# Emergent Feature Modularization

**Márcio Ribeiro**

Paulo Borba

{mmr3, phmb}@cin.ufpe.br




# Features with Preprocessors

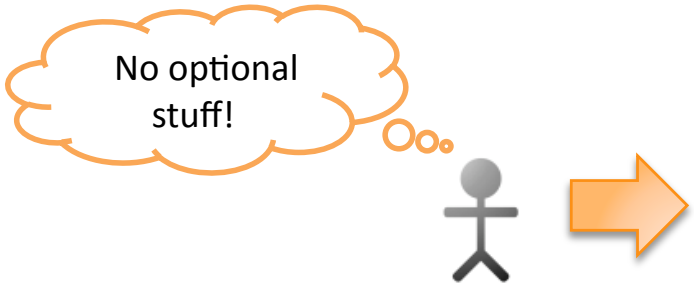
```
public void computeLevel() {
   ...
   totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
         + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
         + gc_levelManager.getCurrentCountryId();

   #ifdef ARENA
   NetworkFacade.setScore(totalScore);
   #endif
   ...
   #ifdef NOKIA
   ...
   #elif MOTOROLA
   ...
   #endif
   ...
   #ifdef ARENA
   ...
   #endif
}
```

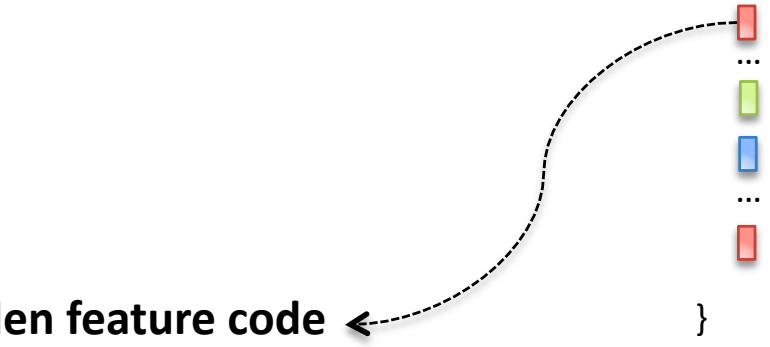
**Difficult to read!**
**Poor maintainability**
**Poor code quality**

**DEPENDENCIES!**

# Virtual Separation of Concerns (VSoC)




```
public void computeLevel() {
   ...
   totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
              + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
              + gc_levelManager.getCurrentCountryId();
```




**Hidden feature code**

Red is Arena

Green is Nokia Sounds

Blue is Motorola Sounds

**Important for comprehensibility:**
**hides code of no interest**

Centro de Informática U·F·P·E

# Virtual Separation of Concerns (VSoC)

To some extent, important for feature **comprehensibility**

What about **maintaining** features?

What about **developing** new features?

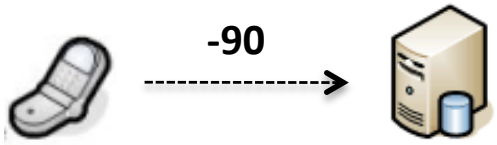
# Maintenance example:
# Let the score be negative!

```
public void computeLevel() {
   ...
   totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
              + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
              + gc_levelManager.getCurrentCountryId()
              - totalLapTime * SRC_TIME_MULTIPLIER;
   ...
   Arena feature
}
```

```
NetworkFacade.setScore(totalScore);
NetworkFacade.setLevel(this.gc_getCurrentLevel());
```


**Negative score appears correctly in the game! Let's commit!**

```
public void setScore(int s) {
   score = (s < 0) ? : 0 : s;
   }
```




