Übergehypte neue Sprachen auf der Java VM – eine differenzierte Analyse

a presentation for Java User's Group Switzerland

by Philipp H. Oser

04/ 2011

ELCA We make it work.

Übergehypte neue Sprachen auf der Java VM – eine differenzierte Analyse

An Java Konferenzen werden die neuen JVM-Sprachen wie Groovy oder Scala gehypt. Was bleibt nach genauerer Betrachtung? Sprachen sollen die Architektur-Governance unterstützen, z.B. durch Reduktion der Komplexität, möglichst früher Detektion von Fehlern, langfristige Stabilität, Überwachung der eingesetzten Sprachfeatures. Diese Sprachen sind für langfristige Standardprojekte suboptimal ausgerichtet. Wo sollte es hingehen? Eine differenzierte Analyse.

Short Poll

Groovy: Who has heard of it?

Who knows it a bit?

Who has really used it to implement something?

Who is really proficient in it?

What do you think of it? (+ thumb up,
- thumb down,
~ fist up)

Short Poll

Scala: Who has heard of it?

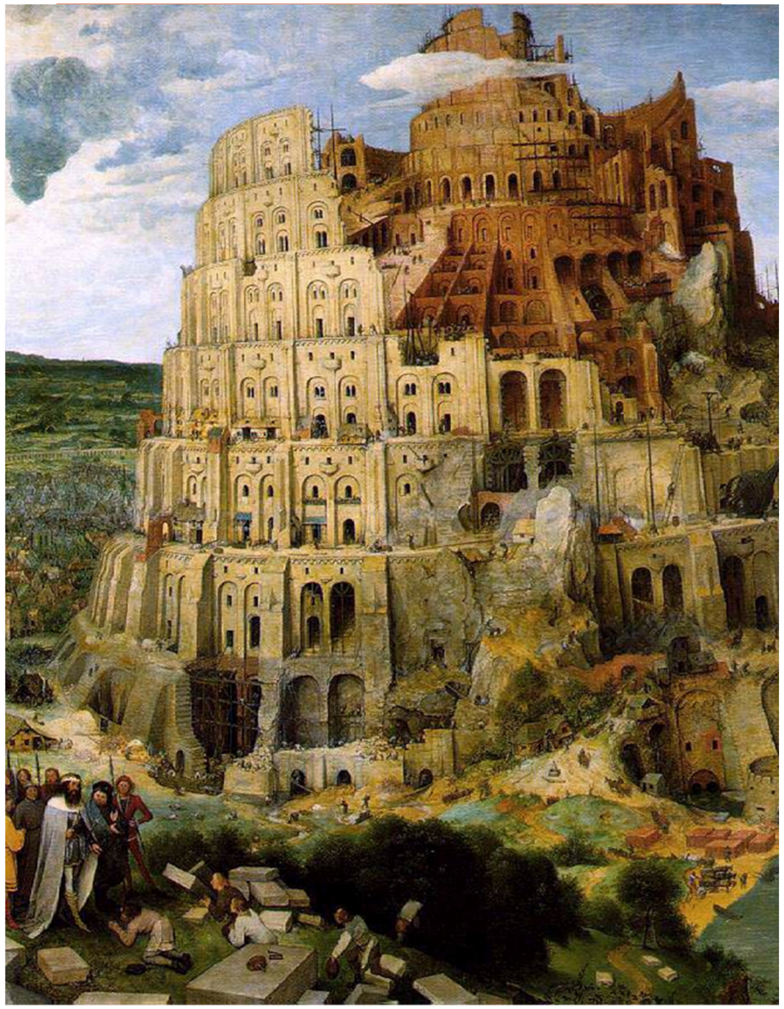
Who knows it a bit?

Who has really used it to implement something?

Who is really proficient in it?

What do you think of it? (+ thumb up,
- thumb down,
~ fist up)

Agenda


Turmbau zu Babel, Pieter Brueghel der Ältere, 1563

- Intro
- Concrete experiences with new JVM languages
- My current issues/ My wish list for Java and its environment
- Alternatives from different sources

My employer: ELCA

ELCA is one of the Swiss main independent IT companies in the field of software development and systems integration.

We develop, integrate, operate and maintain IT solutions using custom developed applications, as well as industry standards.

