

Implementación de DSLs con Ruby

Rubén Heradio Gil

Dpto. de Ingeniería de Software y Sistemas Informáticos
Universidad Nacional de Educación a Distancia

Concepto de SPL generativa

Un ejemplo sencillo

Implementación de una SPL generativa

Conclusiones

Para profundizar: bibliografía y ejercicios

Economía de alcance

Reutilización sistemática de software

Enfoque generativo

Resumen

Desarrollo de Líneas de Productos Software mediante un Enfoque Generativo

Economías de escala y de alcance

Fase de desarrollo



Fase de producción



Economía de alcance



Economía de escala



Desarrollo no sistemático de n productos similares



Ingeniero de Requisitos



Ingeniero de Requisitos



Ingeniero de Requisitos



Desarrollador



Desarrollador



Desarrollador



Tester



Tester



Tester

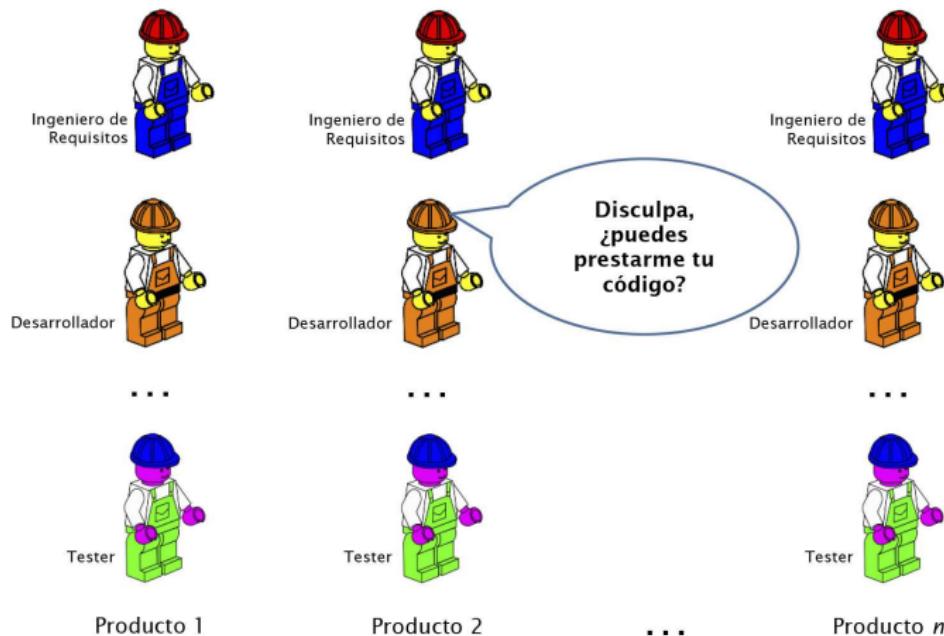
Producto 1

Producto 2

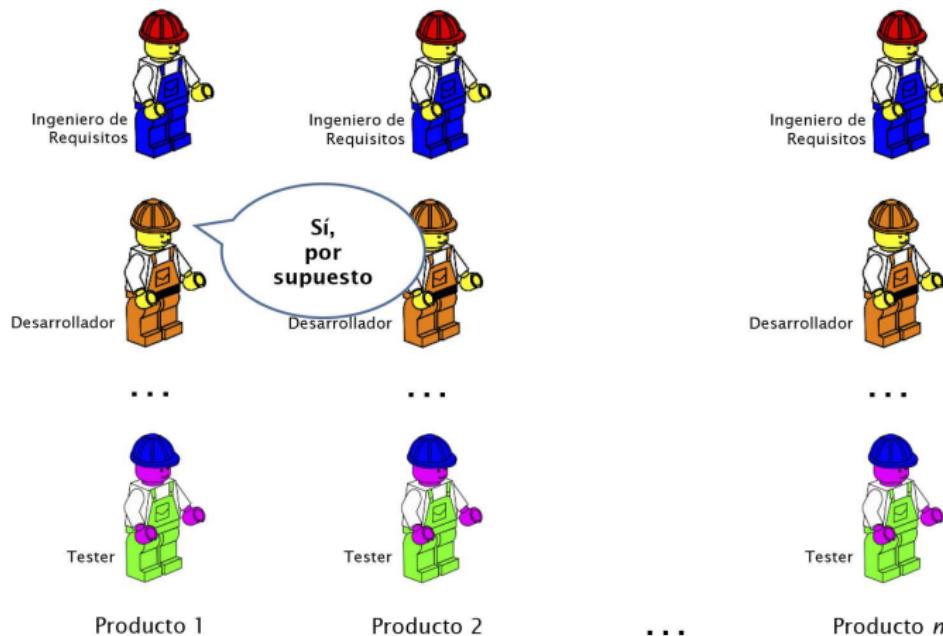
...

Producto n

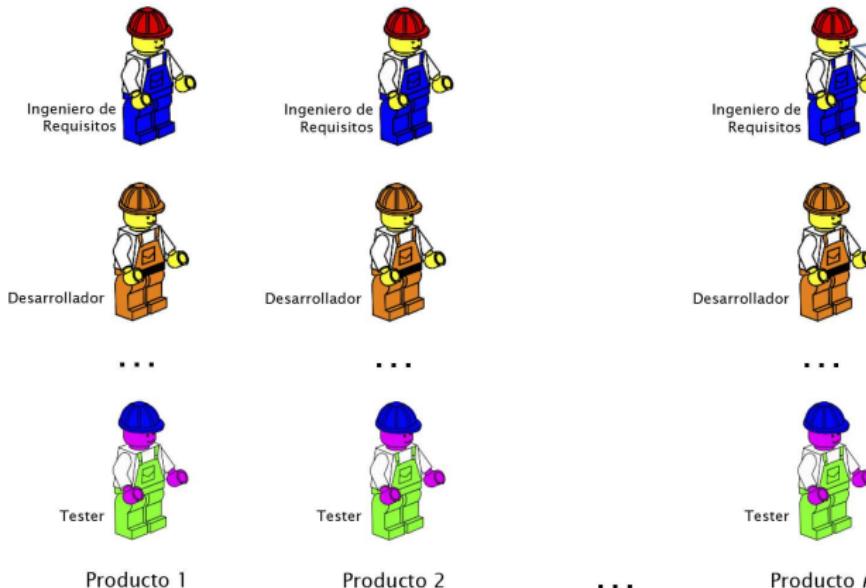
Desarrollo no sistemático de n productos similares



