

Utilizing Performance Unit Tests To Increase Performance Awareness

Vojtěch Horký, Peter Libič, Lukáš Marek,
Antonín Steinhauser and Petr Tůma

February 4, 2015

Charles University in Prague



Motivation: Choosing a Plotting Library

We need to plot graphs for our web application in Java.

Our choice is driven by various requirements.

- Available features
- Price
- Sane and documented API
- Performance
- ...

Motivation: Choosing a Plotting Library

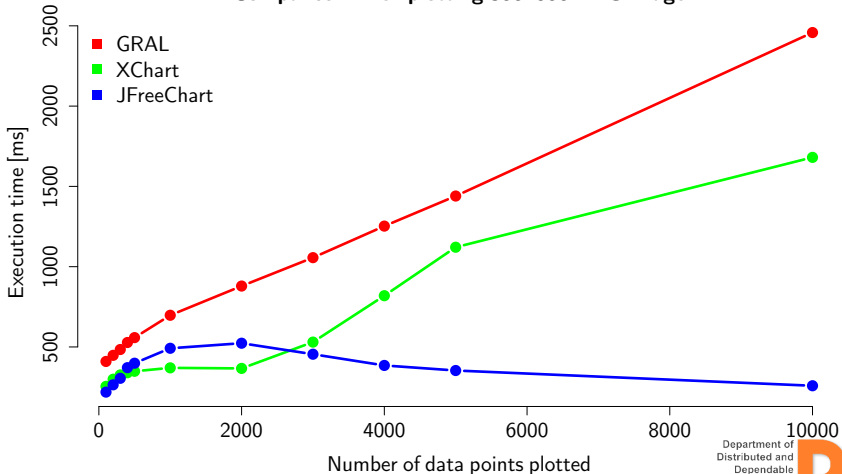
We need to plot graphs for our web application in Java.

Our choice is driven by various requirements.

- Available features
- Price
- Sane and documented API
- **Performance**
- ...

Motivation: Choosing a Plotting Library Based on its Performance

Comparison when plotting 800x600 PNG image



Issues With Low-Impact Performance Decisions

To decide which of the libraries is faster we had to

- design and implement a test
- and evaluate the results.

Issues With Low-Impact Performance Decisions

To decide which of the libraries is faster we had to

- design and implement a test
- and evaluate the results.

This takes time. Often, we

- assume the performance differences are negligible
- or fallback to previous experience with similar task.

Our Goal

Help the developer with decisions that have low performance impact.

- Without extra effort from the developer.
- Give the answers as fast as possible.

Make developers aware of the actual performance of their code.

(We do not aim to correct bad architectural & design decisions.)

The Idea: Extend API Documentation with Performance Information

Why API documentation?

- We target methods and classes.
- Available even in IDE as context help.

The Idea: Extend API Documentation with Performance Information

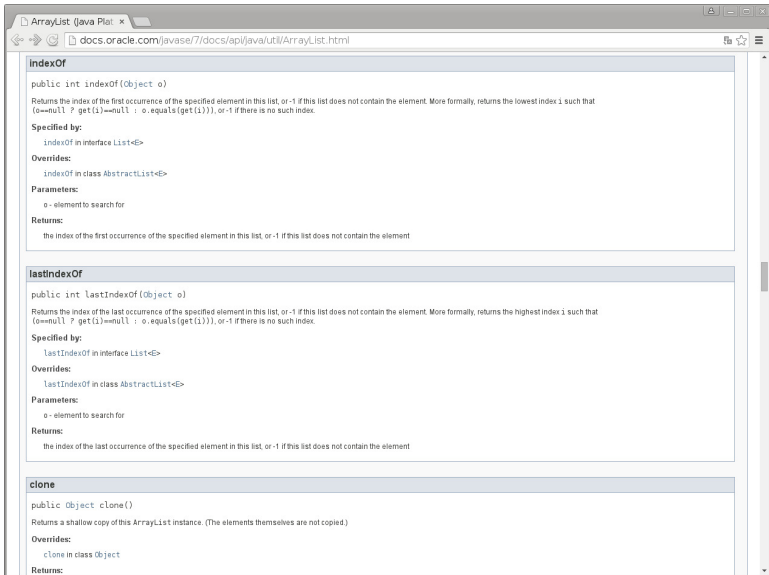
Why API documentation?

