- Motivation
- Overspecification and plasticity
- Initial insights
- Framework

Relieving over-specification through plastic zones

Marcelino Rodriguez-Cancio,
Director: B. Baudry
Advisor: B. Combemale

Motivation

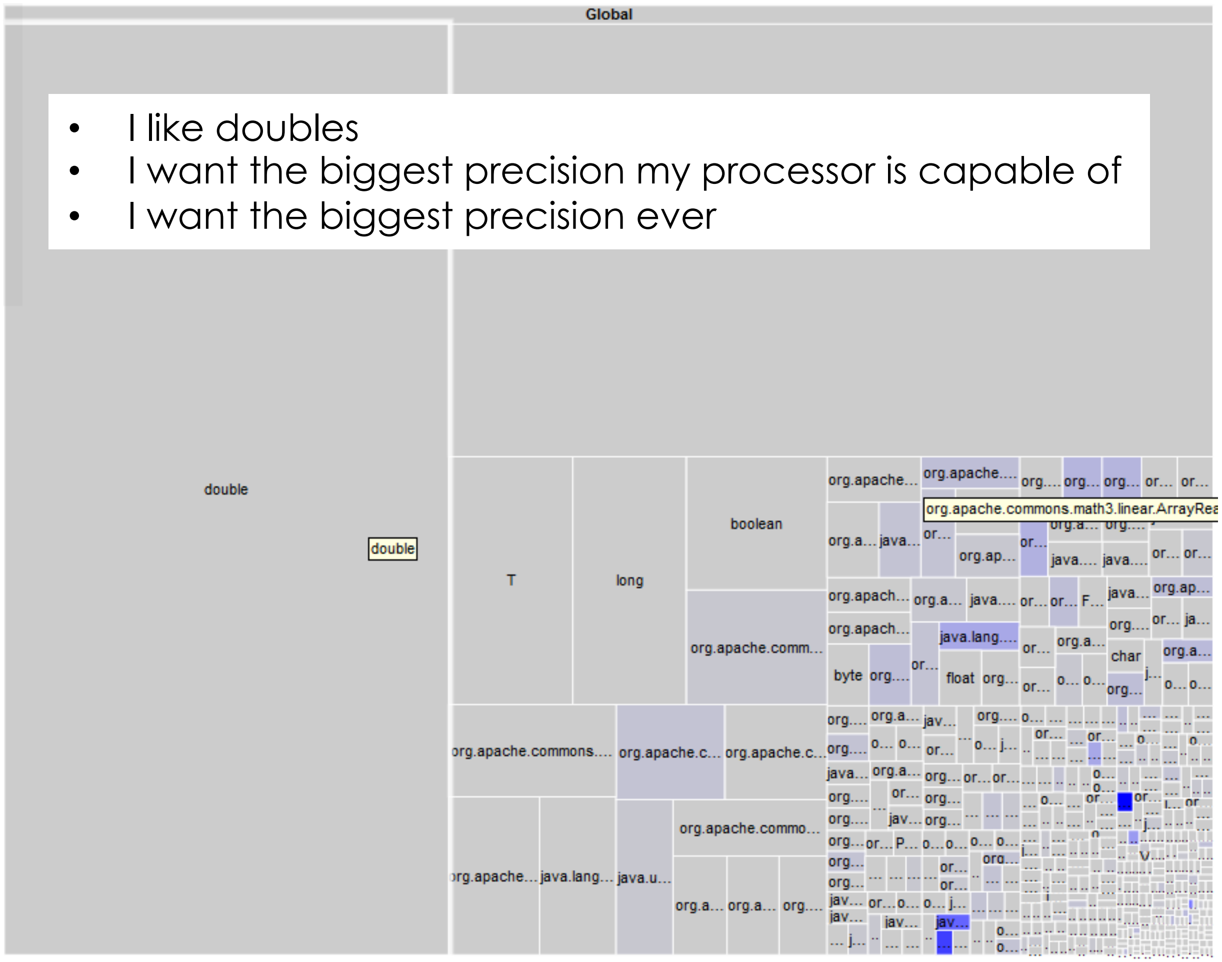
- Old vision (rigid, fragile) vs new vision (plastic, maleable)

This is what we do at

DiverSE

Diversity-Centric Software Engineering

Variable Space Math 3.2

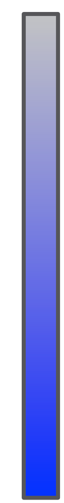


- I like doubles
- I want the biggest precision my processor is capable of
- I want the biggest precision ever

We counted all variables in the project.