**Zero instead of -90!**

# Problems

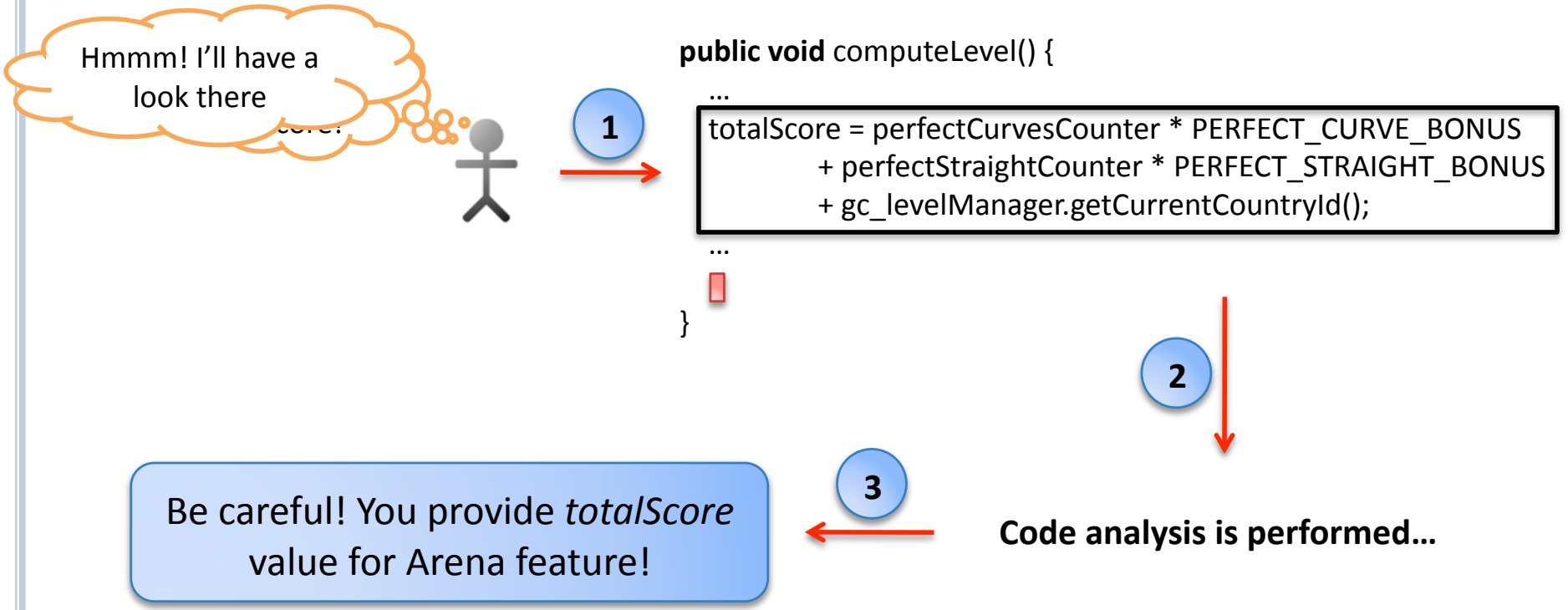
- ✓ Preprocessors, even with VSoC, do not support feature modularization
  - Maintaining one feature might require non-visible changes to other features
  - Maintaining one feature may lead to **errors** in another
- ✓ **Bugs** may be introduced…
- ✓ … leading to **lower** productivity!

```
int x = 0;
…
#ifdef FEATURE
method(x);
#endif
```

Centro de Informática U·F·P·E

# Our proposal in a nutshell

- First, the developer selects code to be maintained

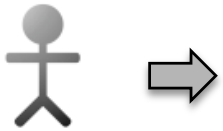


# Towards Feature Modularization

- So, we establish **contracts** between features…
- Emergent Feature Modularization
  - At first, there is **no predefined interfaces** for features…
  - Instead, interfaces **emerge on demand** during a maintenance…
  - Developers do not implement interfaces

# So, instead of hiding everything…

- … emergent interfaces capture **dependencies** between features and **abstract** their details!