- We target methods and classes.
- Available even in IDE as context help.

How it would be used?

When coding, developers would see the performance information together with the method detail.

Tools: From JavaDoc ...



The screenshot shows a web browser window with the title "ArrayList (Java Platform 7 SE) | Oracle" and the URL "docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html". The page displays the JavaDoc for the `ArrayList` class, specifically the `indexOf`, `lastIndexOf`, and `clone` methods.

indexOf

```
public int indexOf(Object o)
```

Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. More formally, returns the lowest index *i* such that (`o==null ? get(i)==null : o.equals(get(i))`), or -1 if there is no such index.

Specified by:

- `indexOf` in interface `List<E>`

Overrides:

- `indexOf` in class `AbstractList<E>`

Parameters:

- `o` - element to search for

Returns:

- the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element

lastIndexOf

```
public int lastIndexOf(Object o)
```

Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. More formally, returns the highest index *i* such that (`o==null ? get(i)==null : o.equals(get(i))`), or -1 if there is no such index.

Specified by:

- `lastIndexOf` in interface `List<E>`

Overrides:

- `lastIndexOf` in class `AbstractList<E>`

Parameters:

- `o` - element to search for

Returns:

- the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element

clone

```
public Object clone()
```

Returns a shallow copy of this `ArrayList` instance. (The elements themselves are not copied.)

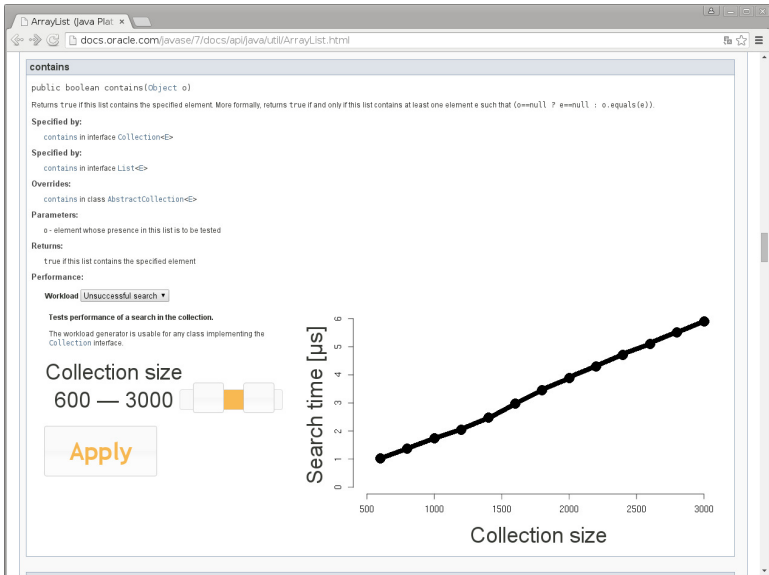
Overrides:

- `clone` in class `Object`

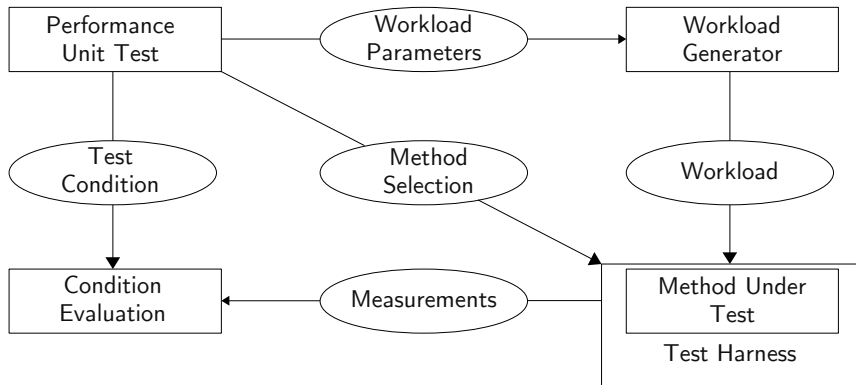
Returns:

- a shallow copy of this `ArrayList` instance.