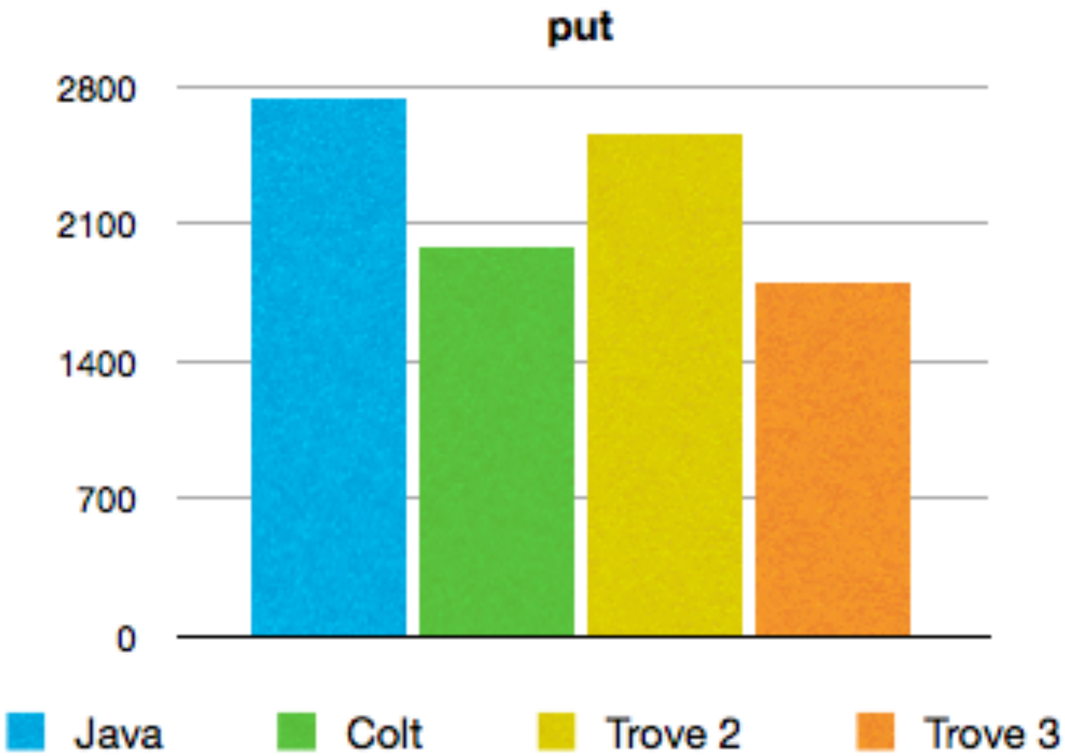
The bigger the box, the more variables of the type

Less inheritors



More inheritors

Use case, speed evolution of Trove



Your design decision is currently right.

Tomorrow it may be wrong.

You need to evolve.

The problem

- Developers don't want to overspecify
- Languages forces to
- You can always create new languages
- But eventually you will over-specify with them because of the capabilities given
- The language updates
- Then you must update your laaaaaaarge codebase.
- Or you can embed the capability to adapt at runtime according **to your new needs**

Q: Where in the code we find plasticity?
A: In over-specified zones

Current programs are not an specification of the solution:

They are an ***over-specification*** of the solution.

- An over-specified zone can be changed
- If we know ***how***, then is a plastic zone

Novelty: Realizing that over-specification ~ plasticity.

Our ambition…

To find new ways of ***executing*** code that constantly and ***dynamically*** adapting existing code bases without changing its functional properties

- Plastic zones exist in already made code bases
- Since we control the MOC to be variability aware, we know when and what to change. This improvements aren't *static* they are *dynamic*

