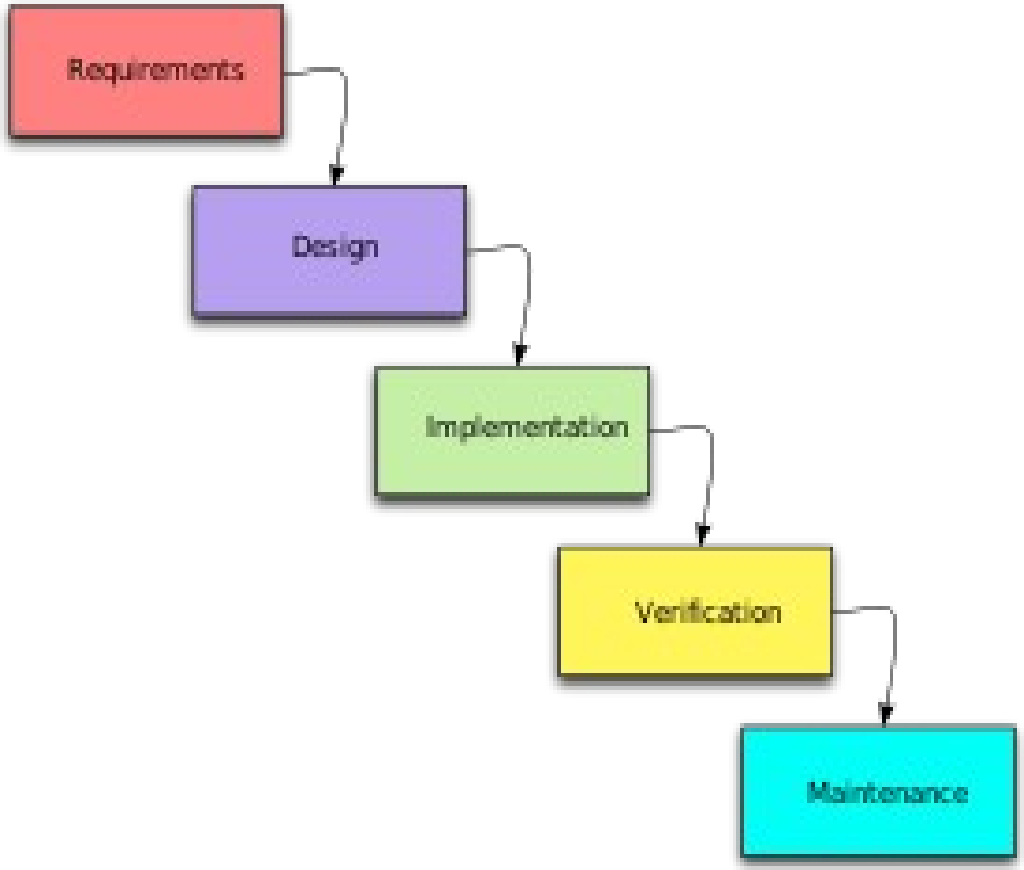
Agile development with Ruby

Eng. Khaled Al Habache

What's Agile development?

- Do you remember the old waterfall model?
- The waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design (validation), Construction, Testing and maintenance.

What's Agile development?