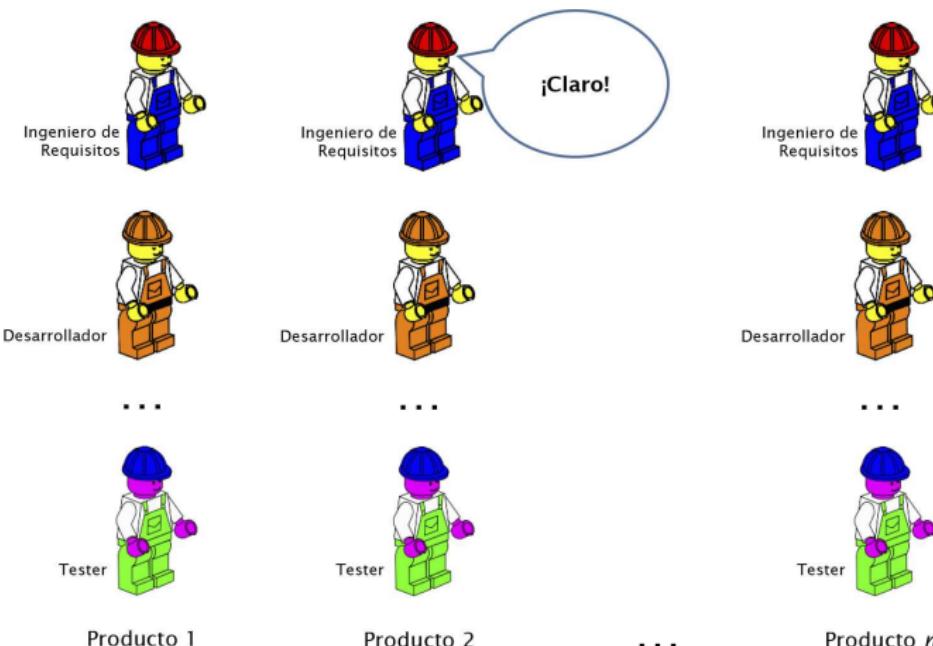
Desarrollo no sistemático de n productos similares



Desarrollo no sistemático de n productos similares



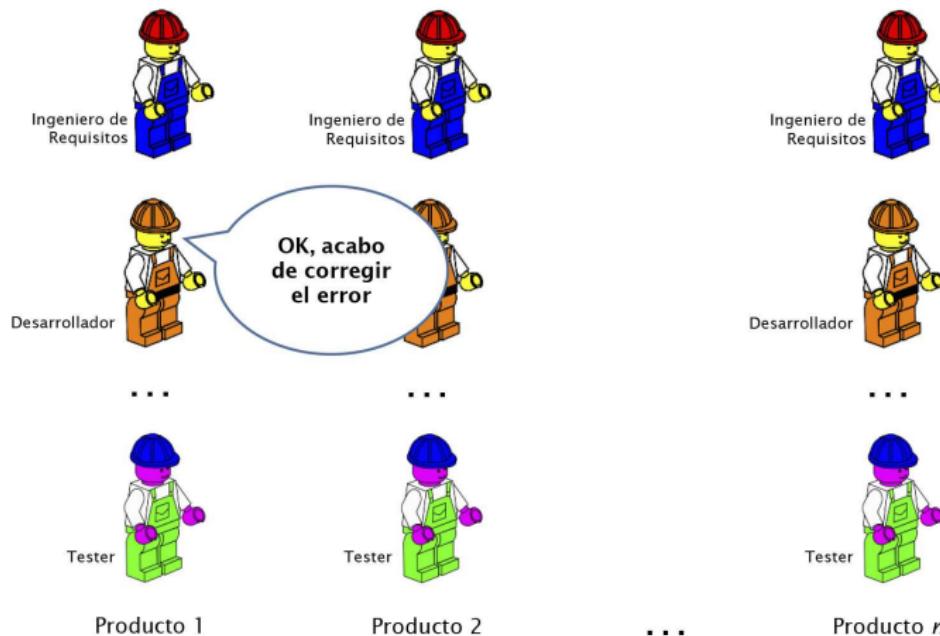
Desarrollo no sistemático de n productos similares



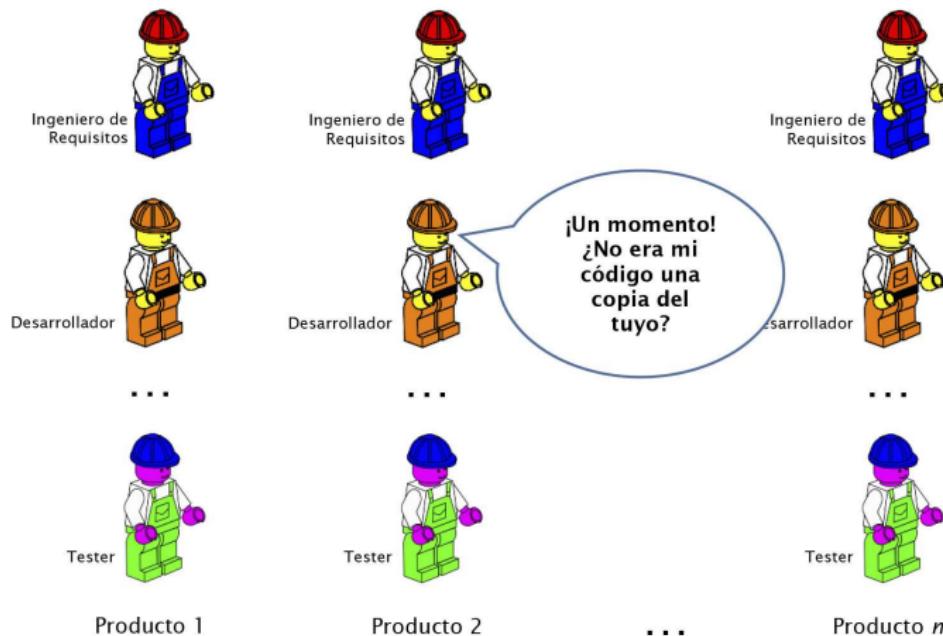
Desarrollo no sistemático de n productos similares



