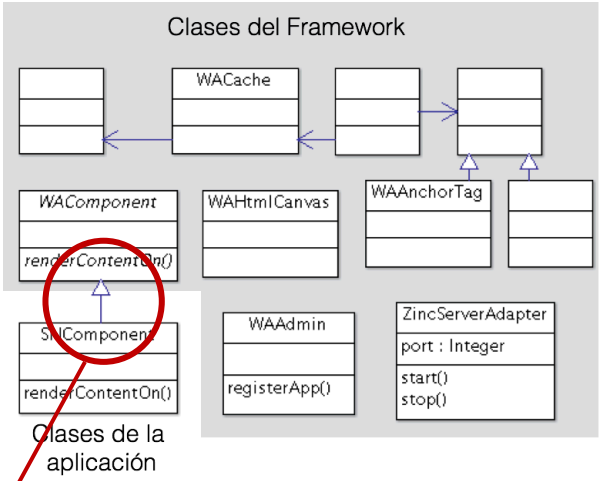
Evolucionando frameworks

el rol de los patrones

Recordamos…



- Hotspot: son las partes en un framework donde ocurre la adaptación.
- Whitebox: si ocurre por subclasificación
- Blackbox: si ocurre por composición
- Frozenspots: Son las partes del framework que no cambian
- Instanciar/especializar un framework es completar/configurar sus hotspots para obtener una aplicación particular. A esas modificaciones las llamamos “incrementos”

Algunos patrones para tener en mente

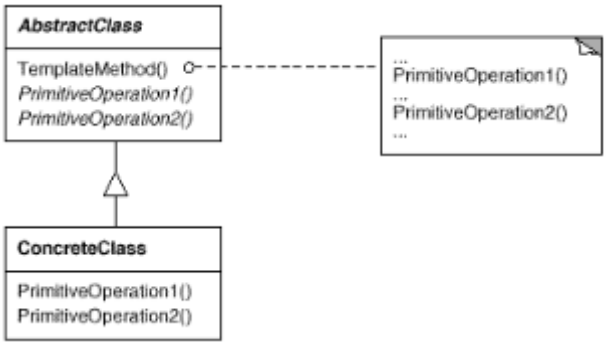
Los patrones encapsulan/separan lo que varía, ¿qué encapsulan cada uno de los siguientes patrones?

- Template method: ???
- Strategy: ???
- Adapter: ???
- Factory method, abstract factory: ???

Algunos patrones para tener en mente

- Template method: para capturar algoritmos comunes (frozenspot) y dejar ganchos
- Strategy: para encapsular algoritmos intercambiables
- Adapter: para aprovechar objetos, en lugares donde no encajan
- Factory method, abstract factory: a la hora de crear objetos

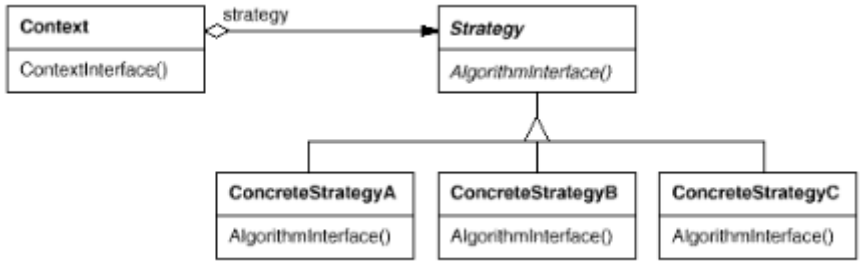
Template method



- Separo un algoritmo de los pasos que lo componen
- El algoritmo (plantilla) nunca cambia (frozenspot)
- Los pasos evolucionan por separado (hotspots)
  - En subclases (herencia)
  - En otro objeto (composición) *

* No es el template method del libro, pero es la misma idea

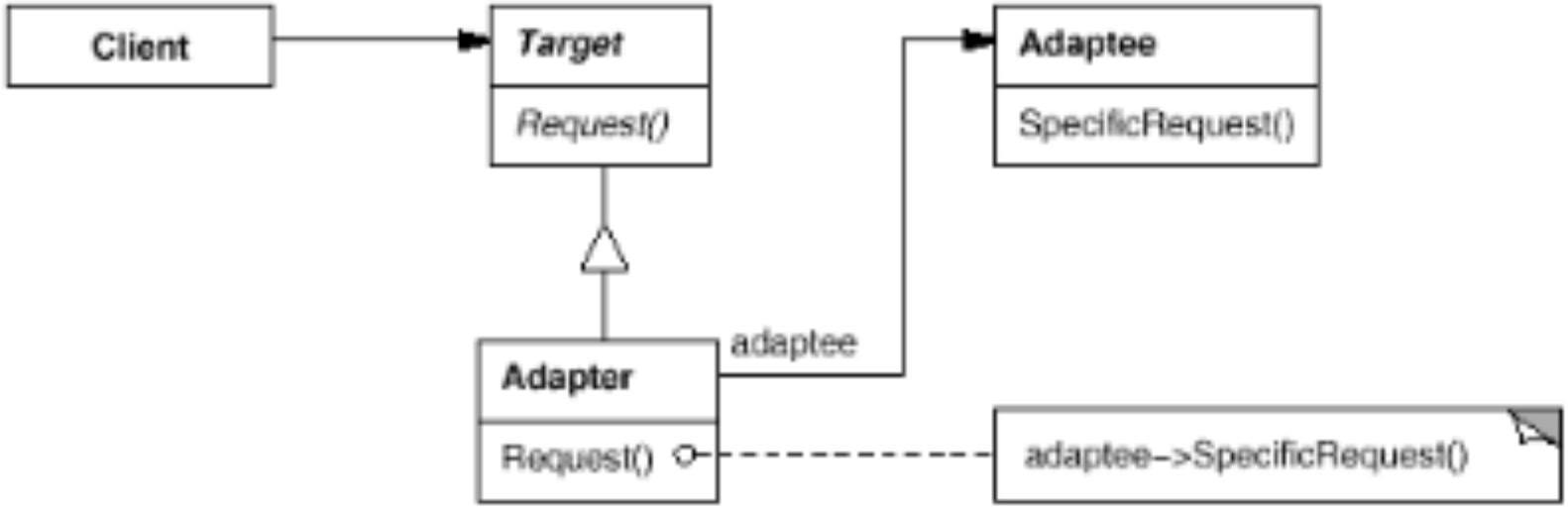
Strategy



- Encapsulo un algoritmo (forma de hacer algo) en un objeto al que se delega
- Lo desacoplo del objeto que lo usa
- Puedo cambiar ese algoritmo en runtime (como en un hotspot por composición)
  - Ofrezco varios algoritmos para intercambiar
- Puedo subclasificar para agregar nuevos algoritmos