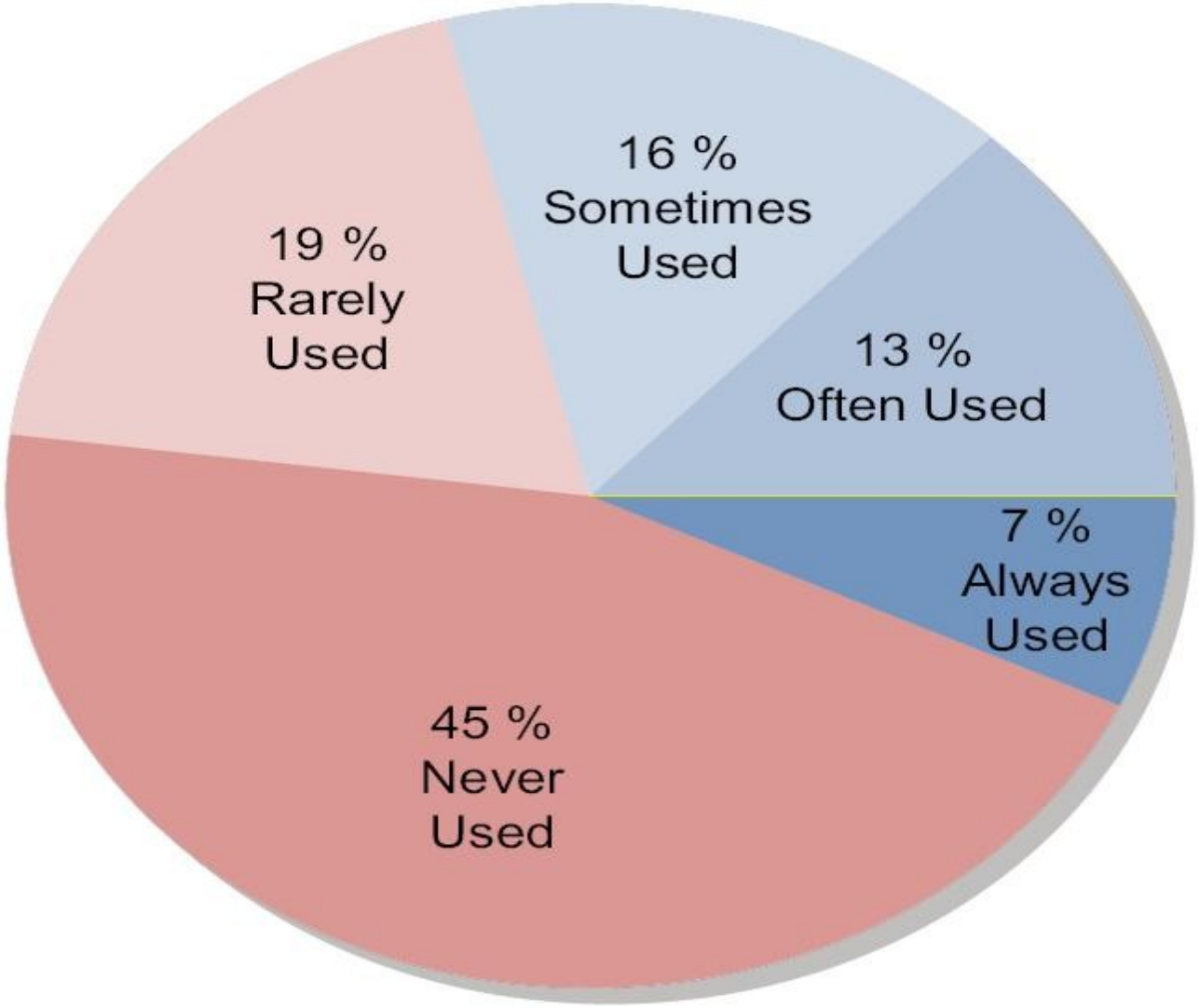


What's Agile development?

- Wide criticism:

1. It is impossible, for any non-trivial project, to get one phase of a software product's lifecycle perfected before moving on to the next phases and learning from them.
2. Clients may not be aware of exactly what requirements they want before they see a working prototype and can comment upon it; they may change their requirements constantly, and program designers and implementers may have little control over this.