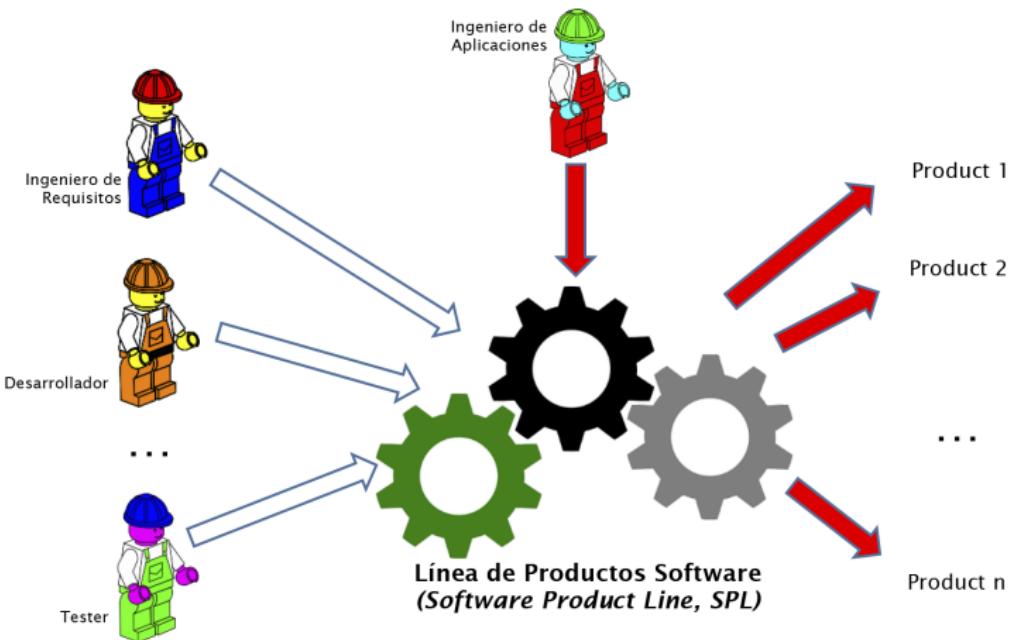
Desarrollo no sistemático de n productos similares



Desarrollo no sistemático de n productos similares



Ingeniería de líneas de productos software



Enfoque generativo: facilidad de reutilización

Librería clásica



Framework



Modelo generativo



Facilidad de reutilización

Lenguaje específico de dominio

DSL
Domain Specific Language



Compilador/Intérprete



Producto

Resumen

Nombre de la asignatura	Explicación
Desarrollo de líneas de productos software mediante un enfoque generativo	Economía de alcance + Reutilización sistemática
	Facilidad de reuso

Concepto de SPL generativa

Un ejemplo sencillo

Implementación de una SPL generativa

Conclusiones

Para profundizar: bibliografía y ejercicios

Enunciado del problema

Esquema de reutilización oportunista

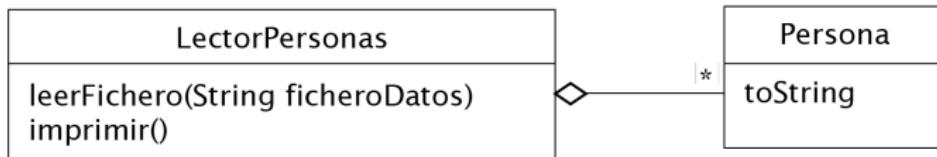
Deficiencias de esta forma de reutilización

