Field study and classification of faults in Eiffel

Raluca Borca-Mureşan
ETHZ
June, 14, 2006

Project description

Problem:

- Testing - an important step in the software development process

Problem:

- Testing - an important step in the software development process

- BUT testing is :
  - time consuming
  - tiresome
  - boring
  - . . .

Problem:

- Testing - an important step in the software development process

- BUT testing is :
  - time consuming
  - tiresome
  - boring
  - . . .

- use automatic testing tools (AutoTest) that:
  - provide a list of test cases that generate bugs
  - NO debugging and interpretation of results

Problem:

- Testing - an important step in the software development process

- BUT testing is :
  - time consuming
  - tiresome
  - boring
  - . . .

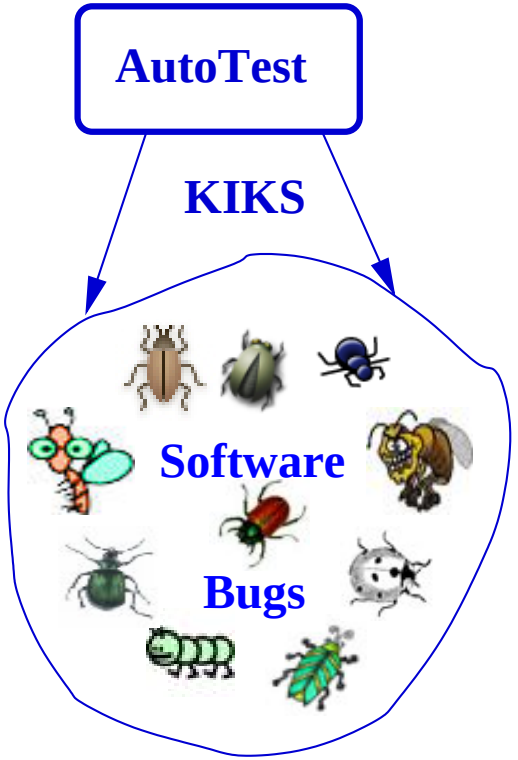
- use automatic testing tools (AutoTest) that:
  - provide a list of test cases that generate bugs
  - NO debugging and interpretation of results

- HOW TO BUILD A CLASSIFICATION SCHEME?