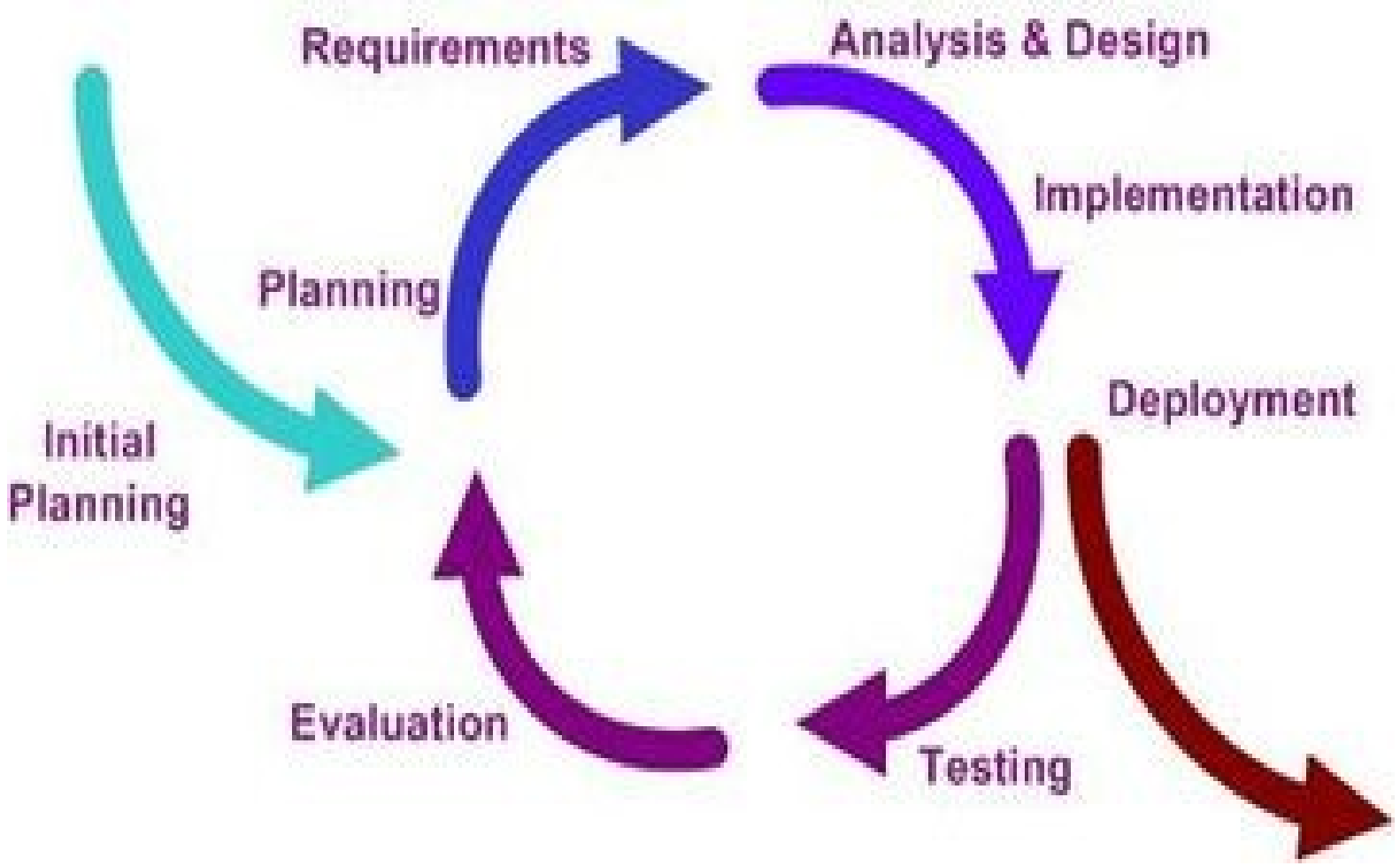
What's Agile development?



What's Agile development?

- Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

What's Agile development?



What's Agile development?

- Agile methods break tasks into small increments with minimal planning, and do not directly involve long-term planning.
- Iterations are short time frames that typically last from one to four weeks.
- Each iteration involves a team working through a full software development cycle including planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- The goal of an iteration is to have an available release (with minimal bugs) at the end of each iteration.

Why Agile development?

- Minimize overall risk by adapting to changes quickly.
- The ultimate goal is to REDUCE COST.

Agile development: Strategies to reduce cost

Maintain the Theory of the Code:
Great teams have a shared understanding of how the software system represents the world.