Founded in	1968
Employees	~ 500 employees
Offices	Lausanne (head-quarters), Zurich, Geneva, Bern, London, Madrid, Paris, Ho Chi Minh City (Vietnam)
Turnover	CHF 71 millions
Quality Standards	ISO 9001 since 1993 CMMI Level 3 since 2007
Awards	SWISS TECHNOLOGY AWARD; THE EUROPEAN IST PRIZE WINNER; SOFI SWISS ORGANISATION FOR FACILITATING INVESTMENTS; Swiss Rising Star 2008 Microsoft Dynamics; Swiss ICT Award 2008 InfoWeek CHAMPION

My Bias

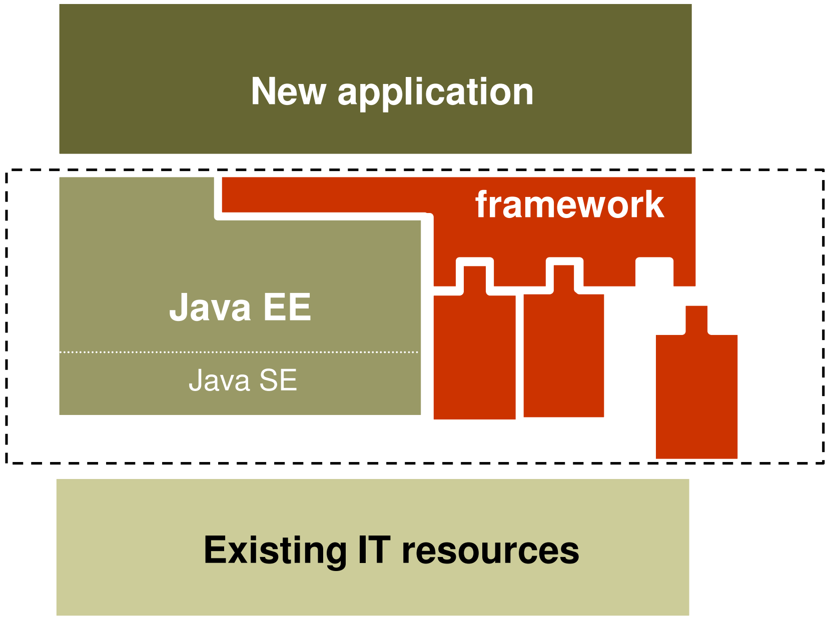
Lead Architect at ELCA Zurich

- 10+ years of experience in technology stack development/ integration
  - Responsible for two Java EE technology stacks
    - LEAF (proprietary) used in ~ 20 projects
    - EL4J (open) used in 30+ projects
- Technology watch
  - Framework updates
  - New languages
    - Includes “what languages are appropriate”? (includes new languages & Domain-Specific Languages (DSLs)/ and Model-Driven Software Engineering (MDSE))
- Architecture governance
  - Coordination of Architecture Reviews in Zurich Office (a mandatory quality gate for projects of certain size)
  - Collaboration in project governance (at ELCA and at client’s sites)
  - Review mandates

20+ years of programming experience

Some context/ my view of the world

- The libraries/ frameworks are also important for the success of a language. Sometimes the border framework/ language is not so strict.