Differences with refactoring the code base

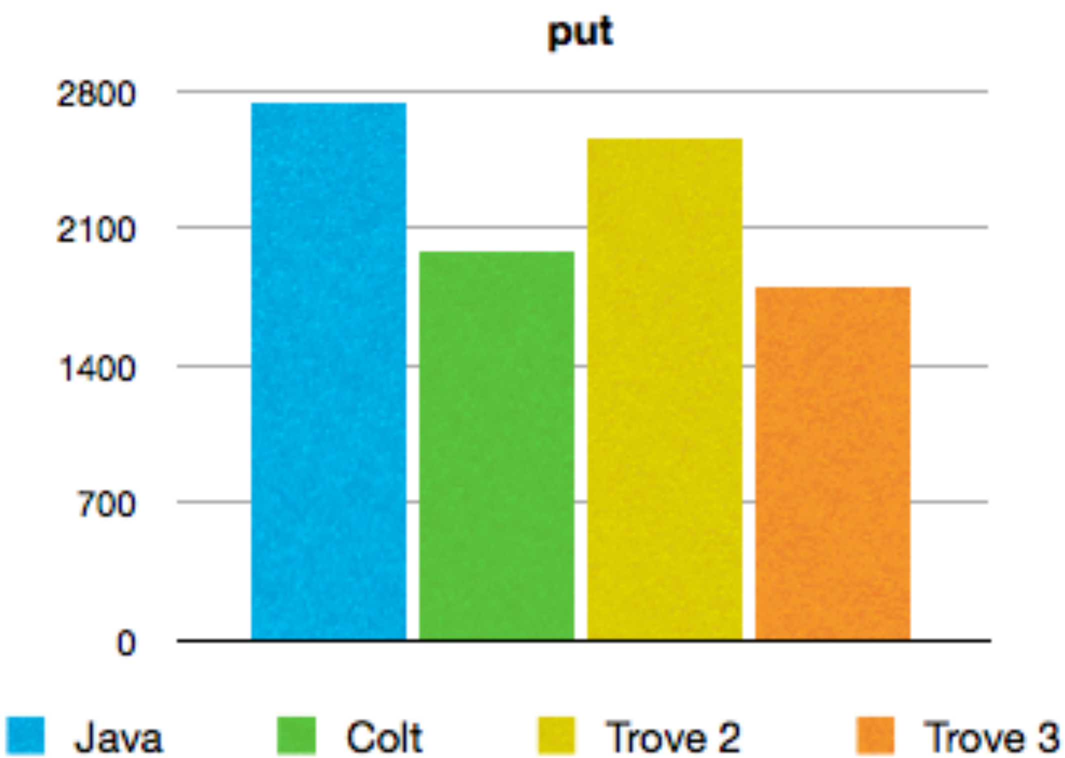
- Transformations can be only done in compilation (build time)
- Can alter the behavior of programs not designed for an specific purpose in runtime.

- You don't have to change the code if you can adapt the code execution

Use cases

Motivation

Use case, speed evolution of Trove



Your design decision is currently right.

Tomorrow it may be wrong.

You need to evolve.

Parallelization

- Separate the definitions the most possible of the uses

```
List<Data> datas = new List<>()
for ( Equis x : equis ){
   Data d = lengthyCalculation(x);
   d *= 5;
   datas.add (d);
}
```

```
List<Future<Data>> datas = new List<>()
for ( Equis x : equis ){
   Future<Data> d = lengthyCalculation(x);
   datas.add(d);
   Future<data> d1 = data * Future<5>;
}
for (Future<Data> d : datas) d.resolve();
```

Use case, obfuscation

1. Transform the code
2. Execute the code using a different MOC

$$f_n : \{ I_n \}$$

```java
// set up parameters
final DerivativeStructure[] dsX = new DerivativeStructure[point.length];
for (int i = 0; i < point.length; ++i) {
    dsX[i] = new DerivativeStructure(point.length, 1, i, point[i]);
}

// compute the derivatives
final DerivativeStructure[] dsY = f.value(dsX);
```

$$f_n : \{ \; I_n \{ \blacksquare n \text{ if } n=0, 1, Mn+1 \text{ if } n \text{ even} n-1 \text{ if } n \text{ odd} \; \}$$

M: Number of statements

```java
// set up parameters
final DerivativeStructure[] dsX = new DerivativeStructure[point.length];
// compute the derivatives
final DerivativeStructure[] dsY = f.value(dsX);

for (int i = 0; i < point.length; ++i) {
    dsX[i] = new DerivativeStructure(point.length, 1, i, point[i]);
}
```