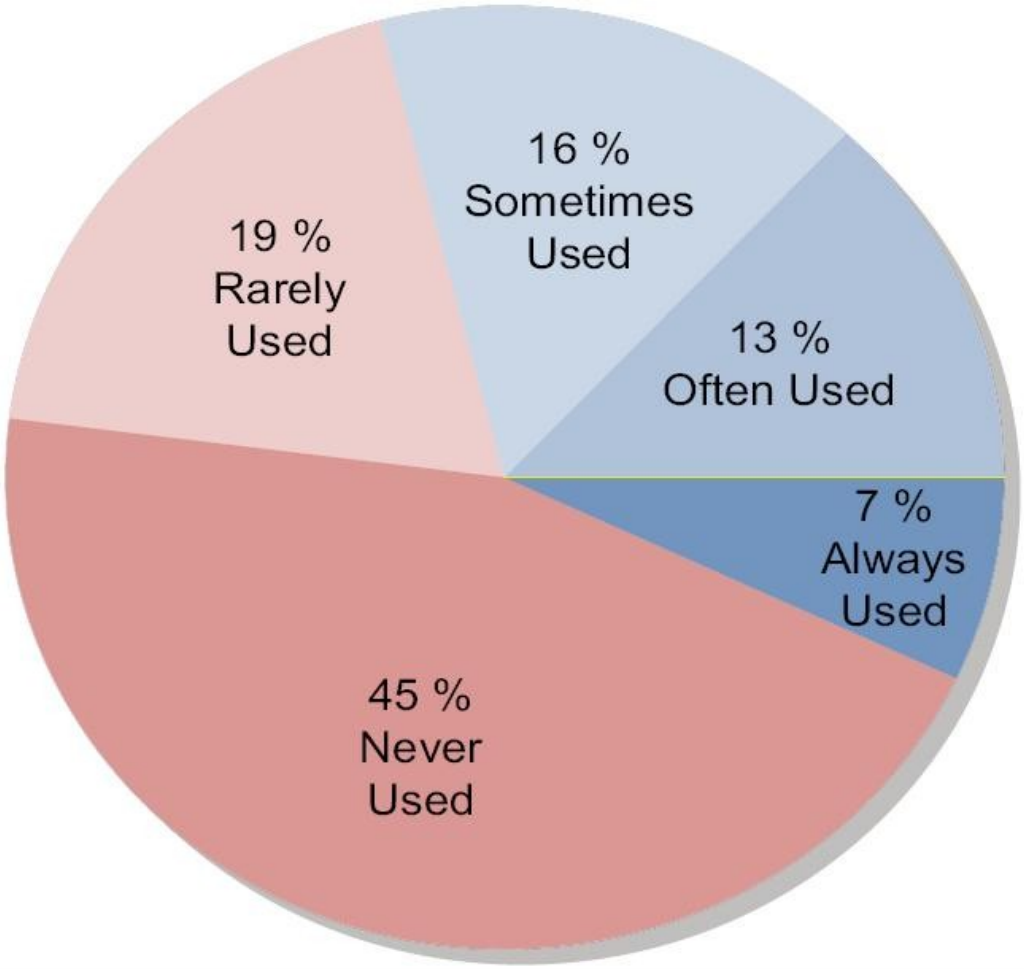
- Therefore they know where to modify the code when a requirement change occurs.
- They know exactly where to go hunting for a bug that has been found.
- They communicate well with each other about the world and the software.

Agile development: Strategies to reduce cost

Build Less:

- It has been shown that we build many more features than are actually used.
- In fact only about 20% of functionality we build is used often or always.
- More than 60% of all functionality built in software is rarely or never used!

Agile development: Strategies to reduce cost



Agile development: Strategies to reduce cost

Pay Less for Bug Fixes:

- Typically, anywhere between 60%-90% of software cost goes into the maintenance phase.
- Most of our money goes into keeping the software alive and useful for our clients after the initial build.

