The Ruby Programming Language...
it's really fun and it feels good!

John Pagonis
Pragmaticomm Limited,
Athens Ruby Meetup III, December 2009

Menu

- Why I got into Ruby?
- In search of a better way to code
- Ruby in twenty minutes (or thereabouts:-)
- Why we should all have a look at it?

Before we start… a word from the wise

A reminder from Fred. P. Brooks ‘’ No Silver Bullet - essence and accidents of software engineering’’,1986

There is NO silver bullet!

Language disclaimer

- I am NOT a language bigot!
- I am a C++ programmer (actually a Symbian OS C++ programmer) and I like C++ for getting to the metal.
- I have programmed enough Java.
- I've also taught ‘C’ to students and professionals because it is so basic (and blunt:-) and because it helps explain to students how things work.
- I am also interested in Virtual Machines, Objective-C, Smalltalk and other languages and runtimes.
- ..even ISO C0x C++ sometimes :-)

My experience with Ruby (mostly from inside the VM:-)

- First got involved in 'Skunkworks' while at Symbian :-)
- Ported with Pragmaticomm the Ruby 1.9.0.0 and Ruby 1.9.1p1 VM and some extensions to Symbian OS v9.1 (for Nokia's Symbian Research dept.)
- I've used it for mobile programming, text filtering, classification and Genetic Algorithm related work

Lately in my life (does it look familiar to you?)

- There is a a lot of stuff I need to automate
- There are a lot of stuff I want to develop
- There are a lot of platforms I need to be using
- I need to be more efficient when coding.
- I have realised that my time and memory is **MUCH** more expensive than my CPUs' time and RAM.
- I haven't been getting any younger
- I haven't been getting much smarter :-)
- I think faster than I code!
- I am running out of time….

Consequently

Life is too short, to not have fun…

I have to cheat!

There must be a better way to program… there must be!!

There must be a better way to program… there must be!!

..to clarify that

There must be a much better way than C,C++ (and maybe Objective-C) to code software that

- … doesn't need to talk to the metal directly
- … doesn't need to be as efficient at runtime
- … allows me to solve problems fast
- … lets me change my mind and accommodate change
- … is fun and feels good
- … is not ugly and doesn't get in the way
- … has a lot of frameworks to get the job done
- … I can use in many domains and on many platforms
- … supports open-world system reuse

In my time...

- I've seen code, a lot of code, ugly code
- I've seen developers struggling to write Symbian OS C++ apps (among other things)
- I've seen people spending a lot of time developing code that will not work (on mobiles) correctly
- I see enterprises wanting apps yesterday
- I see people not having enough time to code
- I see enthusiasts and students not learning or wanting to learn C++
- I see re-use going down the drain

I felt so weak

So I went out to find a silver bullet!!!

Then I remembered that there isn't one and that F.P. Brooks said in 1975...

