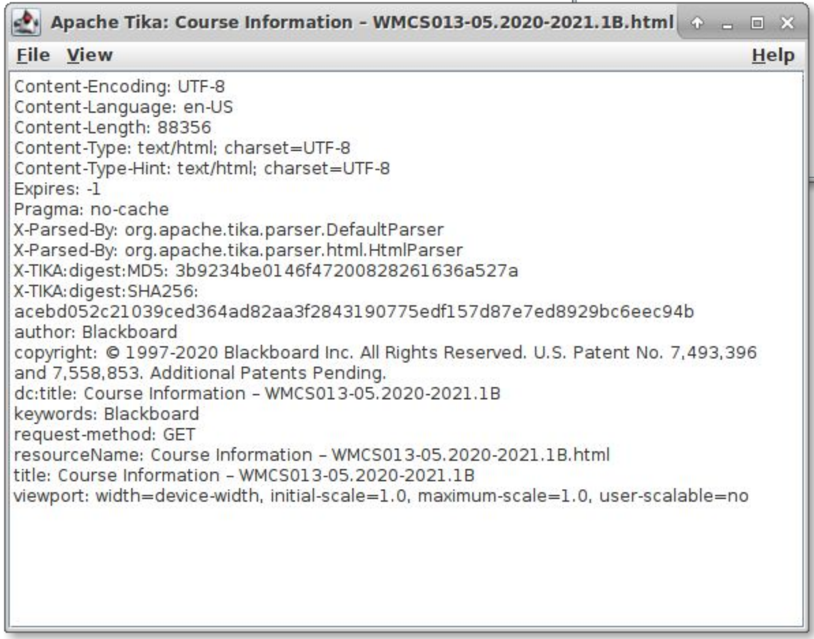
[i] Connie U Smith and Lloyd G Williams. Software performance antipatterns, pages 127–136, 2000
[ii] God object Wikipedia page
[iii] https://edmundkirwan.com/general/spaghetti.html

What is
Apache **Tika** ?

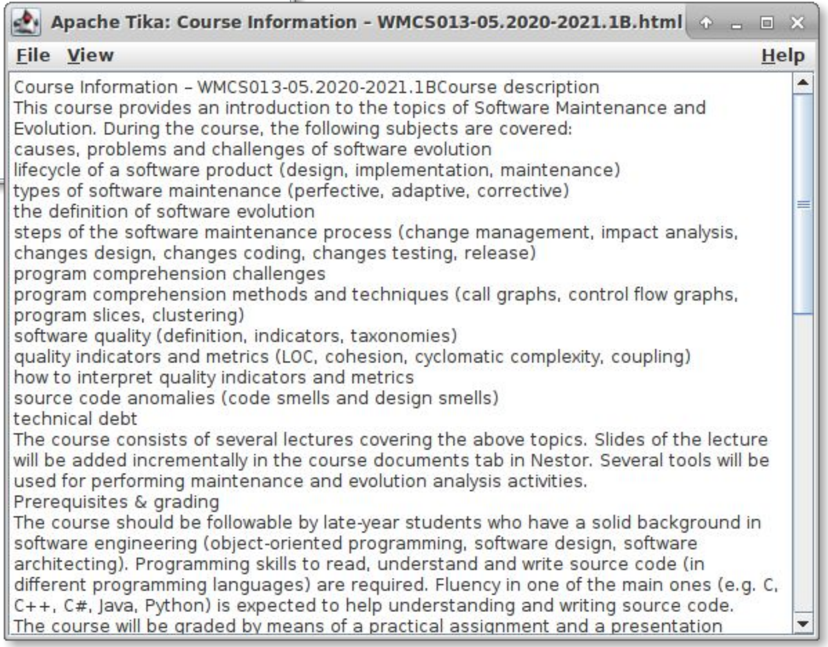


Welcome to Apache Tika version 1.24.1!

To see what Tika can do, just drop
a file or a URL to this window.
Use the View menu to switch views.


Apache Tika: Course Information - WMCS013-05.2020-2021.1B.html
File
View
Help

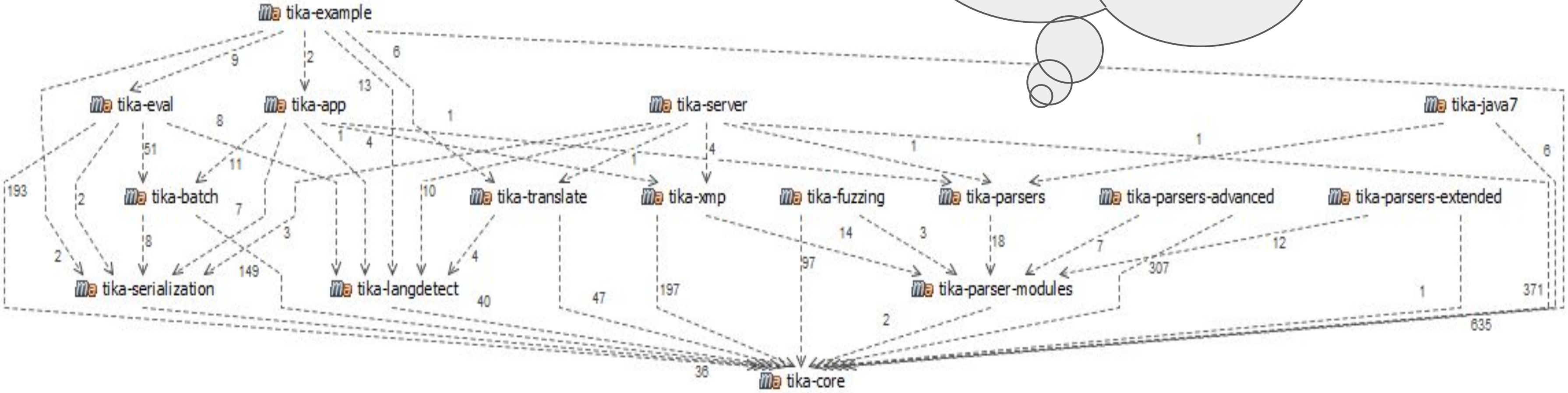
```
Content-Encoding: UTF-8
Content-Language: en-US
Content-Length: 88356
Content-Type: text/html; charset=UTF-8
Content-Type-Hint: text/html; charset=UTF-8
Expires: -1
Pragma: no-cache
X-Parsed-By: org.apache.tika.parser.DefaultParser
X-Parsed-By: org.apache.tika.parser.html.HtmlParser
X-TIKA:digest:MD5: 3b9234be0146f47200828261636a527a
X-TIKA:digest:SHA256:
acebd052c21039ced364ad82aa3f2843190775edf157d87e7ed8929bc6eec94b
author: Blackboard
copyright: © 1997-2020 Blackboard Inc. All Rights Reserved. U.S. Patent No. 7,493,396
and 7,558,853. Additional Patents Pending.
dc:title: Course Information - WMCS013-05.2020-2021.1B
keywords: Blackboard
request-method: GET
resourceName: Course Information - WMCS013-05.2020-2021.1B.html
title: Course Information - WMCS013-05.2020-2021.1B
viewport: width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no
```