Adapter

- En un hotspot (que espera un objeto que entienda ciertos mensajes) utilizo uno que no los entiende pero hace lo que quiero



Encapsular la creación…

MyComponent

```
renderContentOn: htmlCanvas
    htmlCanvas anchor
        callback: [ self reset ];
        with: 'Reset'
```

WAHtmlCanvas

```
anchor
    ^ self brush: WAAnchorTag new

form
    ^ self brush: WAFormTag new
```

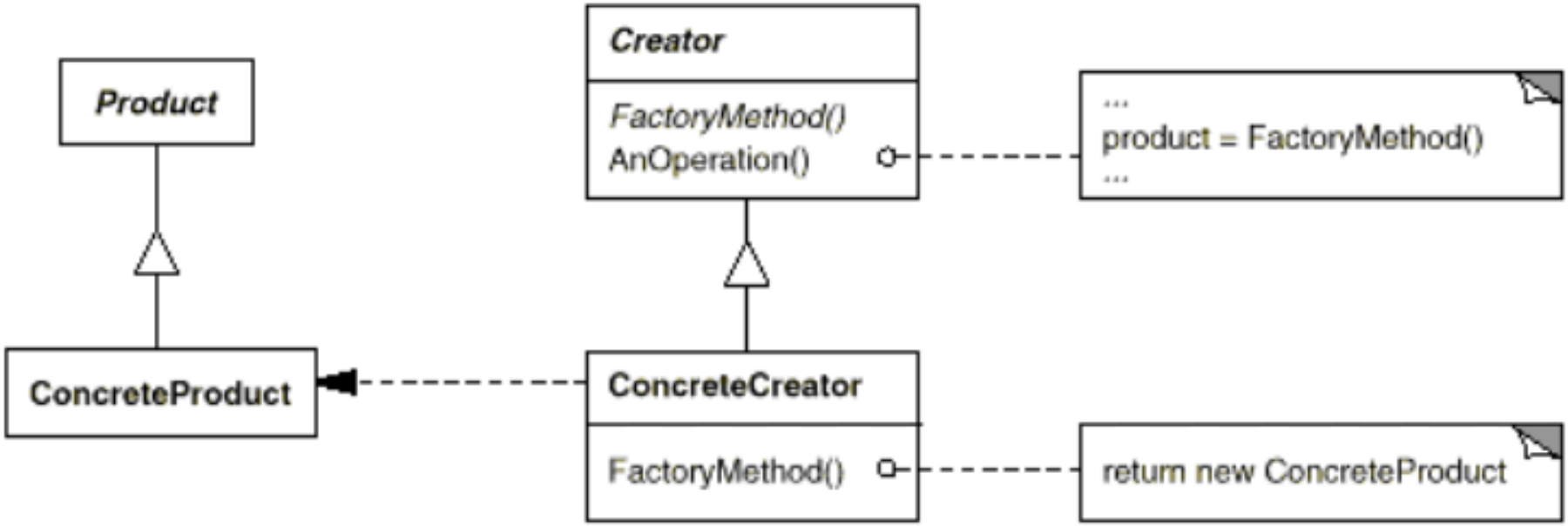
No me preocupa de que clase son los objetos que uso

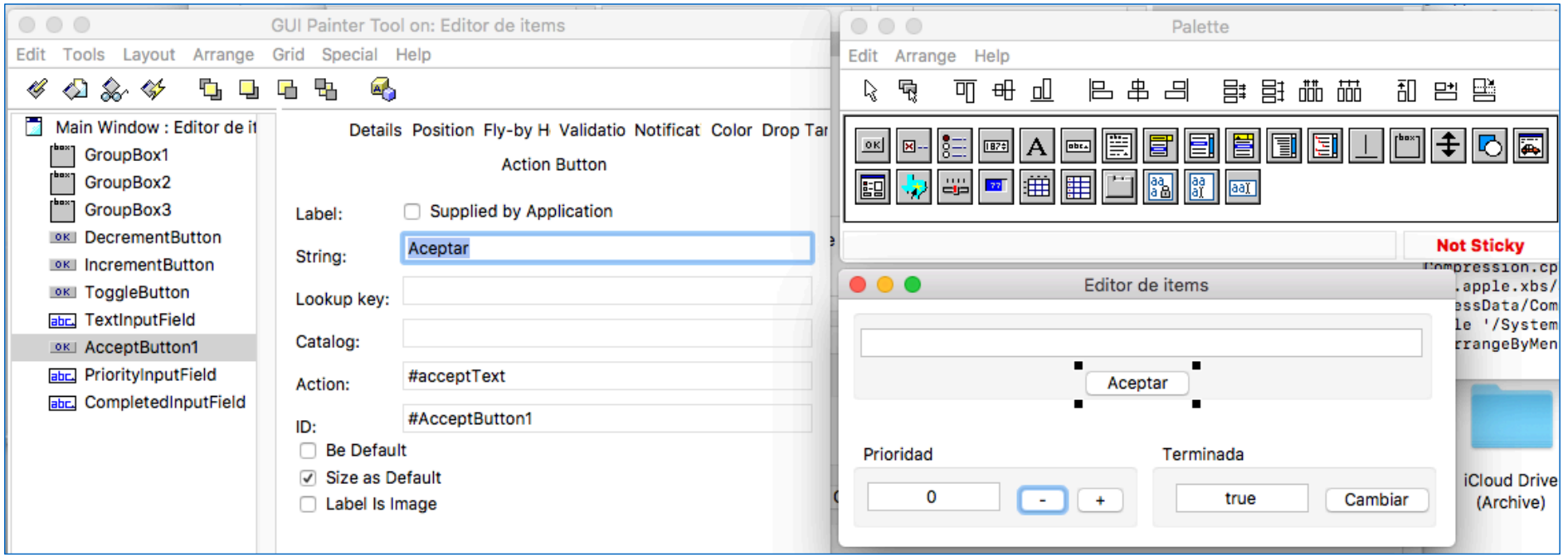
Envío mensajes para obtenerlos y luego los configuro

El htmlCanvas es como una fábrica (y podría haber otras)

Factory Method

Definir la interfaz para crear un objeto, pero dejar que las subclases decidan que clase instanciar. Delegar la instanciación a las subclases.