Hacia un modelo generativo

Un ejemplo sencillo

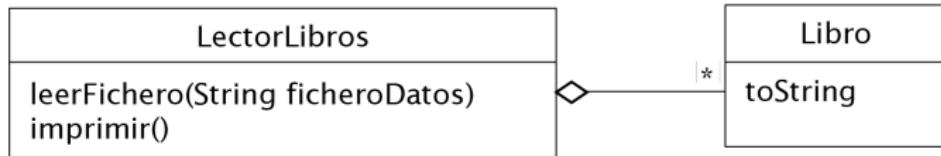
Lector de datos de personas

Pedro, 30
Juan Antonio, 35
Carmen, 29

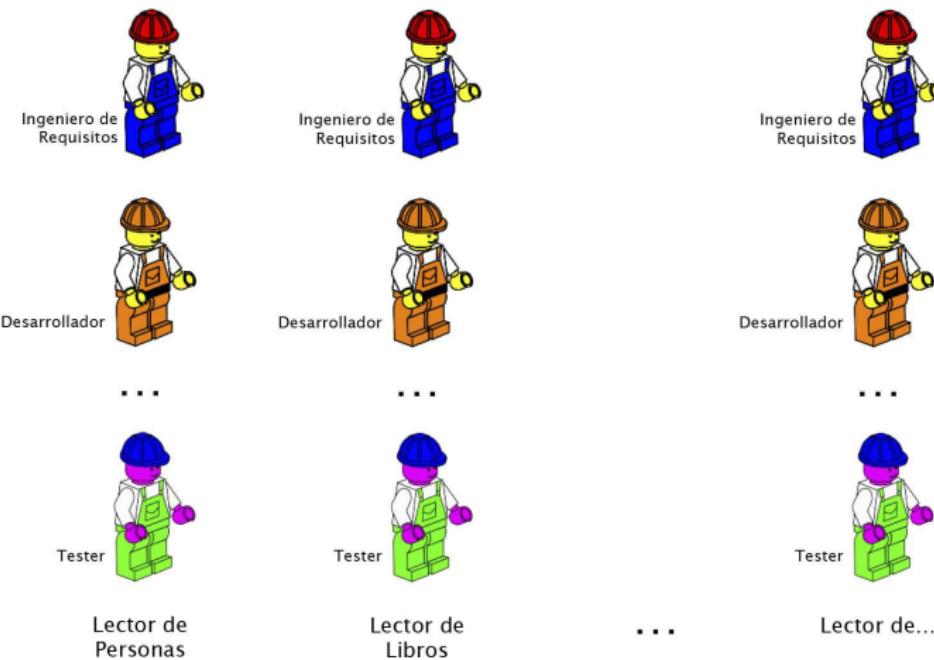


Lector de datos de libros

Gabriel García Márquez, Cien años de soledad, 23
Mario Vargas Llosa, La ciudad y los perros, 18



Esquema de reutilización oportunista



```

public class LectorPersonas {
    private ArrayList<Persona> datos = new ArrayList<Persona>();
    public ArrayList<Persona> leerFichero(String ficheroDatos) {
        String linea;
        try {
            BufferedReader buffer =
                new BufferedReader(new FileReader(ficheroDatos));
            linea = buffer.readLine();
            while (linea != null) {
                procesarLinea(linea);
                linea = buffer.readLine();
            }
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return datos;
    }
    private void procesarLinea(String linea) {
        Scanner scanner = new Scanner(linea).useDelimiter("\\s*,\\s*");
        String nombre = scanner.next();
        int edad = scanner.nextInt();
        datos.add(new Persona(nombre, edad));
    }
    public void imprimir() {
        for( Persona persona : datos ) {
            System.out.println(persona);
        }
    }
}

```

```

public class Persona {
    private String nombre;
    private int edad;
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public String toString() {
        return "nombre = " + nombre + "\n" +
            "edad = " + edad + "\n" +
            "-----";
    }
}

```

```

public class LectorLibros {
    private ArrayList<Libro> datos = new ArrayList<Libro>();
    public ArrayList<Libro> leerFichero(String ficheroDatos) {
        String linea;
        try {
            BufferedReader buffer =
                new BufferedReader(new FileReader(ficheroDatos));
            linea = buffer.readLine();
            while (linea != null) {
                procesarLinea(linea);
                linea = buffer.readLine();
            }
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return datos;
    }
    private void procesarLinea(String linea) {
        Scanner scanner = new Scanner(linea).useDelimiter("\\s*,\\s*");
        String autor = scanner.next();
        String titulo = scanner.next();
        int precio = scanner.nextInt();
        datos.add(new Libro(autor, titulo, precio));
    }
    public void imprimir() {
        for( Libro Libro : datos ) {
            System.out.println(Libro);
        }
    }
}

```

```

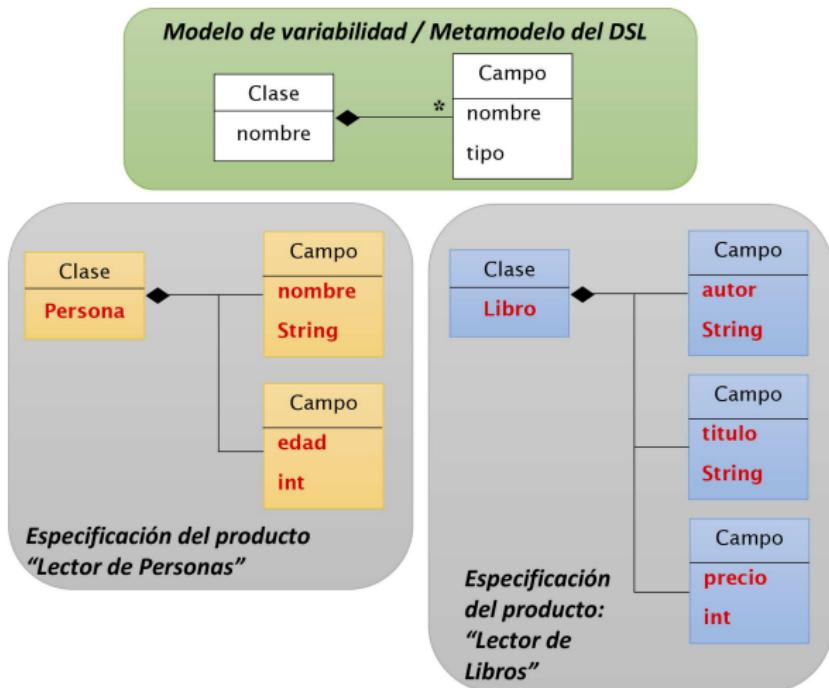
public class Libro {
    private String autor;
    private String titulo;
    private int precio;
    public Libro(String autor,
                String titulo, int precio) {
        this.autor = autor;
        this.titulo = titulo;
        this.precio = precio;
    }
    public String toString() {
        return "autor = " + autor + "\n" +
               "titulo = " + titulo + "\n" +
               "precio = " + precio + "\n" +
               "-----";
    }
}

```

Deficiencias de esta forma de reutilización

- ➊ Antes hablábamos del mantenimiento...
- ➋ Para reutilizar es necesario entender la implementación
- ➌ Hay que escribir más de lo necesario

¿Qué es imprescindible para especificar un producto?