Apache Tika: Course Information - WMCS013-05.2020-2021.1B.html
File
View
Help

```
Course Information - WMCS013-05.2020-2021.1BCourse description
This course provides an introduction to the topics of Software Maintenance and
Evolution. During the course, the following subjects are covered:
causes, problems and challenges of software evolution
lifecycle of a software product (design, implementation, maintenance)
types of software maintenance (perfective, adaptive, corrective)
the definition of software evolution
steps of the software maintenance process (change management, impact analysis,
changes design, changes coding, changes testing, release)
program comprehension challenges
program comprehension methods and techniques (call graphs, control flow graphs,
program slices, clustering)
software quality (definition, indicators, taxonomies)
quality indicators and metrics (LOC, cohesion, cyclomatic complexity, coupling)
how to interpret quality indicators and metrics
source code anomalies (code smells and design smells)
technical debt
The course consists of several lectures covering the above topics. Slides of the lecture
will be added incrementally in the course documents tab in Nestor. Several tools will be
used for performing maintenance and evolution analysis activities.
Prerequisites & grading
The course should be followable by late-year students who have a solid background in
software engineering (object-oriented programming, software design, software
architecting). Programming skills to read, understand and write source code (in
different programming languages) are required. Fluency in one of the main ones (e.g. C,
C++, C#, Java, Python) is expected to help understanding and writing source code.
The course will be graded by means of a practical assignment and a presentation
```

How does **Apache Tika** look like under the hood?

Which are God Components?



How does **Apache Tika** look like under the hood?

Which are God Components?

Designite

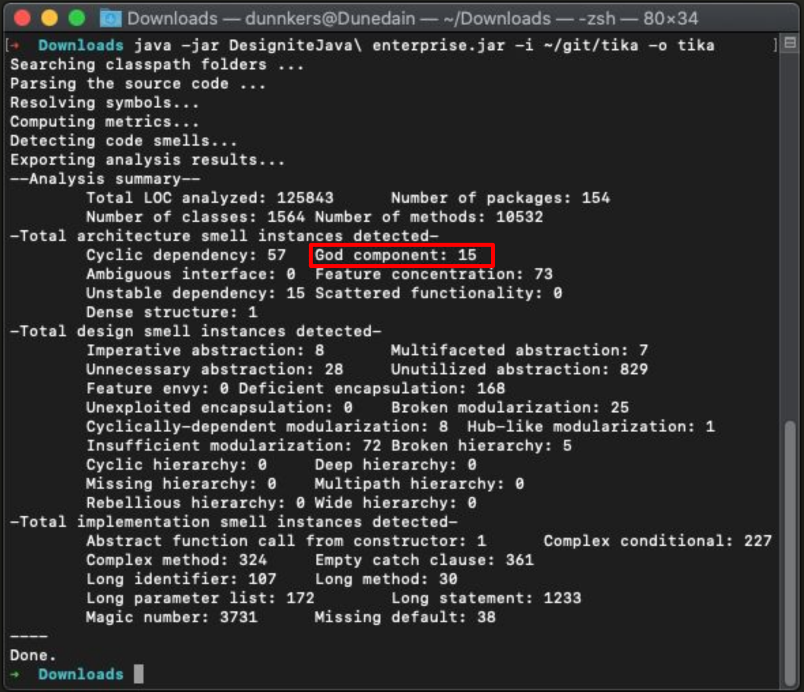
Running Designite

TERMINAL

```
$ git clone
github.com/apache/tika.git
~/git/tika



$ java -jar Designite.jar -i
~/git/tika -o ./out
```



```
→  Downloads java -jar DesigniteJava\ enterprise.jar -i ~/git/tika -o tika
Searching classpath folders ...
Parsing the source code ...
Resolving symbols...
Computing metrics...
Detecting code smells...
Exporting analysis results...
--Analysis summary--
        Total LOC analyzed: 125843      Number of packages: 154
        Number of classes: 1564 Number of methods: 10532
-Total architecture smell instances detected-
        Cyclic dependency: 57   God component: 15
        Ambiguous interface: 0  Feature concentration: 73
        Unstable dependency: 15 Scattered functionality: 0
        Dense structure: 1
-Total design smell instances detected-
        Imperative abstraction: 8       Multifaceted abstraction: 7
        Unnecessary abstraction: 28     Unutilized abstraction: 829
        Feature envy: 0 Deficient encapsulation: 168
        Unexploited encapsulation: 0    Broken modularization: 25
        Cyclically-dependent modularization: 8  Hub-like modularization: 1
        Insufficient modularization: 72 Broken hierarchy: 5
        Cyclic hierarchy: 0     Deep hierarchy: 0
        Missing hierarchy: 0    Multipath hierarchy: 0
        Rebellious hierarchy: 0 Wide hierarchy: 0
-Total implementation smell instances detected-
        Abstract function call from constructor: 1      Complex conditional: 227
        Complex method: 324     Empty catch clause: 361
        Long identifier: 107    Long method: 30
        Long parameter list: 172        Long statement: 1233
        Magic number: 3731      Missing default: 38
----
Done.
→  Downloads
```