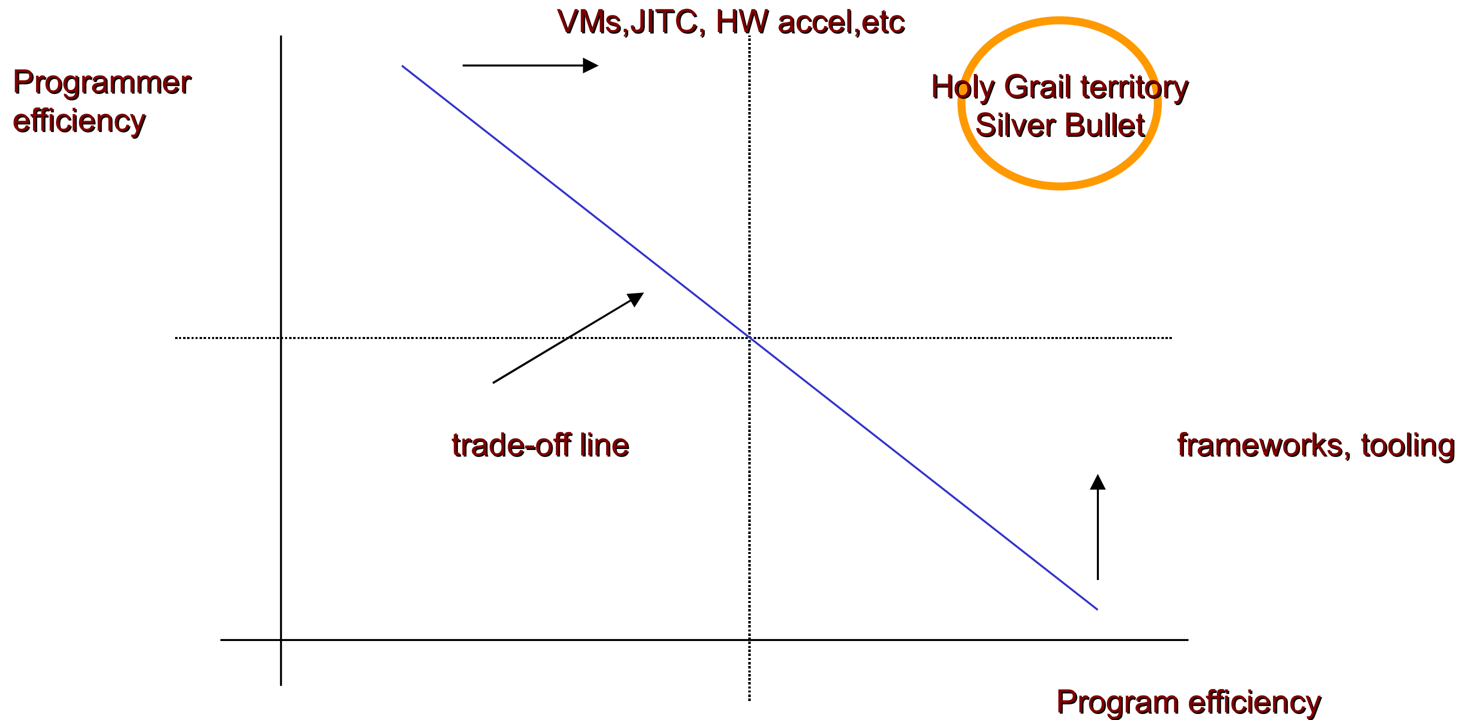
…that compilers are three times as hard to write as application programs and systems programs are three times as hard to write as compilers (may not be true for C++:-)

… that the use of a suitable high-level language may dramatically improve programmer productivity

So lets keep C++ for operating systems coding..

So what I need is:

A fun language for non system level stuff which is mainly programmer efficient as opposed to program efficient and which is also general purpose (that excludes Fortran I guess:-).



Software is all about people and economics

- The economics have changed
  - Now it pays even more to have **programmer efficiency**
  - More functionality yields higher system complexity that leads to more entropy which leads to higher cost (because you need more energy:-)
  - Shorter time from idea to production is vital in order to compete
- People have changed
  - People **care less** about how things work (hence know less)
  - People have **more things to do** and much more information to assimilate and keep in their heads
- The fewer people you have, the better is the quality of the code and cheaper is the project at the end.... but to be on-time they need to be very productive!

What if there was a silver bullet?

- How much more money/time would I save if I could develop things **N** times faster?
- How much more competitive would my organisation be if they could deliver **N** times faster?
- What if the claims which Smalltalk makes that you get 10 times the programmer efficiency was true?
- What if there is something out there today that could give me some of that **N** for the problems that I want to solve?

So I started the search for my silver bullet:

- ***Objective-C*** is very clever, simple, cute and powerful, but not ubiquitous enough, although it is excellent for open-world compiled components. (Symbian OS frameworks almost got written in it).
- ***Java*** is 'Objective-C in bondage' due to static typing and gets in my way, (many feel that it has also become the asylum of the incompetent C++ coder). Plus, I can deal with memory management myself. But it has a lot of frameworks, good VMs and tooling (and it pays well:-).
- ***Perl*** is powerful for many things, but ugly as hell.
- ***Python***, is interesting and powerful with good VM support, has a lot of frameworks and tool support, but it carries some baggage that I don't enjoy, will get better though in Python 3000.
- ***Smalltalk*** is my favourite, but is really secluded :-(
- ***Lisp*** is assembler for the brain and quite difficult and niche for me

Of course the answer is

multi-language and multi-paradigm programming....

