

Meeting the challenge of reusable software

Bertrand Meyer
ETH Zurich
(at today's conference with Marco Piccioni)

Winterthur, 21 February 2013

Chair of Software Engineering, ETH

Since end 2001

1 professor, 6 postdocs/senior researchers, 11 PhD students

Areas of research: object-oriented programming, Eiffel, program proofs, program testing, program fixing, concurrency, persistence, distributed development, IDEs, robotics software, SE education

Extensive teaching activity, incl. introductory programming

ERC Advanced Investigator Grant (2012-2017):
"Concurrency Made Easy"

Basis: Eiffel

Method and language for software development
Covers all phases, starting from analysis

Full implementation of object-oriented concepts, including multiple inheritance, genericity, and functional programming features

Design by Contract: preconditions, postconditions, class invariants,

3

Contracts in Eiffel

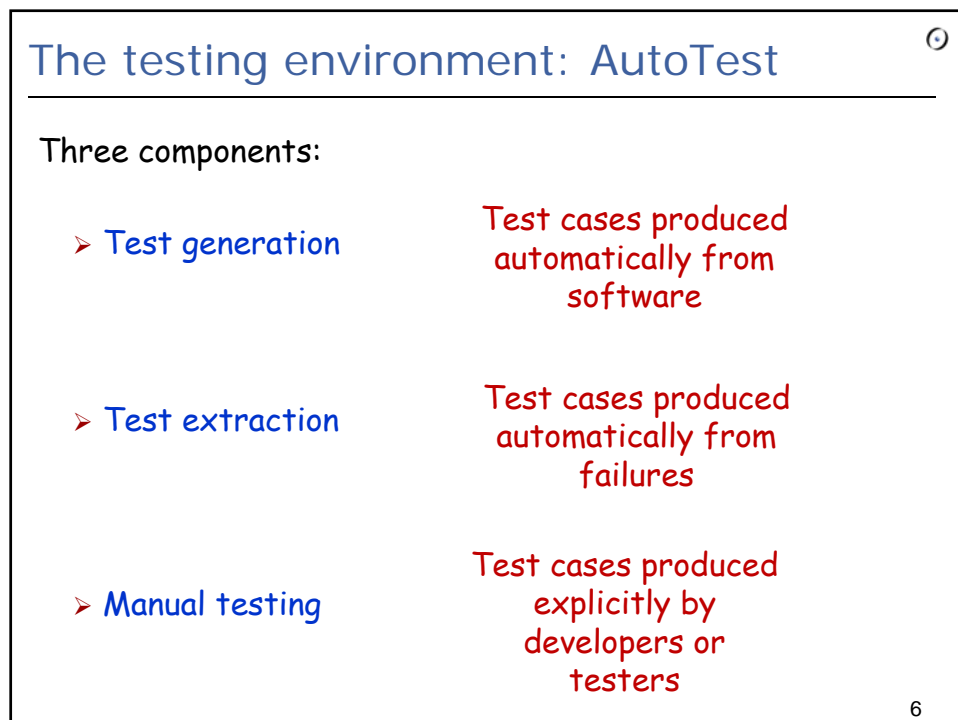
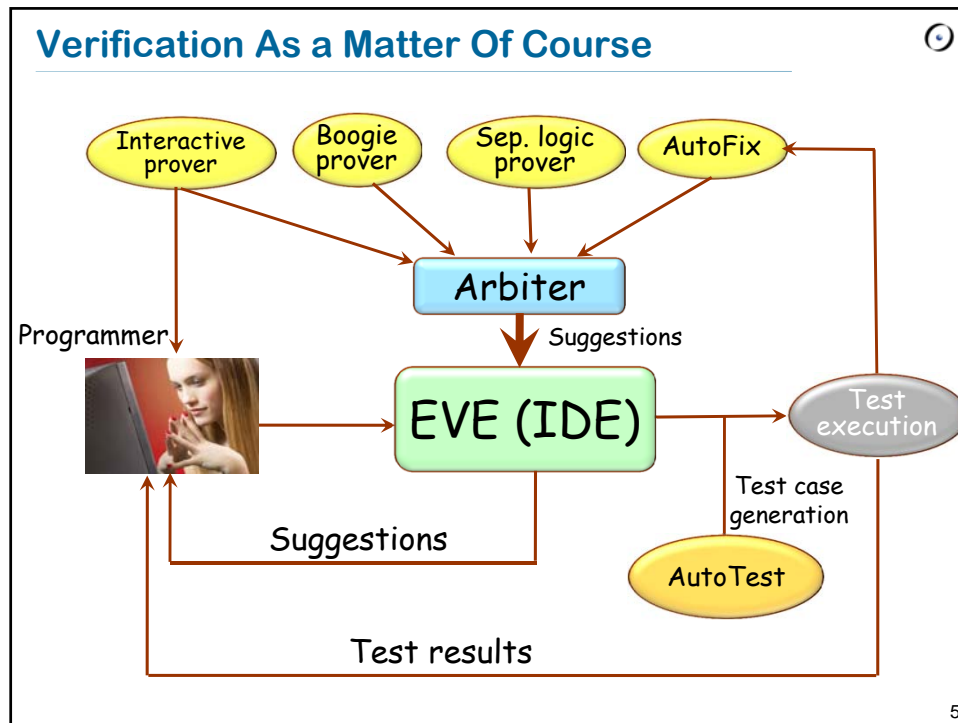
Eiffel contracts are generally too weak:

```
class STACK [G]
  item: G
  count: INTEGER
  -- Top element
  -- Number of elements

  put (x: G)
  -- Push `x' on top
  ensure
    item = x
    count = old count + 1
  -- Old elements are still there
  end
  ...
end
```