Use case, obfuscation

```
L1
 LINENUMBER 48 L1
 ICONST_0
 ISTORE 3
L2
 ILOAD 3
 ALOAD 1
 ARRAYLENGTH
 IF_ICMPGE L3
L4
 LINENUMBER 49 L4
 ALOAD 2
 ILOAD 3
 NEW org/apache/commons/math3/analysis/differentiation/DerivativeStructure
 DUP
 ALOAD 1
 ARRAYLENGTH
 ICONST_1
 ILOAD 3
 ALOAD 1
 ILOAD 3
 DALOAD
 INVOKESPECIAL org/apache/commons/math3/analysis/differentiation/DerivativeStructure.<init> (IIID)V
 AASTORE
L5
 LINENUMBER 48 L5
 IINC 3 1
 GOTO L2
L3
 LINENUMBER 53 L3
 ALOAD 0
 GETFIELD org/apache/commons/math3/analysis/differentiation/JacobianFunction.f : Lorg/apache/commons/math3/analysis/dif
 ALOAD 2
 INVOKEINTERFACE org/apache/commons/math3/analysis/differentiation/MultivariateDifferentiableVectorFunction.value ([Lor
 ASTORE 3
```

SWITCH

SWITCH

Software Synthesizers

- Math function with a very fancy GUI where user input parameters

$$f(notes, conf) \to \mathbb{R}\uparrow n$$

- The function constantly produces array of numbers that are sent to the sound card. The bigger the array, the better the quality

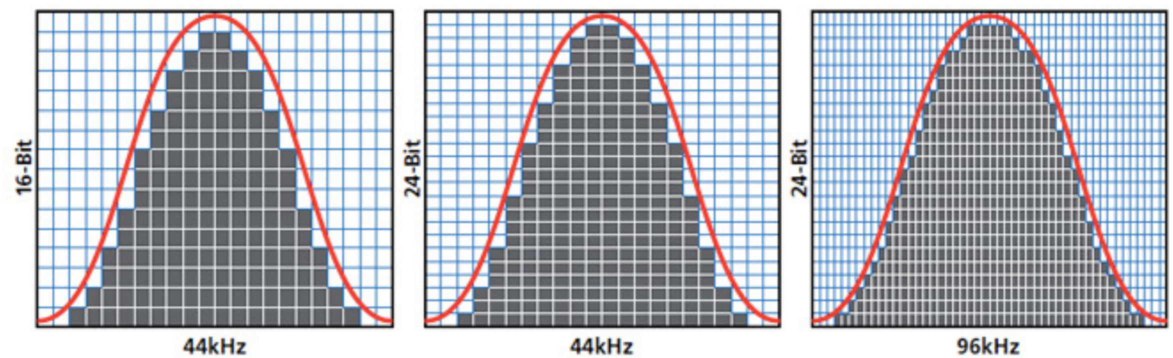
```
for (int i = 0; i < arraySize; i++) {
    bufferToTheSoundCard[i] = magicSynthetizeFunction(userParams, i)
}
magicSendToSoundCard(bufferToTheSoundCard[i])
```

Sound quality trade-off

- More elements in the array, more quality of sound
- Less elements in the array, faster processing, less consumption

Higher quality of sound!!

Bit depth = Type of the buffer array



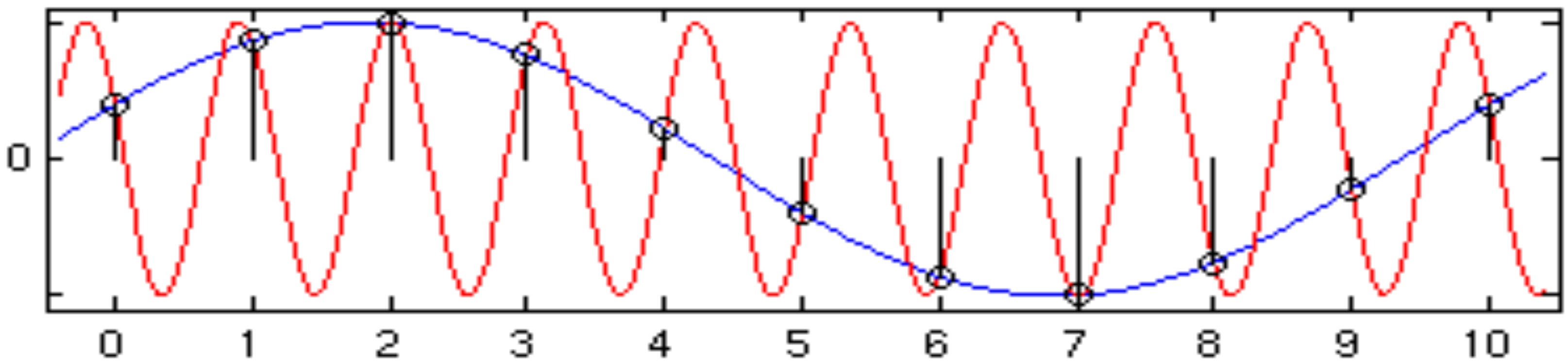
As Bit depth and sample rate increase, more information is captured, this resulting in higher quality audio.