(but this is another topic for another presentation :-)

Then I came across Ruby

Actually it was at a Python seminar at ACCU 2006 !!!

- Ruby was first released to the public in 1995 by Matz (Yukihiro Matsumoto)
- Ruby was more popular than Python in Japan already by 2000 and allegedly almost as popular as Perl by 2004
- Ruby is a play on the word Perl :-)
- Ruby was designed to be beautiful, fun and ‘’stay out of the way’’
- It comes with a standard documentation system ‘RDoc’, a standard packaging system ‘RubyGems’, an interactive shell ‘irb’, the Ruby VM, a hell of a lot of frameworks and a very useful standard library.
- It also has an efficient official VM these days (in Ruby 1.9.x)!

Ruby in 20 minutes (mostly ripped of the Ruby website)

- Ruby is a genuine object-oriented language
- The result of every expression is an object
- Objects are garbage collected
- Like with Smalltalk and Objective-C, objects respond to messages
- Such messages contain a method’s name together with the parameters that the method may need
- Ruby is a single inheritance language
- Classes can include the functionality of any number of ‘mixins’
- Ruby is a dynamic (late-binding) language
- There is access control in Ruby
- You can use curly braces {} if you want to :-) (because it is important:-)

Ruby execution

- Ruby executes usually under the support of a 'runtime' which is usually a bytecode Virtual Machine or interpreter.
- Ruby runtimes can be as diverse as JavaScript interpreters to compilers that output ARM binary code.
- Ruby can be embedded inside another application (such as MacRuby Mac OS X apps and even Symbian OS C++ apps).
- The official Ruby 1.9.0 VM written by Koichi Sasada is 330KLOC written in K&R style 'C' and can easily be extended by native extensions. It assumes Posix APIs and applies native multithreading. It is also a parser and a bytecode compiler (but there is no JIT compilation yet).
- In-fact the core VM plus the OpenSSL and TCP/IP extension is only 140KLOC
- Compared to the 1.8 (MRI) interpreter the Ruby 1.9 VM executes up to 20 times faster in some cases!

The Ruby world

- VMs: JRuby JVM, .NET (DLR), MacRuby, Smalltalk MagLev, SmallRuby, CRuby, MRI, Rubinius, BlueRuby, XRuby
- IRB, you can try your idea on the command line
- Web frameworks and containers: Rails, Ramaze, Webrick, Rack, Rango...
- GUIs: Shoes, FXRuby, QT/Ruby, MonkeyBars
- Compilers: RubyScript2Exe, Ocelot, Atomic-Ruby
- Packaging: Ruby Gems, Gem Bundler
- IDEs: FreeRIDE, Aptana, Netbeans, RadRails, RubyMine, Eclipse, Ruby In Steel (VS), XCode...
- Application frameworks: ....

Reuse (a.k.a better economics)

With Ruby and the plethora of VMs you can reuse your existing components

- Reuse of Java components (GUIs as well) through JRuby
- Reuse of Smalltalk components through MagLev and SmallRuby
- Reuse of .NET components with IronRuby and Ruby.NET
- Reuse of Objective-C components and Mac OS X GUI through MacRuby
- Reuse of 'C' components and native OS APIs through CRuby extensions
- You can also reuse Ruby code from C/C++/Objective-C

Hello Ruby (the 2nd most important slide)

```
irb(main):001:0> puts 'Hello Ruby world'
Hello Ruby world
=>nil
```

This is more interesting though:

```
10.times {p "better"}
```

The **message** 'times' is sent to the object '10', with the code 'p "better"' as **payload**, packaged in a block denoted by '{}'

and this

```
"Hello Ruby world".length        -→ 16
```


John Pagonis, Pragmaticomm Limited, 2009

Duck typing

If it walks like a duck, talks like a duck and looks like a duck…well it must be a duck

...basically it is about **late-binding**, where the implementation of a request is determined at run time dynamically according to the message and receiver involved

...so if an object responds to some **message protocol** we don't care what its type really is, only that it adheres to it

Defining a method

```
def say_hi
   puts “hi”
end
```

Is called by:

`say_hi` Or `say_hi()`

```
...
def say_hi(name)
   puts “hi #{name}”
end
...
def say_hi(name = “there”)
   puts “hi #{name}”
end
```

Defining a class

```
class Greeter
    def initialize(name = “world”)
        @name = name
    end
    def say_hi
        puts “Hi #{name.capitalize}”
    end
end

g = Greeter.new(“john”)
```

Note the coding naming convention here!

