Are Java Programmers Transitioning to Multicore? A Large Scale Study of Java FLOSS

Weslley Torres Gustavo Pinto Benito Fernandes
João Paulo Oliveira Filipe Ximenes **Fernando Castor**

Informatics Center - Federal University of Pernambuco

October 23, 2011

Introduction

Problem

1. In spite of multicore and...
2. many languages providing constructs for concurrent programming...
3. we have no idea about how developers use these constructs in practice.

Introduction

Problem

1. In spite of multicore and...
2. many languages providing constructs for concurrent programming...
3. we have no idea about how developers use these constructs in practice.

Our Study

1. A study targeting a large-scale FLOSS repository
2. To discover what concurrency mechanisms programmers use.
3. The frequency of use and system evolution along time.

Introduction

Implications for Research and Practice

1. **Researchers**
   - 1.1 To design new mechanisms.
   - 1.2 To improve existing ones, based on development practice.
2. **Software Developers**: Might lead to more efficient use of existing abstractions
3. **It is important to know!**

Introduction

Implications for Research and Practice

1. **Researchers**
   1.1 To design new mechanisms.
   1.2 To improve existing ones, based on development practice.
2. **Software Developers**: Might lead to more efficient use of existing abstractions
3. **It is important to know!**

Why Java?

1. Widely used object-oriented programming language.
2. Supports for multi-threading (low level and high level).
3. Programming language with more projects at SourceForge (46.665 rojects).

Research Questions

- Two dimensions: **Spatial** and **Temporal**
- RQ1 - How often are the Java concurrency constructs employed in real applications?
- RQ2 - Are programmers aware about the transition from singlecore to multicore?

RQ1 - Metrics

Metric Category	Element
Concurrent collections	Instantiations of BlockingQueue,ConcurrentMap, SynchronousQueue, ConcurrentHashMap,
Synchronized	Occurrences of synchronized methods and blocks.
Atomic data types	Uses of AtomicInteger, AtomicLong, and AtomicBoolean
Barriers	Uses of CyclicBarrier and CountDownLatch
Locks	Instantiations of LockSupport, ReentrantLock
Others	extends Thread/Runnable, implements Runnable import java.util.concurrent (and its subpackages)
Size	Lines of Code