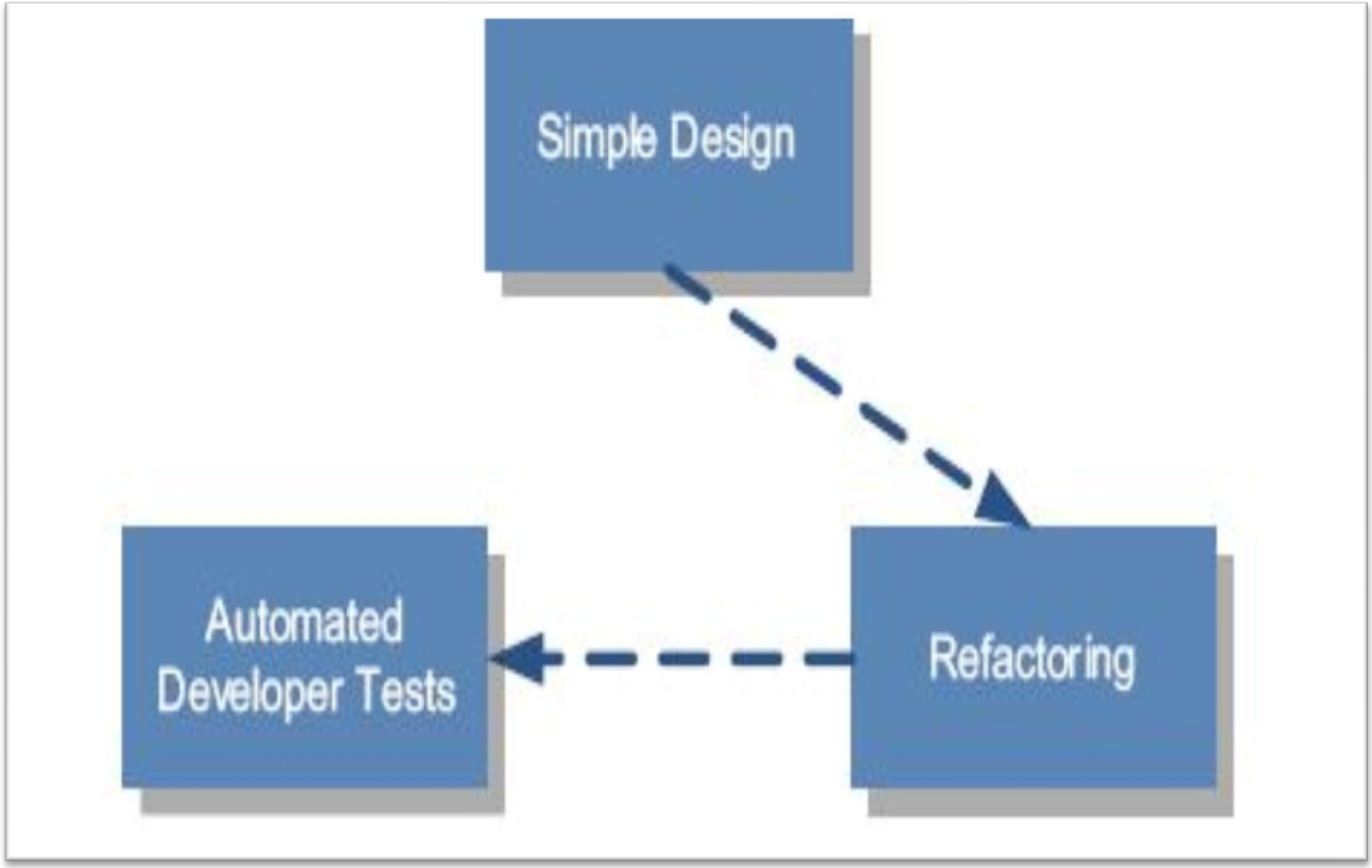
Agile development: Strategies to reduce cost

- Pay Less for Changes:
  The only thing constant in today's software market is change. If we can embrace change, plan for it, and reduce its cost when it eventually happens we can make significant savings.

Reduce cost: A summery

- Agile development => Set of chosen practices => Reduce cost =>
  1- Maintain code theory.
  2- Build less.
  3- Pay less for bugs.
  4- Pay less for changes

Agile practices : 3 basic elements



Agile practices: Simple design

- Simple design meets the requirements for the current iteration and no more.
- Simple design reduces cost because you build less code to meet the requirements and you maintain less code afterwards.
- Simple designs are easier to build, understand, and maintain.
- Simple design<=> Build less

Agile practices: Refactoring

- The practice of Refactoring code changes the structure of the code while maintaining its behavior.
- Costs are reduced because continuous refactoring keeps the design from degrading over time, ensuring that the code is easy to understand, maintain, and change.
- Refactoring <=> Pay less for change

Agile practices: Automated Developer

- Automated developer tests are a set of tests that are written and maintained by developers to reduce the cost of finding and fixing defects —thereby improving code quality—and to enable the change of the design as requirements are addressed incrementally.
- Automated developer tests reduce the cost of software development by creating a safety-net of tests that catch bugs early and enabling the incremental change of design.
- Have a look at TDD or BDD.
- Developer Tests <=> Pay less for bugs fixing

Ruby Language

- Ruby was conceived on February 24, 1993 by Yukihiro Matsumoto(Matz) who wished to create a new language that balanced functional programming with imperative programming. According to Matsumoto he "wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language".

