

***Static Typing or Dynamic Typing:
A Real Issue or a Simple Case of
Tribalism***

Dr Russel Winder

Concertant LLP

russel.winder@concertant.com

Aims and Objectives

Survey some of the static typing vs. dynamic typing issues.

Convince people that the static vs. dynamic typing issue is an opportunity not a problem.

Show how symbiosis can be achieved easily, at least on the Java Platform.

Foment thinking and debate.

Exception handling and dynamic binding are a core part of the structure of this session.

Subliminal (!) Advertising

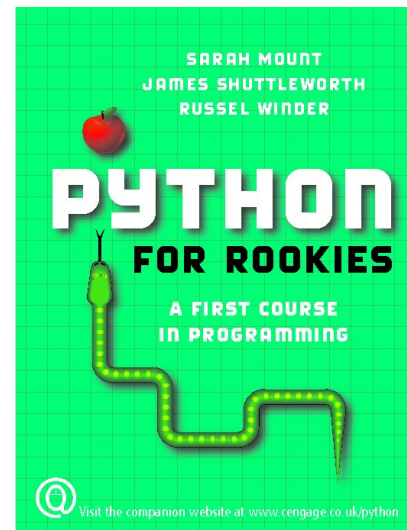
Python for Rookies

Sarah Mount, James Shuttleworth and

Russel Winder

Thomson Learning

Now Cengage Learning.



*Draft cover, things
may change.
Python is, after all,
a dynamic language.*

Learners of Python need this book.



The Players in the Game

Statically typed:

C	Ada
C++	Haskell
Java	Fortran
Scala	OCaml
C#	Eiffel

Dynamically typed:

Smalltalk
Python
Ruby
Groovy
Lua

Perl

Typing the Typing

The major keywords are:

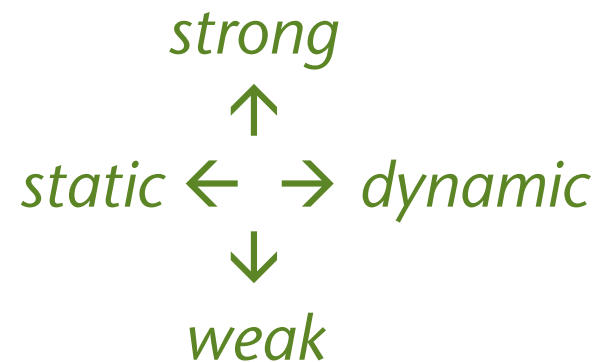
Strong typing.

Weak typing.

Static typing.

Dynamic typing.

Two orthogonal dimensions:



There is a third dimension:

manifest ← → *inferred*

The Strong Typing Imperative

Software engineering dogma demands abstract data types and strong (\equiv static) typing.

Java, Ada, Eiffel, Scala, Haskell are paradigms of languages meeting the requirements of software engineering doctrine.

