Can an Introductory Programming Language Support the Teaching of Software Engineering?

James Noble
Andrew Black
Kim Bruce
Michael Homer

1 Why Now?

Happy teaching Java next 3-5 years
In 2015, Java will be 20 years old
Java 8 is coming...
State of the art has advanced...

2 Grace

Grace User Model

First year students in CS1 or CS2
Second year students
Researchers & TAs - assignments and libraries
Faculty & TAs - assignments and libraries
Language Designers wanting a good example

3 Grace Example

method average(in : InputStream) -> Number
// reads numbers from in, streams and averages them

var total := 0
var count := 0
while (! in.atEnd) do 

# total := total + in.readNumber
# count := count + 1
# count := count + 1

if (count == 0) then 

return 0

return total / count
Object Constructors

```java
object {
  def x = 2
  def y = 3
  method distanceTo(other) {
    ((x - other.x)^2 + (y - other.y)^2)
  }
}
```

Object Constructors

```java
object {
  def x = 2
  def y = 3
  method distanceTo(other) {
    ((x - other.x)^2 + (y - other.y)^2)
  }
}
```

Method Requests

```java
aPerson.printOn(outputStream)
printOn(outputStream)
```

Operators are requests

```java
while { ! in.atEnd } do { print (in.readNumber) }
```

multi-part method name

```java
println(outputStream)
```

operators are requests

```java
(x + y) < 2
```

implicit self

```java
println(outputStream)
```
Can an Introductory Programming Language Support the Teaching of Software Engineering?

Consistency

Syntactic Consistency:

```
while (x > 0) {
  other.iterate
}
```

```
while (x > 0) {
  other.iterate
}
```

Semantic Consistency:

```
if (count == 0) then return 0
```

```
if (count == 0) then return 0
```

```
while (x > 0) {
  other.iterate
}
```

```
while (x > 0) {
  other.iterate
}
```

```
new(x', y') {
  def x = x'
  def y = y'
  method distanceTo(otrher) {
    ((x - otrher.x)^2 + (y - otrher.y)^2)
  }
}
```

```
new(x', y') {
  def x = x'
  def y = y'
  class CartesianPoint(new(x', y')) {
    def x = x'
    def y = y'
    method distanceTo(otrher) {
      ((x - otrher.x)^2 + (y - otrher.y)^2)
    }
  }
}
```
Static vs. Dynamic Types

class CartesianPoint.new(x', y') {
def x = x'
def y = y'
method distanceTo(other) {
  ((x - other.x)^2 + (y - other.y)^2) }
}

Types vs Classes

type Point = {
x -> Number
y -> Number
distanceTo(other: Point) -> Number
}

Implicit vs. Explicit Declarations

JavaScript, FORTRAN:
```javascript
counter = counter + delta
```

Pascal, C, Java, Ada:
```pascal
var counter = 0
def delta = 3
counter = counter + delta
```

JavaScript, FORTRAN:
```javascript
def delta = 3
var counter = 0
counter = counter + delta
```

Types vs Classes

class CartesianPoint.new(x, y) {
def x : Number = x
def y : Number = y
method distanceTo(other: CartesianPoint) {
  (x - other.x)^2 + (y - other.y)^2}
}

Static vs. Dynamic Types

class CartesianPoint.new(x', y') {
def x : Number = x'
def y : Number = y'
method distanceTo(other: CartesianPoint) {
  (x - other.x)^2 + (y - other.y)^2}
}

Types need to be defined separately

Types are separate from classes

JavaScript:
```javascript
def distanceTo(other: Point) : Number
```

Pascal, C, Java, Ada:
```pascal
def distanceTo(other: CartesianPoint) : Number
def distanceTo(other: Point) : Number
```
def joe = object {
  var forename := "Joe"
  var surname := "Bloggs"
  var id := 234567

  method asString
    return "Name: {forename} {surname} Id: {id}"  // error here

  print "joe is {joe}."
}

// works now
print "joe is {joe}.

def joe = object {

  var forename
    is public, readable
    := "Joe"

  var surname
    is public, readable
    := "Bloggs"

  var id := 234567

  method asString
    return "Name: {forename} {surname} Id: {id}"

  print "joe is {joe}.

  // works now
  print "joe is {joe}".
}

assert {m >= 0) & (n >= 0) & ((m != 0) | (n != 0)}

var a := \max(m, n)
var b := \min(m, n)
while \{b != 0\}
  invariant {a >= b}
  do {def remainder = a % b

  a := b
  b := remainder

  variant {b}

  return a

}
Formal Reasoning

method for (collection) inv:inv (inv) blk:blk { if (!inv.apply) then { InvariantFailure.raise "Loop invariant not satisfied." } blk.apply (element) } if (!inv.apply) then { InvariantFailure.raise "Loop invariant not satisfied." }

Dialects

dialect "loopinvariant"
import "mocollections" as collections
def data = collections.list.new(2, 3, 4, 5)
var sum : Number := 0
for (data) invariant { sum > 0 } do { sum := sum + item }

19

Grace: objects and method requests
Consistency: syntactic vs semantics
Static vs Dynamic Types
Types vs Classes
Information Hiding
Formal Reasoning
Dialects

Can an Introductory Programming Language Support the Teaching of Software Engineering?

Questions
Comments
Suggestions
Brickbats

No conclusions — we aren’t done yet

20

21

22