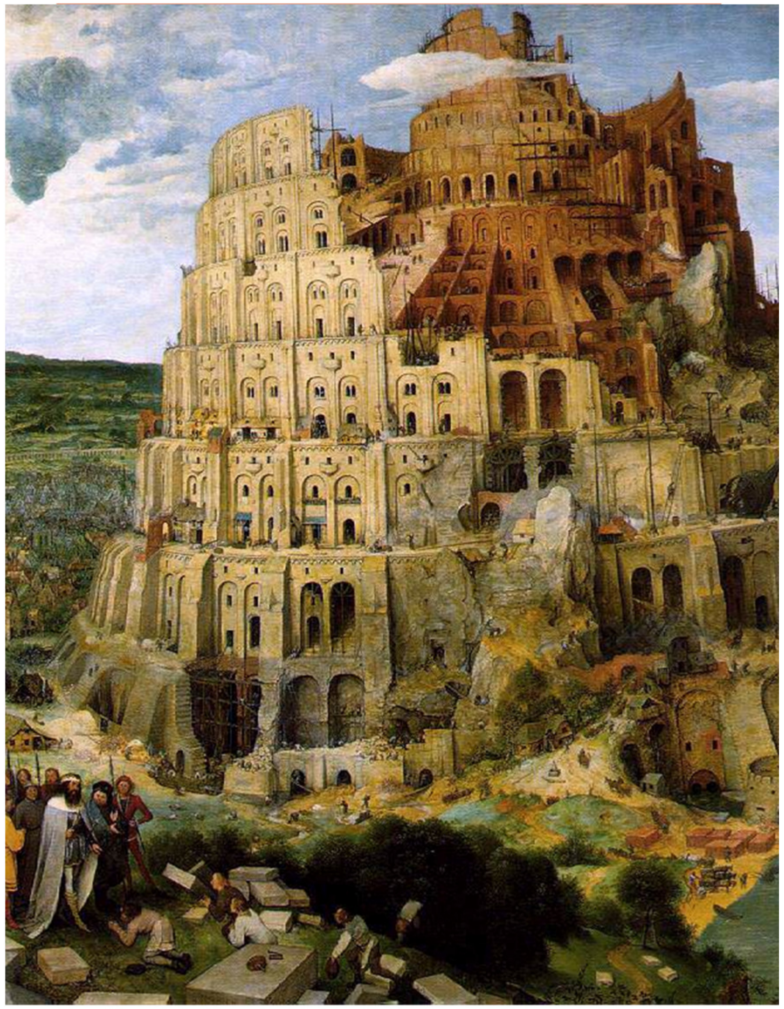
- I am interested in technologies for **strategic systems** (for tactical projects, the long-term viability/ the governability/ the cleanliness is less important)
- This presentation represents **my opinion**. I play the devil's advocate (as we discuss about replacing something that works).
  Please don't sent me death threats. ;-)

Some context/ my view of the world

- A DSL := a Domain-Specific Language (DSL) is a programming language or specification language dedicated to a particular problem domain
  - E.g.: SQL, Regexp, Yacc, XSLT, BPMN 2,
    own specific languages (e.g. for investment products)

- IMHO: Creating a real new DSL should be something that is *very explicit* (special team & very visible in code): otherwise it can be hard to understand code.

Concrete experiences with new JVM languages



- Intro
- ▶ Concrete experiences with new JVM languages
- My current issues/ My wish list for Java and its environment
- Alternatives from different sources

Concrete experiences with Groovy

Groovy

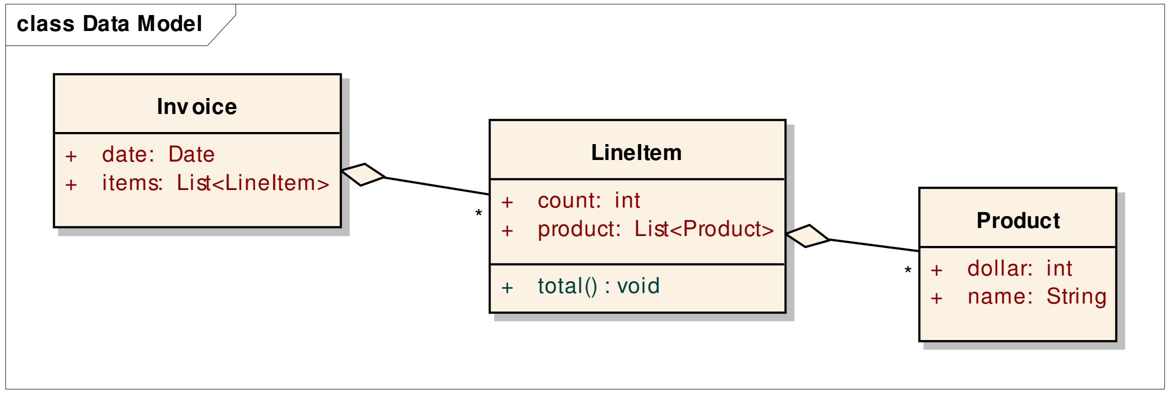
- Many cool concepts
  - Less boilerplate code: properties, closures,
  - Interceptors in language (MOP)
  - Library extensions (add stuff to existing classes)
  - Support for easier internal DSLs
- I read the book „Grooyv in Action“
- Wow!