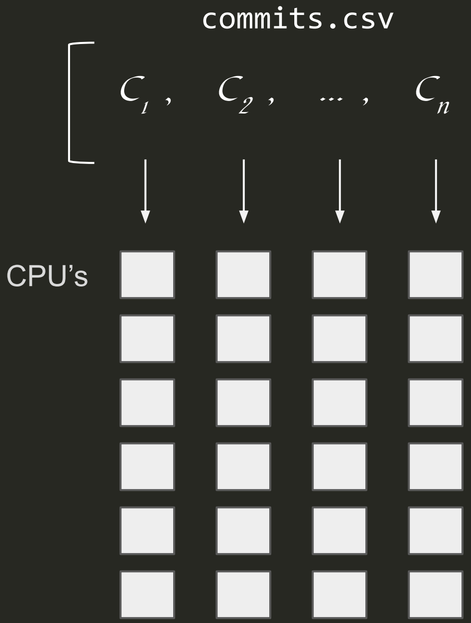
For every version
(commit) of the code

Would take
55 hours

Running Designite using Peregrine

TERMINAL

```
$ git log
--pretty=... >
commits.csv
```



Tika has about 5K commits

Designite runs are distributed across available CPU's
e.g. quad-core or 24-core

Running Designite

statistics.ipynb

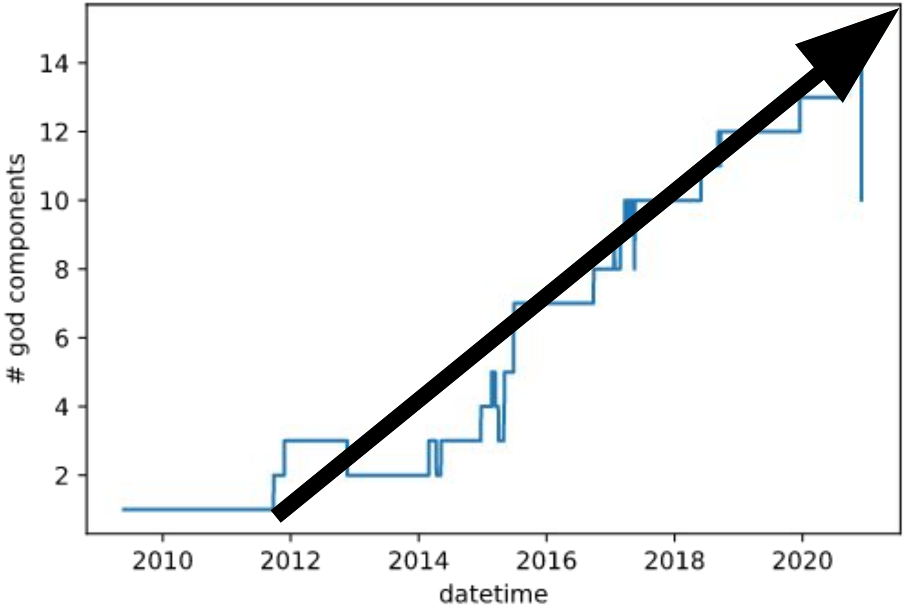
```python
import pandas as pd

all_reports = pd.read_csv('output/all_reports.csv')
all_reports.head()
```

commit	repo	package	smell	cause	metric
49bb4691393c016d8d65e6b11febca9e56feedef	tika-cpu_21	org.apache.tika.example	God Component	MANY_CLASSES	49
49bb4691393c016d8d65e6b11febca9e56feedef	tika-cpu_21	org.apache.tika.batch	God Component	MANY_CLASSES	31
49bb4691393c016d8d65e6b11febca9e56feedef	tika-cpu_21	org.apache.tika.detect	God Component	MANY_CLASSES	31

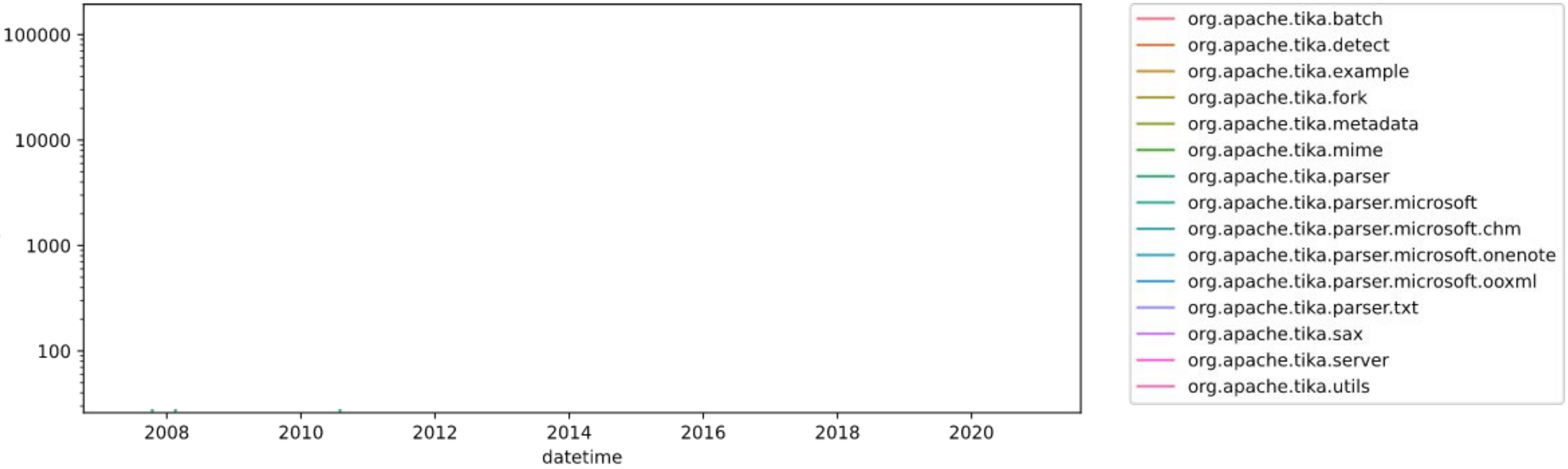
So what can we tell from the data?

So what can we tell from the data?



