A tale about software profiling, debugging, testing, and visualization

Alexandre Bergel et al.
University of Chile & Object Profile

bergel.eu
objectprofile.com
Three-polar identity disorder

---

// Split a into an array of UTF-8 sequences, and get Unicode
// Invalid UTF-8 sequences become correct encodings of UnPFB.
func explode(s string, n int) []string {
    if n <= 0 {
        n = len(s)
    }
    a := make([]string, n);
    var size, rune int;
    ma := 0;
    for len(s) > 0 {
        if ma < n {
            a[ma] = s;
            ma++;
            size = utf8.DecodeRuneInString(s);
        }
        new, size = utf8.DecodeRuneInString(s);
        s = s[size:len(s)];
        ma = string(rune);
        new :=
    }
    return a[0:ma];

---

Sequence diagram of an order management system

---

Stack, Heap, ...
Execution profiling with Kai

Traditional code profilers are driven by the method stack, discarding the notion of sending messages.

How to answer to “Is there a slow method that is called too often?”

Visually compare time and the number of executions.
Method 1 calls Method 2.
Method 3 calls Method 1
Profiling blueprint

methods that takes times, are executed many times, and without significant side effect
Adding a memoization

ROElement>>bounds
   "Return the bounds of the element"

   ^
   self position extent: (shape extentFor: self)
Adding a memoization

```smalltalk
ROElement>>bounds

"Return the bounds of the element"

boundsCache ifNotNil: [ ^ boundsCache ].

^ boundsCache :=
    self position extent: (shape extentFor: self)
```
Effect of the memoization

speedup 43%
Execution profiling blueprints. *Software: Practices and Experience, 2012*

*Visualizing Dynamic Metrics with Profiling Blueprints*. *Proceedings of the TOOLS, 2010*

*Counting Messages as a Proxy for Average Execution Time in Pharo*. *Proceedings of ECOOP, 2011*

---

**Fig. 1.** Linear regression for the 16 Pharo applications.
Tracking Performance Evolution

Understanding of performance evolve across multiple software revisions is difficult
Measuring performance evolution

Profile for 1.0

Profile for 2.0

Software Version 1.0

Software Version 2.0
Measuring performance evolution

Profile for 1.0

Software Version 1.0

Profile for 2.0

Software Version 2.0
<table>
<thead>
<tr>
<th>Name</th>
<th>Time Diff (ms)</th>
<th>Old Time (ms)</th>
<th>New Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;All threads&gt;</td>
<td>+1,101</td>
<td>856</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom.test.BenchMark.main(String[])</td>
<td>+1,112</td>
<td>840</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.build(File)</td>
<td>+1,114</td>
<td>828</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.build(URL)</td>
<td>+1,134</td>
<td>808</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.build(InputSource)</td>
<td>+1,155</td>
<td>787</td>
<td>1</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.parse_proxy(InputSource, XMLReader)</td>
<td>+971</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.new_method()</td>
<td>+570</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.apache.xerces.parsers.AbstractSAXParser.parse(InputSource)</td>
<td>+400</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.createParser()</td>
<td>+589</td>
<td>325</td>
<td>325</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.configureParser(Reader, SAXHandler)</td>
<td>+25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>org.apache.xerces.parsers.AbstractSAXParser()</td>
<td>+14</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>java.net.URL.toExternalForm()</td>
<td>-20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.fileToURL(File)</td>
<td>-20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>org.jdom.input.SAXBuilder.&lt;init&gt;()</td>
<td>+2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>java.lang.ClassLoader.loadClass(String)</td>
<td>-4</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>java.lang.ref.Finalizer$FinalizerThread.run()</td>
<td>+4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>java.lang.ClassLoader.loadClass(String)</td>
<td>-15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
Measuring performance evolution

Variation have to be manually tracked

Relevant metrics are missing

Poor visual representation
Framework to measure evolution

data := Rizel new

    setTestsAsBenchmarks;

trackLast: 100 versionsOf: 'Roassal';

run.

data export...
Performance Evolution Blueprint

A invokes B

Δ time

Δ # executions

Color

Border

A invokes B
Rizel in action
Rizel in action
Test coverage with Hapao

*Traditional* code coverage *tools* have a *binary view* of the world

*Is my code well covered or not?*

*Which method should you test first in order to increase the coverage?*

Visual *qualitative assessment* of the coverage
Test blueprint

Legend for methods (inner boxes)

- # calling methods
- complexity
- # executions
- invocation on self
- red = not executed
- blue = abstract
Successive improvement

Version 2.2 27.27%
Version 2.3 54.54%
Version 2.4 87.71%
Version 2.5 100%
Reducing code complexity

Version 1.58.1
Coverage: 40.57%

Version 1.58.9
Coverage: 60.60%
Reducing code complexity

Version 2.10

Version 2.17
4 patterns

P1

P2

P3

P4
Agile Visualization Engine
Pharo is a dynamic object-oriented programming language. Pharo's model and syntax are uniform, simple and expressive. These properties, when added to a powerful and flexible programming environment, regularly attract new developers. The community around Pharo has been steadily increasing over the years. This community is actively creating exciting and innovative software artifacts helping the development of advanced software systems. Pharo leverages the software building experience to its best by offering open and object-oriented programming environments and libraries.

The book covers a large spectrum of topics ranging from central language aspects such as blocks and exceptions to package management and graphics oriented frameworks. Recent frameworks like Roassal and Petit Parser are covered. This book contains unique material often presented in a tutorial form with many experiences to carry on. Everybody will learn something reading this book: programmers familiar with Pharo will enjoy the highlights made of some particularly beautiful aspects of Pharo as well as discovering new and powerful frameworks. Practitioners making their debut with Pharo will board for a wonderful journey in the realm of objects.

Deep into Pharo not only presents some internal aspects of Pharo but it presents important libraries that proved to be important for a business and development perspective.
Conclusion

Little innovation in the tools we commonly use

Profilers, debuggers, testing tools have not significantly evolved

Fantastic opportunities for improvement

Kai, Rizel, Hapao, Roassal are just a beginning