Example feature: GPaths (Groovy Paths)

```
this.class.methods.name.grep(~/get.*/).sort()
```

may lead to (is function of what `this` is):

```
["getBinding", "getClass", "getMetaClass", "getProperty"]
```



```
Invoice invoices = … // get some content
invoices.items.grep { i -> i.total() > 7000 }.product.name
```

Is quite readable, takes 8 lines of code to implement in Java

GPaths (Groovy Paths)

The same in Java

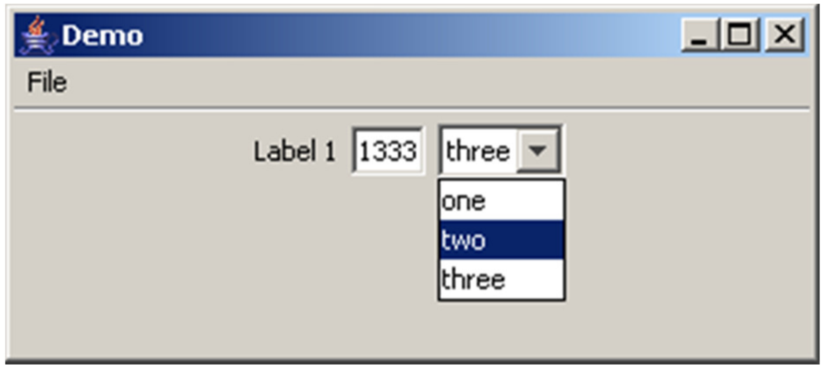
```
List result = new LinkedList();
for (int i = 0; i < invoices.length; i++) {
  List items = invoices[i].getItems();
  for (Iterator iter = items.iterator(); iter.hasNext();) {
    LineItem lineItem = (LineItem) iter.next();
    if (lineItem.total() > 7000){
      result.add(lineItem.getProduct().getName());
    }
  }
}
```

Groovy:

```
invoices.items.grep { i -> i.total() > 7000 }.product.name
```

Builders for GUIs

```
import groovy.swing.SwingBuilder

swing = new SwingBuilder()
frame = swing.frame(title:'Demo') {
    menuBar {
        menu('File') {
            menuItem ('New', actionPerformed: { println "new" })
            menuItem 'Open'
        }
    }
    panel {
        label 'Label 1'
        textField(id:'text-box', 1333)
        comboBox(items:['one','two','three'])
    }
}
frame.pack()
frame.show()
```



Closure

XML parsing

```
xml = """
 <people>
  <person name="Rod">
    <pet name="Misha" age="9"/>
    <pet name="Bowie" age="3"/>
  </person>
  <person name="Eric">
    <pet name="Poe" age="5"/>
    <pet name="Doc" age="4"/>
  </person>
 </people>"""“

people = new groovy.util.XmlParser().parseText( xml )
println people.person.pet['@name']
```

Output

```
-> ['Misha','Bowie','Poe','Doc']
```

More Groovy concepts

- Access to the AST of closures
- Seamless integration with Java
- Many more

Concrete experiences with Groovy

- Then I wrote first scripts (for code analysis & to automate development tasks)
  - My intent was not clear to Groovy: „why does it think this is a string, I wanted it to be a number“ ;-). Due to weaker typing.
  - Builders: hard to understand error message when there is a problem
  - There are many more possibilities: danger of over-use of language features

- Later experiences
  - 2 Grails developers told me (independently), when I challenged the weaker typing of Groovy: „yes, I just write more tests“ . This is IMHO the wrong direction. => I should state my intent when I write my code (that’s when I know) and my intent should be where I declare things (that’s where I look for it).
  - Experience of „The compiler is our friend“: it should rather be stricter than in Java, not weaker: in Eclipse I sometimes think what is the compiler complaining about?! (then I get humble and happy it is paying attention) ;-)
  - Groovy is Java + many things => more complexity

A strength of Groovy: Grails

THE web framework of Groovy. Well done (great simplification, good scaffolding, big adoption in market)

BUT:

- A step back from the evolution: not UI component (=widget)-oriented
  - e.g. lower level of abstraction (basically on http/html-level)
  - e.g. not easy to provide own (non-trivial widgets (from the extjs js framework => no server-side widget abstraction))
- Only basic set of widgets integrated
- A typical pattern of Groovy: weakly typed (web state is just a Map with fields for web state)
- No conversation/ page state isolation (no browser-tab isolation)