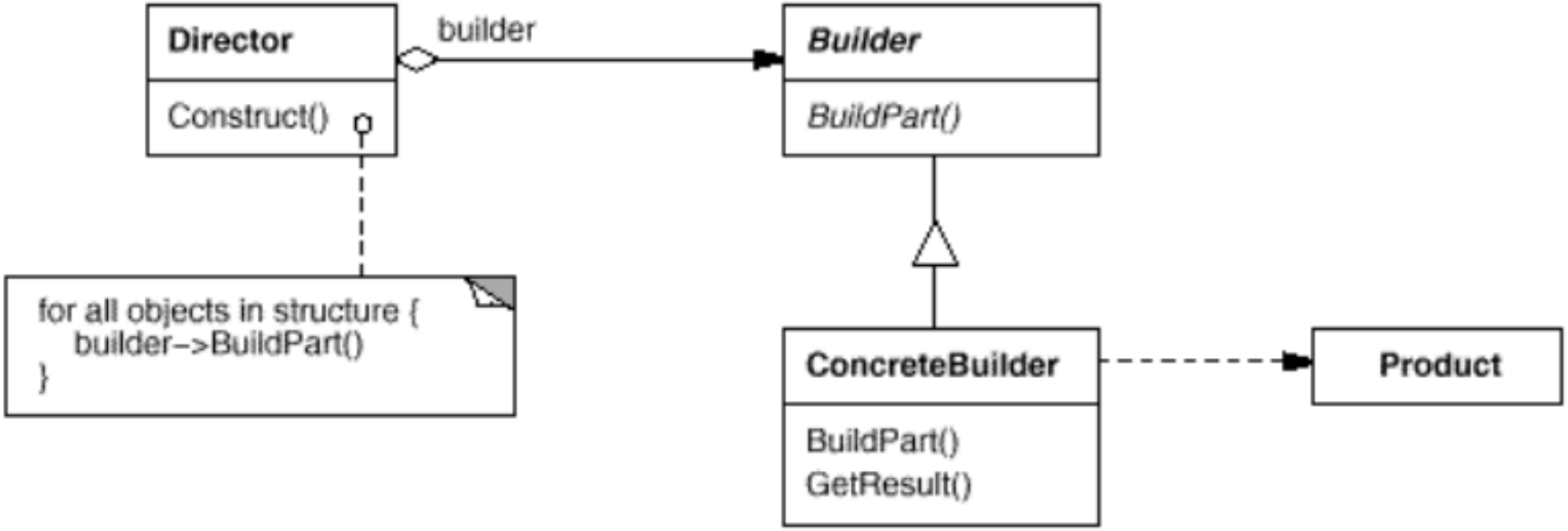


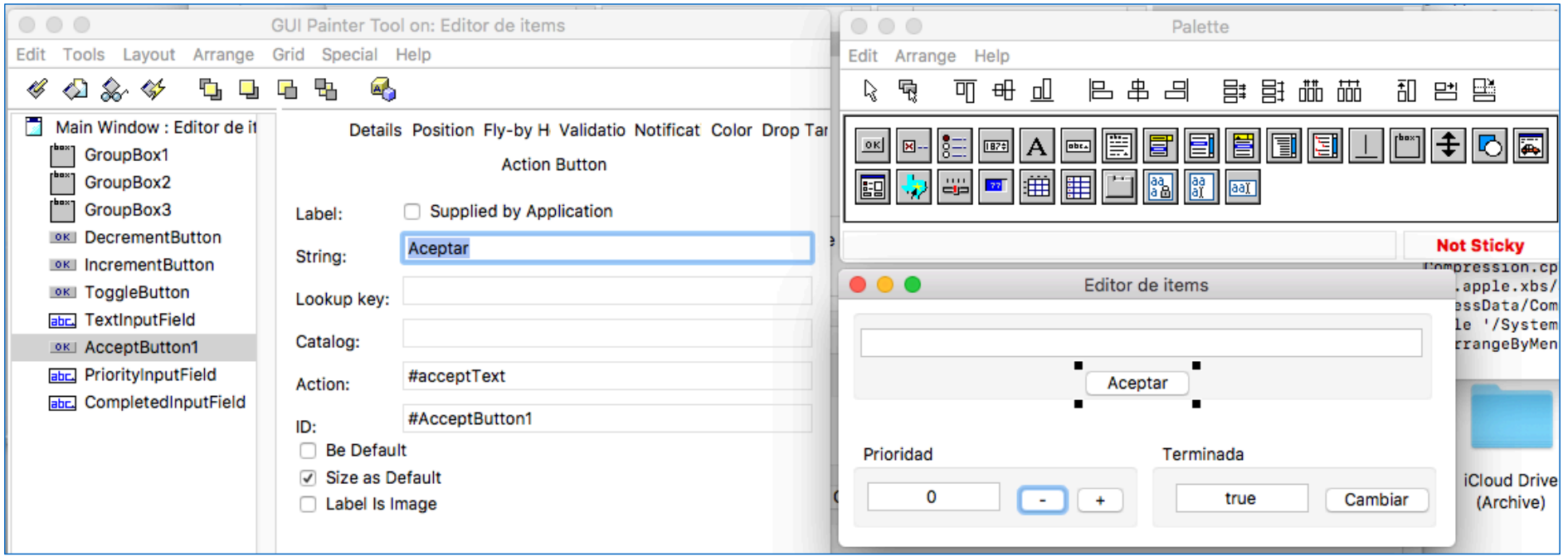



```
<resource: #canvas>
^#(#{UI.FullSpec}
    #window:
    #(#{UI.WindowSpec}
        #label: 'Editor de items'
        #bounds: #(#{Graphics.Rectangle} 512 388 962 568 ) )
    #component:
    #(#{UI.SpecCollection}
        #collection: #(
            #(#{UI.GroupBoxSpec}
                #layout: #(#{Graphics.Rectangle} 8 105 221 170 )
                #name: #GroupBox1
                #label: 'Prioridad' )
            #(#{UI.ActionButtonSpec}
                #layout: #(#{Graphics.Rectangle} 132 134 166 155 )
                #name: #DecrementButton
                #model: #decrement
                #label: '-'
                #defaultable: true )
            #(#{UI.ActionButtonSpec}
                #layout: #(#{Graphics.Rectangle} 174 134 208 155 )
```

Builder

Separate the construction of a complex object from its representation so that the same construction process can create different representations.