Trend

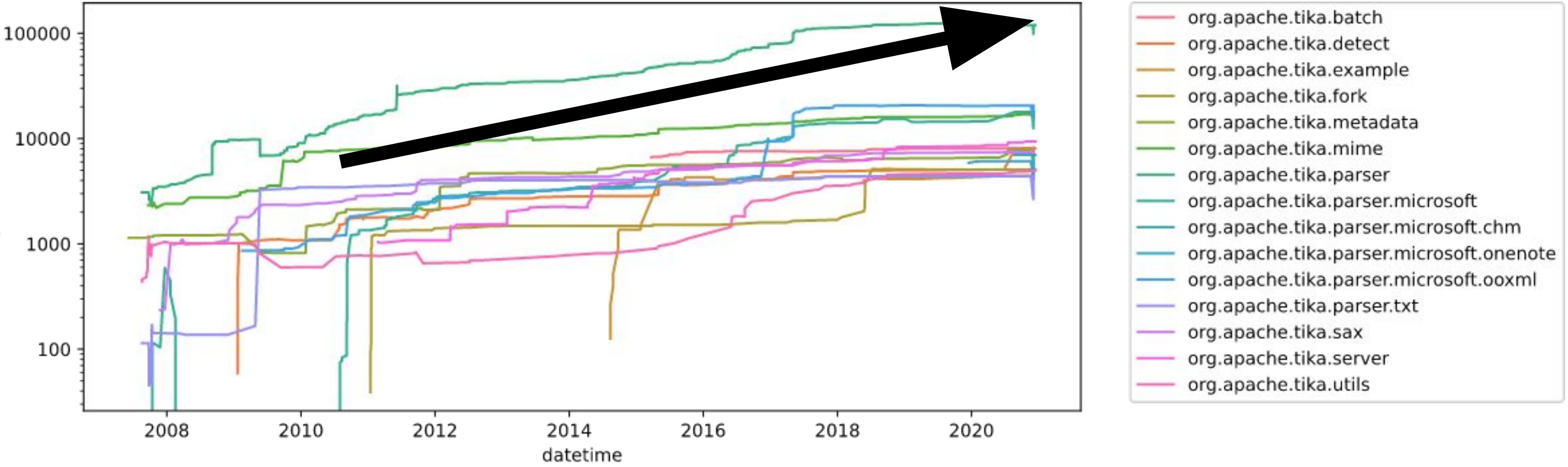
So what can we tell from the data?



Lines Of Code

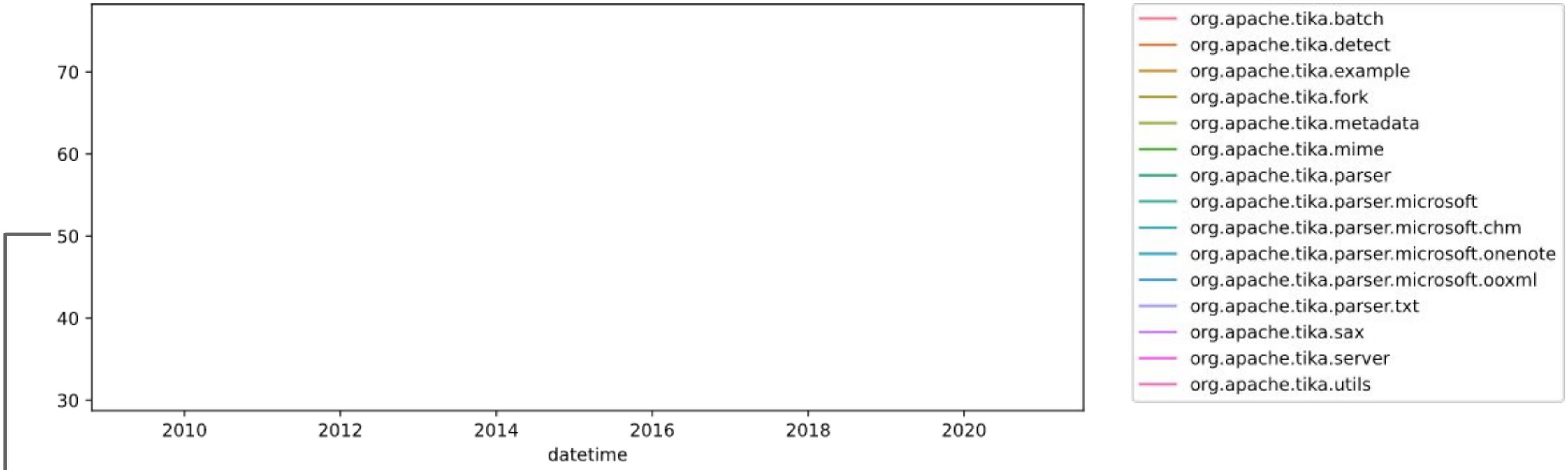
So what can we tell from the data?

Trend



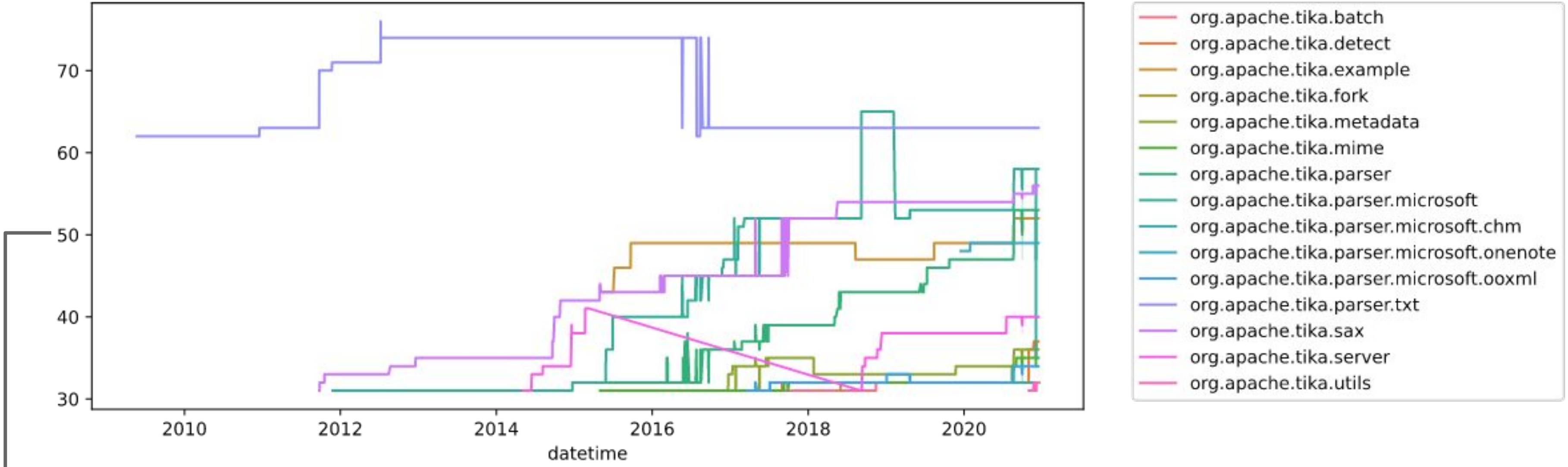
Lines Of Code

So what can we tell from the data?