Under the object's skin

```
irb(main):010:0> g.@name
SyntaxError: compile error
(irb):52: syntax error
         from (irb):52
```

Let's see what is inside the object:

```
irb(main):039:0> Greeter.instance_methods
=> ["method", "send", "object_id", "singleton_methods",
    "__send__", "equal?", "taint", "frozen?",
    "instance_variable_get", "kind_of?", "to_a",
    "instance_eval", "type", "protected_methods", "extend",
    "eql?", "display", "instance_variable_set", "hash",
    "is_a?", "to_s", "class", "tainted?", "private_methods",
    "untaint", "say_hi", "id", "inspect", "==", "===",
    "clone", "public_methods", "respond_to?", "freeze",
    "__id__", "=~", "methods", "nil?", "dup",
    "instance_variables", "instance_of?"]
```

That was… a lot of methods!

- `Greeter`'s ancestor is `Object`, where it got all those methods

So let's get only the ones we defined then:

```
irb(main):040:0> Greeter.instance_methods(false)
=>["say_hi"]
```

And even check them out

```
irb(main):041:0> g.respond_to?("name")
=> false
irb(main):042:0> g.respond_to?("say_hi")
=> true
irb(main):043:0> g.respond_to?("to_s")
=> true
```


John Pagonis, Pragmaticomm Limited, 2009

Altering Classes—It's Never Too Late

- In Ruby, you can open a class up again and modify it.
- The changes will be present in any new objects that you create and even in existing objects of that class.

Now consider how powerful this is in the context of C++ binary compatibility and over the air upgrades or of a 24/7 system that you can debug and alter when it is live !!

You can even override methods implemented natively (in 'C') inside the VM!

Opening up a class and adding an accessor

```
class Greeter
    attr_accessor :name
End


irb(main):048:0> g.respond_to?("name")
=> true
irb(main):049:0> g.respond_to?("name=")
=> true
```

Using the `attr_accessor` defined two new methods (as opposed to doing it manually) for the existing `Greeter` class which became available to the instantiated `g` object! The new methods know how to deal with the instance `@name` variable.

You cannot access `@name` directly from neither the same 'package' a la Java nor from another object instance of the same class a la C++!

John Pagonis, Pragmaticomm Limited, 2009

Let us see some more tricks

..doing more

```
def say_hi
    if @names.nil?
      puts "..."
    elsif @names.respond_to?("each")
      # @names is a list of some kind, iterate!
      @names.each do |name|
        puts "Hello #{name}!"
      end
    else
      puts "Hello #{@names}!"
    end
end
```

Iteration, using ... iterators

```
...
  @names.each do |name|
    puts "Hello #{name}!"
  end
```

If the object pointed by @names responds to the message ‘each’ then you can iterate over its elements… it must be some kind of list!

‘each’ is a **message** that carries a **block** of code, it then executes that block of code for every element in a list. The bit between ‘do’ and ‘end’ is just such a block. A block is like an anonymous function or ‘lambda’ (for Lispians). The variable between the pipe characters is the parameter for this block

More iteration

```
for i in 1..100
    print "Now at #{i}. Restart? "
    retry if gets =~ /^y/i
end

Now at 1. Restart? n
Now at 2. Restart? y
Now at 1. Restart? n
...
0.upto(10) do |x|
    print x, " "            # print from 0 to 10 inclusive
end
0.step(10,2) {|x| print x, " "}   # 0,2,4,6,8,10
```

…and there is much more

Blocks

- Blocks are chunks of code (whoa that's new …not)
- Blocks play well with iterators
- Iterators are methods that invoke a block of code repeatedly (usually for each element of a collection)
- A block may appear only in the source adjacent to a method call
- The code in the block is not executed at the time it is encountered
- Ruby remembers the context in which the block appears and then enters the method