Difficult to express formally

4



“Automated testing”

What can be automated:

- Test suite execution
- Resilience
- Regression testing
- Test case generation
- Test result verification (*oracles*)
- Test extraction from failures
- Test case minimization

7

Contracts for testing

Contracts provide the right basis:

- A fault is a discrepancy between intent and reality
- Contracts describe intent

A contract violation always signals a fault:

- Precondition: in *client*
- Postcondition or invariant: in *routine* (supplier)

In EiffelStudio: select compilation option for contract monitoring at level of class, cluster or system.

8

8

AutoTest: Test generation

- Input: set of classes + testing time
- Generates instances, calls routines with automatically selected args
- Oracles are contracts:
 - Direct precondition violation: skip
 - Postcondition/invariant violation: bingo!
- Value selection: Random+ (use special values such as 0, +/-1, max and min)
- Add manual tests if desired
- Any test (manual or automated) that fails becomes part of the test suite

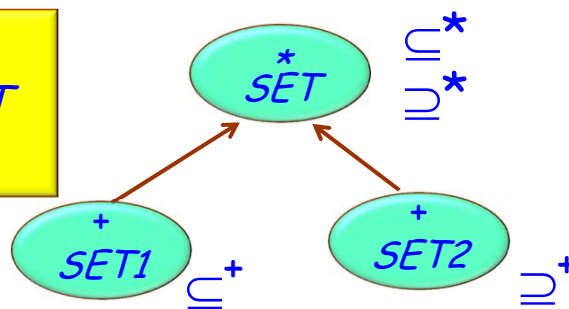
Ilinca Ciupa
Andreas Leitner
Manuel Oriol
Yi Wei
Arno Fiva
et al.

9

Random testing: example bug found

Test:

$s1, s2: SET$
 $s2 \subseteq s1$



*: Deferred
+: Effective

10

Bernd Schoeller

Test extraction

Andreas Leitner, Manuel Oriol, Arno Fiva

"Contract-Driven Development": like Test-Driven Development, but

- Tests derived from spec (contracts)
- Not the other way around!

Record every failed execution, make it reproducible by retaining objects

Turn it into a regression test

11

Specified but unimplemented routine

```
class BANK_ACCOUNT
  inherit ANY
  redefine default_create
end

deposit (an_amount: INTEGER) is
  do
  ensure
    balance_increased: balance > old balance
    deposited: balance = old balance + an_amount
  end
end

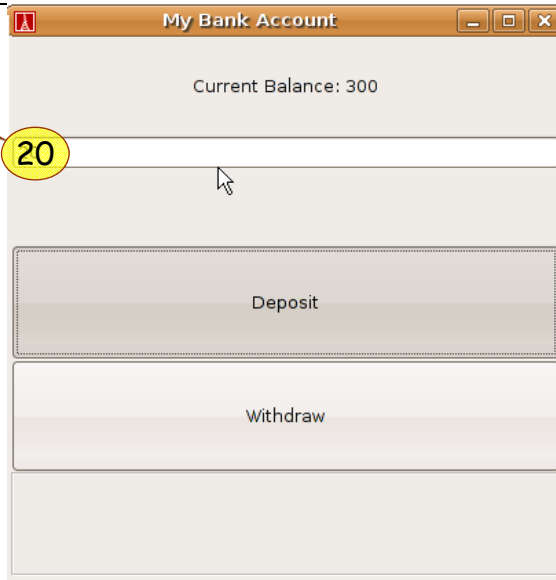
withdraw (an_amount: INTEGER) is
  do
  ensure
    balance_decreased: balance < old balance
    withdrawn: balance = old balance - an_amount
  end
end

invariant
  balance_not_negativ: balance >= 0
```

12

Running the system and entering input

(erroneous)