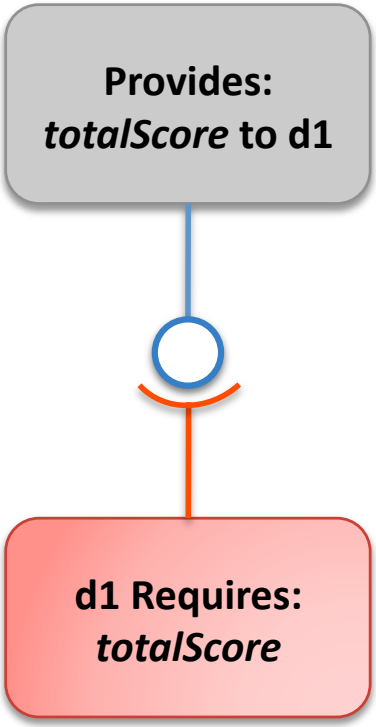
```
public void computeLevel() {
    totalScore = …
    [ ]
    [ ]
    [ ]
}
```

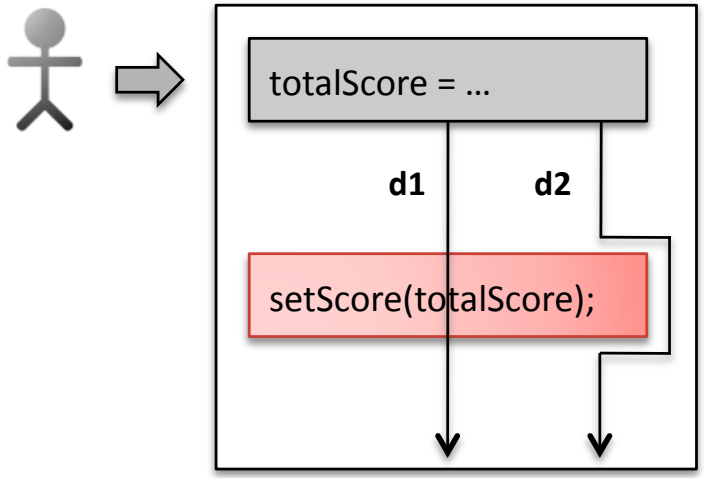
**Emergent Interface: provides *totalScore* to Arena**

- ✓ Abstract feature details