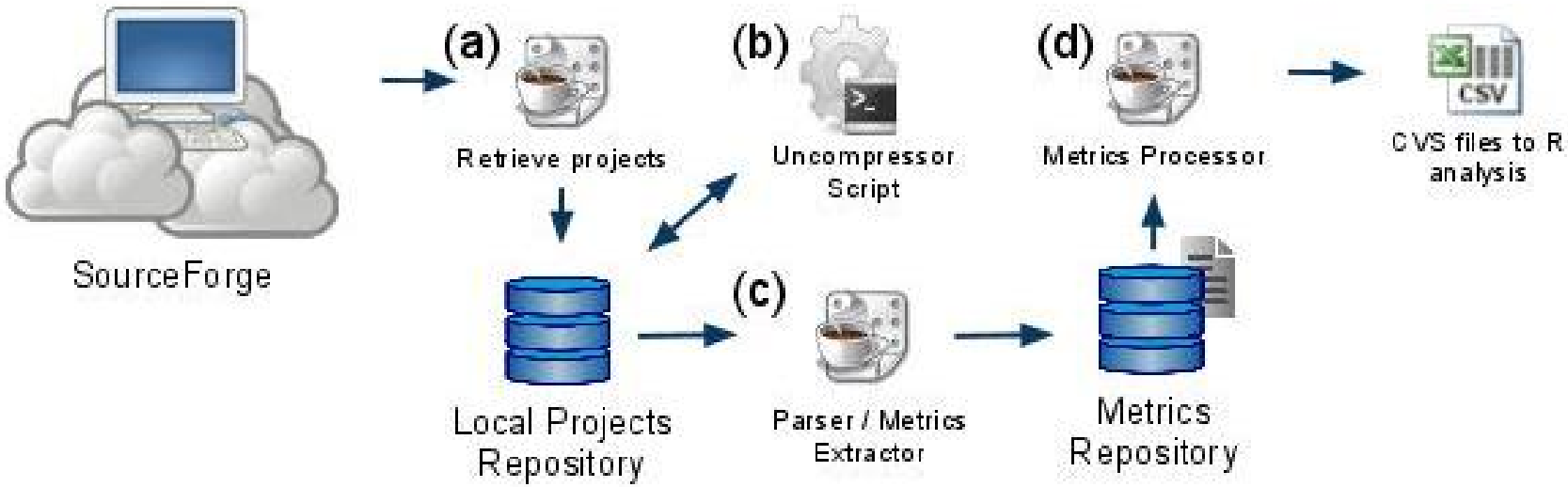
Table: Some Metrics

RQ2 - Are programmers aware about the transition from singlecore to multicore?

We have broken this question into three more:

- RQ2.1 - Are developers employing more concurrent programming constructs?
- RQ2.2 - Are developers wasting opportunities to use j.u.c.?
- RQ2.3 - Have threads been used to improve concurrency or parallelism?

Intrastructure



- Only projects with some version released after 2004
- Many projects were discarded

General information

#Projects (subprojects included)	2.343
#Small concurrent projects	1.300
#Small non-concurrent projects	489
#Medium concurrent projects	635
#Medium non-concurrent projects	32
#Big concurrent projects	199
#Big non-concurrent projects	0
# of LoC of the last version of the biggest project	1.702.972
Size on disk (all versions of all projects)	124GB

Table: General information about the projects.

RQ1 - How often are the Java concurrency constructs employed in real applications?

Metrics	Median			Mean			Std. Dev.			#Projects		
#1	4	14	51	11.2	39.47	119.7	22.56	65.88	189.73	703	535	189
#2	6	24	90	12.6	50.53	152.3	20.07	74.02	188.66	941	586	196
#3	1.5	3	6	2.41	4.8	9.46	2.17	6.07	11.32	520	399	152
#4	2	3	6	2.71	6.38	13.04	3.02	10.3	17.8	596	416	168
#5	1	2	1	1.91	2.86	6.05	1.24	3.60	14.17	6	9	17

Table: Metricts by categories (small/medium/big projects, respectively)

where:

- #1: synchronized blocks
- #2: synchronized methods
- #3: classes extending Thread
- #4: implementing Runnable
- #5: interfaces extending Runnable

RQ1 - How often are the Java concurrency constructs employed in real applications?

metrics	Median			Mean			Std. Dev.			#Projects		
# 1	2	3	6	3.07	9.05	16.02	2.71	16.16	21.63	39	70	47
# 2	1	2	4	2.32	7.17	6.77	3.48	17.91	8.70	43	82	53
# 3	2	2	2	3.80	3.61	2.88	3.83	3.79	2.26	31	42	26
# 4	2	3	4	3.17	7.12	13.07	4.12	11.17	21.01	86	97	61
# 5	2	2	3	2.25	6.44	5.8	1.81	26.35	8.07	71	92	60

Table: Metricts by categories (small/medium/big projects, respectively)

where:

- #1: atomic data types
- #2: locks
- #3: futures
- #4: concurrent collections
- #5: executors

Some facts

- 49% of the big **concurrent** projects, 32.4% of the medium ones, and only 15.5% of the small ones employ `java.util.concurrent`
  - For `synchronized` blocks, the percentages are 94.97%, 84.64%, and 54.07%, respectively