Concrete experiences with Scala

Scala

- Looks very cleanly engineered, a “scalable language”
  - Very strong typing: the compiler is strong as “it should be”, more intelligent than in Java, it really helps the developer
  - Shorter language spec than Java
  - Much less boilerplate, closures
  - Matchers: programming involves a lot of detection of certain patterns and reactions to it
  - Great possibilities for internal DSLs (e.g. Actors (Erlang-like language for distributed and concurrent programming))
- I read the book “Programming Scala” by Odersky (first 50%, then selectively)
- Wow!

Scala examples: reduction of boilerplate

```
class Graduate(name: String , age: Int, degree: String)
      extends Person(name, age) {

   // any method can access "name",  "age", or "degree"

   def  this(name: String, age:Int ) {

       // call the "primary" constructor

      this(name, age, "Bachelor");

   }

}
```

The features of Groovy are (mostly) also available.

Scala: code parsing can become difficult for humans

Easy to understand:

- `val names = new Regex(pattern).findAllIn(path).toList`

But one can also say:

- `val names = new Regex(pattern) findAllIn path toList`

I found the following Scala program (a CSP DSL):

- `v(c) (c ? P | c ! Q)`

Exercise for you: what are the symbols here?

Source: http://blog.razie.com/2010/09/scala-workflow-engine-and-dsl.html

Scala: code parsing can become difficult for humans

One can define own control structures:

```
def whileAwesome(conditional: => Boolean)(f: => Unit) {
  if (conditional) {
    f
    whileAwesome(conditional)(f)
  }
}
```

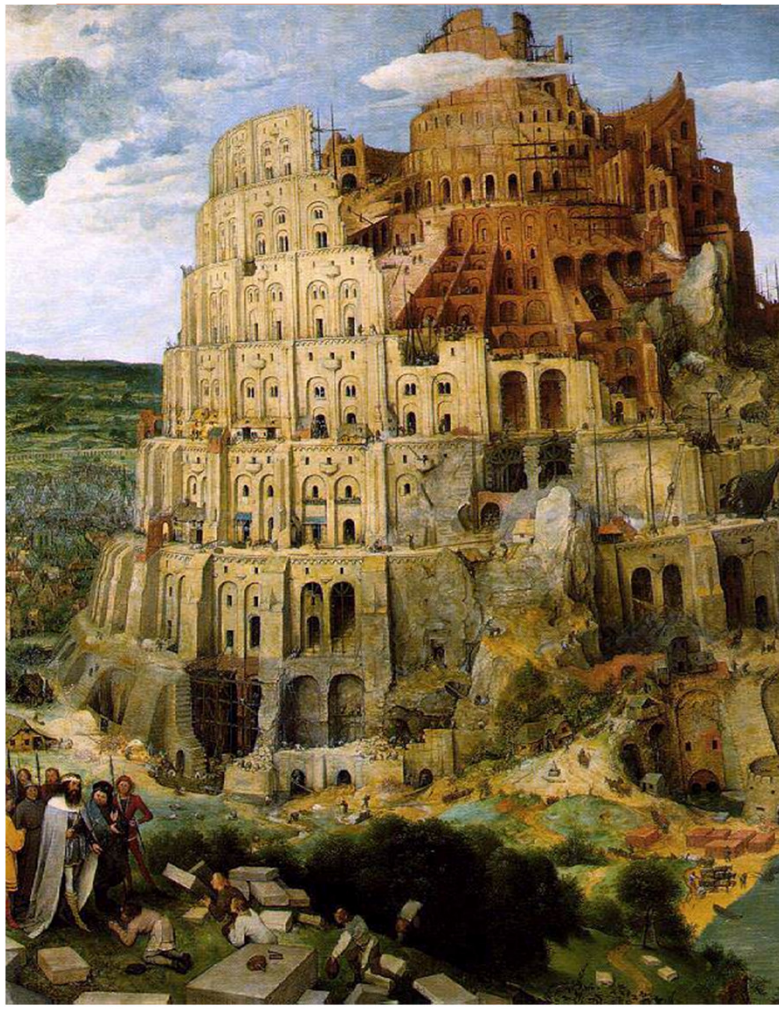
```
var count = 0

whileAwesome(count < 5) {
  println("still awesome")
  count += 1
}
```