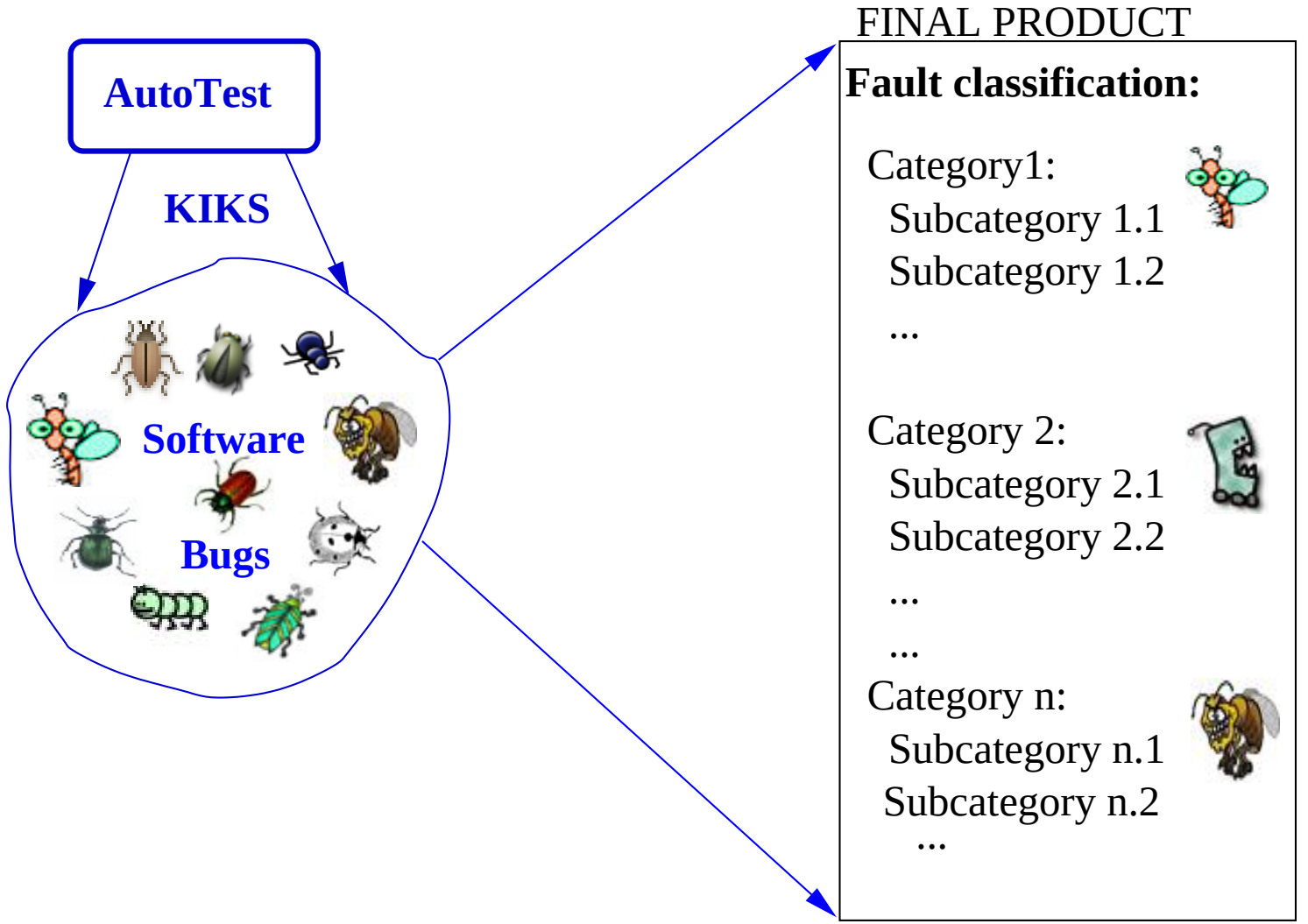
Solution

AutoTest

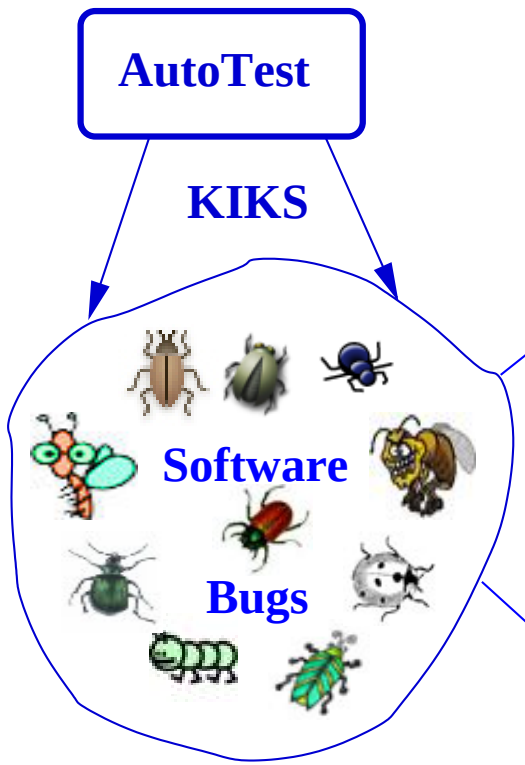
Solution



Solution



Solution



*Large field case study

*Analysis and Interpretation

*Debugging

FINAL PRODUCT

Fault classification:

Category1:
- Subcategory 1.1
- Subcategory 1.2
- ...

Category 2:
- Subcategory 2.1
- Subcategory 2.2
- ...
- ...

Category n:
- Subcategory n.1
- Subcategory n.2
- ...

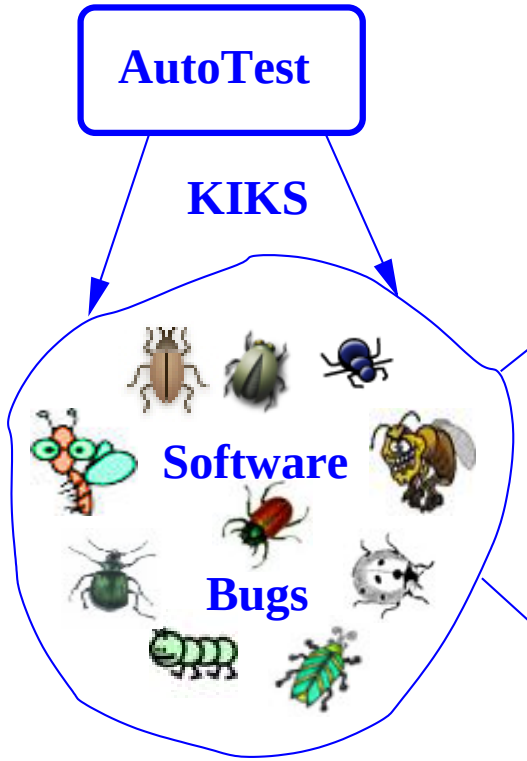
Field of study

AutoTest was run on the following libraries and applications:

- EiffelBase
- Gobo
- PerfectDeveloper - mathematical library
- DrC
- EWG

Classification scheme

Bug Types



*Large field case study

*Analysis and Interpretation

*Debugging

FINAL PRODUCT

Fault classification:

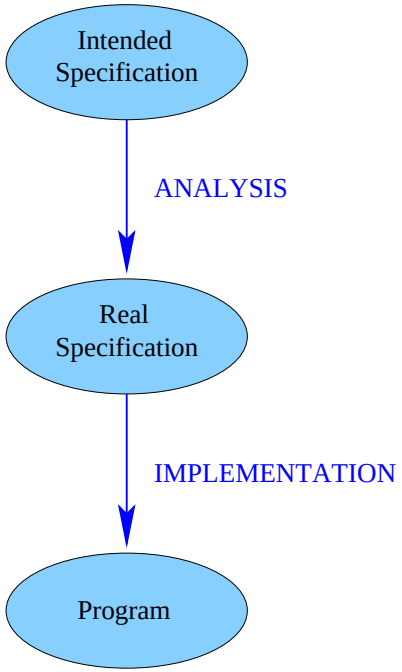
Category1:

- **Implementation**
- **Specification**
- **Dunno**

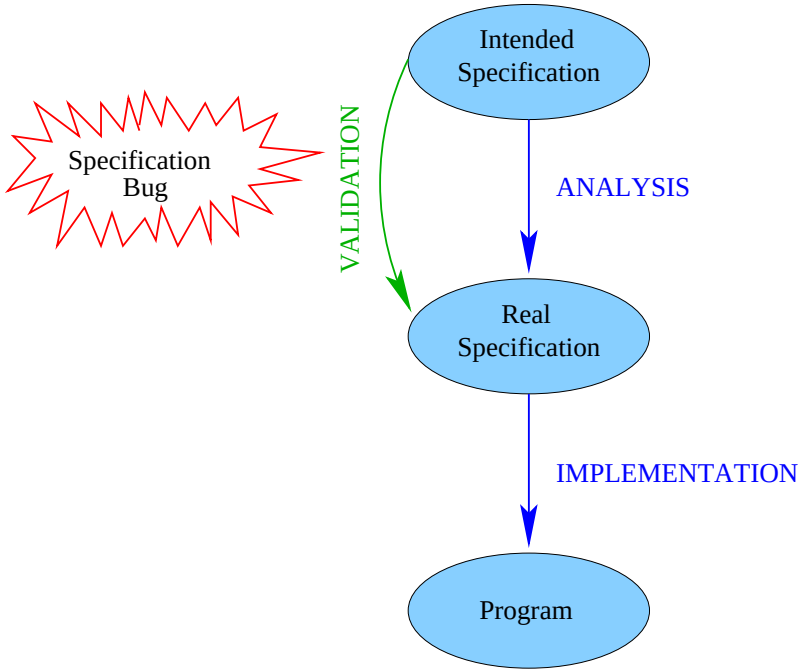
Category 2:

- **Faulty supplier**
- **Inheritance**
- **Wrong export status**
- **External fault**
- **Void target**
- **Other**