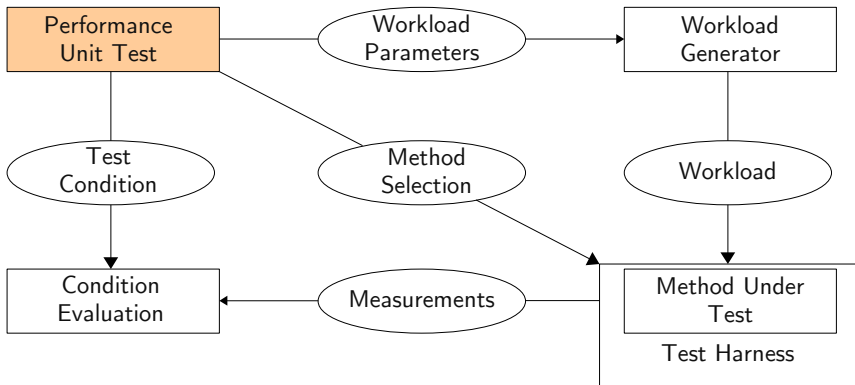
Tools: ... to PerfJavaDoc



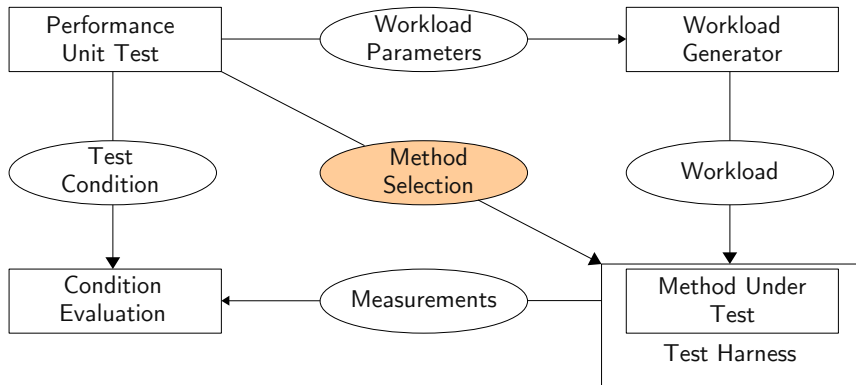
Getting Workloads from Performance Unit Tests



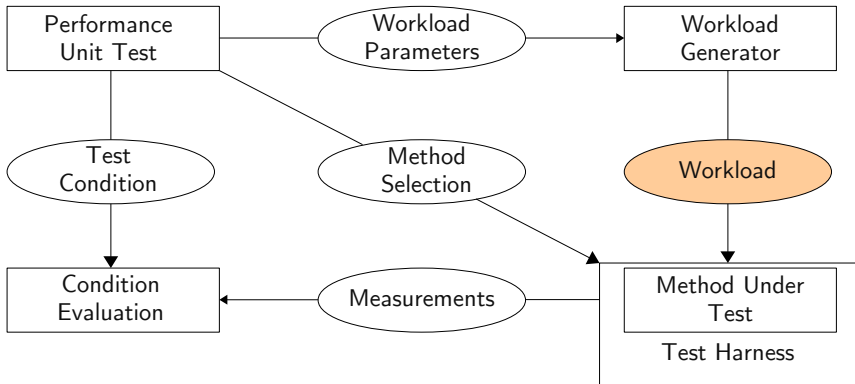
Getting Workloads from Performance Unit Tests