Mac L&F

Windows L&F

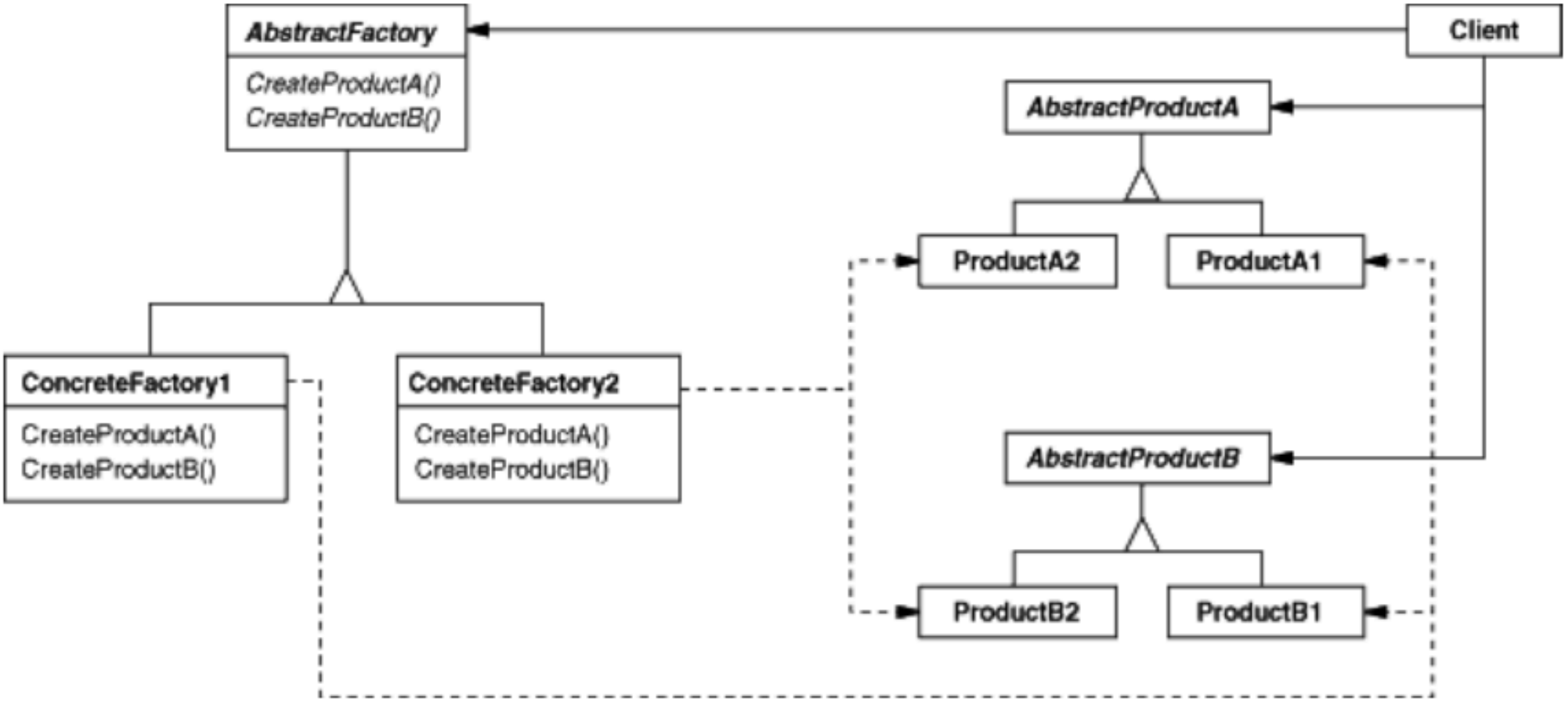
Vs

Android L&F

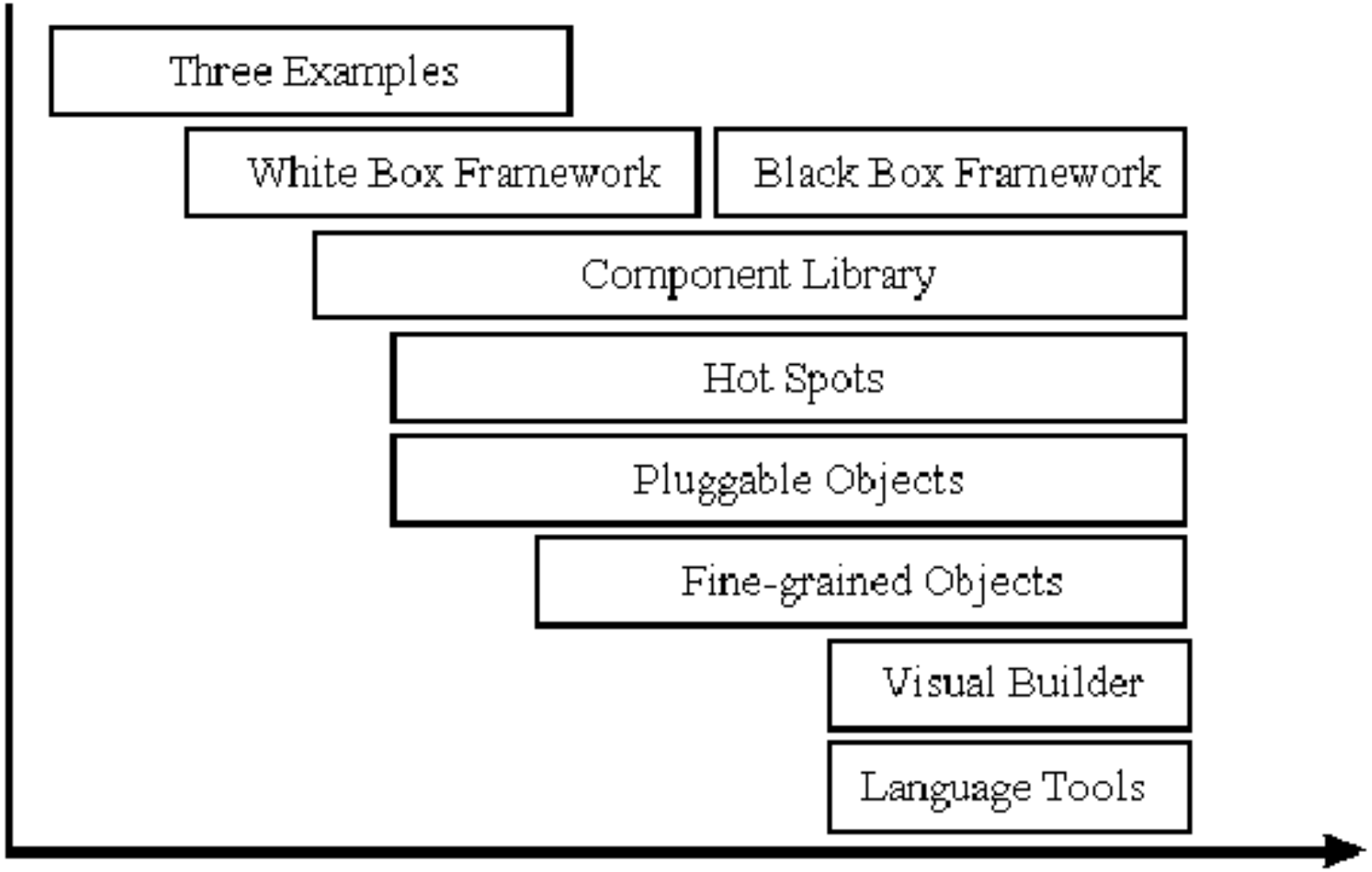
IPhone L&F

Abstract Factory

Proveer una interfaz para crear familias de objetos relacionados sin indicar sus clases concretas