Less memory, energy and CPU consumption

Sample rate = Size of the buffer array (number of elements)

Motivation

Never under sample! i.e. : make the array large enough to digitalize the desired sound



Under sampling can result in complete lost of sound.

Sound we got

Sound we wanted to digitalize

New SoC synths, new problem

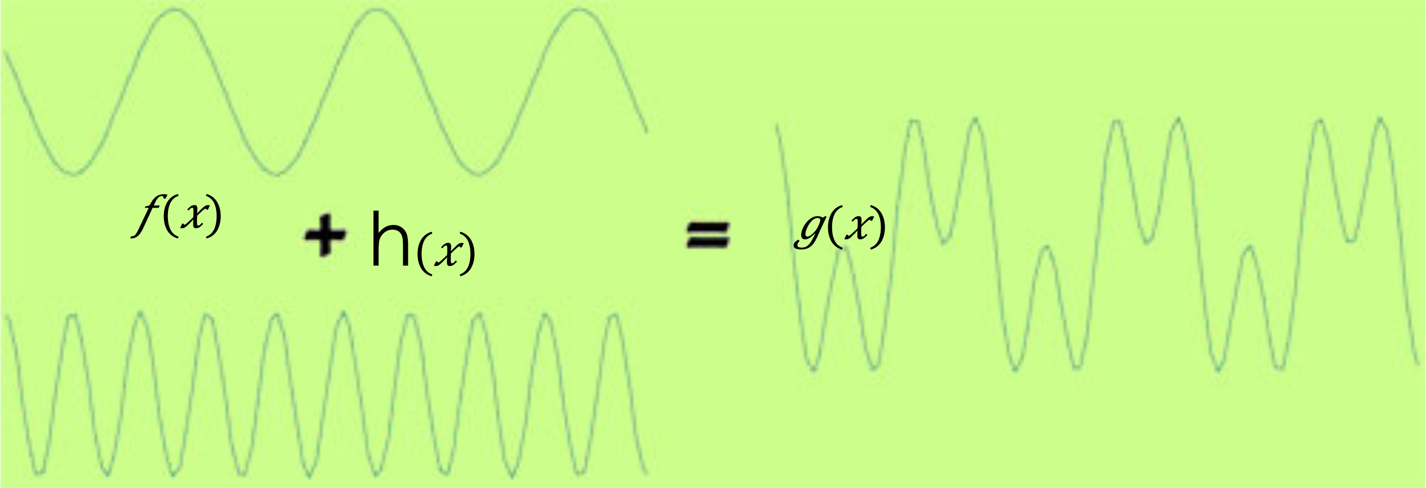
Akai Advance 25

Pioneer XDJ-RX

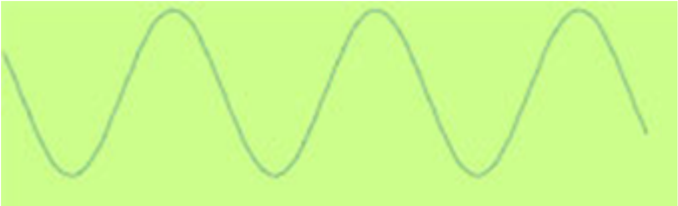
The market is moving towards inserting Systems on a Chip in the instruments. These SoCs run software synthesizers inside with a GPU

Plasticity application

- Music synthesizers can be decomposed into simpler functions that does not need such high resolution like in additive synthesis technique