- ✓ Two features do not need to be expanded and analyzed
- ✓ Developers may know the maintenance impact (i.e., testing)

# And how we capture dependencies?

- **Feature-sensitive** dataflow analysis




**d1:** Arena

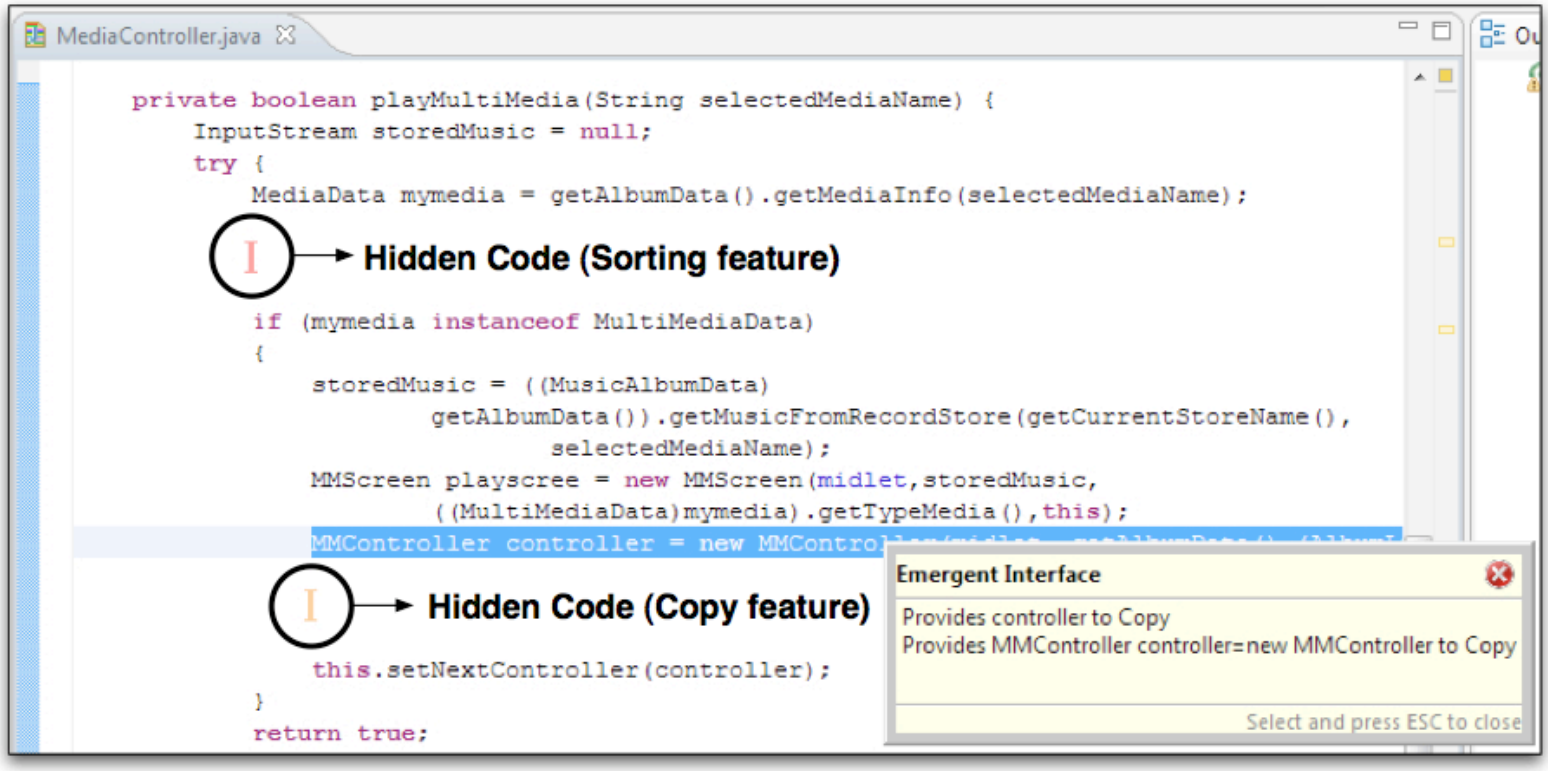
**d2:** not(Arena)

