Wollok: relearning how to teach Object-Oriented Programming

Javier Fernandes[1,2] Nicolás Passerini[1,2,4]
Pablo Tesone[3,1,2,4] Débora Fortini[1,4]
Nahuel Palumbo[4] Juan Contardo[4] Carlos Raffellini[4]

[1]Universidad Nacional de Quilmes
[2]Universidad Nacional de San Martin
[3]Universidad Nacional del Oeste
[4]Universidad Tecnológica Nacional - F.R. Buenos Aires.

Dec 1, 2015

Agenda

Context: what do we do

- We teach object-oriented programming
- Most of the time in engineering careers
  - i.e. people who are supposed to produce industrial software
  - and have little previous (structured) programming experience
- Planning to translate this experience to highschools in 2016
- Some experience working with smaller kids
  ...but is not what we plan to talk about today

Context: what do we do

- We teach object-oriented programming
- Most of the time in engineering careers
  - i.e. people who are supposed to produce industrial software
  - and have little previous (structured) programming experience
- Planning to translate this experience to highschools in 2016
- Some experience working with smaller kids
  ...but is not what we plan to talk about today

Context: what we try to solve

- Low approval rates
- Bad programming practices enforced
- Low understanding of the fundamental concepts
- Low quality of software produced

Why is that?

- Low abstraction capabilities
- Little mathematical background
- And also behavioral issues
  - Lack of concentration
  - High abandonment rates

Context: what we try to solve

- Low approval rates
- Bad programming practices enforced
- Low understanding of the fundamental concepts
- Low quality of software produced

Why is that?

- Low abstraction capabilities
- Little mathematical background
- And also behavioral issues
    - Lack of concentration
    - High abandonment rates

Introducción
Why is it so difficult to learn OOP?

- Focus on a particular language
- Too much concepts to be learnt

```
package examples;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

- Limited or inadequate **development environments**
- To learn programming demands to **create and handle abstractions**

Introducción
Why is it so difficult to learn OOP?

- Focus on a particular language
- Too much concepts to be learnt

```
package examples;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

- Limited or inadequate **development environments**
- To learn programming demands to **create and handle abstractions**

Introduction
Influences and previous work

Ozono http://ozono.uqbar-project.org/

- Based on Pharo Smalltalk
  - Image-based
  - Dynamic language
- **Incremental learning path**[1]

Gobstones http://www.gobstones.org/

- Careful selection of syntactic elements
- No need of input/output
- Separation of pure and effectful elements

[1] Lombardi, Passerini and Cesario, FRBA, 2007

Introduction
Influences and previous work

Ozono http://ozono.uqbar-project.org/

- Based on Pharo Smalltalk
  - Image-based
  - Dynamic language
- **Incremental learning path**[1]

Gobstones http://www.gobstones.org/

- Careful selection of syntactic elements
- No need of input/output
- Separation of pure and effectful elements

[1] Lombardi, Passerini and Cesario, FRBA, 2007

Our first proposal

1. Think about the learning path
   - Introduce concepts gradually
   - Start with fundamental ones:
     object - message - references - polymorphism
   - Postpone others:
     classes - inheritance - ...
2. Build a customized tool
   - Programming language
   - Development environment
   - Visualization tools
     - (Dynamic) object diagram
     - (Static) class diagram

Our first proposal

1. Think about the learning path
    - Introduce concepts gradually
    - Start with fundamental ones:

      object - message - references - polymorphism
    - Postpone others:

      classes - inheritance - ...
2. Build a customized tool
    - Programming language
    - Development environment
    - Visualization tools
        - (Dynamic) object diagram
        - (Static) class diagram

After seven years of learning...

Ozono was a succesful idea:

- Approval rates raised (from 40 - 50% to 80 - 90%)
- Exported to other universities: UNQ, UNSAM, UNO, FRD, ...
- Big community ($>$ 30 teachers and/or developers)
- Research projects

But...

After seven years of learning...