Concepto de SPL generativa

Un ejemplo sencillo

Implementación de una SPL generativa

Conclusiones

Para profundizar: bibliografía y ejercicios

SPL 1: XML/Ruby/Java

SPL 2: Ruby/Ruby/Java

SPL 3: Ruby/Ruby/Ruby

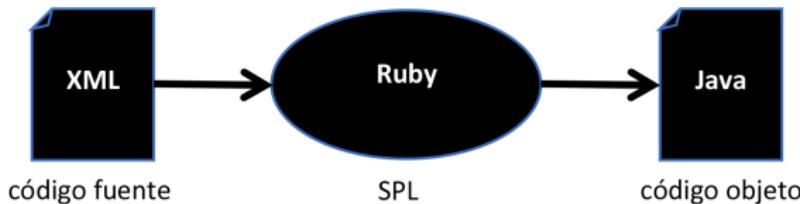
SPL 4: Ruby/Ruby/Ruby-Java

Implementación de una SPL generativa

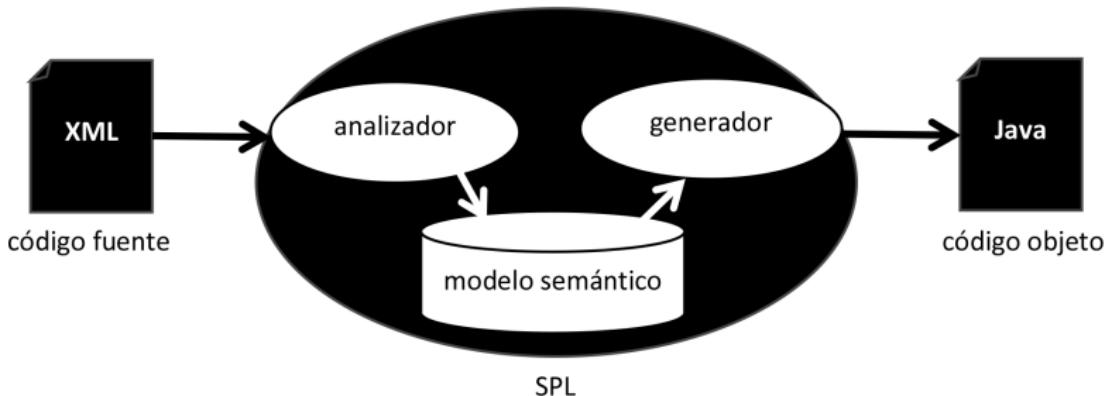
SPL 1: XML/Ruby/Java

```
<dsl clase="Persona">
  <campo nombre="nombre" tipo="String"/>
  <campo nombre="edad" tipo="int"/>
</dsl>
```

```
<dsl clase="Libro">
  <campo nombre="autor" tipo="String"/>
  <campo nombre="titulo" tipo="String"/>
  <campo nombre="precio" tipo="int"/>
</dsl>
```



SPL 1: XML/Ruby/Java



```

# ModeloSemantic
class ModeloSemantic
  attr_accessor :nombre_clase, :campos
  def initialize()
    @campos = Array.new
  end
end

# Analizador
require 'rexml/document'
modelo_semantico = ModeloSemantic.new
doc = REXML::Document.new(File.open("dsl_personas.xml"))
modelo_semantico.nombre_clase = doc.root.attributes['clase']
doc.root.each_element('campo') { |campo|
  modelo_semantico.campos << [campo.attributes['nombre'], campo.attributes['tipo']]
}

# Generador
require 'erb'
File.open('plantilla_clase_datos.erb') { |plantilla|
  erb = ERB.new( plantilla.read )
  File.open("#{modelo_semantico.nombre_clase}.java", 'w') { |fichero_obj|
    fichero_obj.write(erb.result( binding ))
  }
}
File.open('plantilla_lector.erb') { |plantilla|
  erb = ERB.new( plantilla.read )
  File.open("Lector#{modelo_semantico.nombre_clase}s.java", 'w') { |fichero_obj|
    fichero_obj.write(erb.result( binding ))
  }
}

```

```

public class LectorPersonas {
    private ArrayList<Persona> datos = new ArrayList<Persona>();
    public ArrayList<Persona> leerFichero(String ficheroDatos) {
        String linea;
        try {
            BufferedReader buffer =
                new BufferedReader(new FileReader(ficheroDatos));
            linea = buffer.readLine();
            while (linea != null) {
                procesarLinea(linea);
                linea = buffer.readLine();
            }
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return datos;
    }
    private void procesarLinea(String linea) {
        Scanner scanner = new Scanner(linea).useDelimiter("\\s*,\\s*");
        String nombre = scanner.next();
        int edad = scanner.nextInt();
        datos.add(new Persona(nombre, edad));
    }
    public void imprimir() {
        for( Persona persona : datos ) {
            System.out.println(persona);
        }
    }
}

```

```

public class Persona {
    private String nombre;
    private int edad;
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public String toString() {
        return "nombre = " + nombre + "\n" +
            "edad = " + edad + "\n" +
            "-----";
    }
}

```

Concepto de SPL generativa

Un ejemplo sencillo

Implementación de una SPL generativa

Conclusiones

Para profundizar: bibliografía y ejercicios

SPL 1: XML/Ruby/Java
SPL 2: Ruby/Ruby/Java
SPL 3: Ruby/Ruby/Ruby
SPL 4: Ruby/Ruby/Ruby-Java

```
public class <%= modelo_semantico.nombre_clase %> {
    <% modelo_semantico.campos.each [campo] %>
    <%= "#{campo[1]} #[campo[0]]" %>
<% } %>
public <%= modelo_semantico.nombre_clase %> {
<%
    i = 0
    while i<(modelo_semantico.campos.length-1)
<%>
    <%= "#{modelo_semantico.campos[i][1]}"
    #[modelo_semantico.campos[i][0]] %>
<% }
    #{
        "#{modelo_semantico.campos[i][1]}"
        #[modelo_semantico.campos[i][0]] %>
    }
<% modelo_semantico.campos.each [campo] %>
    <%= "#{campo[1]} #[campo[0]]" %>
<% } %>
}
public String toString() {
return
<% modelo_semantico.campos.each [campo] %>
    <%= "\\"#{campo[0]}\\\" = \\"#{campo[0]}\\\" + \\"\\n\\\" + "
<% } %>
"-----"
}
```

