Documentation

for modern languages

Gilles Dubochet & Donna Malayeri
École Polytechnique Fédérale de Lausanne, Switzerland

Keeping pace

- Rushing ahead:
  - languages
- Trying to catch up:
  - tools
- Left behind:
  - documentation, practices

Javadoc paradigm

- Java, Objective-C, Smalltalk…
- For simple object languages:
  - class based
  - only locally defined members

In the 21st century

- Beyond object oriented (Python, Java 5…)
- Scala:
  - object and functional
  - rich type system
  - big deal in industry

✓ Mismatching code and documentation

2. Two instances of the problem (in Scala)
3. One solution (in Scaladoc)
4. Experimental assessment
5. Calling language developers to prudence

Multiple inheritance

- Compositional programming
- But “List” has 24 parent classes:
  - finding a member is tricky

Type instantiation

- Inheritance changes signature types
  ```
  class A[T] { def m(x:T): List[T] }
  class B extends A[Int]
  ```
- Brain is wrong tool for type instantiation

Code base	Inherited methods	Changed signatures
Scala std. library	69%	33%
Scala compiler	53%	22%
Google data API	70%	24%
Java utilities	37%	8%
Google collections	23%	5%

Two birds, one stone

- Inherited and local members
- Instantiated types

Demo

Study

- 60 people in 30 min. online study
- Inter-subject with independent variable:

alphabetic members instantiated types	members by parent instantiated types
✖	members by parent declaration types

- Coding and comprehension task
  - *Complete the following method:*

```
def printAllEvents(
    theFeed: CalendarEventFeed
) = ...
```

- Dependent variables:
  - response results & time
  - satisfaction

Data

- Satisfaction:
  - instantiated types increase 70% to 92%
    - member ordering has no effect (p=0.45)
- Comments confirm quantitative data
- No difference in result & time
  - except for simplest “what is type” task

Results

- For small programming tasks:
  - document with instantiated types
  - no recommendation for ordering
- Satisfaction but no impact on performance

Usability accounting

- Abstract types:
  - are widely used in real life APIs
  - are not documented well
- And implicits, annotations…
- Languages improve, not documentation

Can a good language feature become counter-productive because it hurts documentation?