Blocks, where the magic starts…

- Within a method the block may be invoked using the `yield` statement (Python 3000 will also have one)
- Whenever a `yield` is executed it invokes the block
- You can pass parameters to them and receive values from them

e.g.,

```
def two_times
    if block_given?
        yield
        yield
    end
end
two_times {puts "Hello"}
```

Containers

Array

```
myArray = ['a','1',some_object, 3.14]  #creation
myArray[3]  →  3.14                     #indexing
myArray[2]= "c"  →  "c"                 #assignment
myArray  → 'a','1','c',3.14
myArray[1..3]  →  '1','c',3.14          #range
myArray.length →  4
[1,2,6,4].sort.reverse  →  6,4,2,1
otherArray = myArray.dup                #copy
```

Hash

`h = {"key1",1, "key2","value"}` or `h = {"key1" => 1, "key2" => "value"}`

```
h["key1"] → 1
h.has_key?("key1") →  true
hashed_array = Hash[*myArray.flatten]
```

Conditionals

```
...
kind = case year
       when 1850..1889 then "Blues"
       when 1890..1909 then "Ragtime"
       when 1910..1929 then "New Orleans Jazz"
       when 1930..1939 then "Swing"
       when 1940..1950 then "Bebop"
       else "Jazz"
       end
```

Exceptions handling

- There is an `Exception` class and its children
- Every `Exception`, has a stack backtrace and a string
- Exception handling is enclosed in a `begin`/`end` block
- `rescue` tells Ruby which exceptions to handle
- The `ensure` clause contains code that will always be executed
- There is an `else` clause as well
- And there is a way to `retry` !!!

John Pagonis, Pragmaticomm Limited, 2009

Handling exceptions

```
f = File.open(“thefile”)
begin
    # ... Process
rescue IOError => exc
    # ... Some handling and change of state for retry
    retry
rescue SecurityError
    # ...
else
    puts “No errors,well done!”
ensure
    f.close unless f.nil?
end
```

So `retry` takes us back at the beginning of the block!!

John Pagonis, Pragmaticomm Limited, 2009

Raising exceptions

```
raise

raise “message from the deep”

raise SomeException, “Message attached”, caller
```

`Kernel.raise` is the actual call and `Kernel.caller` is the method to get the stack trace

John Pagonis, Pragmaticomm Limited, 2009

Mixins

can be considered as partially completed classes used to introduce a protocol to a class

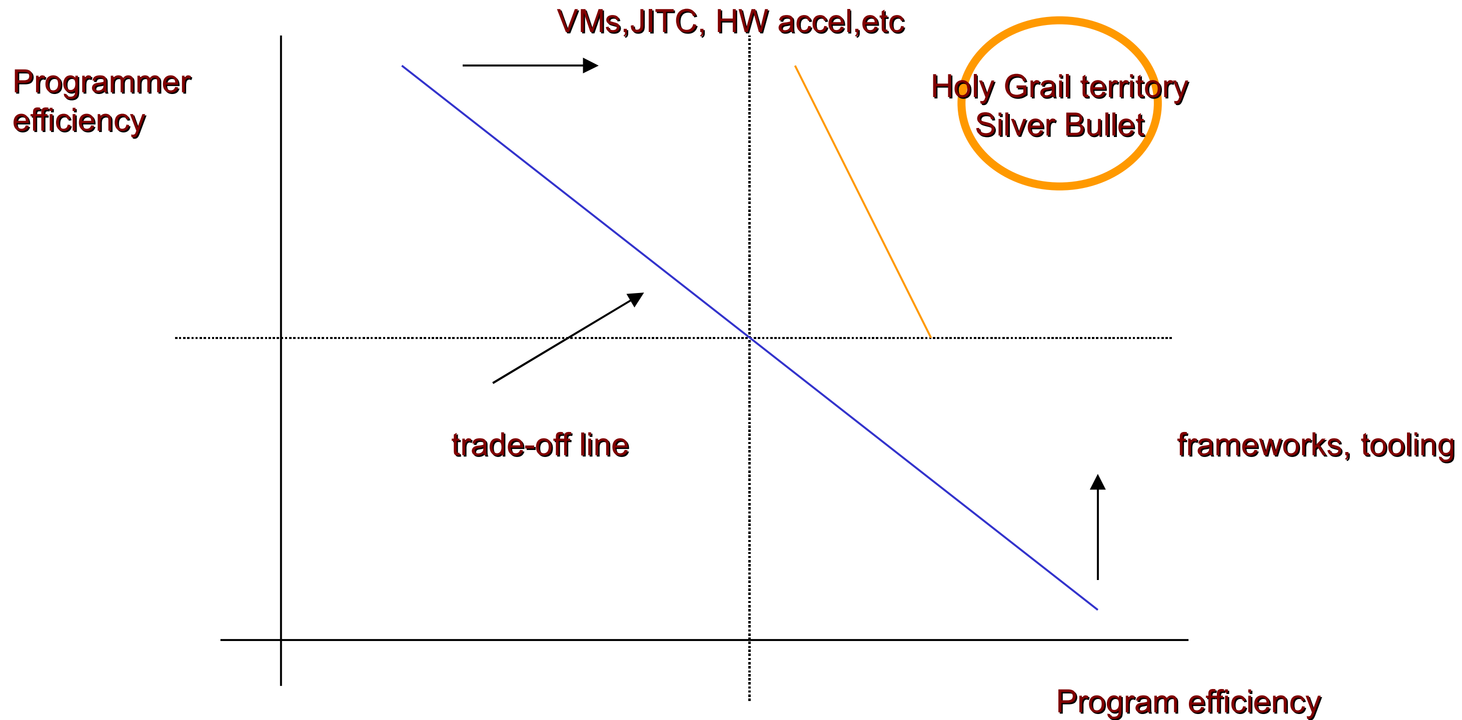
```
module GoodStuff
    def goodness
        ...
    end
end

class MyClass < ParentClass
  include GoodStuff
    def do_good
        goodness
    end
end
gs = MyClass.new()
gs.goodness
gs.do_good
```