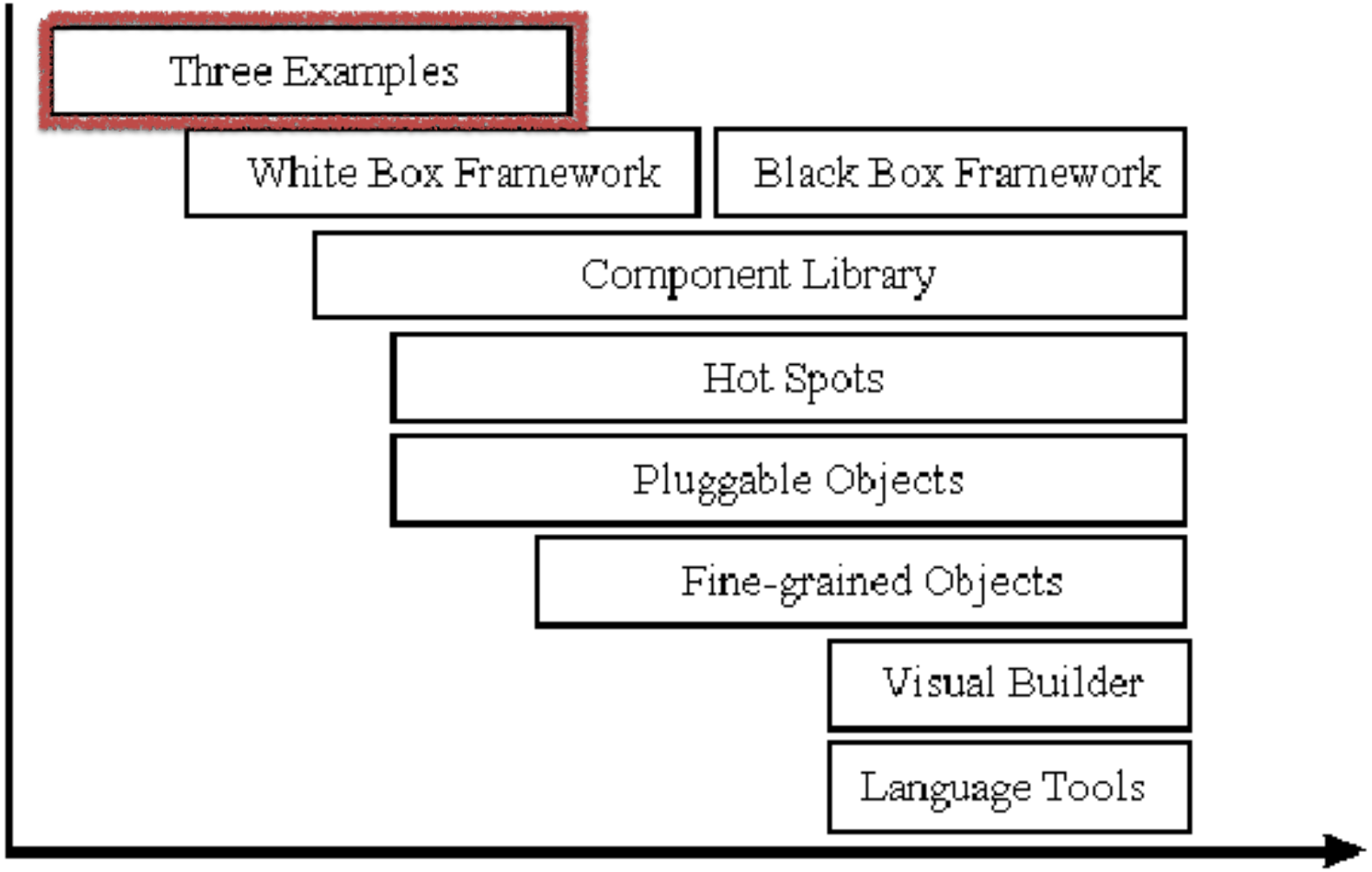


Diseño evolutivo de frameworks



De: Evolving Frameworks - Don Roberts, Ralph Johnson

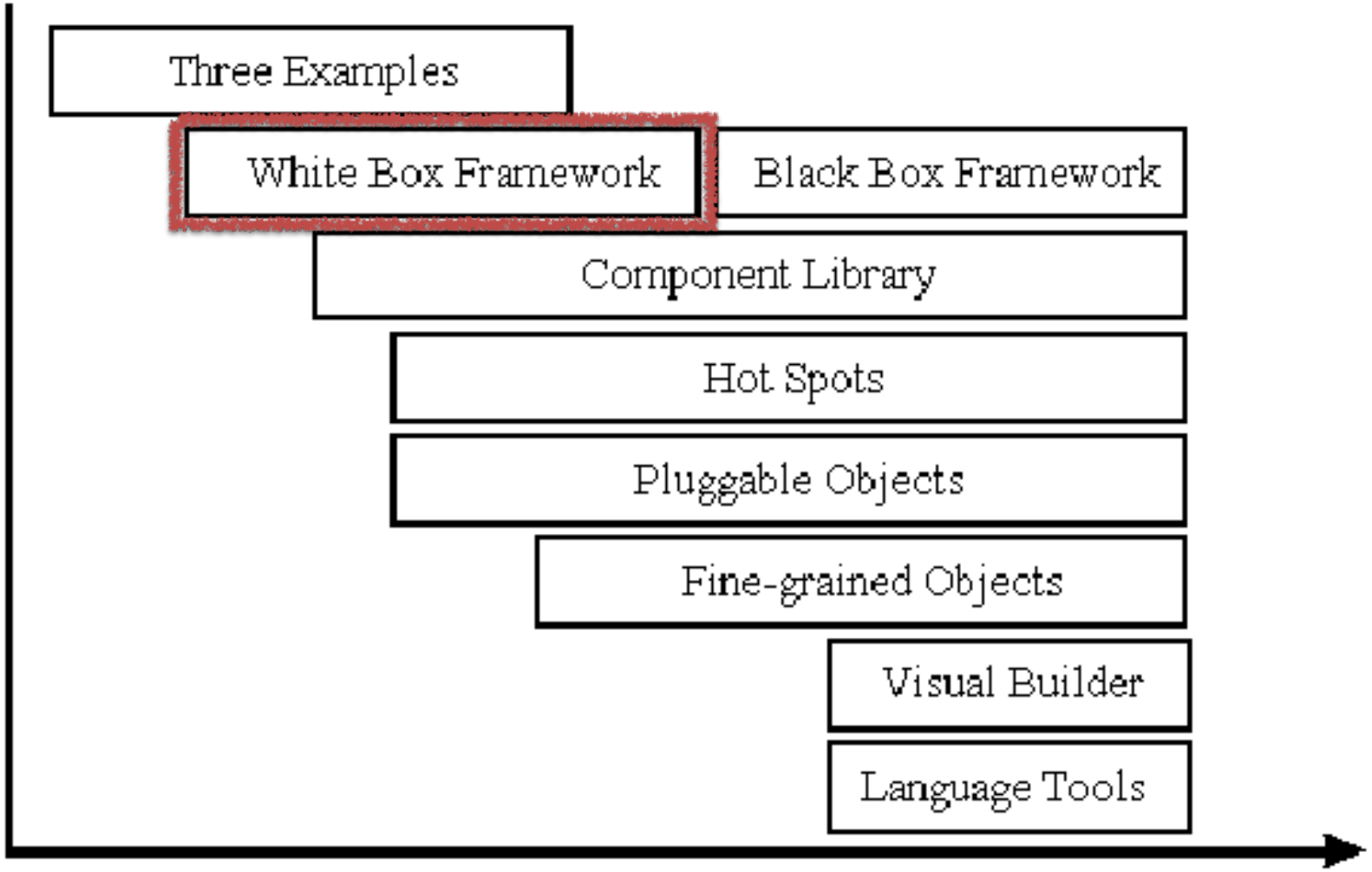
Tres ejemplos



Tres ejemplos

- ¿Por donde empezar?
  - Normalmente no puedes pensar en los hotspots si nunca hiciste una aplicación del dominio. Sobretodo, si es un framework de dominio, no de arquitectura.
- Implementa tres aplicaciones de la familia a la que apunta el framework
  - En cada nueva, diferente, aplicación, intenta reusar y generalizar.
  - Ojo con invitar muchos usuarios muy temprano – el framework sigue evolucionando.