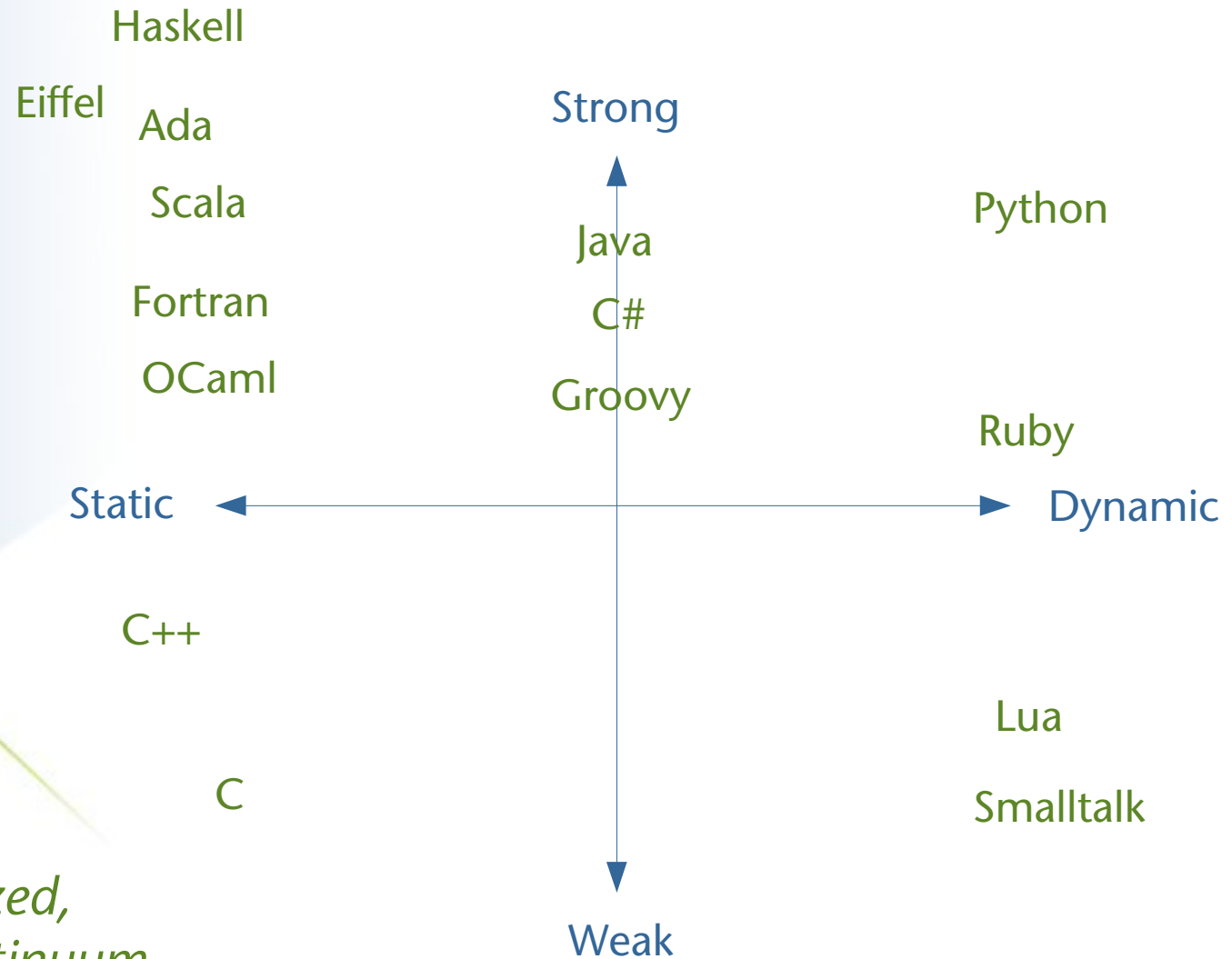
C and C++ are allowed, but are they compatible with doctrine?

Imperative languages are not mandated but:

Object-orientation has hegemony.

Functional languages have been ignored.

Placing Languages in the Space



*Scales are quantized,
there is not a continuum.*

Benefits of Static Type Checking

Type checking as:

Pre-execution testing of code.

A tool for not getting it wrong.

The Problem:

*“My program compiles
so it is correct.”*

Managing Expectations

Unit tests, integration tests, acceptance tests.

Test-driven development – incorporate unit testing as an integral part of the development process.

The testing-oriented philosophy is now part of the dogma and doctrine.

Emphasis on testing is independent of static, dynamic, strong, or weak typing.

Interfaces

Java really popularized *interfaces*, though C++, Eiffel etc. have them.

Interfaces reify the public interface of an object.

Interfaces define types as well as classes.

```
public interface Blah {  
    int doDah ( String s );  
}
```

```
class Blah {  
    int doDah ( std::string ) = 0 ;  
};
```

Programming to Interfaces

Classes are for creating objects, interfaces are for defining the public interfaces. cf. *List* in Java.

List<T> is the interface defining “listness”.

ArrayList<T>, *LinkedList*<T> provide representations of *List*<T> with different properties.

The Importance of Being Typed

Code uses types not classes.

Correct use of types checked at compile time.

It is not the class that is crucial for execution, it is the public interface exposed by an object.

Polymorphism

Generic programming and polymorphic functions increasingly seen as the way forward.

Programming to interfaces supports polymorphism.

In statically typed languages, there has to be an underlying type relationship to achieve polymorphism: *inheritance* or *conformance* required.

Do we agree that programming to interfaces and polymorphism are good?

Being Manifestly Typed

C++, Java, Ada, Eiffel, etc. require explicit naming of types in all places – duplication of typing (on the keyboard).