And much much more

- Modules and mixins
- `catch` and `throw`
- Parallel assignment
- Lambda
- Hash tables, lists, ranges, arrays, slicing etc
- C bindings
- DRb
- Regular expressions
- Closures
- More conditionals
- More iteration
- Message interception
- Dynamic evaluation of code at runtime
- Profiler
- Debugger
- And much more

So what we need is to move over the trade-off line

In many cases absolute program efficiency becomes a moot point thanks to Moore’s law, while programmer efficiency has become an acute problem. Ruby can help today!



What is happening out there

- Ruby is one of the most discussed and hyped new programming ‘’thing’’
- Sun/IBM are adding JSR 292 to the JVM
- Sun hired the JRuby team .. but now they went away due to the Oracle acquisition
- The (Perl) Parrot VM is designed to accommodate Python and Ruby
- Google's Python embrace has proven (again) that dynamic languages are production ready, popular and powerful (it took us some 30+ years but anyway)
- Python and now Ruby is becoming popular at universities and the industry (through domain specific languages as well)
- Java is getting somehow tired (but is here to stay for good)
- You don’t need permission or voodoo to hack and extend the Ruby VM
- There is a plethora of Ruby VMs and runtimes (CRuby, Rubinius, MacRuby, MagLev, IronRuby, JRuby, etc.)
- Frameworks like Rails are becoming increasingly popular in the industry as well

Ruby projects that I want to see happening

- Other languages running natively on the Ruby 1.9 VM's bytecode and vice versa.
- A Ruby friendly JSR292 enabled JVM.
- VM optimisations such as a JIT Compiler, pre-compilation and independence from the GVL (Giant(global) VM Lock).
- Good support for multi-core CPU/GPGPU programming
- A pure Ruby GUI for desktop and small devices
- A Smalltalk-like IDE and environment (but NOT like Eclipse)
- Instrumentation and tool support for live and interactive on-target development and debugging
- A framework for mobile applications

To probe further

- www.ruby-lang.org
- www.ruby-doc.org
- www.rubyinside.com
- “Programming Ruby 1.9” by Dave Thomas, 2009 (also online is the 2nd ed)
- “The Mythical Man-month” by F.P. Brooks, 1975
- “Object-Oriented Programming: An evolutionary approach” by Brad. J. Cox and A. J. Novobilski, 1986
- “Smalltalk-80: The interactive programming environment” by Adele Goldberg, 1984

John Pagonis, Pragmaticomm Limited, 2009