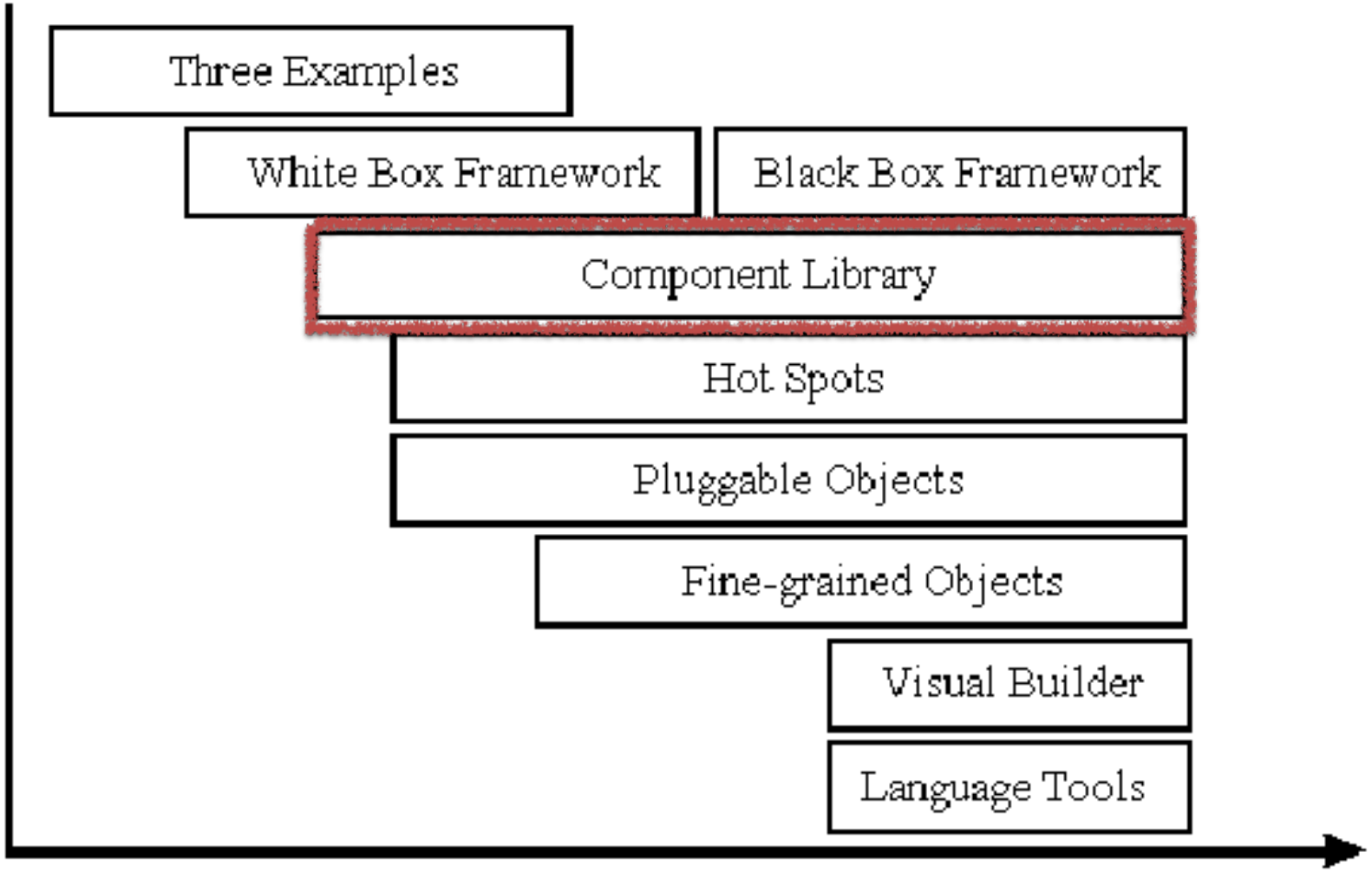
Caja blanca


Three Examples
White Box Framework
Black Box Framework
Component Library
Hot Spots
Pluggable Objects
Fine-grained Objects
Visual Builder
Language Tools

Caja blanca

- ¿Herencia o composición?
  - Herencia permite cambiar cosas que no estaba previsto que cambien pero requiere programación y conocimiento del diseño
  - Composición implica conocer todo lo que puede cambiar