Implementation/ Specification bug



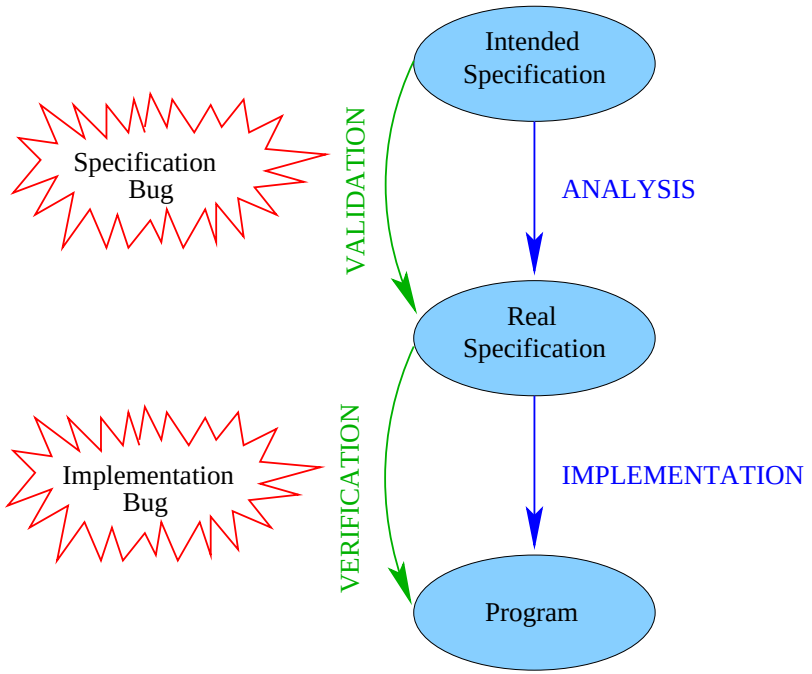
Implementation/ Specification bug



Definitions:

- *A specification bug* appears because of the discrepancy between the intended specification and the real specification.

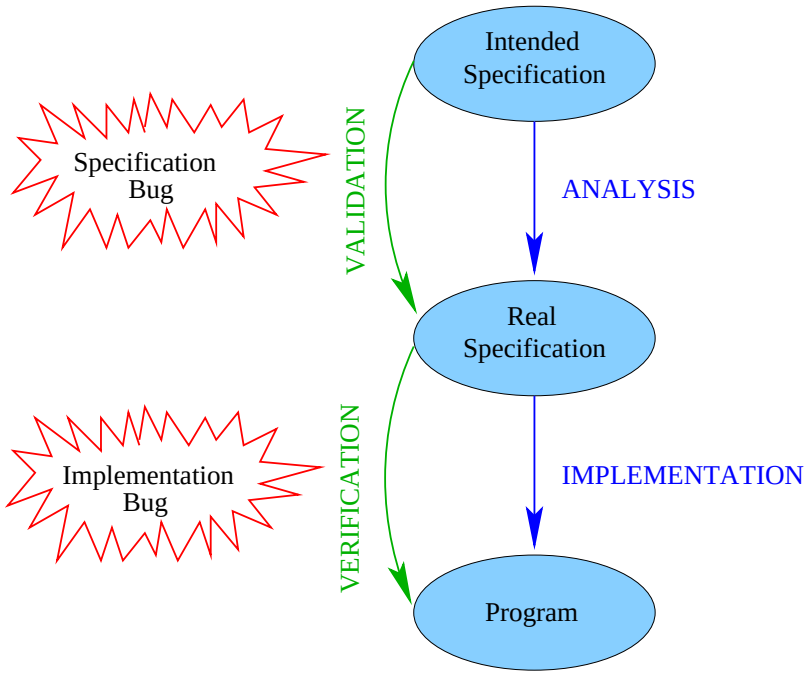
Implementation/ Specification bug



Definitions:

- *A specification bug* appears because of the discrepancy between the intended specification and the real specification.
- *An implementation bug* appears because the implementation of the routine does not fulfill the real specification of the routine.

Implementation/ Specification bug



Definitions:

- *A specification bug* appears because of the discrepancy between the intended specification and the real specification.
- *An implementation bug* appears because the implementation of the routine does not fulfill the real specification of the routine.
- *Dunno* : prove that a bug can be interpreted as being a specification and an interpretation bug.

Specification bug- example

ARRAYED_TREE

```
fill   (other: TREE [G]) is
      -- Fill with as many
      -- items of 'other'
      -- as possible. The
      -- representations
      -- of 'other' and
      -- current node
      -- need not be the
      -- same. (from TREE)
  do
     replace (other.item)
      fill_subtree   (other)
  end
```

Test case

```
create {ARRAYED_TREE[STRING] } v_1.make (0,Void )
v_1. fill   (Void)
```

In EiffelBase: 28 specification bugs.

Implementation bugs

An implementation bug is the result of an implementation of a routine not fulfilling the real specification of the routine.

In EiffelBase: 27 implementation bugs.

Implementation bug -example

BASIC_ROUTINES

```
abs (n: INTEGER): INTEGER
        is
-- Absolute value of 'n'
do
   if  n < 0 then
      Result := – n
   else
      Result := n
   end
   ensure
      non_negative_result:
               Result >= 0
end
```

Test case

```
create {BASIC_ROUTINES}v_15
v_16 := v_15.abs(–2147483648)
```

Dunno bug example

STRING

```
adapt (s: STRING): like Current is
        -- Object of a type conforming
        -- to the type of 's',
        -- initialized with
        -- attributes from 's'
    do
        Result := new_string (0)
        Result.share (s)
    ensure
        adapt_not_void:
            Result /= Void
        shared_implementation:
            Result.shared_with (s)
    end
```

Test case

```
create {STRING} v_1.make_empty
v_3 := v_1.adapt (Void)
```

In EiffelBase: 2 dunno bugs

Supplier-induced (SI) bug

A supplier-induced bug appears in the following cases: suppose routine r_1 calls routine r_2 (which is faulty). One can have:

Specification SI bug

```
r1 is
  require
    r2 -- !!! buggy
    do
     ...
  end
```