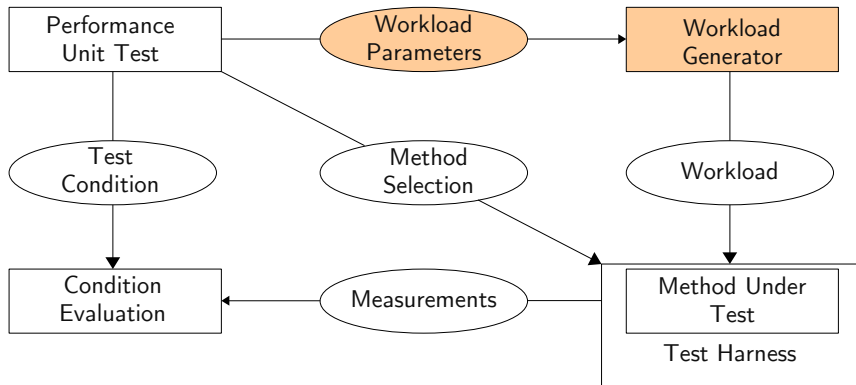
Getting Workloads from Performance Unit Tests



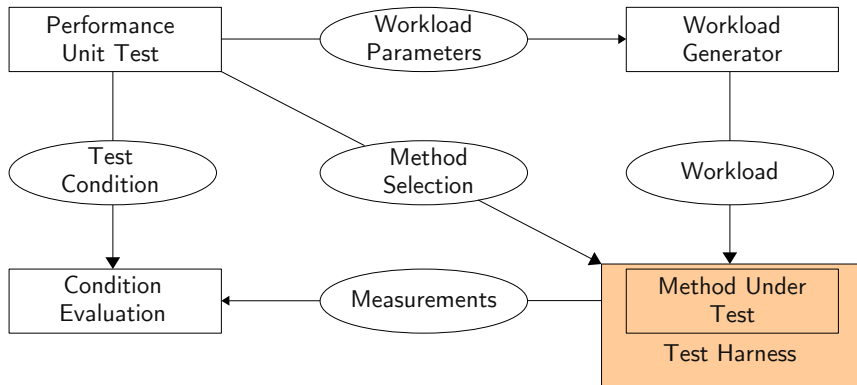
Getting Workloads from Performance Unit Tests



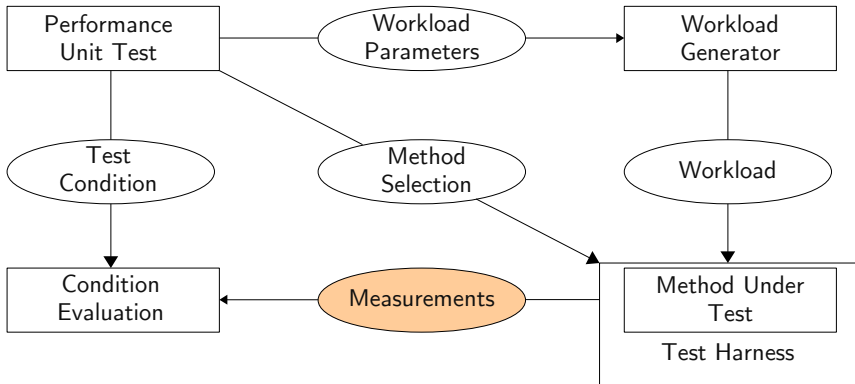
Getting Workloads from Performance Unit Tests



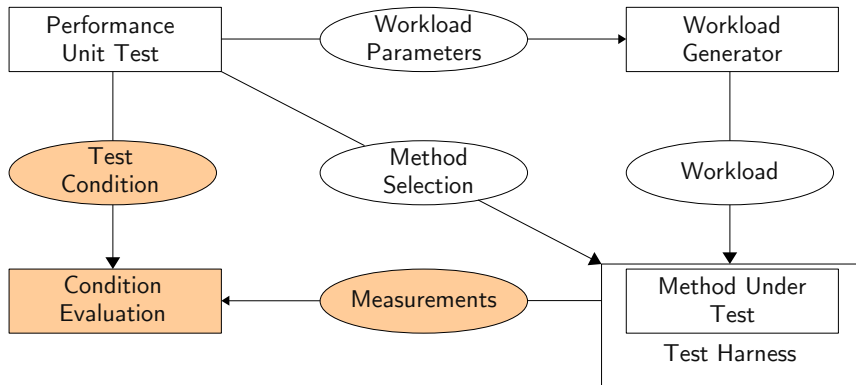
Getting Workloads from Performance Unit Tests