# classes

So what can we tell from the data?

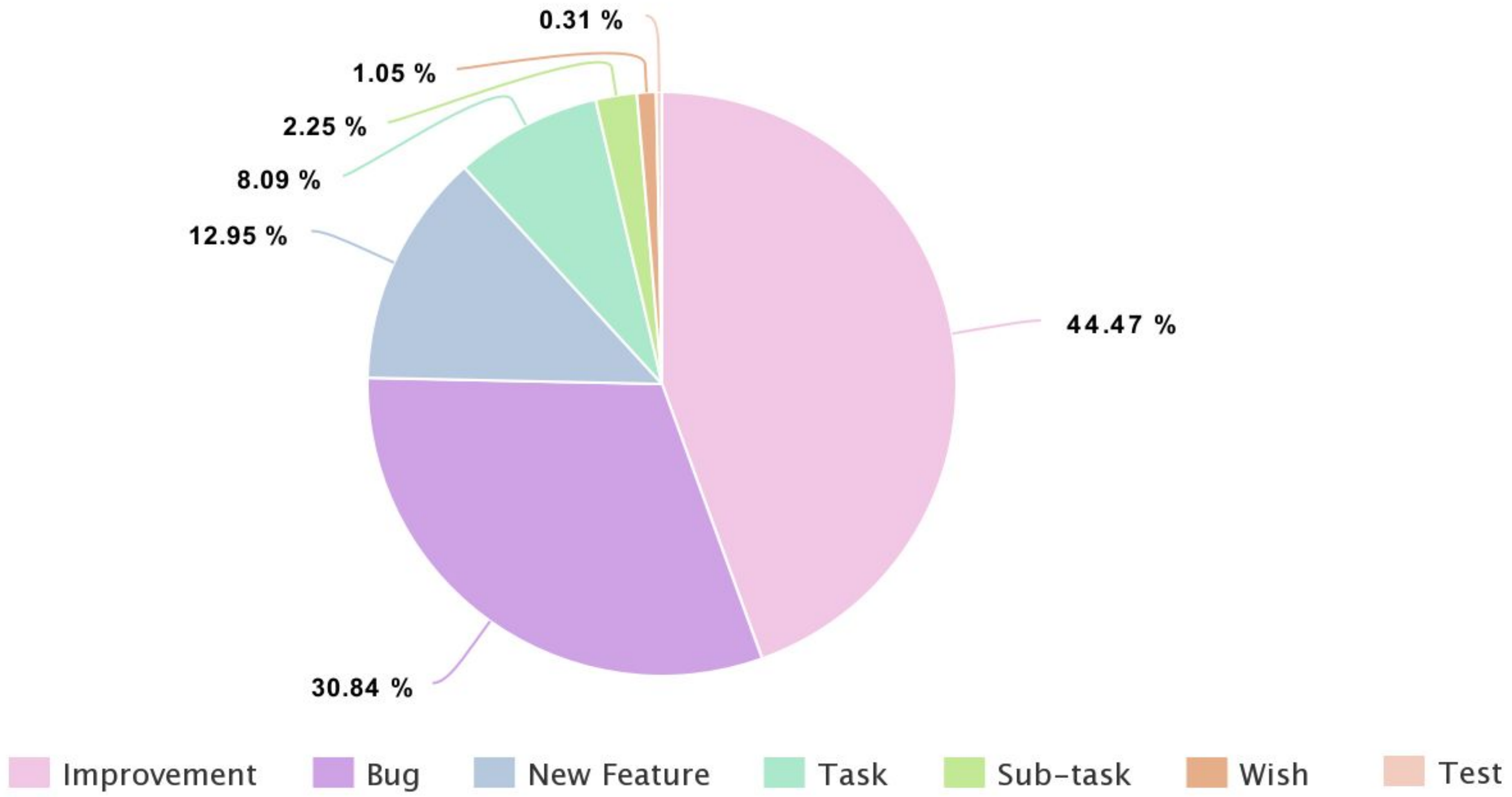


# classes

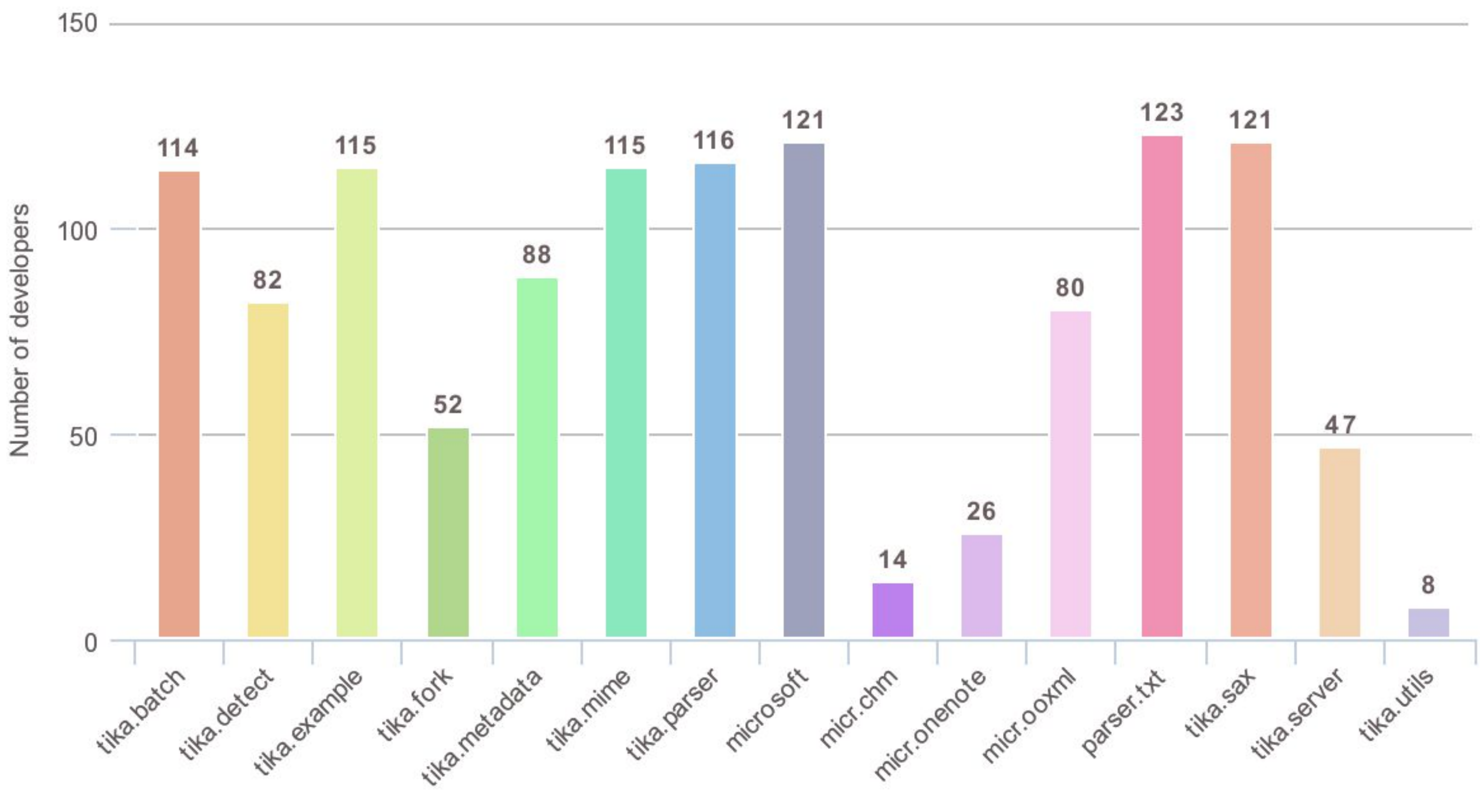
Let’s combine more data sources.

+ Developers in git repository