Some facts

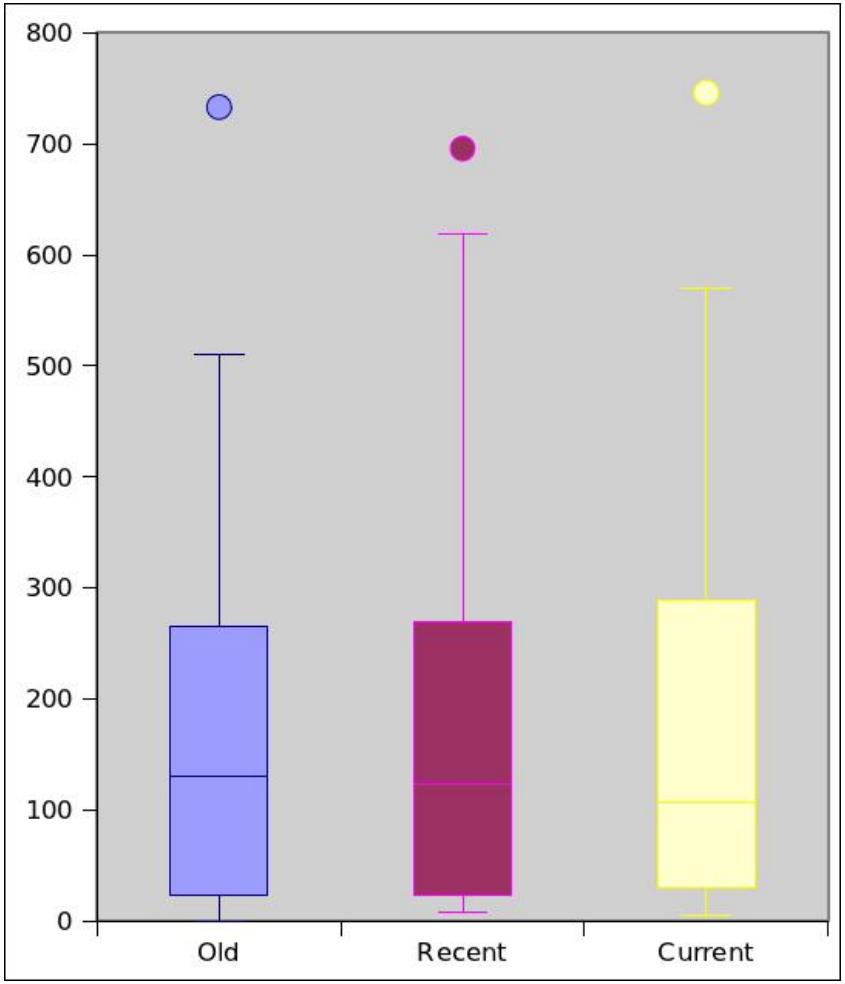
- 49% of the big **concurrent** projects, 32.4% of the medium ones, and only 15.5% of the small ones employ `java.util.concurrent`
  - For `synchronized` blocks, the percentages are 94.97%, 84.64%, and 54.07%, respectively
- 139 projects define threads but do not use `synchronized`
  - Often worker threads in small ($<$ 10KLoC) projects

Some facts

- 49% of the big **concurrent** projects, 32.4% of the medium ones, and only 15.5% of the small ones employ `java.util.concurrent`
  - For `synchronized` blocks, the percentages are 94.97%, 84.64%, and 54.07%, respectively
- 139 projects define threads but do not use `synchronized`
  - Often worker threads in small ($<$ 10KLoC) projects
- 44 projects employ the `java.util.concurrent` library but not the `synchronized` keyword
- 10% of the analyzed projects employ concurrent collections, particularly `ConcurrentHashMap`