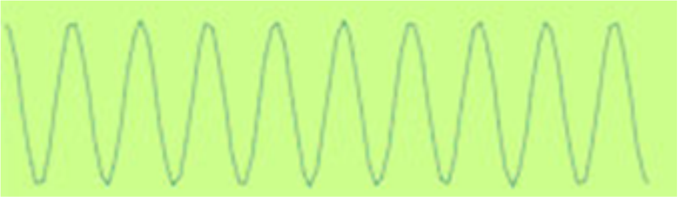


Plasticity application



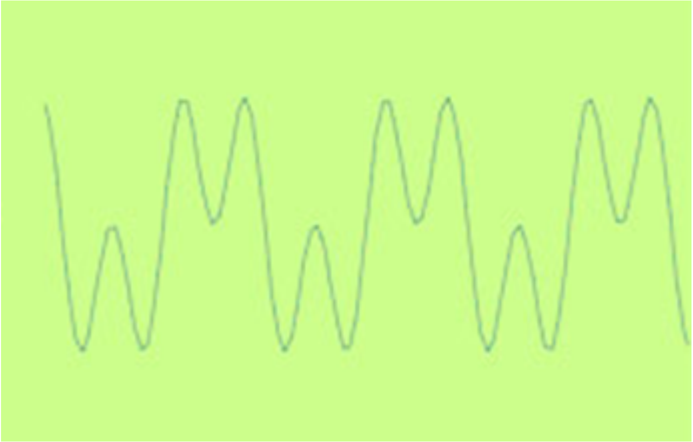
$f(x)$

Here you don't need that much sample rate. **LOOP PERFORATION**



$h(x)$

Here you don't need that much bit depth! **TYPE PLASTICITY**



$g(x)$

To produce this we need f(x) + g(x) so lets send h(x) to the GPU! **CONCURRENCY PLASTICITY**

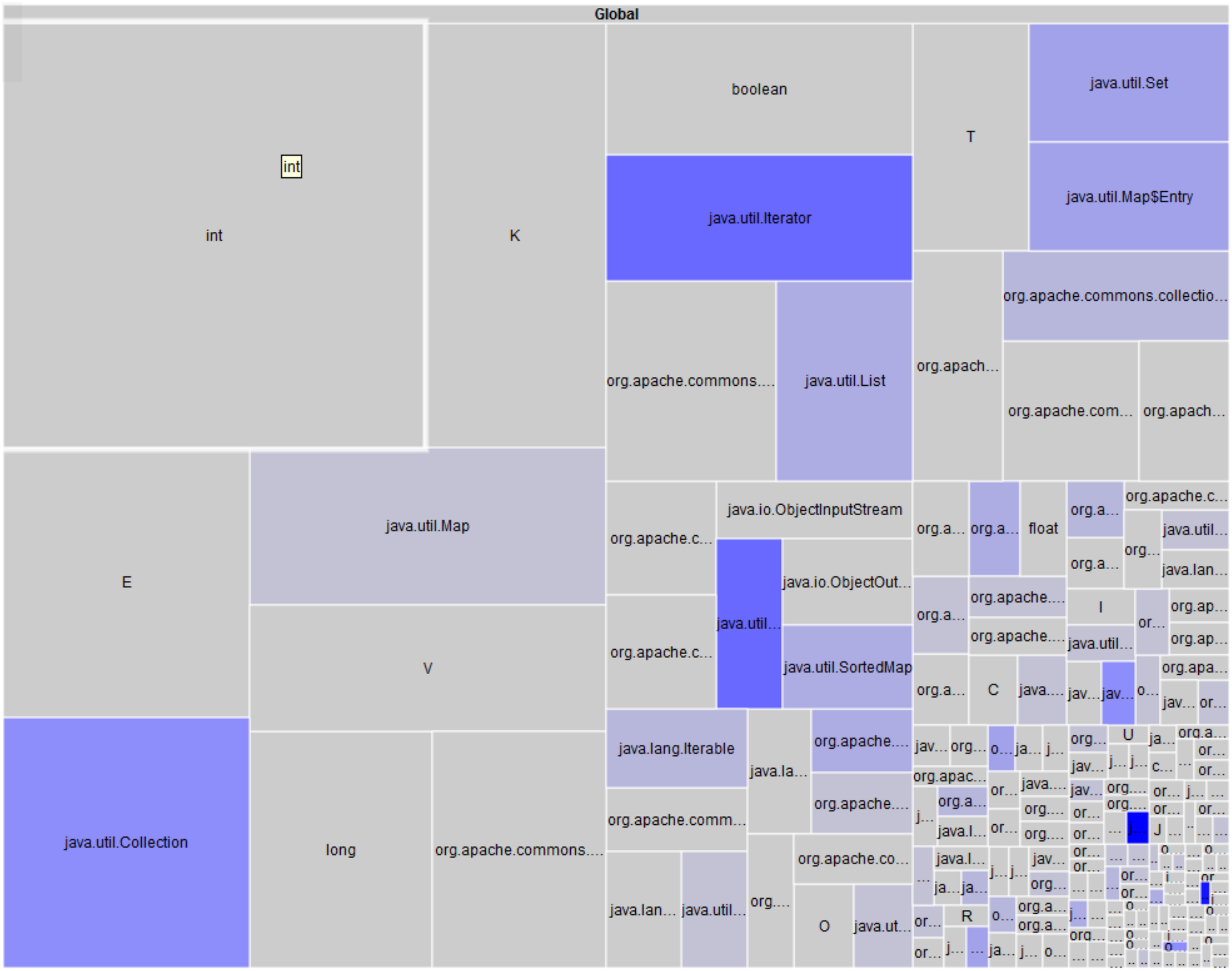
What kinds of plastic zones they are?