Ruby Language: Examples

Selecting even numbers from a range:

```
(1..25).select {|x| x % 2 == 0 }

#=>  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
```

Ruby Language: Examples

Cumulative sum:

[1,2,3,4] => [1,3,6,10]

```
sum = 0
[1,2,3,4].map{|x| sum += x}

#=> [1,3,6,10]
```

Ruby language

- Ruby is a dynamic, reflective, general purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features. Ruby originated in Japan during the mid-1990s and was initially developed and designed by Yukihiro "Matz" Matsumoto. It is based on Perl, Smalltalk, Eiffel, Ada, and Lisp.
- Ruby supports multiple programming paradigms, including functional, object oriented, imperative and reflective. It also has a dynamic type system and automatic memory management; it is therefore similar in varying respects to Python, Perl, Lisp, Dylan, and CLU.

Ruby Language: Dynamic

- Dynamic programming language is a term used broadly in computer science to describe a class of high-level programming languages that execute at runtime many common behaviors that could include extension of the program, by adding new code, by extending objects and definitions, or by modifying the type system, all during program execution.

Ruby Language: Dynamic

- Eval: Evaluating code on runtime.
  eval “x=1; x+=5” #=> 6
  instance_eval, class_eval
- Higher order functions: Passing functions as arguments. In Ruby we pass blocks (anonymous functions)
  (1..25).select {|x| x % 2 == 0 }
- Reflection: Modifying program structure and behavior; treating code like data.

Ruby Language: Dynamic

- Monkey patching:
- Example: Sum array values

```
class Array
   def sum
      self.inject{|sum,current|
sum+current }
   end
end
[1,5,7,8].sum #=> 21
```

Ruby language: Reflective

- In computer science, reflection is the process by which a computer program can observe and modify its own structure and behavior.
- Ruby has a very wide set of methods for introspection and reflection.

Ruby language: Reflection examples

```
class Foo
  def hi
    puts “hi”
  end

  def bye
    puts “bye”
  end
end
# Introspection API
Foo.new .methods(false)
#=> [“hi”,”bye”]
```

Ruby language: Reflection examples

```
class Foo
end
f = Foo.new
f.methods(false) #=>[]
f.define_method(:hi) { puts “hello
   world!" }
f.methods(false) #=>[“hi”]
f.hi #=> hello world!
```

Ruby language: Reflection examples

Ruby's attr_accessor is an example of Ruby's reflection use:

```
class Foo
  attr_accessor :name
  # getter
  # def name ; @name ; end
  # setter
  # def name=(sn); @name = sn; end
end
```

Ruby language: Everything is Object

- Everything is object even null values!
  nil.class #=> NilClass
- This is called Unified Object Model UOM.
- Cool, you don’t have to think of primitive and reference values, everything is an object and has its methods.
- Treating everything as an object will extend the language dynamicity to unlimited scopes.
- You can treat methods as objects as well.

Ruby language: Support for functional paradigm

- Everything is evaluated as an expression in Ruby.
- Example:

You can do this:

```
if num == "one" then val = 1
elsif num == "two" then val = 2
else then val = 3 end
```

But this is better:

```
val = if num == "one" then 1
      elsif num == "two" then 2
      else 3 end
```

Ruby language: Support for functional paradigm

- Methods are higher order functions; you can pass code blocks to methods, this makes you focus on the ‘what’ part instead of the ‘how’ one of your problem.
- Example: selecting even numbers from a range:

  Traditional imperative way:

```
res = []
input = 1..25
for  x in input
  res <<  x if x % 2 == 0
end
```