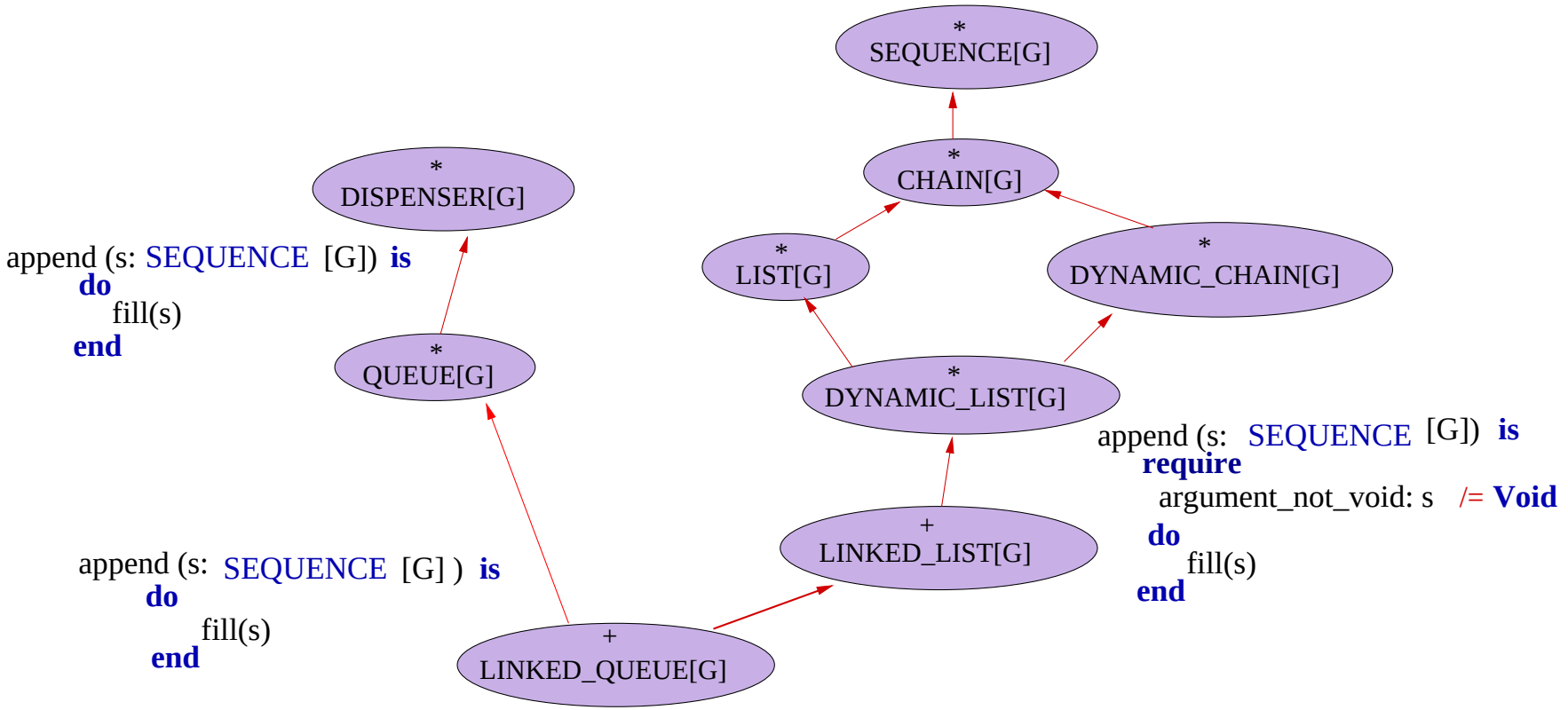
Implementation SI bug

```
r1 is
    do
    r2 -- !!! buggy
  end
```

In EiffelBase: 9 specification SI bugs

In EiffelBase: 30 implementation SI bugs

Inheritance contract bug



Test case

```
create {LINKED_QUEUE[ANY] v_1.make}
v_1.append(Void)
```

Invariant inheritance bug

class *ML_SORT_SET*

```
default_create is
  -- Process instances of
  -- classes with no creation
  -- clause.
  -- (Default: do nothing.)
  -- (from ANY)
  -- (export status {NONE})
do
end
```

Test case

```
create {ML_SORTED_SET[
      STRING]} v_10
```

The invariant

```
  set_size_not_negative:
    # Current >= 0
```

prefix “#”

```
prefix "#": INTEGER is
-- how many elements are currently in the set
-- (i.e. cardinality )? (from ML_SET)
do
  Result := # implementation
ensure -- from ML_MODEL
  is_empty_means_count_is_zero: is_empty
        implies (Result = 0)
end
```