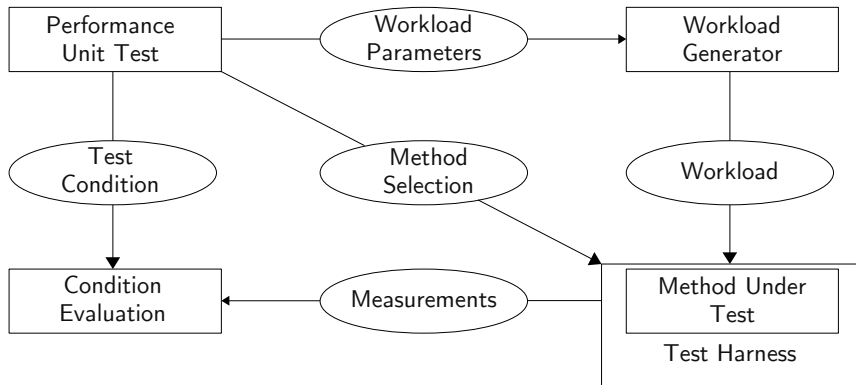
Getting Workloads from Performance Unit Tests



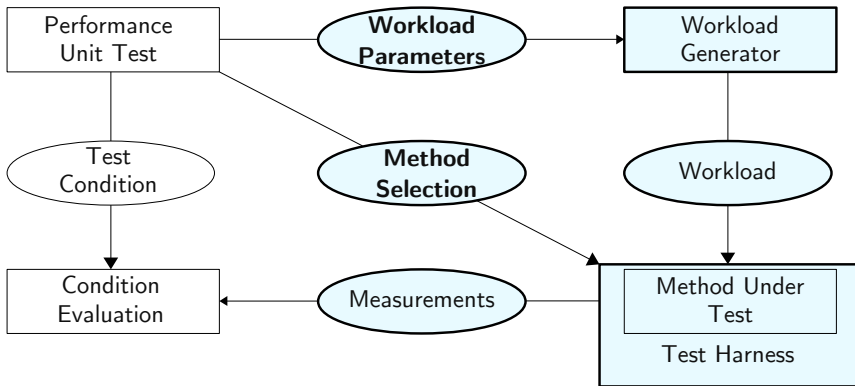
Getting Workloads from Performance Unit Tests



Getting Workloads from Performance Unit Tests



Getting Workloads from Performance Unit Tests



This provides data for the performance documentation.

From Performance Unit Tests to Documentation

Connecting the workload to the method

```
void sort(long[] data) {  
    ...  
}
```

From Performance Unit Tests to Documentation

Connecting the workload to the method

```
@Workload("pkg.Workload.longArray")  
void sort(long[] data) {  
    ...  
}
```

From Performance Unit Tests to Documentation

Connecting the workload to the method

```
@Workload("pkg.Workload.longArray")  
void sort(long[] data) {  
    ...  
}
```

```
long[] longArray(                                int size)  
    ...  
}
```


From Performance Unit Tests to Documentation

Connecting the workload to the method

```
@Workload("pkg.Workload.longArray")  
void sort(long[] data) {  
    ...  
}
```

Getting labels for the plots

```
long[] longArray(                                int size)  
    ...  
}
```

From Performance Unit Tests to Documentation

Connecting the workload to the method

```
@Workload("pkg.Workload.longArray")
void sort(long[] data) {
    ...
}
```

Getting labels for the plots

```
@Descr("Generate array filled with random longs")
long[] longArray(                int size)
    ...
}
```