- Lacks a transition from object-based to class-based
- Environment shortcommings
  - Very attached to Pharo (Eg. debugger)
  - Some tools are not suited for learning
- Sometimes we miss static type information
  - Some simple errors are difficult to detect
  - It is more difficult to guide the programmer
- Far from *mainstream* languages
  - Image-baed
  - Unsuitable for some industrial tools (eg. github)

After seven years of learning...

- Lacks a transition from object-based to class-based
- Environment shortcommings
  - Very attached to Pharo (Eg. debugger)
  - Some tools are not suited for learning
- Sometimes we miss static type information
  - Some simple errors are difficult to detect
  - It is more difficult to guide the programmer
- Far from *mainstream* languages
  - Image-baed
  - Unsuitable for some industrial tools (eg. github)

After seven years of learning...

- Lacks a transition from object-based to class-based
- Environment shortcommings
    - Very attached to Pharo (Eg. debugger)
    - Some tools are not suited for learning
- Sometimes we miss static type information
    - Some simple errors are difficult to detect
    - It is more difficult to guide the programmer
- Far from *mainstream* languages
    - Image-baed
    - Unsuitable for some industrial tools (eg. github)

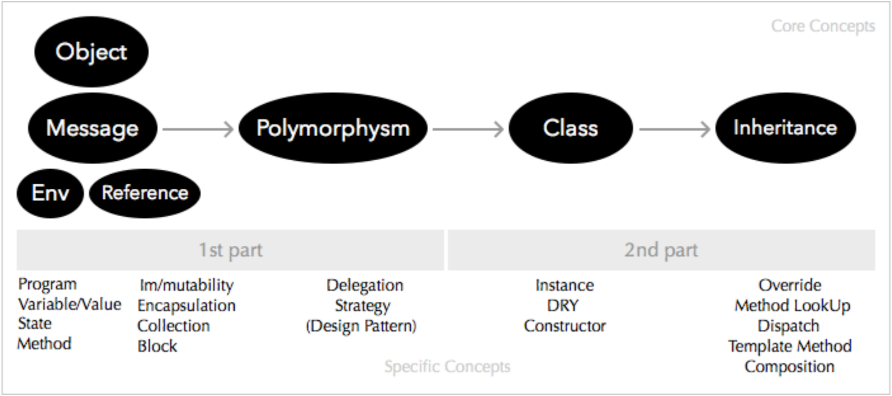
After seven years of learning...

- Lacks a transition from object-based to class-based
- Environment shortcommings
  - Very attached to Pharo (Eg. debugger)
  - Some tools are not suited for learning
- Sometimes we miss static type information
  - Some simple errors are difficult to detect
  - It is more difficult to guide the programmer
- Far from *mainstream* languages
  - Image-baed
  - Unsuitable for some industrial tools (eg. github)

Wollok
The big picture

- Methodology: incremental learning path
  - Additive meta-model
  - Progressive examples & exercises
- Language + IDE (+ lots of related tools)
- Optimized for teaching
  but close to their *mainstream* industrial counterparts!
- Empower the students to use the best development practices
- Integrated assignment submission, correction and grading (starting)

Wollok the methodology
Incremental learning path



Wollok the language
Emphasize the programming concepts

- Combines *object-based* with class-based programming
- Everything is an object
- *Almost* everything is done through messages [2]
- *Educative* syntax (eg: method & inherits keywords)
- Selected concepts (eg: differentiate val vs. var)
- *Pluggable* type system (in progress)

[2] We have if and try/catch constructs.

Wollok the language (+ tools)

Warning: Do not miss the evolution of industrial tools!

- Light and *modern* syntax

  Eg. lambdas, literals, exceptions, constructors
- Mixins (planned)
- Ad-hoc testing constructs
- File-based object environment
- Simplified code repository integration (starting)

Discussion: Why a new language?

Because it allows for 100% customization.

- Better error detection (eg: mandatory return)
- Avoid overloaded APIs (eg: collections)
- Integrated tools

Additive meta-model

- Start only with objects
- Add classes playing nicely with preexisting programs
- Another example: import system

Discussion: Why a new language?