  Functional way:

```
res = (1..25).select{|x| x % 2 == 0}
```

- How about odd numbers?

```
res = (1..25).select{|x| x % 2 > 0}
```

Ruby language: Support for functional paradigm

- Functional programming emphasize immutability(pure methods) thus no side effects!

```
s = "hello" #=> "hello“
s2 = s.upcase #=> "HELLO“
puts s.upcase! #=> "HELLO“
puts s , s2 #=> "HELLO“, “hello”
```

Ruby Language : TDD

- Ruby community embraces Test Driven Development, and recently the more recent type of it called Behavior Driven Development BDD.
- TDD or BDD => Automated test suit => reduce the cost of finding and fixing defects => improving code quality => Productivity ⇔ reduce cost

Ruby language: Productive

- Remember: Agile development => reduce cost.
- Ruby code is at least 50% more concise than code written in other languages like Java, which means write less and get the same => productivity.
- Having less code means less bugs to fix and less code to maintain(refactor) => productivity.
- Ruby code is so readable, Ruby follows the principle of least surprise(POLS). Readable code means self documenting => productivity.
- TDD – BDD => improving code quality => Productivity

Ruby Language : Internal DSLs

- A Domain-specific language(DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem. Regular expressions and CSS are 2 examples of DSLs.
- Any software language needs a parser and an interpreter(or compiler or a mix), but in DSLs, we have 2 types: external ones which need parsers and interpreters, and internal ones which rely on the hosting language power to give the feel of a particular language, and thus they don't require their own parsers and they use the language's interpreter.

Ruby Language : Internal DSLs Example

```
RobotTasksExecuter.start do
  stack :boxes => 5 do
    fetch do
      rotate :rectangle => 60
      pick :speed => 'slow',:height => 15
      rotate :rectangle => -60
      free :speed => 'slow'
    end
    package do
      lock
      seal
    end
  end
end
```

Ruby Language : Callbacks

```ruby
class Colors
  def mix(*colors)
    if block_given?
      yield(colors)
    else
      puts "Mixing: #{colors.join ' + '}"
    end
  end

  def method_missing(*args, &block)
    # First arg is a symbol representing the name of the method.
    method_name = args.shift
    # Extract the colors from the method name and collecting them in an array
    # This is a very basic treatment.
    colors = method_name.id2name.split(/and/i).collect{|color| color.downcase}
    # Send the colors and the block to 'mix' method.
    self.send :mix, colors ,&block
  end
end

c = Colors.new
c.mix "orange" , "cyan" #=> Mixing: orange + cyan
c.mix("green","red"){|colors| puts "A nice mix: #{colors.join(' + ')}"} #=> A nice
c.greenAndBlue #=> Mixing: green + blue
c.greenAndWhiteAndBrown #=> Mixing: green + white + brown
c.grayAndYellow{|colors| puts "A nice mix: #{colors.join(' + ')}"} #=> A nice mix:
```

Ruby Language : Callbacks

ActiveRecord ORM:

Traditional:

```
user =
  User.find(:conditions=>[“name=?
  and age = ? ”, “khelll”, 18])
```

Cool:

```
user =
  User.find_by_name_and_age(“khelll”,
  26)
```

Ruby language : Real World Examples

- Ruby is used in almost all fields.
- The most common use is RubyOnRails or RoR.
- RoR is a inspiring web framework that was implemented in almost all living languages.
- RoR is used by yellowpages.com and twitter.com.
- Ruby is used by system admins as it’s easier to write than shell scripts.
- Ruby is used in telecomunication systems.