By no means this is a definitive list:

- Concurrency plasticity
- Type plasticity (a super case of polymorphism)
- Abstract construction (source code) plasticity

Novelty: Definition of plastic zones.

Variable Space Collections 4.0

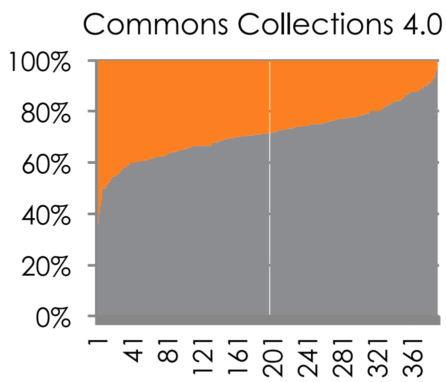


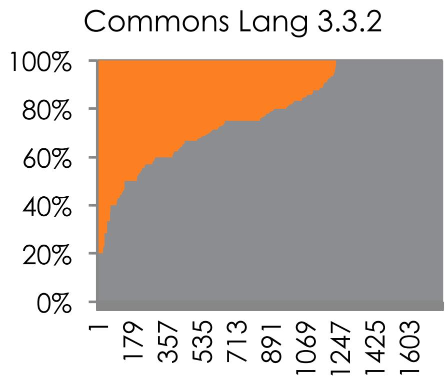
Bigger:
More variables

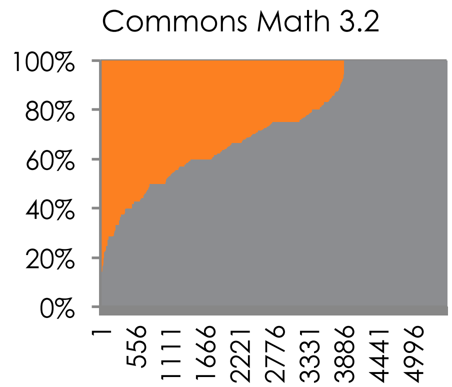
Less inheritors

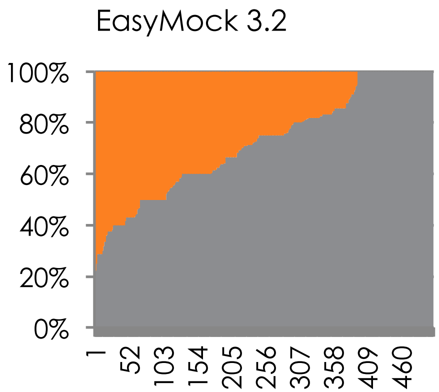
More inheritors

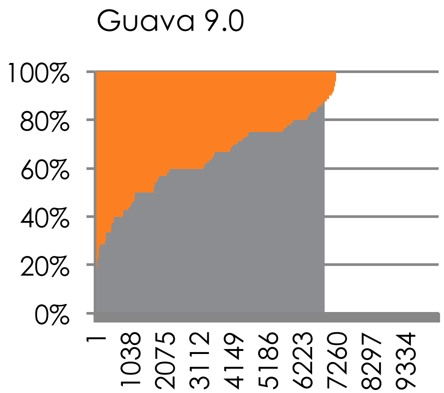
Concurrency plasticity (Over-approximated)