# Ongoing work

# Problem's dimension

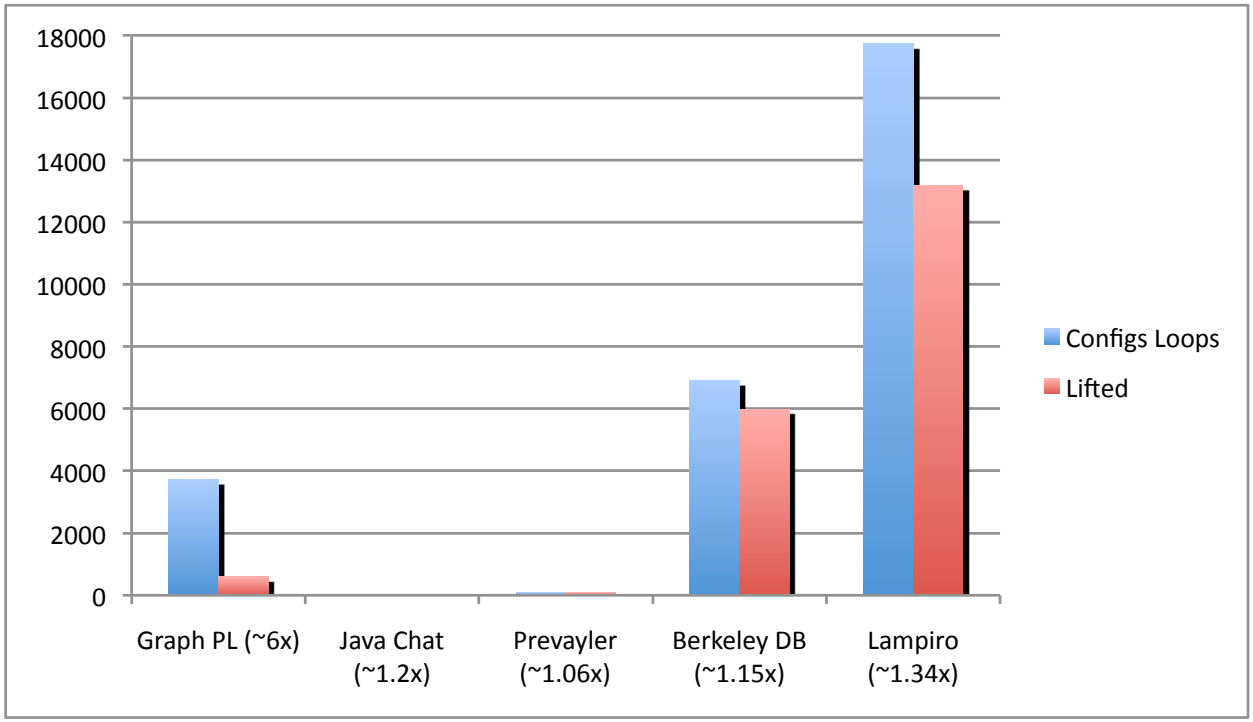
# Supporting developers

- Tool prototype



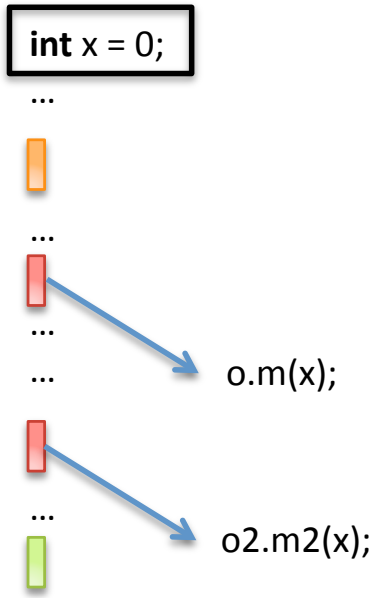
# Evaluation: feature-sensitive DFA performance

- Loops for each configuration *versus* Lifted



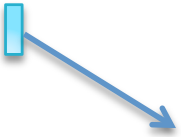
# Evaluation: effort

- Hypothesis: when using EIs, developers analyze less artifacts




```
class Other2 {
   void m2(int a) {
      ...
   }
}
```

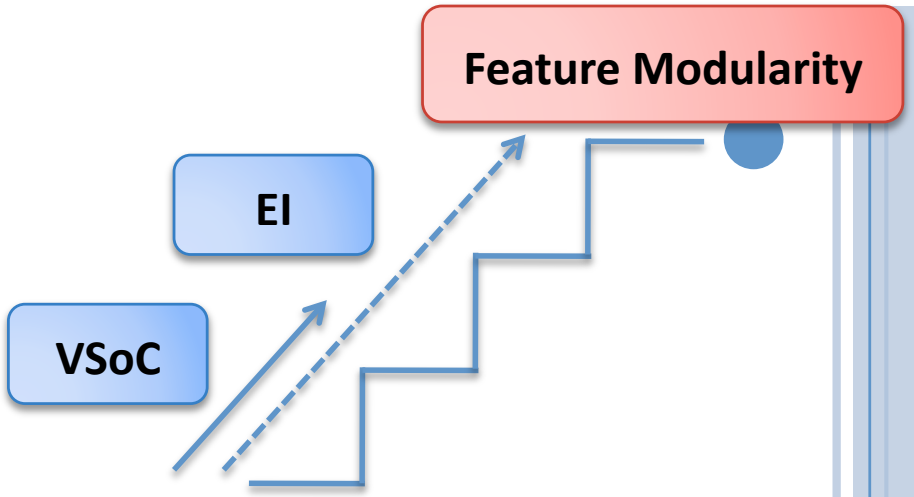
```
class Other {
   void m(int a) {
      if (a > 0) ...
   }
}
```




Fragments = 5
Classes = 2
Methods = 2
Features = 4

Fragments = 3
Classes = 2
Methods = 2
Features = 2

# Concluding Remarks




- Emergent Interfaces
  - **Abstract** hidden features **details**…
  - … and document dependencies between features
  - May prevent errors during maintenance and development…
  - … improving productivity!
  - Summarizes the impact of a maintenance
    - How many products are affected if I remove a variable?
    - How many products should I test after finishing the task?

# Emergent Feature Modularization

**Márcio Ribeiro**

Paulo Borba

{mmr3, phmb}@cin.ufpe.br