public class
<%=
modelo_semantico.
nombre_clase %> {

```
import java.io.*;
import java.util.*;

public class Lector<%=modelo_semantico.nombre_clase%> {
    private ArrayList<%=modelo_semantico.nombre_clase%> datos = new
    ArrayList<%=modelo_semantico.nombre_clase%>();
    public ArrayList<%=modelo_semantico.nombre_clase%> leerFichero(String
ficheroDatos) {
        String linea;
        try {
            BufferedReader buffer = new BufferedReader(new FileReader(ficheroDatos));
            linea = buffer.readLine();
            while (linea != null) {
                procesarLinea(linea);
                linea = buffer.readLine();
            }
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return datos;
    }
    private void procesarLinea(String linea) {
        Scanner scanner = new Scanner(linea).useDelimiter("\\s+\\s+");
        <% modelo_semantico.campos.each [campo]
        if campo[1] == "int"
<%>
            <%= "#{campo[1]} #[campo[0]] = scanner.nextInt(); %>
        <% else %
            <%= "#{campo[1]} #[campo[0]] = scanner.next(); %>
        <% end
        >%>
        datos.add(new <%=modelo_semantico.nombre_clase%>(
<%
        i = 0
        while i<(modelo_semantico.campos.length-1)
<%>
        <%= "#{modelo_semantico.campos[i][1]}"
        #[modelo_semantico.campos[i][0]] %>
    );
    i = i + 1
    end %
<%= "#{modelo_semantico.campos[i][0]] %> ));
}
public void imprimir() {
    for( <%=modelo_semantico.nombre_clase%>
        <%=modelo_semantico.nombre_clase%> : datos ) {
        System.out.println(<%=modelo_semantico.nombre_clase%>);
    }
}
}
```

		Especificación de los productos	
		Fácil	Difícil
Desarrollo y mantenimiento del analizador	Fácil		XML, JSON, etc.
	Difícil	Compiladores clásicos: Lex & Yacc	

XML facilita el “parseo”, no la escritura de especificaciones abstractas

A or B and not C

```
<or>
  A
  <and>
    B
    </not C>
  </and>
</or>
```

		Especificación de los productos	
		Fácil	Difícil
Desarrollo y mantenimiento del analizador	Fácil		XML, JSON, etc.
	Difícil	Compiladores clásicos: Lex & Yacc	

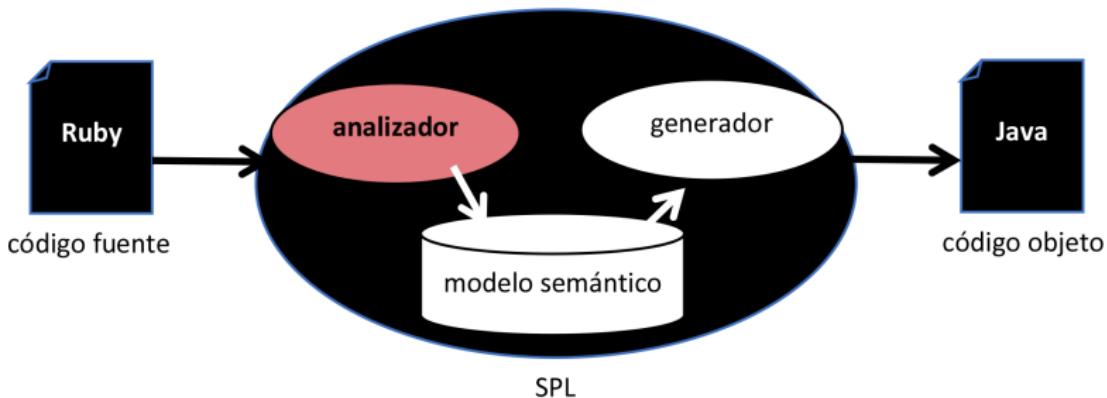
		Especificación de los productos	
		Fácil	Difícil
Desarrollo y mantenimiento del analizador	Fácil		XML, JSON, etc.
	Difícil	Compiladores clásicos: Lex & Yacc	

		Especificación de los productos	
		Fácil	Difícil
Desarrollo y mantenimiento del analizador	Fácil		XML, JSON, etc.
	Difícil	Compiladores clásicos: Lex & Yacc	

SPL 2: Ruby/Ruby/Java

		DSL	
		Compilado	Interpretado
DSL	Externo	XML Java Ruby	X
	Interno	Ruby Java Ruby	Ruby Ruby Ruby

SPL 2: Ruby/Ruby/Java



```
<dsl clase="Persona">
  <campo nombre="nombre" tipo="String"/>
  <campo nombre="edad" tipo="int"/>
</dsl>
```

```
<dsl clase="Libro">
  <campo nombre="autor" tipo="String"/>
  <campo nombre="titulo" tipo="String"/>
  <campo nombre="precio" tipo="int"/>
</dsl>
```

```
clase "Persona" do
  campo "nombre", "String"
  campo "edad", "int"
end
```

```
clase "Libro" do
  campo "autor", "String"
  campo "titulo", "String"
  campo "precio", "int"
end
```

```
clase "Persona" do
  campo "nombre", "String"
  campo "edad", "int"
end
```

```
clase "Libro" do
  campo "autor", "String"
  campo "titulo", "String"
  campo "precio", "int"
end
```

```
# Analizador
def clase(nombre)
  modelo_semantico = ModeloSemantico.new
  modelo_semantico.nombre_clase = nombre
  define_method :campo do |nombre, tipo|
    modelo_semantico.campos << [nombre, tipo]
  end
  yield
  modelo_semantico
end
modelo_semantico = eval File.read('dsl_libro.rb')
```