+ Jira issue tracker

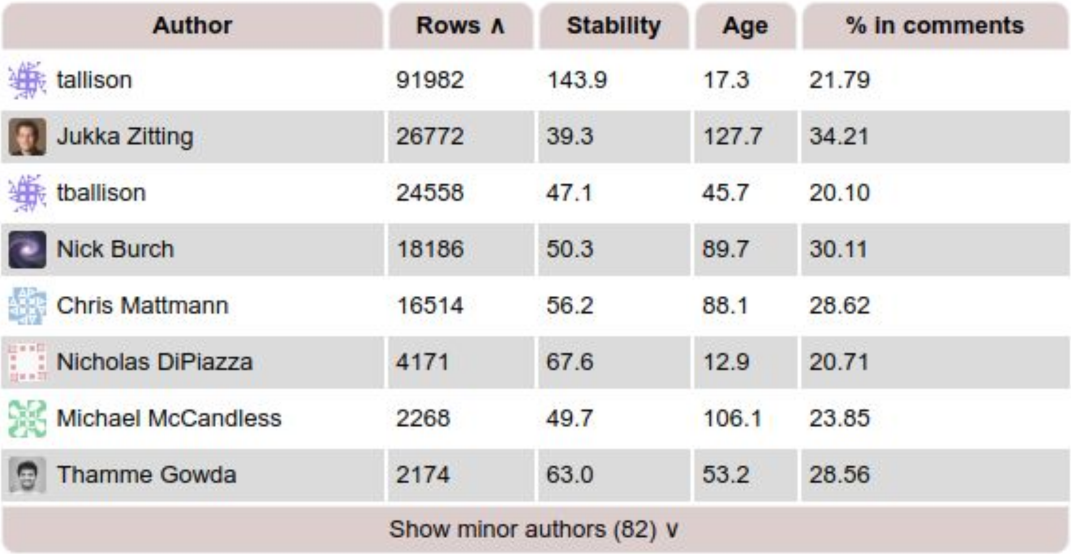
Types of Jira issues.



Developers per GC.

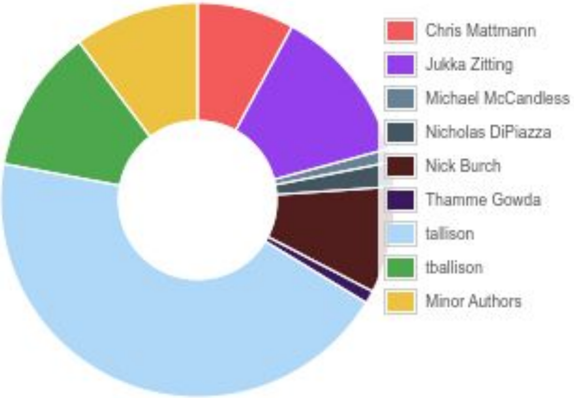


Developers in codebase.