Because it allows for 100% customization.

- Better error detection (eg: mandatory return)
- Avoid overloaded APIs (eg: collections)
- Integrated tools

Additive meta-model

- Start only with objects
- Add classes playing nicely with preexisting programs
- Another example: import system

Discussion: Why a new language?

Because it allows for 100% customization.

- Better error detection (eg: mandatory return)
- Avoid overloaded APIs (eg: collections)
- Integrated tools

Additive meta-model

- Start only with objects
- Add classes playing nicely with preexisting programs
- Another example: import system

Some details

- Explicit receiver, always write obj.msg()
- Many literal objects: collections, positions, date & time (planned)
- Initial values for variables & constants
- Objects can inherit from classes
- Operator precedence
- Abreviated syntax for mathematical assignments $a+=10$

More discussions...

- Standalone objects are visible globally
- *Impure* language
  - If construct
  - Try/catch construct
  - Constructors
- Image vs. file based

- Why not have properties?
- Object literals => a class-less way of code sharing?

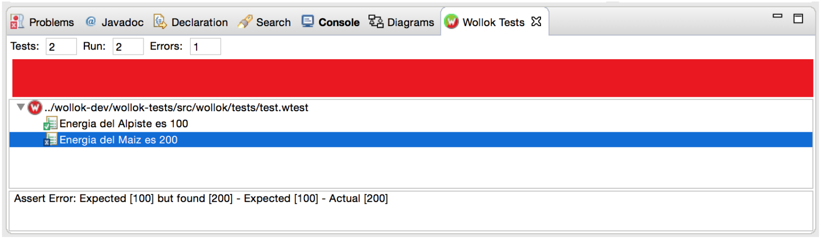
Advanced features

- Debugger
- Automatic class & object diagrams
- Integrated testing framework
- Navigation (eg: go to the definition)
- Content assist, autocomplete
- Quick fixes
- Automatic refactorings (in progress)
- I18N
- Wizards / templates (eg. create project/class/object/test)
- Integrated groupware (in progress)
- Integration in other editors (sublime, ace)
- Configurable syntax (prototypes)

Advanced Features
Debugger

- Integrated to Eclipse Debugger
- Breakpoints
- Step, into, out
- Inspect variables
- Object diagram

Advanced Features
Unit testing



Tests: 2 Run: 2 Errors: 1

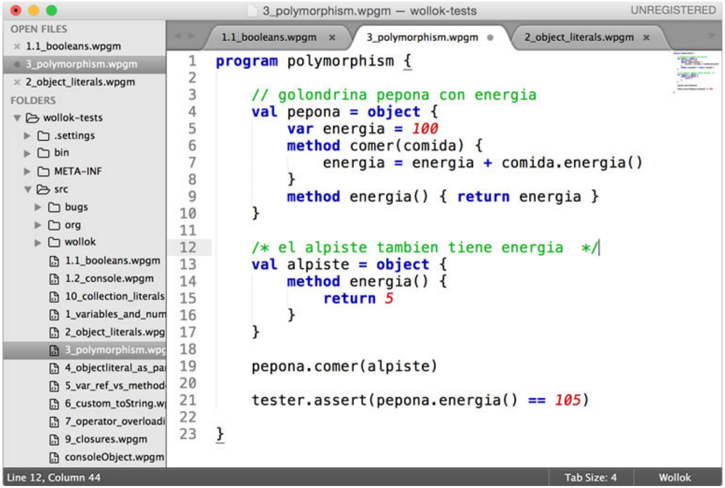
../wollok-dev/wollok-tests/src/wollok/tests/test.wtest
Energia del Alpiste es 100
Energia del Maiz es 200

Assert Error: Expected [100] but found [200] - Expected [100] - Actual [200]