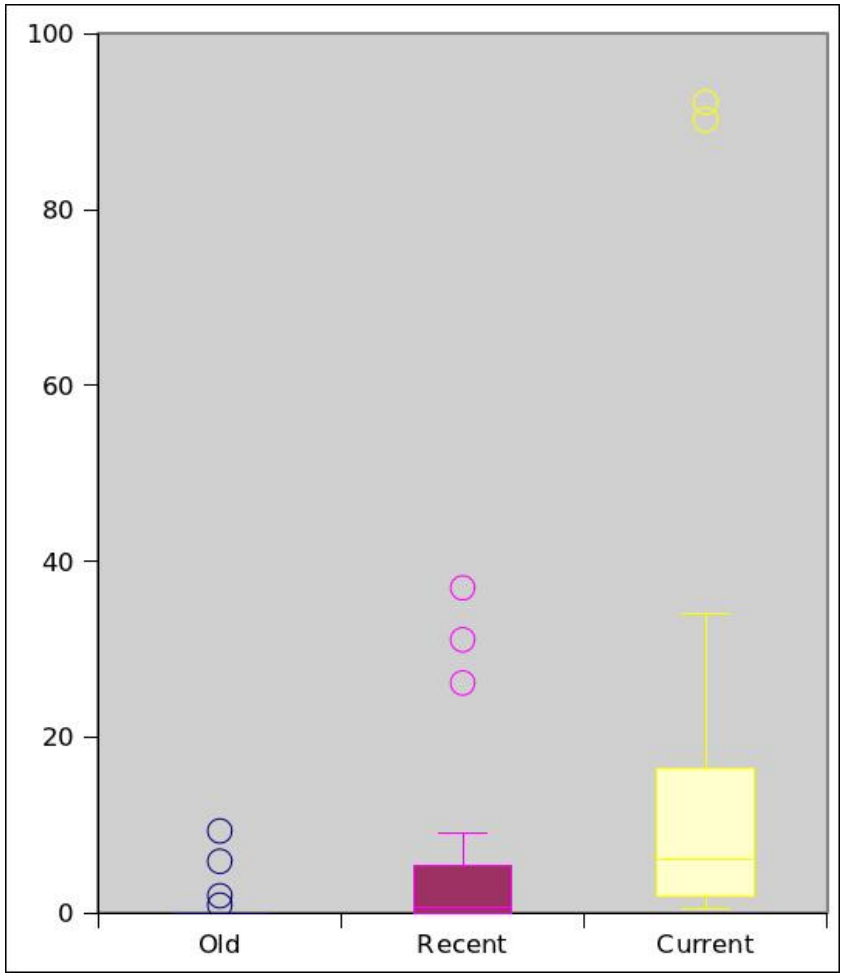
RQ2.1 – Usage of the `synchronized` keyword