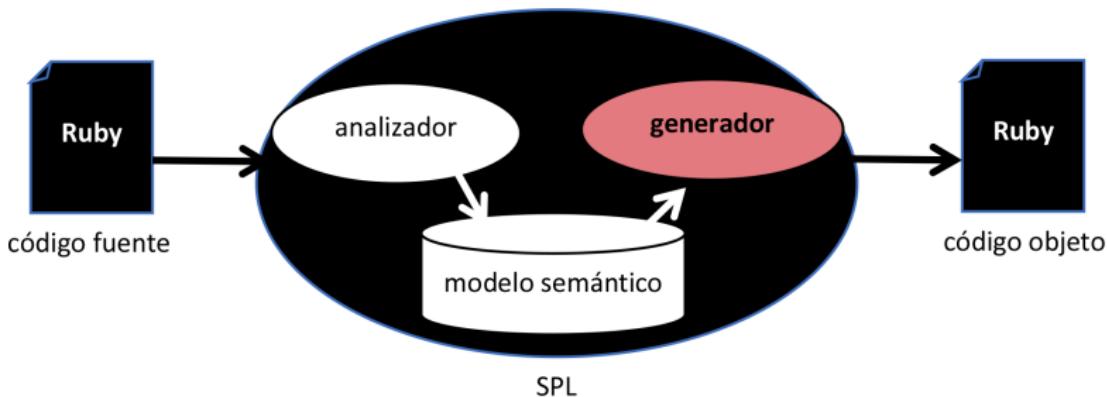
Ventajas

- ➊ Código fuente bastante legible
- ➋ El analizador está embebido en un lenguaje de propósito general:
 - Nuestro DSL puede acceder a librerías del lenguaje “host”
 - La detección de errores léxicos, sintácticos y semánticos se simplifica: el analizador de Ruby hace gran parte del trabajo

SPL 3: Ruby/Ruby/Ruby

		DSL		
		Compilado	Interpretado	
DSL	Externo	XML Java Ruby	X	
	Interno	Ruby Java Ruby	Ruby Ruby Ruby	

SPL 3: Ruby/Ruby/Ruby



```

public class LectorPersonas {
    private ArrayList<Persona> datos = new ArrayList<Persona>();
    public ArrayList<Persona> leerFichero(String ficheroDatos) {
        String linea;
        try {
            BufferedReader buffer =
                new BufferedReader(new FileReader(ficheroDatos));
            linea = buffer.readLine();
            while (linea != null) {
                procesarLinea(linea);
                linea = buffer.readLine();
            }
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return datos;
    }
    private void procesarLinea(String linea) {
        Scanner scanner = new Scanner(linea).useDelimiter("\\s*,\\s*");
        String nombre = scanner.next();
        int edad = scanner.nextInt();
        datos.add(new Persona(nombre, edad));
    }
    public void imprimir() {
        for( Persona persona : datos ) {
            System.out.println(persona);
        }
    }
}

```

```

public class Persona {
    private String nombre;
    private int edad;
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public String toString() {
        return "nombre = " + nombre + "\n" +
            "edad = " + edad + "\n" +
            "-----";
    }
}

```

```

# Generador
# Clase que almacena los datos
eval "class #{modelo_semantico.nombre_clase} end"
clase_datos = Object.const_get(modelo_semantico.nombre_clase)
clase_datos.class_eval do
  define_method :initialize do |parametros|
    i = 0
    while i<parametros.length
      self.instance_variable_set("@#{modelo_semantico.campos[i][0]}".to_sym, parametros[i])
      i += 1
    end
  end
  define_method :to_s do
    resultado = ""
    modelo_semantico.campos.each do |nombre, tipo|
      resultado += "#{nombre} = "
      self.instance_variable_get("@#{nombre}".to_sym).to_s + "\n"
    end
    resultado + "-----"
  end
end
end

# Clase lectora
eval "class Lector#{modelo_semantico.nombre_clase}s end"
clase_lectora = Object.const_get("Lector#{modelo_semantico.nombre_clase}s")
clase_lectora.class_eval do
  attr_reader :datos
  define_method :initialize do |*parametros|
    @datos = Array.new
  end
  define_method :leer_fichero do |ficheroDatos|
    File.readlines(ficheroDatos).each { |linea|
      procesar_linea(linea)
    }
  end
  define_method :procesar_linea do |linea|
    valores = linea.split(/\s*,\s*/)
    valores.map! { |valor| valor.strip }
    @datos << clase_datos.new(*valores)
  end
  def imprimir
    @datos.each { |dato|
      puts dato
    }
  end
end

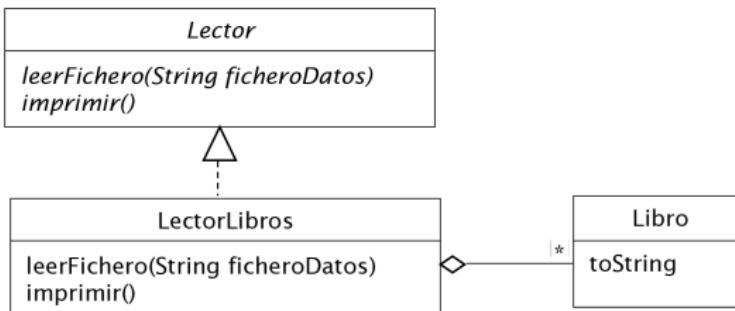
```

SPL 4: Ruby/Ruby/Ruby-Java

		DSL	
		Compilado	Interpretado
DSL	Externo	XML Java Ruby	X
	Interno	Ruby Java Ruby	Ruby Ruby Ruby Ruby Ruby/JAVA Ruby



```
public static void main(String[] args) throws IOException{  
  
    ScriptingContainer interpreteRuby = new ScriptingContainer();  
    String codigoSPL = readFile("spl.rb");  
    interpreteRuby.put("$dsl", "dsl_libro.rb");  
    interpreteRuby.runScriptlet(codigoSPL);  
  
    Lector lector = (Lector) interpreteRuby.runScriptlet("LectorLibros.new");  
    lector.leerFichero("ejemplo_libros.txt");  
    lector.imprimir();  
}
```



```

public static void main(String[] args) throws IOException{
    ScriptingContainer interpreteRuby = new ScriptingContainer();
    String codigoSPL = readFile("spl.rb");
    interpreteRuby.put("$dsl", "dsl_libro.rb");
    interpreteRuby.runScriptlet(codigoSPL);

    Lector lector = (Lector) interpreteRuby.runScriptlet("LectorLibros.new");
    lector.leerFichero("ejemplo_libros.txt");
    lector.imprimir();
}
  
```

```

# Clase lectora
eval "class Lector#{modelo_semantico.nombre_clase}s < Java::Lector\n end"
...
  