Advanced Features
Sublime editor support

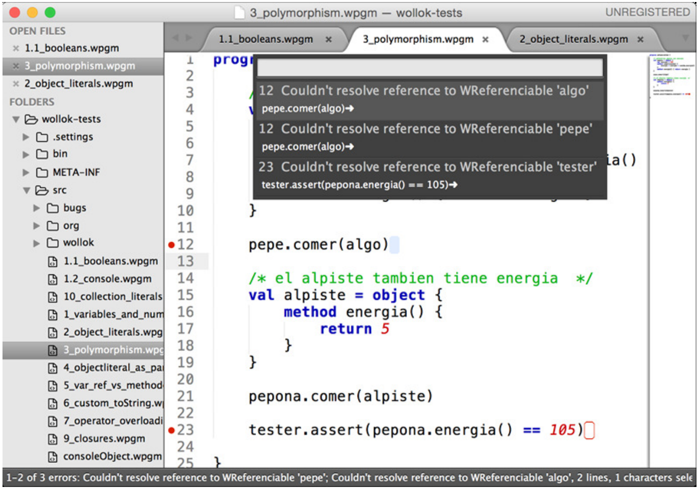
- WDK
    - No IDE
    - $\sim$ 70MB (vs $\sim$ 140)
    - Headless: wchecker, winterpreter, wtest
- Syntax highlight
- Templates
- Linter

Sublime Support
Syntax Highlight



```
program polymorphism {

    // golondrina pepona con energia
    val pepona = object {
        var energia = 100
        method comer(comida) {
            energia = energia + comida.energia()
        }
        method energia() { return energia }
    }

    /* el alpiste tambien tiene energia  */
    val alpiste = object {
        method energia() {
            return 5
        }
    }

    pepona.comer(alpiste)

    tester.assert(pepona.energia() == 105)

}
```

Sublime Support
Linter



3_polymorphism.wpgm — wollok-tests

12 Couldn't resolve reference to WReferenciable 'algo'
pepe.comer(algo)➔
12 Couldn't resolve reference to WReferenciable 'pepe'
pepe.comer(algo)➔
23 Couldn't resolve reference to WReferenciable 'tester'
tester.assert(pepona.energia() == 105)➔

```
}

pepe.comer(algo)

/* el alpiste tambien tiene energia  */
val alpiste = object {
    method energia() {
        return 5
    }
}

pepona.comer(alpiste)

tester.assert(pepona.energia() == 105)
}
```

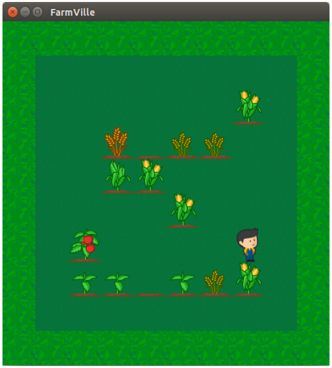
1-2 of 3 errors: Couldn't resolve reference to WReferenciable 'pepe'; Couldn't resolve reference to WReferenciable 'algo', 2 lines, 1 characters sele

Wollok Game

- Herramienta complementaria al testeo unitario y consola interactiva.
- Mejorar la comprensión de conceptos.
- Visualización de comportamiento
- Motivación en el aprendizaje fomentando la participación.

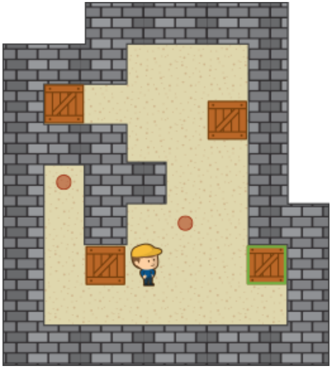
Wollok Game
FarmVille

FarmVille - Demo



Wollok Game
Sokoban

Sokoban - Demo



Wollok Game
Future work

- More types of **Games**
  - Survival
  - Turn-based
- More types of **Interactions**
- Visual features
  - Animations
  - Infinite backgrounds
  - Different view (side view, isometric view, ...)