Author	Rows ∧	Stability	Age	% in comments
tallison	91982	143.9	17.3	21.79
Jukka Zitting	26772	39.3	127.7	34.21
tballison	24558	47.1	45.7	20.10
Nick Burch	18186	50.3	89.7	30.11
Chris Mattmann	16514	56.2	88.1	28.62
Nicholas DiPiazza	4171	67.6	12.9	20.71
Michael McCandless	2268	49.7	106.1	23.85
Thamme Gowda	2174	63.0	53.2	28.56

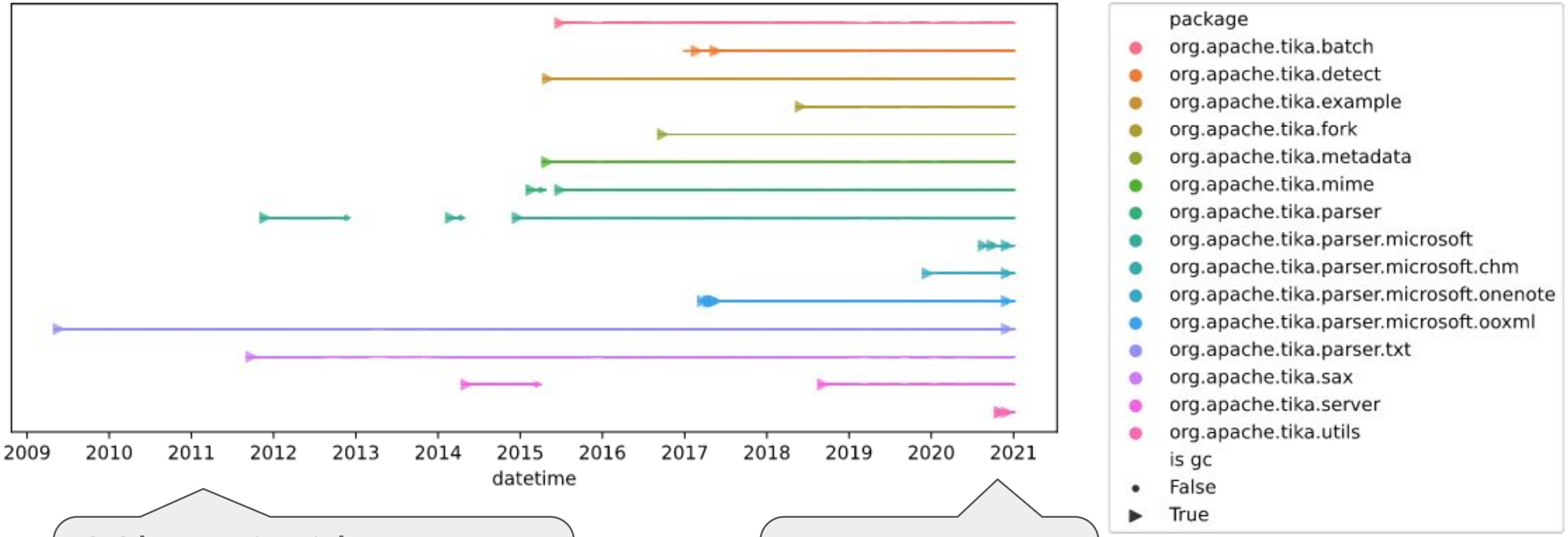
Show minor authors (82) ∨



Most of code was written only by a **handful of authors**

^ Data mined using gitinspector

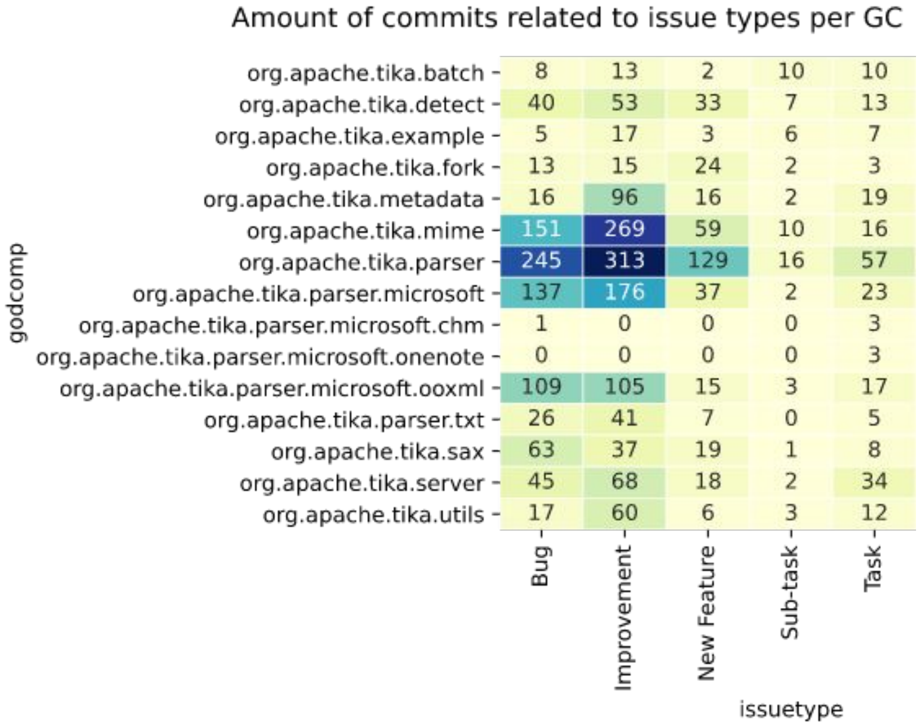
How long do God Components stay inside Tika?



GC's are inside system **~5 years** on average

All GC's are **still there** now

Types of Jira issues per GC.