Source: http://ofps3.vz.oreilly.com/titles/9780596155957/ch08.html

Concrete experiences with Scala

- Then I found some Scala programs and did not even know how to *parse* the programs
  - What is this token (a field, a method, an operation)?
  - How to figure out what the program does?
  - You could define basically any language in Scala (Visual Basic? APL? To fan's of exotic languages, this is your chance to use your language again ;-))
  - How to control (governance) what features are used in a project? How to avoid e.g. inappropriate DSLs?
- I actually asked Martin Odersky (creator of Scala) in November 2010 whether there was a way to constrain Scala. He did not know of one, but said it was an interesting question.
- I think Scala can work in a **closed and stable group** of very skilled programmers. Where you can e.g. spend 1-2 months to bring a new employee up to speed and have very close governance (education/ reviews) about the use of the language (but put this in your ROI!).

My current issues/ My wish list for Java and its environment



- Intro
- Concrete experiences with new JVM languages
- My current issues/ My wish list for Java and its environment
- Alternatives from different sources

My issues with Java

Complexity of the typically used technology stack/ frameworks

- Most clients I know (also ELCA) put a lot of effort to set up their own technology stack.
  There is no consensus what the “right” technology stack is. (Contrary to .NET¨; it’s a half full glass, we also have more evolution)
- Java EE has 5000+ pages of specifications (!), earlier “mistakes” partially still there (EJB entity beans). New “mistakes” still appear (JPA 2 query API).
- Spring download has 100+ (!) jar files included
- Classloader challenges/ parent-last is not standard => this reduces isolation
- Java EE 6: is there now (1.5 y after finalization) a certified container for production use?

Complexity of the language

- Genericity (FAQ of Angelika Langner is 300 pages long). Needs expert to understand fully.
- We need books like *Effective Java*
- There is the „Java Puzzlers“ book
  - Some people love to solve such puzzles: IMHO it shows flaws of the language
- If we are unlucky: complex solution for closures

My issues with Java (2)

Some boilerplate code necessary (bad for “signal to noise ratio”)/ heaviness

- Setter/ getter: a bit heavy: typically 8 LOC per field
- Basic types AND wrapper types
- Arrays AND lists
- Missing closures
- No automatic resource clean up pattern (fixed in JDK 7)
- No support for basic structures (maps, lists) in language

Expressivity could be improved

- Some code examples of e.g. Groovy: very readable with much less code
- Interfaces could be more expressive (to help the compiler/ developer)
  - List<Person> getMatchingPersons(Query q): how are 0 matching elements returned: „null“ or empty list or Exception?
  - Is a method thread safe?
  - Design by contract-like information?
  - Treatment of null?

My issues with Java (3)

Little means for language governance in the language

- Typically use of Checkstyle/ Findbugs/ PMD, many others
- Little limitation of feature use (e.g. avoid Reflection, Synchronization, …)
- Little dependency management (often delegated to Maven/ separate tools)

Internal DSLs are quite limited

- Potential of higher expressivity via special DSLs is harder to achieve (e.g. such as languages similar to Erlang / Scala Actors)
- BUT: as mentioned earlier, IMHO not a big issue (Creating a DSLs should be explicit)

My wish-list for a new language

A) Improvements on the signal-to-noise ratio

- Java is often a bit verbose/ Reduction of such boilerplate-code
- It should be evident for a reader what code does (less opportunity for puzzlers)
- Add stronger language support for frequently needed things (e.g. SQL, XML, Money, Maps)
- Simplification
  - E.g. Genericity

B) Governability

- Restricted set of features or ability to **externally control what features can be used** “some language features can be forbidden”
- Optimal support for early error reduction “power to the compiler & to tools”

C) Technology stacks/ frameworks for common tasks

- Maturity (UI frameworks are still evolving)
- Major simplification and standardization
  - Ideally „**1** set of standard technology stacks“
  - Alternatively an ecosystem of frameworks/ technology stacks