For example in Java:

```
final List<String> l = new ArrayList<String> ( );
```

Inferring the Types

Type inference systems have been around for ages:

Functional languages such as Haskell infer the types:

```
fibonacci 0 = 0
```

```
fibonacci 1 = 1
```

```
fibonacci n = fibonacci ( n - 1 ) + fibonacci ( n - 2 )
```

Scala also has type inference:

```
val l = new Array[T] ( )
```

The type structure and checking are the same but the compiler does the work.

Doing the Inference Thing

Scala and Haskell have type inferencing.

C++ will get some aspects of type inferencing in C++0x.

There appear no plans for type inferencing in Java or C#. Why not?

Are Java and C# wrong to not use type inferencing?

***Strong* \equiv *Static* ?**

The strongly typed languages such as C++, Java, C#, Scala, Haskell are compiled languages, i.e. statically typed.

Are there any dynamic languages that are strongly typed?

Python, Ruby, Groovy

Duck now...

Who Put the Duck in Typing

Inductive reasoning.

If it looks like a duck, walks like a duck, swims like a duck, and quacks like a duck, then it must be a duck.



Not Inferring the Types

Take a simple step:

Enforce programming to interfaces.

Do the check at run time instead of compile time.

Are there adherents?

Smalltalk

Python

Ruby

Groovy

If an object responds to the method, it must be of the right type.

Is this the ultimate expression of polymorphism?

The Role of Testing

Testing is critical for programming.

Programming in C++, Java, Haskell, etc. requires test suites using test frameworks:

Aeryn, TestNG, HUnit.

What is good for statically typed languages is good for dynamically typed languages.

Programming in Smalltalk, Python, Ruby, Groovy, etc. requires test suites using test frameworks:

sUnit, PyUnit, Test::Unit, Test'N'Groove.

Dynamic Languages and Testing

It is true that dynamic languages rely more on unit testing than static languages do.

Is this a problem?

Is Static Typing Irrelevant?

Dynamically typed languages seem to have the same aims and goals as statically typed languages **and** there is no separate compilation phase.

What do statically typed language bring that is actually needed?

Having it Both Ways

Groovy is a dynamic programming language, with strong but dynamic typing.

Groovy is compiled before executed – compiled to JVM bytecodes.

Groovy works symbiotically with Java uses the Java data types – the object forms by default, can also use the primitives when calling Java code.

Systems that are part Groovy and part Java are natural and work very well.

Static where needed, dynamic where wanted.

Summary

Dynamic typing is a technique to use when appropriate.

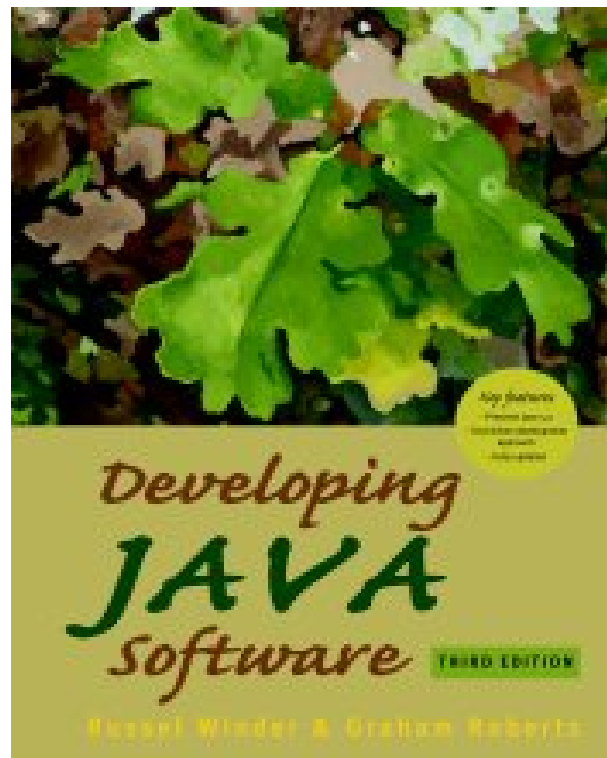
Mixed statically typed and dynamically typed systems have the best of all world.

Java + Groovy make a great team.

Closures change the way you program.

And finally...

And Finally...



*Java learners **need** this.*