13

Error caught at run time as contract violation

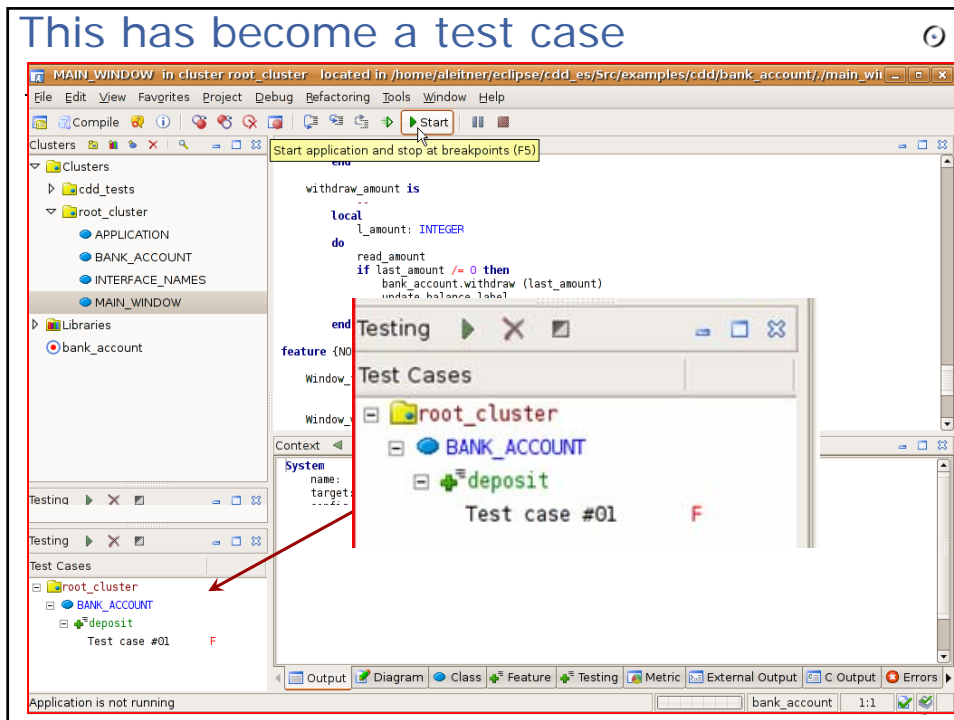
Postcondition violated

Code: 4 (Postcondition violated.) Tag: balance_increased

do
 ensure
 balance_increased: balance > old_balance
 end

The violated clause:
balance > old_balance

Implicit exception pending: Code: 4 (Postcondition violated.) Tag: balance_increased



Automated program fixing

- 16 out of 42 (38%) faults fixed
- Capable of fixing faults due to missing method call
- It takes 3 to 5 minutes to understand a fix
- Some fixes are the same as those from programmers

Proposed fix

duplicate (n: INTEGER): ...

```
do
  pos := cursor
  Result := new_chain
  Result.forth
  from until (counter=n) or after loop
    Result.put_left(item)
    forth
    counter := counter+1
  end
  go_to (pos)
end
```

Faulty
version

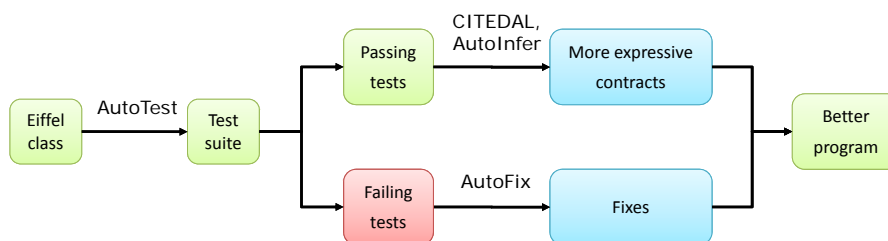
duplicate (n: INTEGER): ...

```
do
  pos := cursor
  Result := new_chain
  Result.forth
  from until (counter=n) or after loop
    if before then
      forth
    else
      Result.put_left(item)
      forth
      counter := counter+1
    end
  end
  go_to (pos)
end
```

Fixed
version

17

Automated testing and fixing



Verifying Function Objects

- Implemented an automatic verifier for a subset of Eiffel
 - Same architecture as Spec# verifier
 - Translation to Boogie
 - Boogie verifier
- Methodology for function objects
 - Using abstract specifications

Verification Assistant: Goals

- Tools run *automatically* in *background*
- Automatic verification using proofs and tests
 - Boogie, jStar, AutoTest
- Automatic inference of code and contracts
 - AutoFix, CITADEL, gin-pink
- Present useful information to user

Possible projects

- Automated program fixing: from libraries to full systems, real-time suggestions
- Distributed development environments and automated configuration management
- Information retrieval engine
- Completely seamless O-O persistence