```

Concepto de SPL generativa

Un ejemplo sencillo

Implementación de una SPL generativa

Conclusiones

Para profundizar: bibliografía y ejercicios

Conclusiones

Conclusiones

- ① ¿Qué es una SPL?
- ② ¿Enfoque generativo?
- ③ ¿Cómo implementar una SPL generativa?
- ④ Técnicas dinámicas y de metaprogramación para facilitar la creación y mantenimiento de DSLs
- ⑤ El código de esta presentación está disponible en
https://rheradio.github.io/ruby_dsl/

Concepto de SPL generativa
Un ejemplo sencillo
Implementación de una SPL generativa
Conclusiones
Para profundizar: bibliografía y ejercicios

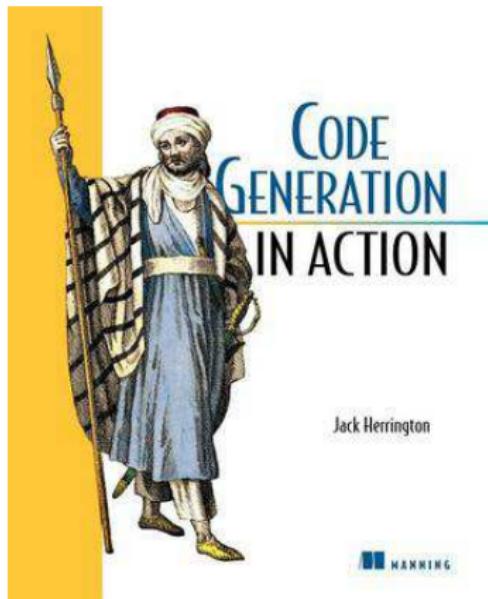
Bibliografía
Ejercicios

Bibliografía y ejercicios

		DSL	
		Compilado	Interpretado
DSL	Externo	XML Java Ruby	X
	Interno	Ruby Java Ruby	Ruby Ruby Ruby Ruby Ruby/Java Ruby

Concepto de SPL generativa
Un ejemplo sencillo
Implementación de una SPL generativa
Conclusiones
Para profundizar: bibliografía y ejercicios

Bibliografía
Ejercicios

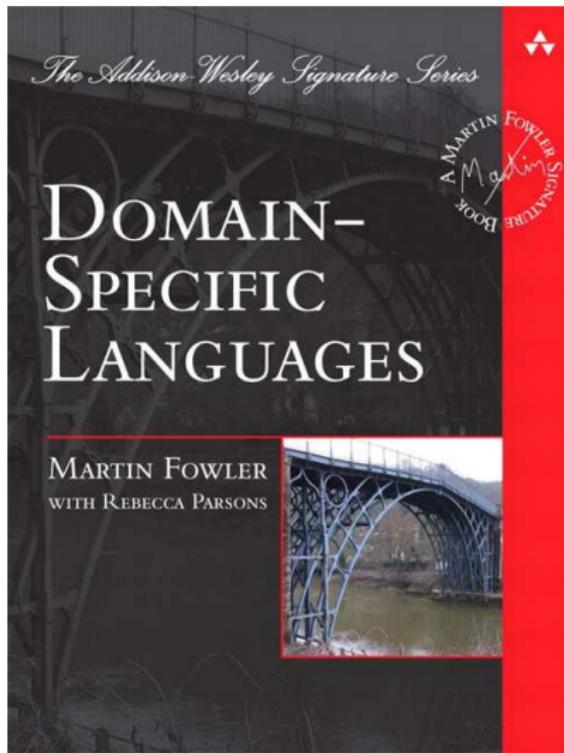


		DSL	
		Compilado	Interpretado
DSL	Externo	XML Java Ruby	X
	Interno	Ruby Java Ruby	Ruby Ruby Ruby Ruby Ruby/JAVA Ruby

Concepto de SPL generativa
Un ejemplo sencillo
Implementación de una SPL generativa
Conclusiones

Para profundizar: bibliografía y ejercicios

Bibliografía
Ejercicios



The
Pragmatic
Programmers

Metaprogramming Ruby 2

Program Like
the Ruby Pros



Paolo Perrotta

Edited by Lynn Beighley

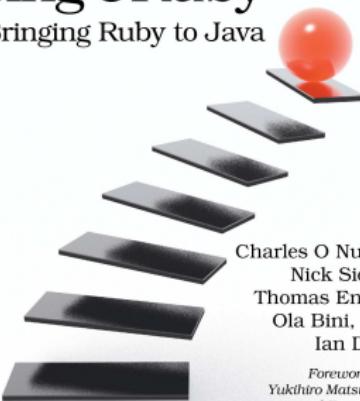
The Facets of Ruby Series

		DSL	
		Compilado	Interpretado
DSL	Externo	XML Java Ruby	X
	Interno	Ruby Java Ruby	Ruby Ruby Ruby Ruby Ruby/Java Ruby



Using JRuby

Bringing Ruby to Java



Charles O Nutter,
Nick Sieger,
Thomas Enebo,
Ola Bini, and
Ian Dees

Forewords by
Yukihiko Matsumoto
and Bruce Tate

Edited by Jacqueline Carter

The Facets of Ruby Series