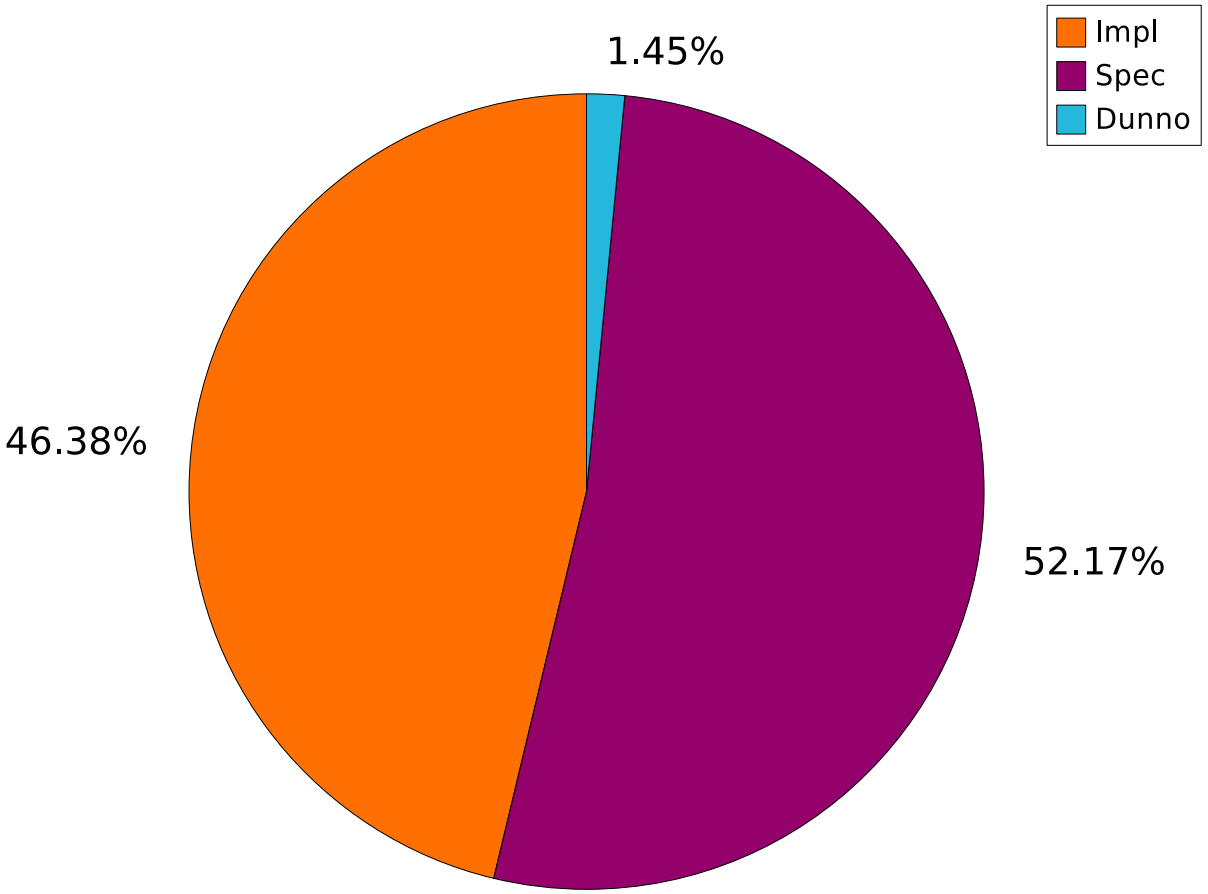
Other bugs in Category 2

- Wrong export status: creation procedures exported to ANY, but they should be exported to NONE!
- External fault
- Void target