D) Maturity/ Adoption

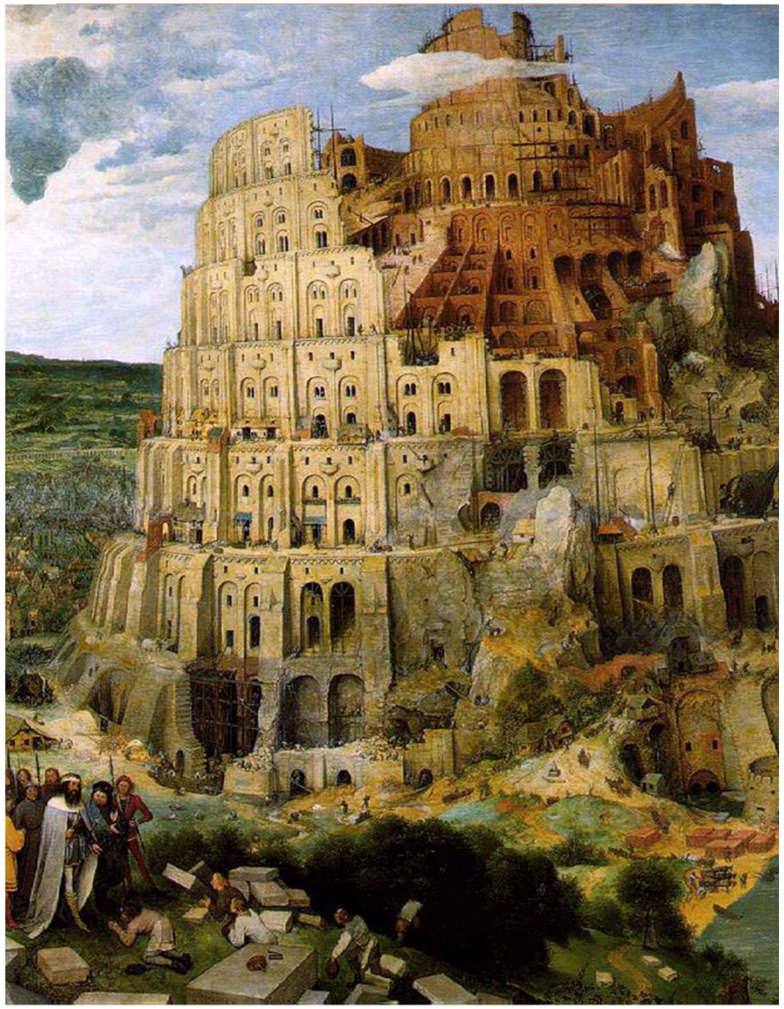
- To ensure know-how availability and support
  - Standards, “big” corporate backing helps, high bus factor
- Bootstrapping is a problem, but can e.g. be overcome via tactical use first (Java first came up in browsers)

Matching the new languages to my wish list

Language /Criteria	**Groovy**	**Scala**	**Clojure**
Better signal to noise	+better	+quite a bit better	-uncommon syntax, quite old & did not emerge
Governability	-less early error prevention 💣 -even more features	+more early error prevention -too complex 💣	-danger of macros 💣 -weakly typed
Frameworks	+Grails -BUT: low abstraction (on level of HTML)	-Lift (smaller than Grails, similar issue) +Actors (Erlang-like)	+Transact. Memory (STM)
Adoption/ Maturity	+not bad (Grails!)	+clean spec (smaller than Java) -not bad, still small 💣	-even smaller 💣

💣 An argument „killing“ the language for strategic systems today

Alternatives from different sources



- Intro
- Concrete experiences with new JVM languages
- My current issues/ My wish list for Java and its environment
- ▶ Alternatives from different sources

Alternatives from different sources

I think these languages are great innovations, but not the final answers for today. We (the Java Community) have a great innovation culture.

Here are some IMHO interesting innovations that improve the language question, e.g.

- (These JVM languages)
- DSL/ MDSE-Toolkits (Xtext, MPS, …)
- Governance tools and processes (Checkstyle, Findbugs, PMD, Maven, custom guidelines, reviews, …)
- Standard technology stacks (Seam, AppFuse, EL4J, …)

- Other new languages in the Java space (Fantome, Gosu, …)
- Small Java Language improvements (Groovy++ (more typed, less dynamic, constrains use of some features), LambdaJ, Lombok (see next slides),)

Lombok

```
// the next annotation means: I want all setters and getters,
//  a default constructor, a toString, hashmap and equals method
//   (the ~100 lines of boring code one typically generates with the IDE)
@lombok.Data
public class DataExample {
   private final String name;
   private int age;
   private double score;
   private String[] tags;
   public void setTags(String[] tags) {
      this.tags = tags;
      notifyChange();  // this is special (it varies from the standard setter)
   }
   protected void notifyChange() {
      // ...
   }
}
```

My first reaction: Yet another generator.