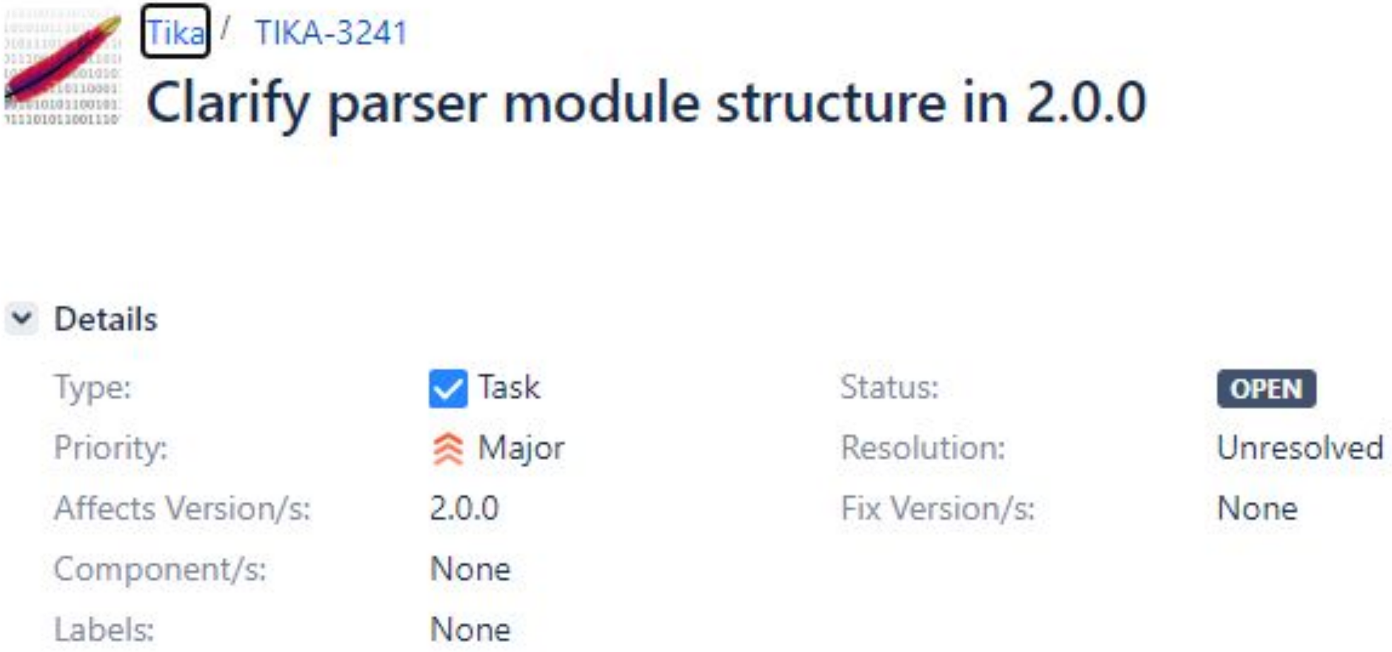


Amount of commits related to issue types per GC

godcomp	Bug	Improvement	New Feature	Sub-task	Task
org.apache.tika.batch	8	13	2	10	10
org.apache.tika.detect	40	53	33	7	13
org.apache.tika.example	5	17	3	6	7
org.apache.tika.fork	13	15	24	2	3
org.apache.tika.metadata	16	96	16	2	19
org.apache.tika.mime	151	269	59	10	16
org.apache.tika.parser	245	313	129	16	57
org.apache.tika.parser.microsoft	137	176	37	2	23
org.apache.tika.parser.microsoft.chm	1	0	0	0	3
org.apache.tika.parser.microsoft.onenote	0	0	0	0	3
org.apache.tika.parser.microsoft.ooxml	109	105	15	3	17
org.apache.tika.parser.txt	26	41	7	0	5
org.apache.tika.sax	63	37	19	1	8
org.apache.tika.server	45	68	18	2	34
org.apache.tika.utils	17	60	6	3	12

So what issue types were involved in re-factoring / buildup of GC’s?

Specific Jira issues.



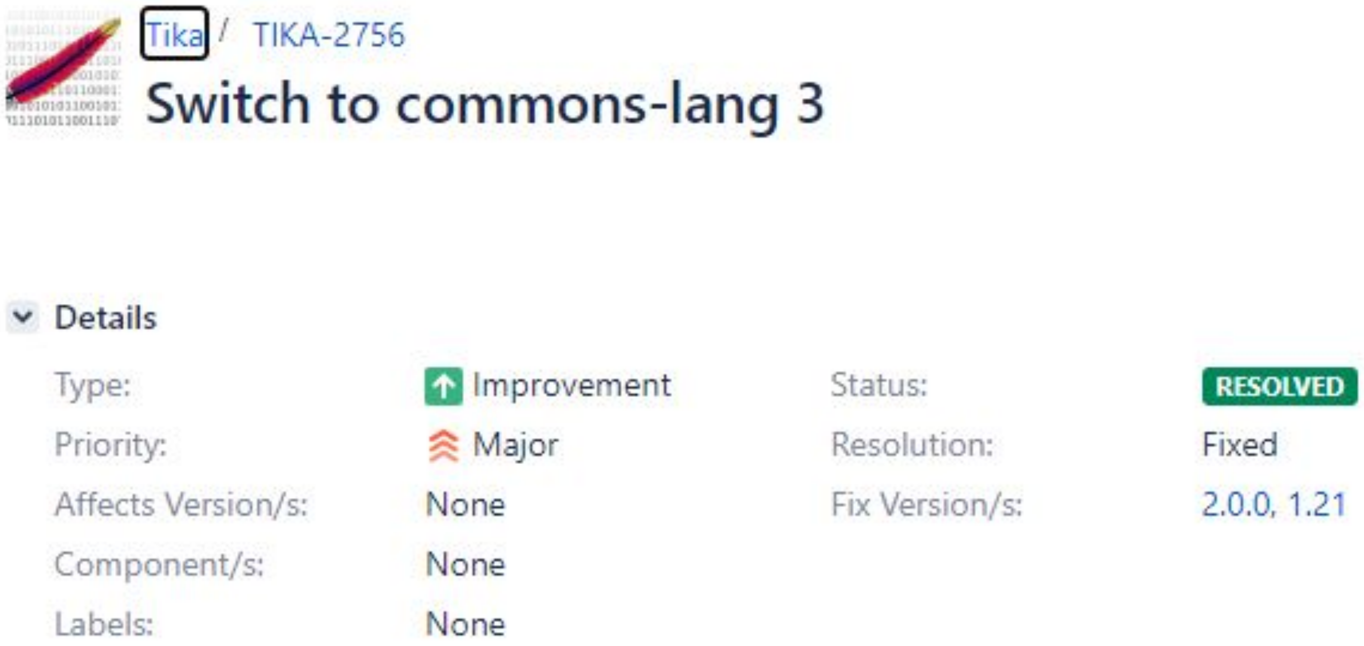
Tika / TIKA-3241

Clarify parser module structure in 2.0.0

Details

Type:	☑ Task	Status:	OPEN
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	2.0.0	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Specific Jira issues.



Tika / TIKA-2756

Switch to commons-lang 3

Details

Type:	Improvement	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	2.0.0, 1.21
Component/s:	None		
Labels:	None		

Conclusion

What is the rationale of the developers regarding the God Components?

So,

So, take care of your **codebase!** 😀

Thanks for listening

Any questions?