Wollok development

- OpenSource: LGPLv3
- Stack: Eclipse XText + Xtend Lang
- SCM:
  - **Code**: GitHub (uqbar-project/wollok)
  - **Build**: Maven + Tycho
  - **Continuous Integration**: Travis
  - **Continuous Deployment**
  - **Coverage**: coveralls + jacoco
- Testing & TDD

Wollok development
Continuous Integration & Deployment

- **GitFlow**
  - Feature Branches
  - Pull-Requests
  - *dev* $\rightarrow$ *master* $\leftarrow$ *hotfixes*
- **Integration**:
  - Travis
  - compile, test, coverage, deploy
- Deployment:
  - **Products** (IDE): multiple platforms
  - **Update Sites**
  - **WDK**
  - 2 Environments: Stable & Dev

Wollok Development
Testing & TDD

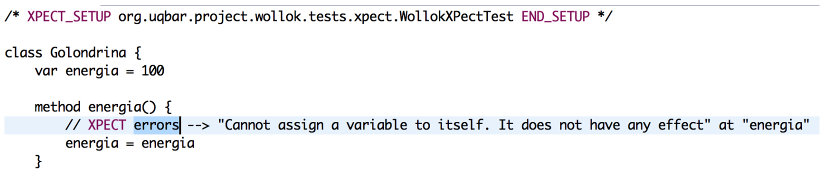
- 87% coverage
- **Runtime**
  - Test program execution
  - Interpreter
  - **JUnit + iDSL**
- **Static**
  - **Check system**: XPect
  - **Type System**: JUnit + iDSL
  - **Autocomplete**: XPect
  - **Formatting**: JUnit + iDSL
- **Pending**
  - Quick-Fixes
  - Refactorings

Testing & TDD
Runtime

Interpreter testing

```
class PostFixOperationTestCase extends AbstractWollokInterpreterTestCase {

    @Test
    def void testPlusPlus() {'''
        program p {
            var n = 1
            n++

            assert.that(n == 2)
        }'''.interpretPropagatingErrors
    }
```

Testing & TDD
Static checks



```
/* XPECT_SETUP org.uqbar.project.wollok.tests.xpect.WollokXPectTest END_SETUP */

class Golondrina {
    var energia = 100

    method energia() {
        // XPECT errors --> "Cannot assign a variable to itself. It does not have any effect" at "energia"
        energia = energia
    }
```

Teaching experience

Students intuitively take advantage of the language and tools:

- Class-based + object-based integration
- REPL is more intuitive than traditional Smalltalk workspaces
- More control over unit testing
- More traditional editors

An **incremental learning path** supported by adequate **tools**,
empowers students' **intuition**
incrementing their **autonomy, creativity and motivation**

Teaching experience

Students intuitively take advantage of the language and tools:

- Class-based + object-based integration
- REPL is more intuitive than traditional Smalltalk workspaces
- More control over unit testing
- More traditional editors

An **incremental learning path** supported by adequate **tools**,
empowers students' **intuition**
incrementing their **autonomy, creativity and motivation**

Teaching experience

Students intuitively take advantage of the language and tools:

- Class-based + object-based integration
- REPL is more intuitive than traditional Smalltalk workspaces
- More control over unit testing
- More traditional editors

An **incremental learning path** supported by adequate **tools**, empowers students' **intuition**

incrementing their **autonomy, creativity and motivation**

Teaching experience

Students intuitively take advantage of the language and tools:

- Class-based + object-based integration
- REPL is more intuitive than traditional Smalltalk workspaces
- More control over unit testing
- More traditional editors

An **incremental learning path** supported by adequate **tools**,
empowers students' **intuition**
incrementing their **autonomy, creativity and motivation**

Próximos pasos

Próximos Pasos

- Varias discusiones sobre la mejor sintaxis (in progress)
- Plataforma p/interacción Alumno $\leftrightarrow$ Docente (starting)
- Herencia basada en mixins
- Implementar wollok-game en el aula
- Block-based editor

Y muchas actividades para sumar más gente al proyecto.

Muchas gracias

¡Muchas Gracias!