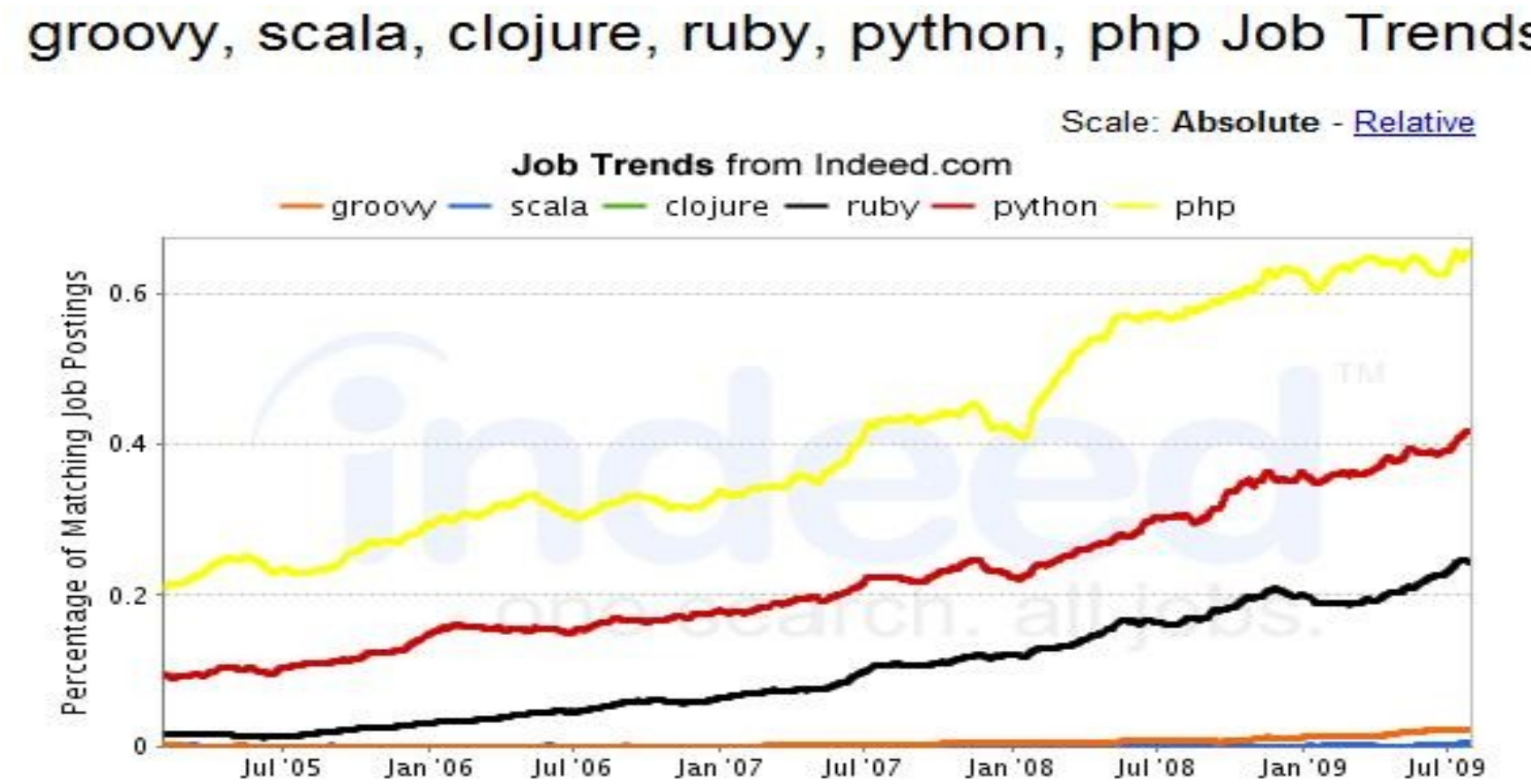
Ruby language: Popularity

TIOBE .com

Position Sep 2009	Position Sep 2008	Delta in Position	Programming Language	Ratings Sep 2009	Delta Sep 2008	Status
1	1	=	Java	19.383%	-1.33%	A
2	2	=	C	16.861%	+1.48%	A
3	5	↑↑	PHP	10.156%	+0.91%	A
4	3	↓	C++	9.988%	-0.73%	A
5	4	↓	(Visual) Basic	9.196%	-1.29%	A
6	7	↑	Perl	4.528%	-0.31%	A
7	8	↑	C#	4.186%	-0.15%	A
8	6	↓↓	Python	3.930%	-1.08%	A
9	9	=	JavaScript	2.995%	-0.14%	A
10	11	↑	Ruby	2.377%	-0.38%	A

Ruby language: Popularity

indeed.com



Ruby language: Implementations

- CRuby, the official Ruby implementation.
- JRuby, Java based implementation that runs on JVM, it integrates Java to be used with Ruby.
- IronRuby, .Net implementation.

Deep thanks!