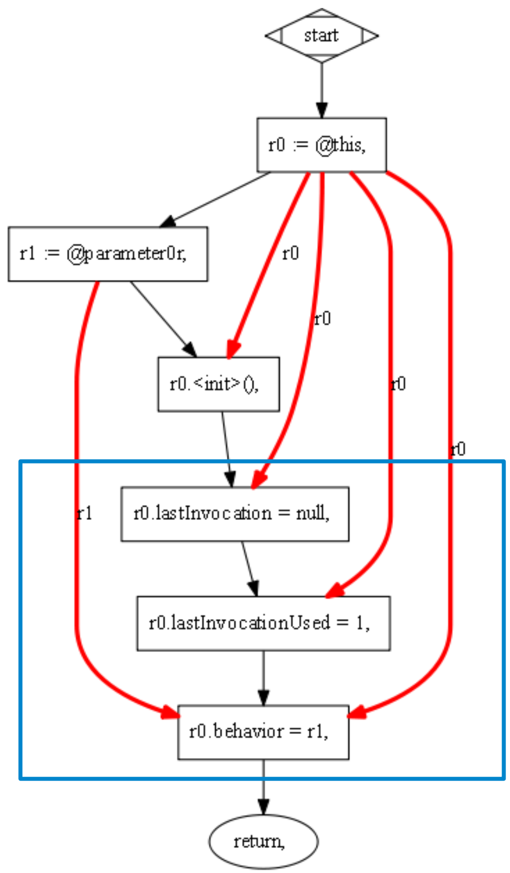








Control and data flow



Context aware, Context Free

- Obfuscator, Parallelization Context Free.
  - They only depend on the code
- Type and Synthesizers: Context aware
  - They depend on external information to the code

How to play with

Mining repositories to indentify plastic zones

Iterate through a HashMap [duplicate]

1202

483

Possible Duplicate:
How do I iterate over each Entry in a Collection Map?

What's the best way to iterate over the items in a `HashMap`?

java loops hashmap iteration

share edit flag

edited 2 days ago
Steve Chambers
4,128 • 5 • 21 • 54

asked Jun 30 '09 at 23:24
burntsugar
9,372 • 13 • 26 • 43

1266

Iterate through the `entrySet` like so:

```
public static void printMap(Map mp) {
    Iterator it = mp.entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry pair = (Map.Entry)it.next();
        System.out.println(pair.getKey() + " = " + pair.getValue());
        it.remove(); // avoids a ConcurrentModificationException
    }
}
```

Read more on `Map`

share edit flag

edited Feb 17 at 16:15
Jean-François Savard
6,819 • 1 • 5 • 28

answered Jun 30 '09 at 23:27
karim79
196k • 31 • 280 • 306

2085

If you're only interested in the keys, you can iterate through the `keySet()` of the map:

```
Map<String, Object> map = ...;

for (String key : map.keySet()) {
    // ...
}
```

If you only need the values, use `values()`:

```
for (Object value : map.values()) {
    // ...
}
```

Finally, if you want both the key and value, use `entrySet()`:

```
for (Map.Entry<String, Object> entry : map.entrySet()) {
    String key = entry.getKey();
    Object value = entry.getValue();
    // ...
}
```

Trying to use a Hashmap<Integer, Integer>

0

1

how would I increment the key [i] by 1 in this situation every time I run through this for loop with the way I currently have it set up all the elements only get mapped to 1. I am trying to find out how many times each number occurs. I have tried +1 in the empty spot after list.get(i) but again only maps each element to 1. thank you.

```
List<Integer> list = new ArrayList<Integer>();
HashMap<Integer,Integer> Mode = new HashMap<Integer, Integer>();
for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr[i].length; j++) {
        list.add(arr[i][j]);
    }
}
System.out.println(list);
int count = 1;
for(int i = 0; i < list.size(); i ++) {
    Mode.put(list.get(i), );
```

3 Answers

active oldest votes

1

If you have the option, you may find it easier to use something like Multiset from Guava.

```
Multiset<Integer> seen = HashMultiset.create();
for (int[] row : arr) {
  for (int elem : row) {
    seen.add(elem); // none of that nasty dealing with the Map
  }
}
// you can look up the count of an element with seen.count(elem)
E mostCommon = null;
int highestCount = 0;
for (Multiset.Entry<Integer> entry : seen.entrySet()) {
  if (entry.getCount() > highestCount) {
    mostCommon = entry.getElement();
    highestCount = entry.getCount();
  }
}
return mostCommon; // this is the most common element
```

Novelty: Plastification through knowledge base.

Conclusions

- Defined plastic zones
- Identified types of plastic zones
- Foreseen applications of Plastic zones
- Initial quantification
- Experimental framework

Thanks!!