From Performance Unit Tests to Documentation

Connecting the workload to the method

```
@Workload("pkg.Workload.longArray")  
void sort(long[] data) {  
    ...  
}
```

Getting labels for the plots

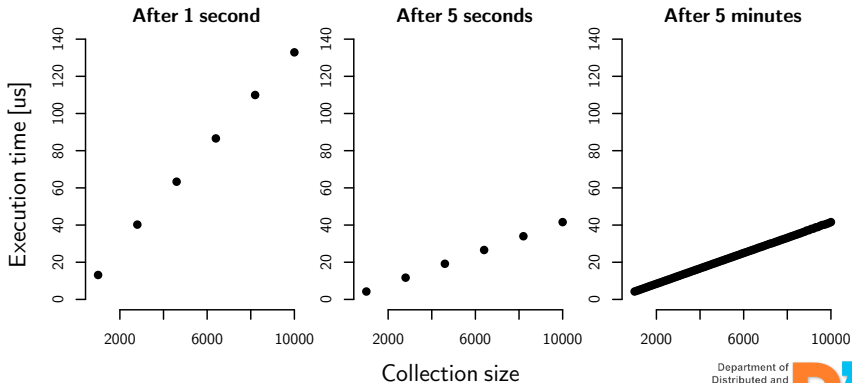
```
@Descr("Generate array filled with random longs")  
long[] longArray(@Param("Array size") int size)  
    ...  
}
```

Displaying Performance Interactively

- Measure on demand
- Cache & share the results
- Refine results continuously (on background)

Displaying Performance Interactively

- Measure on demand
- Cache & share the results
- Refine results continuously (on background)



Experiment: Faster Applications with Extended Documentation

Volunteer students (advanced course of Java).

- Groups with and without performance documentation.
- Task to solve with Java JDOM library.

Experiment: Faster Applications with Extended Documentation

Volunteer students (advanced course of Java).

- Groups with and without performance documentation.
- Task to solve with Java JDOM library.

Results

Participants have problems judging performance of their own code: expected run-times differed in orders of magnitude.

Not feasible to decide whether one group writes faster applications (we would need thousands of students to have statistically reliable comparison of the groups).

Experiment: Improve Existing Applications

Simulate developers caring about performance.

Use similar – but faster – methods guided by results from the extended documentation.

Experiment: Improve Existing Applications

Simulate developers caring about performance.

Use similar – but faster – methods guided by results from the extended documentation.

Applications

- Buildhealth
 - Reports merged results from various test frameworks.
- METS Downloader
 - Gathers bibliographic meta-information.

(Selected because of a reasonable size and use of JDOM.)

Improving Existing Applications: Results

Application	Before	After
Buildhealth		
METS downloader		

Improving Existing Applications: Results

Application	Before	After
Buildhealth	Changed code	
METS downloader		

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
METS downloader			

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
	Whole application		
METS downloader			

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
	Whole application	2.52 s	2.40 s
METS downloader			

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
	Whole application	2.52 s	2.40 s
METS downloader	Changed code		

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
	Whole application	2.52 s	2.40 s
METS downloader	Changed code	131.5 ms	27.4 ms

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
	Whole application	2.52 s	2.40 s
METS downloader	Changed code	131.5 ms	27.4 ms
	Whole application		

Improving Existing Applications: Results

Application		Before	After
Buildhealth	Changed code	938.3 ms	908.4 ms
	Whole application	2.52 s	2.40 s
METS downloader	Changed code	131.5 ms	27.4 ms
	Whole application	120.2 s	120.4 s

Conclusion: Utilizing Performance Unit Tests To Increase Performance Awareness

Extend the API documentation and make the developers aware of performance of small parts of the code and help them write faster applications.

<http://d3s.mff.cuni.cz/software/spl>

Conclusion: Utilizing Performance Unit Tests To Increase Performance Awareness

Extend the API documentation and make the developers aware of performance of small parts of the code and help them write faster applications.

<http://d3s.mff.cuni.cz/software/spl>

Thank You!