But then: Autocompletion of generated methods works in Eclipse & it's a plugin for javac (so works everywhere). Delombok exists (converts to plain Java). (Implementation is a hack).

LambdaJ: clever use of Java

```
List<Person> personInFamily = // init to persons of the family

forEach(personInFamily).setLastName("Fusco");
```

```
List<Car> cars = // init to some cars

List<Double> speeds = extract(cars, on(Car.class).getSpeed());
```

```
List<Sale> sales = // get some Sales

List<Person> buyersSortedByAges = with(sales)
        .retain(having(on(Sale.class).getValue(), greaterThan(50000)))
        .extract(on(Sale.class).getBuyer())
        .sort(on(Person.class).getAge());
```

```
Closure println = closure(); { of(System.out).println(var(String.class)); }

println.apply("one");

println.each("one", "two", "three");
```

A bit limited,
unsure whether such closures are
a good idea.

Exercise for you: find out how it's done

These languages are great innovations, not final solutions

Some even newer languages coming up

- Groovy++ (might also be named Groovy--, is a modification of Groovy): more typed, less dynamic, constrains use of some features
- Fantome (much simplified Java): Java with less boilerplate, simplified libraries
- Gosu (http://gosu-lang.org/): simplified Java (Generics) with closures, interceptors, enhancements of existing code elements

What I would like: a "constrained Scala"

a) fewer features
b) allows limiting its possibilities/ features (for easier governability)
c) a standardized technology stack, with state of art concepts

More established languages are at the JVM (jpython, jruby, …). Subjectively: I do not see a lot of momentum.

Today's Java is a "good enough" language, I don't propose to switch today. Toughest challenges are on level of technology stacks.

For tactical things, we are selectively using Groovy/ Grails.

The Outlook is bright

Great innovation, in many domains:

- Languages
- Technology stacks
- Language Workbenches/ DSL tools/ MDSE-Tools

Combinations of features can happen quickly

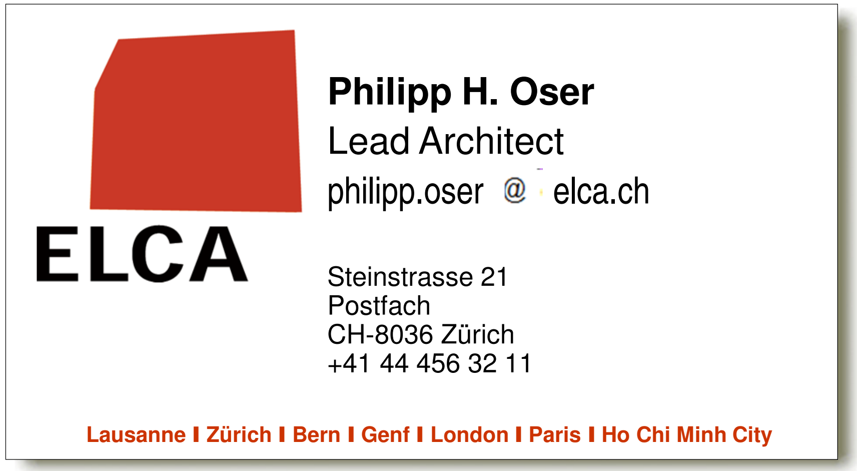
Darwinism and international sharing of OSS/ research developments (brutal, makes mistakes, eventually adapted ones win)

For strategic systems, I propose to stick today with Java & careful selection of technology stack.

Q&A / Discussion

Thank you for your attention

For further information
please contact:



ELCA

Philipp H. Oser
Lead Architect
philipp.oser @ elca.ch

Steinstrasse 21
Postfach
CH-8036 Zürich
+41 44 456 32 11

Lausanne I Zürich I Bern I Genf I London I Paris I Ho Chi Minh City