  - Empieza con un framework caja blanca
    - Usa template method para capturar algoritmos comunes
    - Usa factory method cuando crees objetos
    - Programa por diferencia (subclasifica y redefine)

Librería de componentes

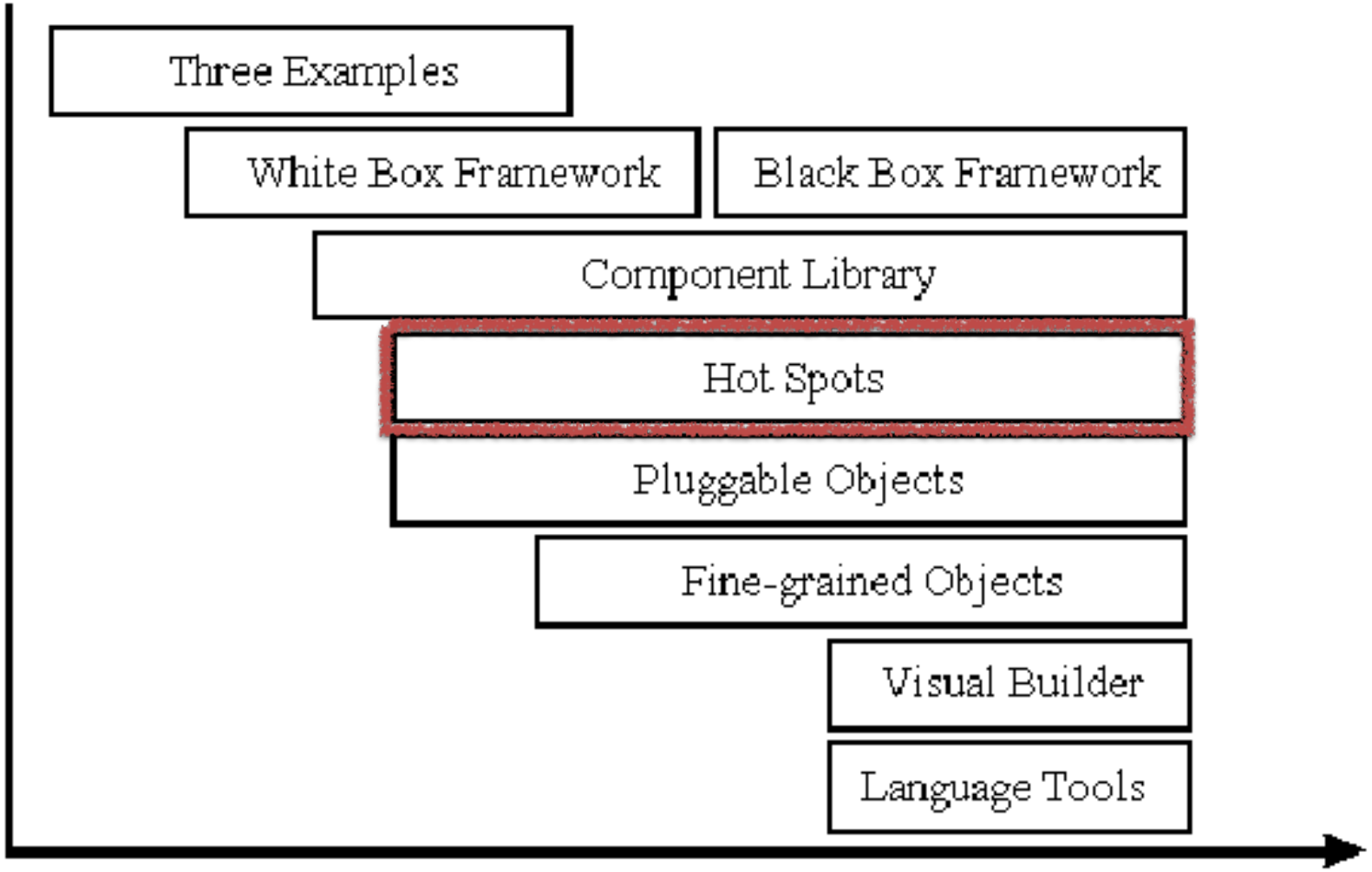


Librería de componentes

- Cada vez que se instancia el framework se implementan objetos similares
  - Los objetos listos para usar (sin programar) simplifican mucho el desarrollo y aprendizaje