Per 100KLoC



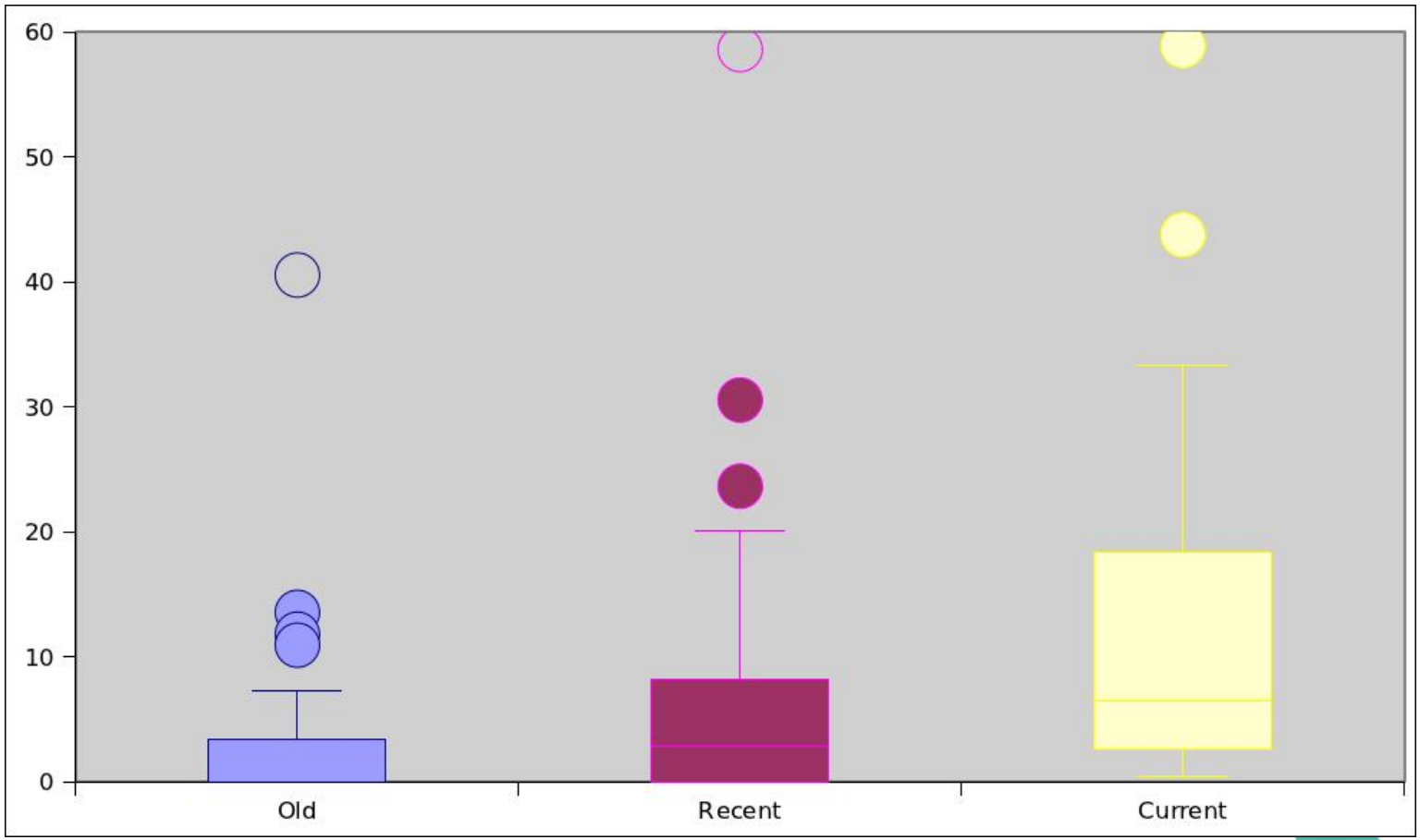
RQ2.1 – Usage of atomic data types

Per 100KLoC



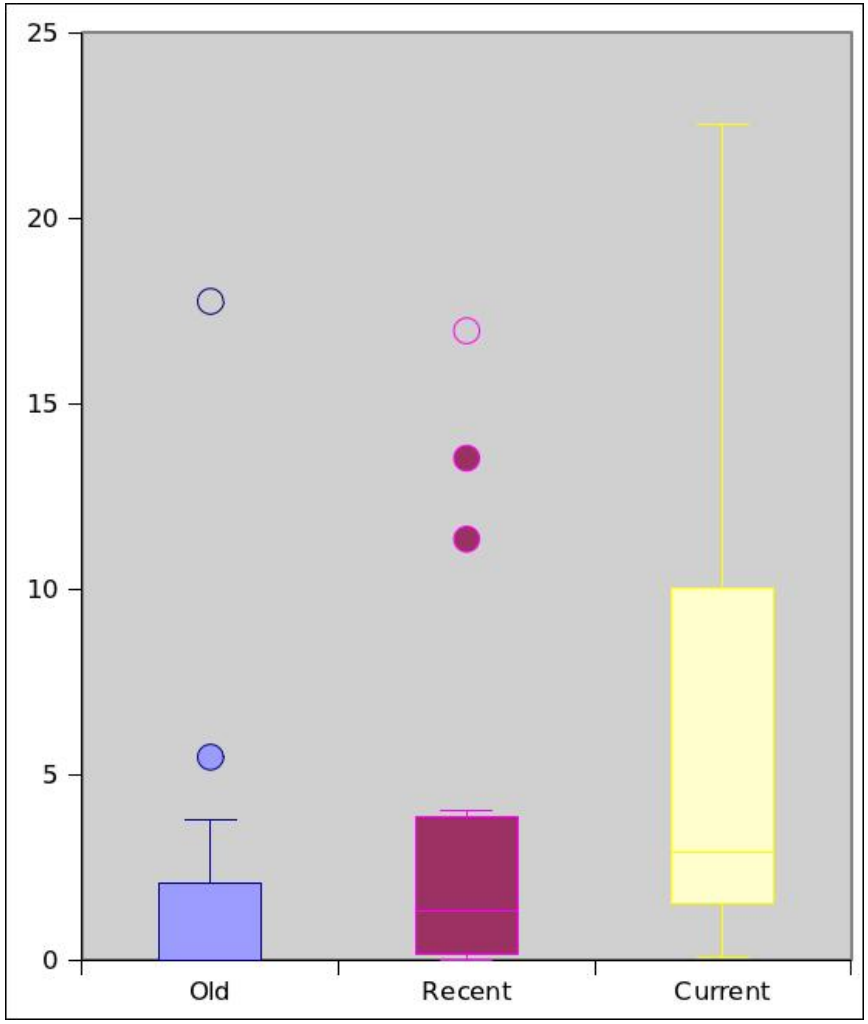
RQ2.1 – Usage of concurrent collections

Per 100KLoC – Mainly `ConcurrentHashMap`



RQ2.1 – Usage of executors

Per 100KLoC – Mainly thread pools



Are developers wasting opportunities to use j.u.c.?