Results

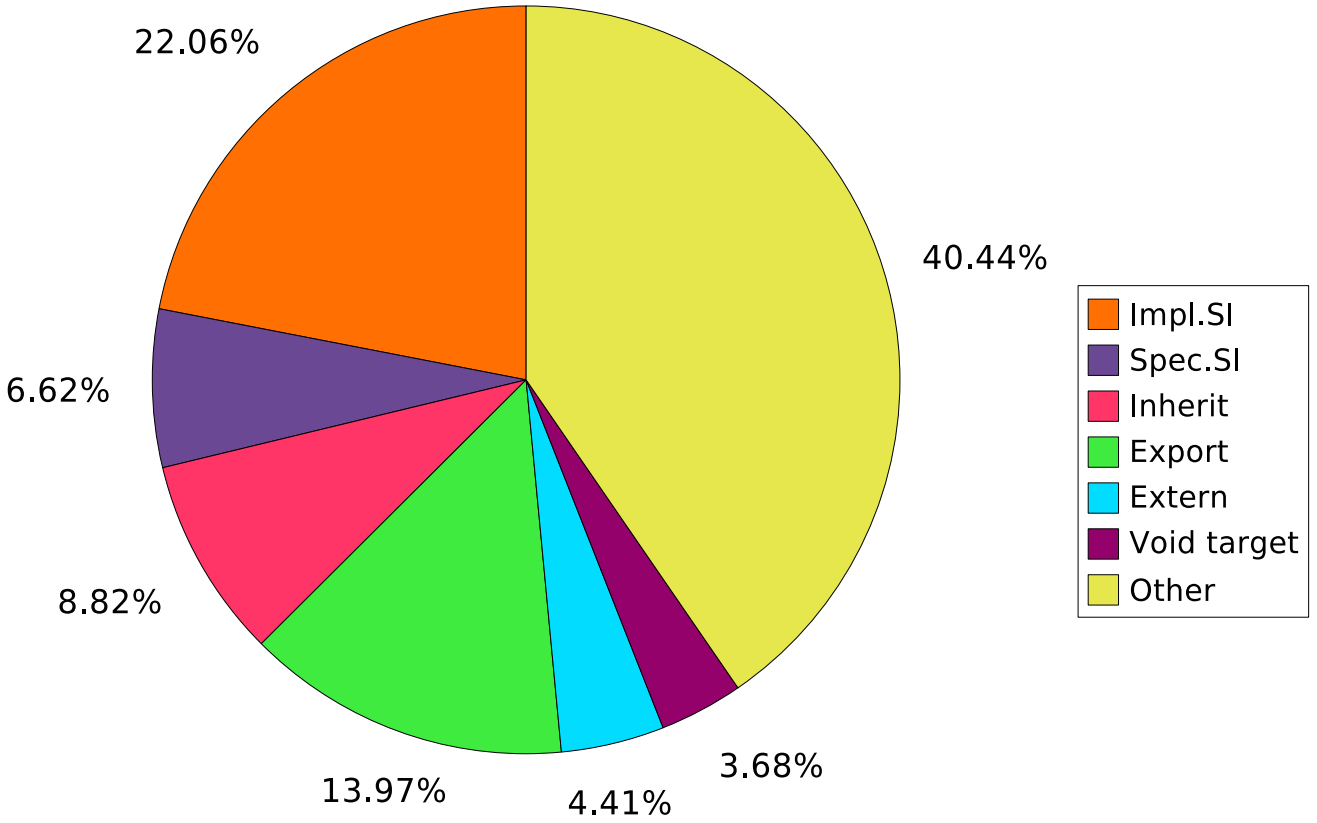
Results - EiffelBase

Category 1: Implementation, Specification , Dunno



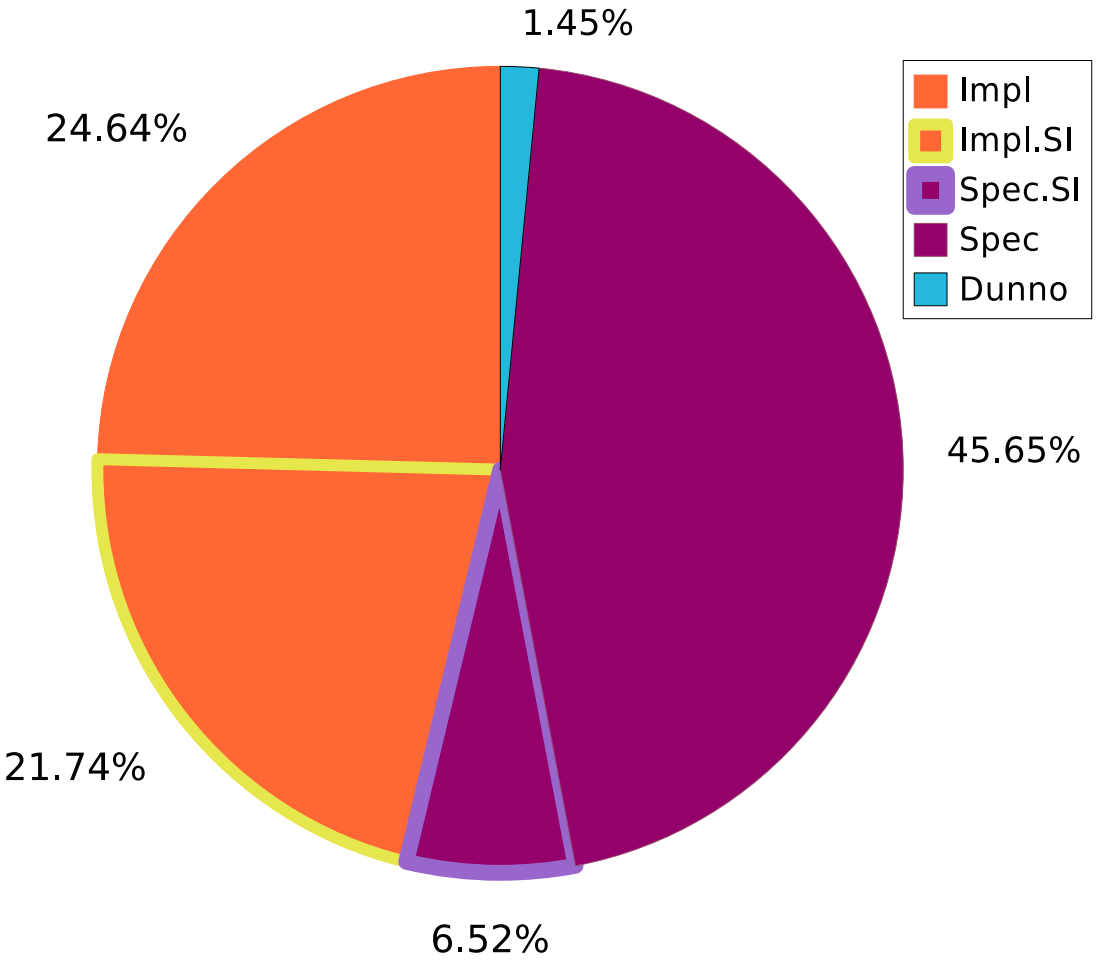
Results - EiffelBase

Category 2: Supplier-induced, Inheritance, Export Status, External fault, Void target, other



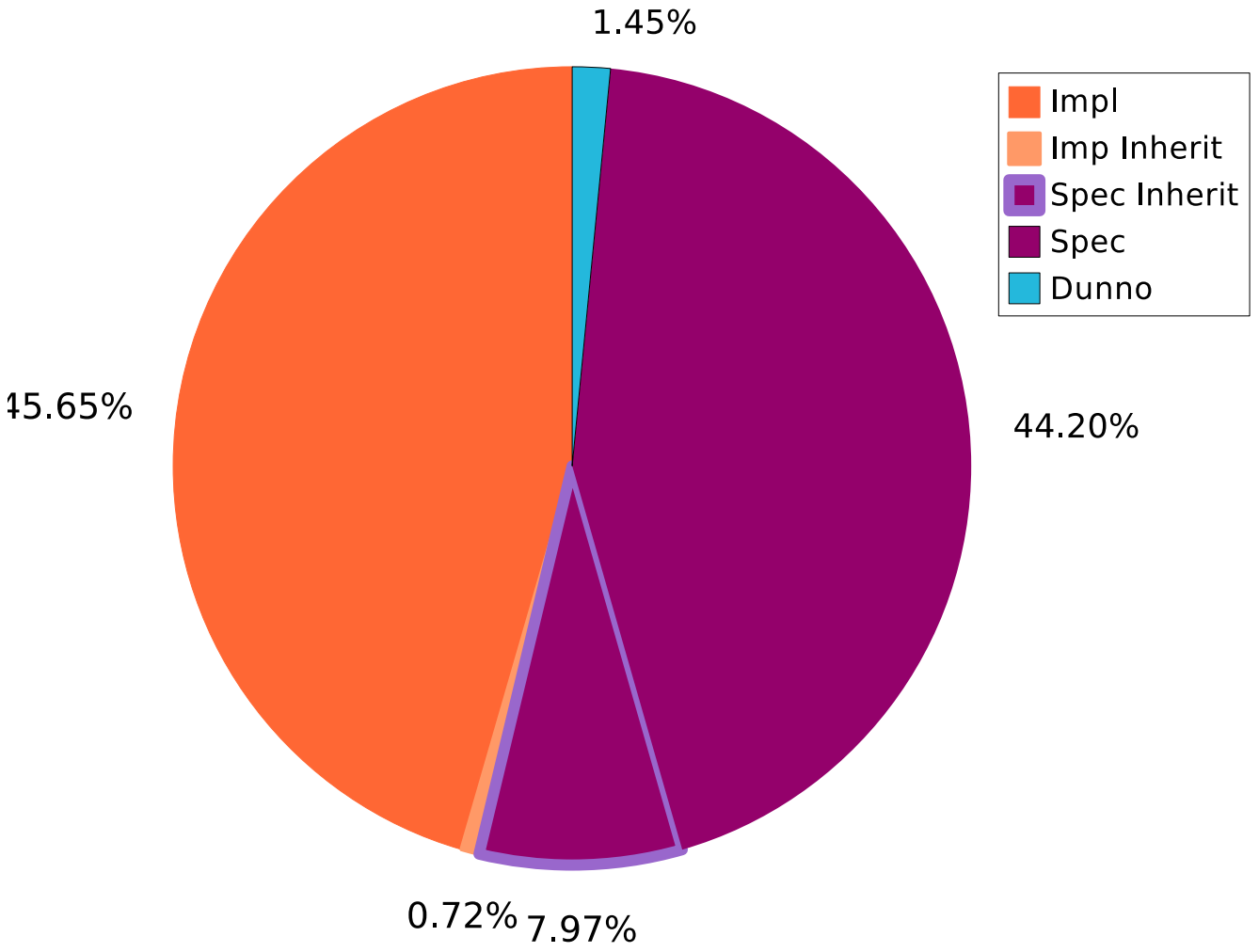
Results - EiffelBase - Overlapping

Faulty Supplier:



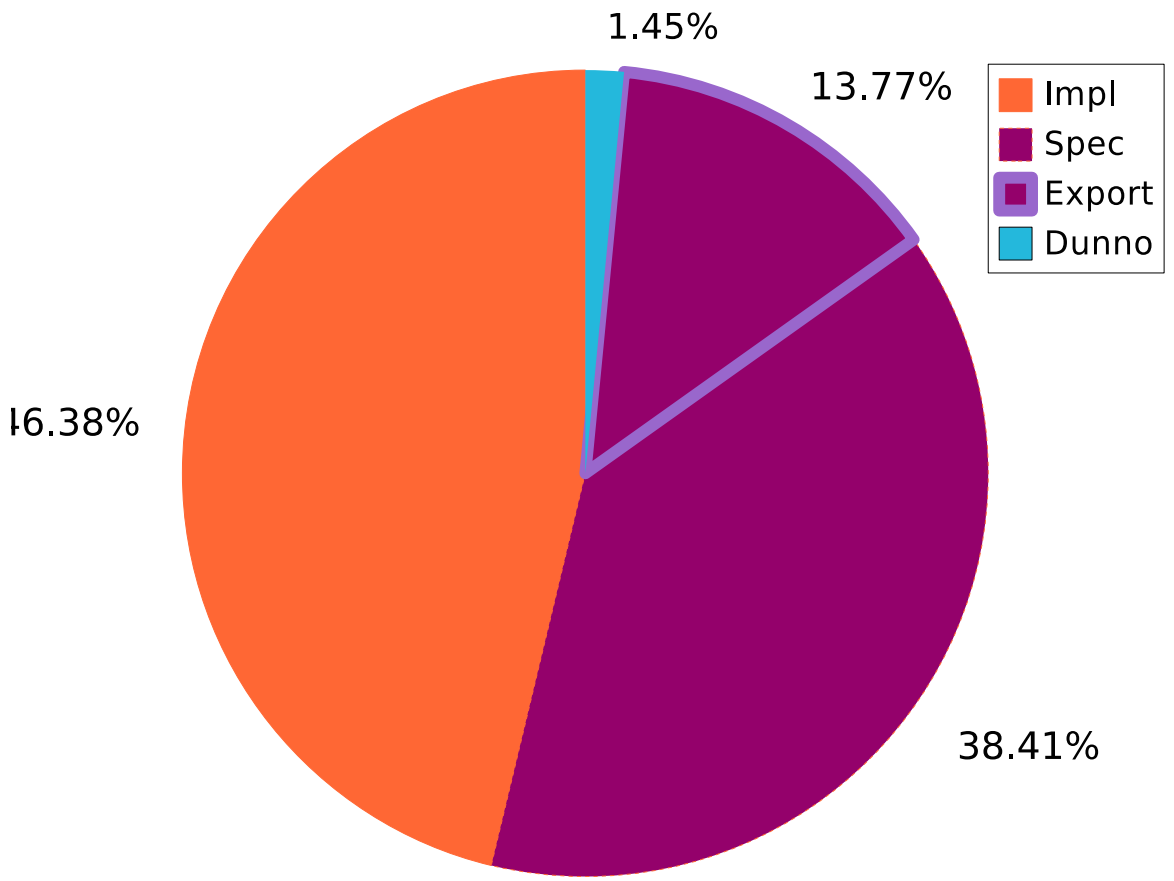
Results - EiffelBase - Overlapping

Inheritance:



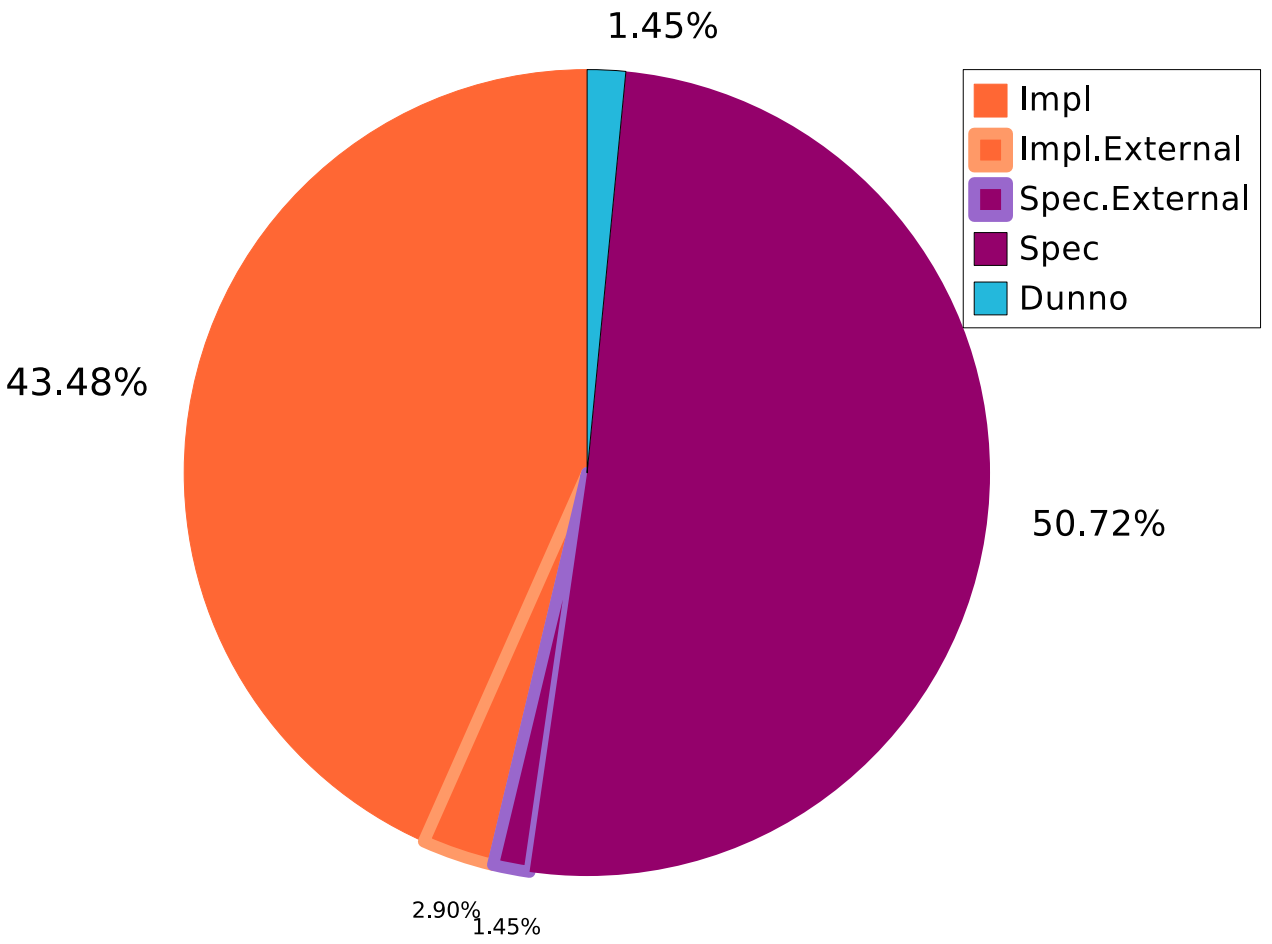
Results - EiffelBase - Overlapping

Export status:



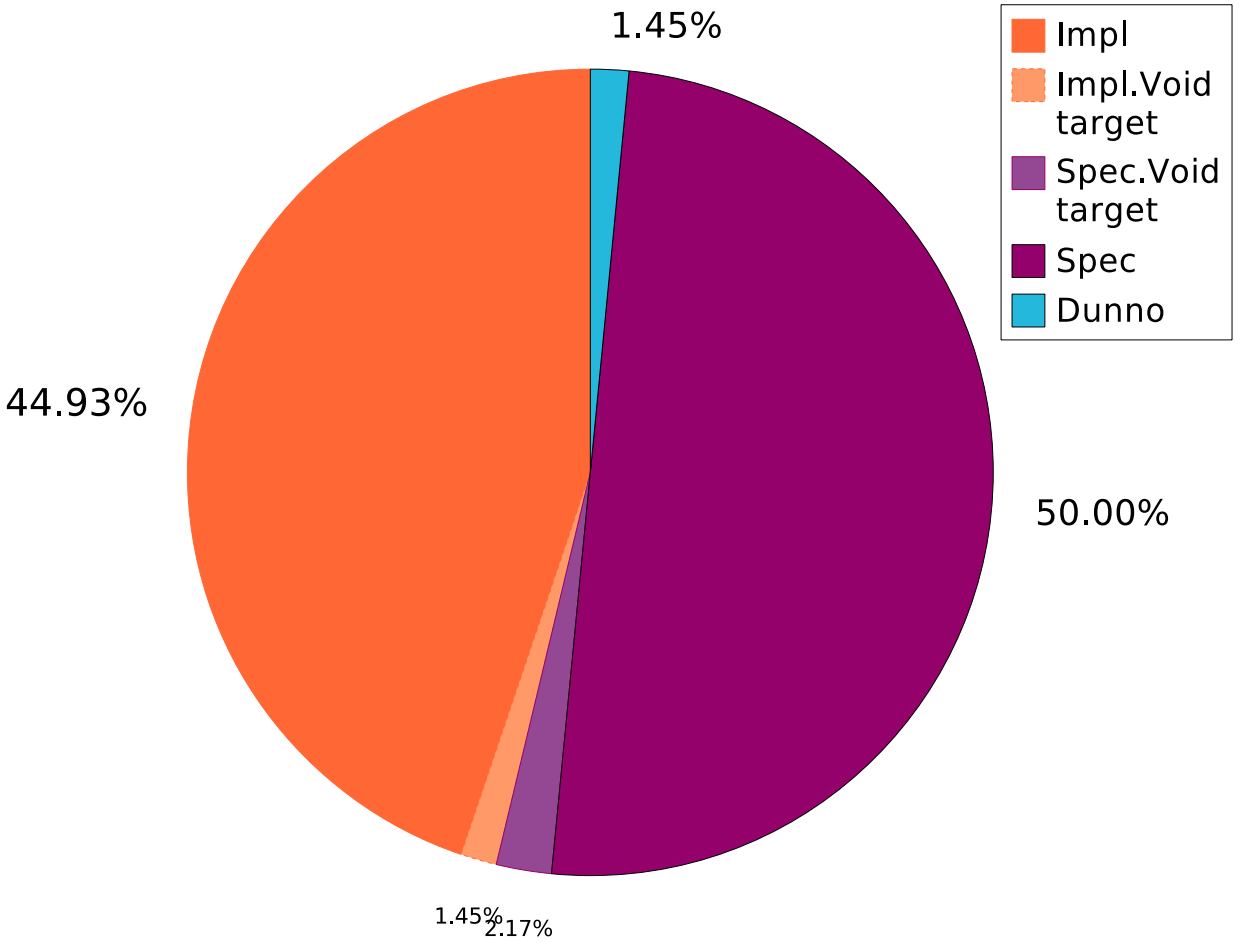
Results - EiffelBase - Overlapping

External fault:



Results - EiffelBase - Overlapping

Void target:



Results

Library/ Application	Bugs	Buggy routines/ total tested	fail tests/ total tests
EiffelBase	127	6.40(127/1984)	3.81%(1513/39615)
base.kernel	16	4.64%(16/343)	1.37%(204/15140)
base.support	23	10.74%(23/214)	5.13%(166/3233)
base.structures	88	6.28(88/1400)	5.38%(1143/21242)
Gobo	26	4.44%(26/585)	3.66%(2928/79886)
Gobo -xml	17	3.85%(17/441)	3.71%(2912/78347)
Gobo - math	9	6.25(9/144)	1.03%(16/1539)
Perfect Devel - math	72	14.11%(72/510)	49.56%(12860/25946)
DoctorC	15	45.45%(15/33)	14.30%(1283/8972)
EWG	8	10.38%(8/77)	1.32%(43/3245)