Methodology

1. Randomly chosen 100 projects out of all the 1830 concurrent projects.

2. Randomly collected 1–3 examples of the use of the `synchronized` keyword in these projects.

3. Analyzed 276 examples of `synchronized` usage.
   - Some systems had fewer than 3 occurrences of `synchronized`.

Are developers wasting opportunities to use j.u.c.?

Results

1. We found 28 cases where the use of `synchronized` could be avoided in 25 projects.

2. 40% of these projects already use j.u.c. somehow.

3. In most cases, the `synchronized` keyword cannot be removed because of the complexity of the operations.

RQ2.3: Concurrency vs. Parallelism

- Still much work to do.
- Existing systems seem to be getting more concurrent
  - But not much...
- Some (e.g., Lucene) are also getting more parallel

RQ2.3: Concurrency vs. Parallelism

- Still much work to do.
- Existing systems seem to be getting more concurrent
  - But not much...
- Some (e.g., Lucene) are also getting more parallel
- How can we know for sure?

RQ2.3: Concurrency vs. Parallelism

- Still much work to do.
- Existing systems seem to be getting more concurrent
  - But not much...
- Some (e.g., Lucene) are also getting more parallel
- How can we know for sure?
- Suggestions?

Future Work

- **Threads for Concurrency** vs. **Threads for Parallelism**

Future Work

- **Threads for Concurrency** vs. **Threads for Parallelism**
- Atomic Data Types for actual variables instead of specific pieces of code.

Future Work

- **Threads for Concurrency** vs. **Threads for Parallelism**
- Atomic Data Types for actual variables instead of specific pieces of code.
- To investigate the organization of concurrency code in the analyzed projects.

Thank you

Why the Web and not a version control system?

- SourceForge's SVN repositories do not have a fixed structure
- It is difficult to know whether a given version is a release or a development version
  - Difficult to know **what** is an actual version
- SourceForge projects employ more than one kind of repository
- On the other hand, SourceForge's Web site organizes things somewhat

Metric Category	Element
Thread methods	Calls to interrupt, join, run, setDaemon, sleep, yield, and getContextLoader
Object methods	Calls to wait, notify, notifyAll
Concurrent collections	Instantiations of BlockingQueue, ArrayBlockingQueue, ConcurrentMap, LinkedBlockingDeque, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, SynchronousQueue, ConcurrentHashMap, DelayQueue, ConcurrentSkipListMap
Synchronized keyword	Occurrences of synchronized methods and blocks.
Executors	Uses of ExecutorService, ForkJoinPool, Executor, Executors, ScheduledExecutorService, AbstractExecutorService, ThreadPoolExecutor, ScheduledThreadPoolExecutor

Table: List of metrics

Metric Category	Element
Atomic data types	Uses of AtomicInteger, AtomicLong, and AtomicBoolean
Barriers	Uses of CyclicBarrier and CountDownLatch
Futures	Uses of Future, Response, RunnableFuture RunnableScheduledFuture, ScheduledFuture, FutureTask, ForkJoinTask, RecursiveAction, RecursiveTask, SwingWorker
Locks	Instantiations of LockSupport, ReentrantLock, ReentrantReadWriteLock, ReentrantReadWriteLockReadLock, ReentrantReadWriteLockWriteLock
Others	extends Thread/Runnable, implements Runnable, volatile modifier, import java.util.concurrent (and its subpackages)
Size	L of Code

Table: List of metrics

RQ1 - How often are the Java concurrency constructs employed in real applications? ALL PROJECTS

metrics	Median	Mean	Std. Dev.	#Projects
# 1	108.77	223.81	362.21	1634
# 2	12.23	25.63	39.08	924
# 3	2.97	8.83	12.41	40
# 4	14.26	32.52	52.42	1031
# 5	31.17	91.85	195.61	470

Table: Metricts per 100KLOC

where:

- #1: `synchronized` keywords
- #2: extends thread
- #3: extends runnable
- #4: implements runnable
- #5: use of j.u.c.