- Crea una librería con los objetos mas obvios, listos para usar
  - Cada vez que hagas una aplicación, fíjate si algunos de tus objetos no pueden ir al framework
  - Con el tiempo solo dejas los que son usados por varias aplicaciones

Hot spots


Three Examples
White Box Framework
Black Box Framework
Component Library
Hot Spots
Pluggable Objects
Fine-grained Objects
Visual Builder
Language Tools

Frozen/Hot spots

- A medida que creas aplicaciones con el framework vas a encontrar escribiendo código ligeramente similar y código claramente diferente
  - Frozen spots y hotspots

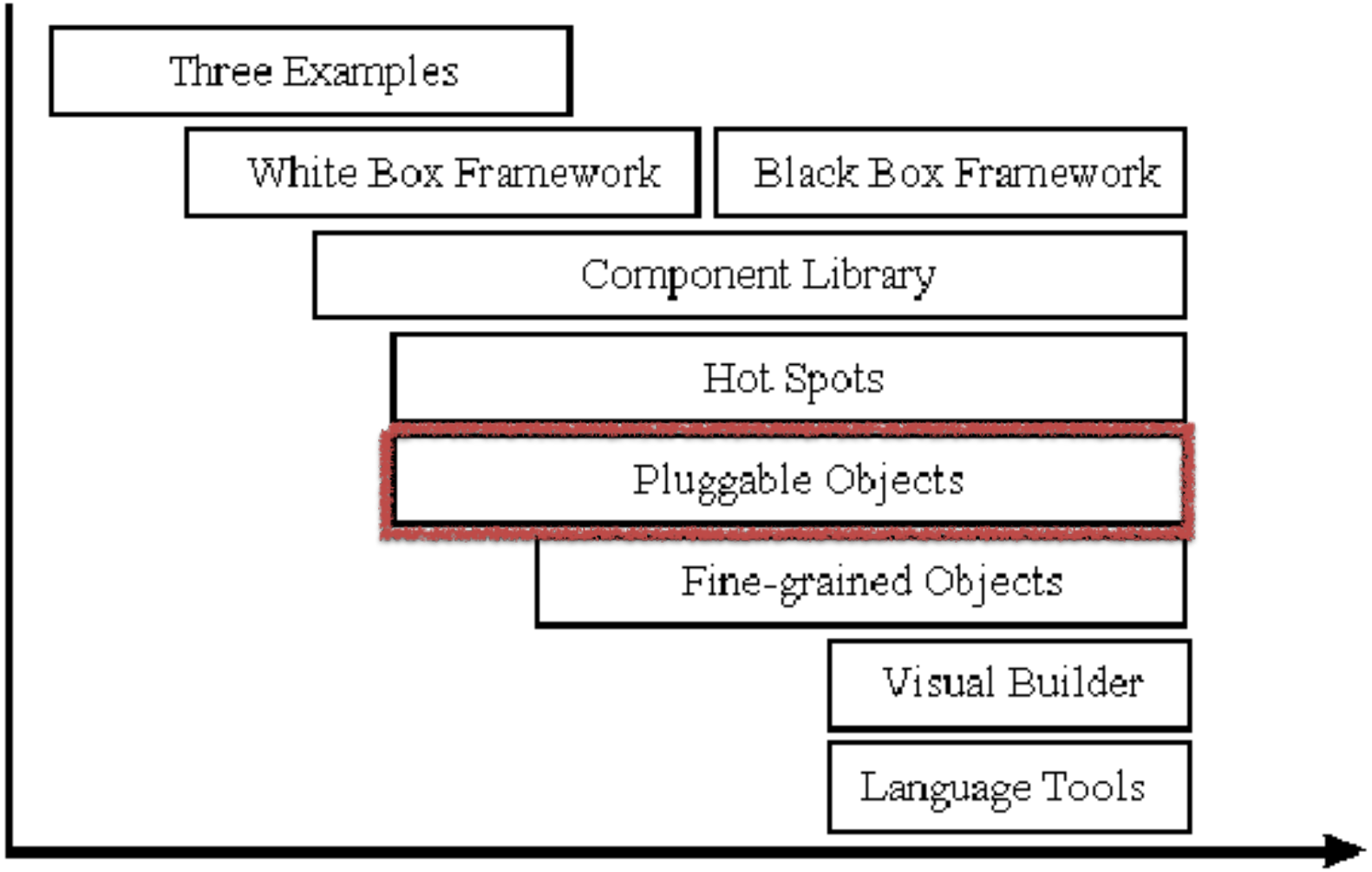
- Separa (p.ej. en paquetes) el código que varía del código que no
  - Encapsula el código que varía en métodos (p.ej. factory methods) u objetos (p.ej. strategies)
  - Si lo encapsulas en objetos, y podrás reusarlos por composición en lugar de herencia

Encapsular con patrones

Algunos patrones sirven para encapsular y separar aspectos variables

¿Qué varía?	¿Cómo lo encapsulo?
Algoritmo	Strategy, Visitor
Acciones	Command
Implementación	Bridge
Respuesta a un cambio	Observer
Interacciones entre objetos	Mediator
Que objeto se crea	Factory method, Abstract Factory, Prototype
La estructura que se construye	Builder
Algoritmo de recorrido	Iterator
Interfaces de los objetos	Adapter
Comportamiento del objeto	Decorator, State

Pluggable objects

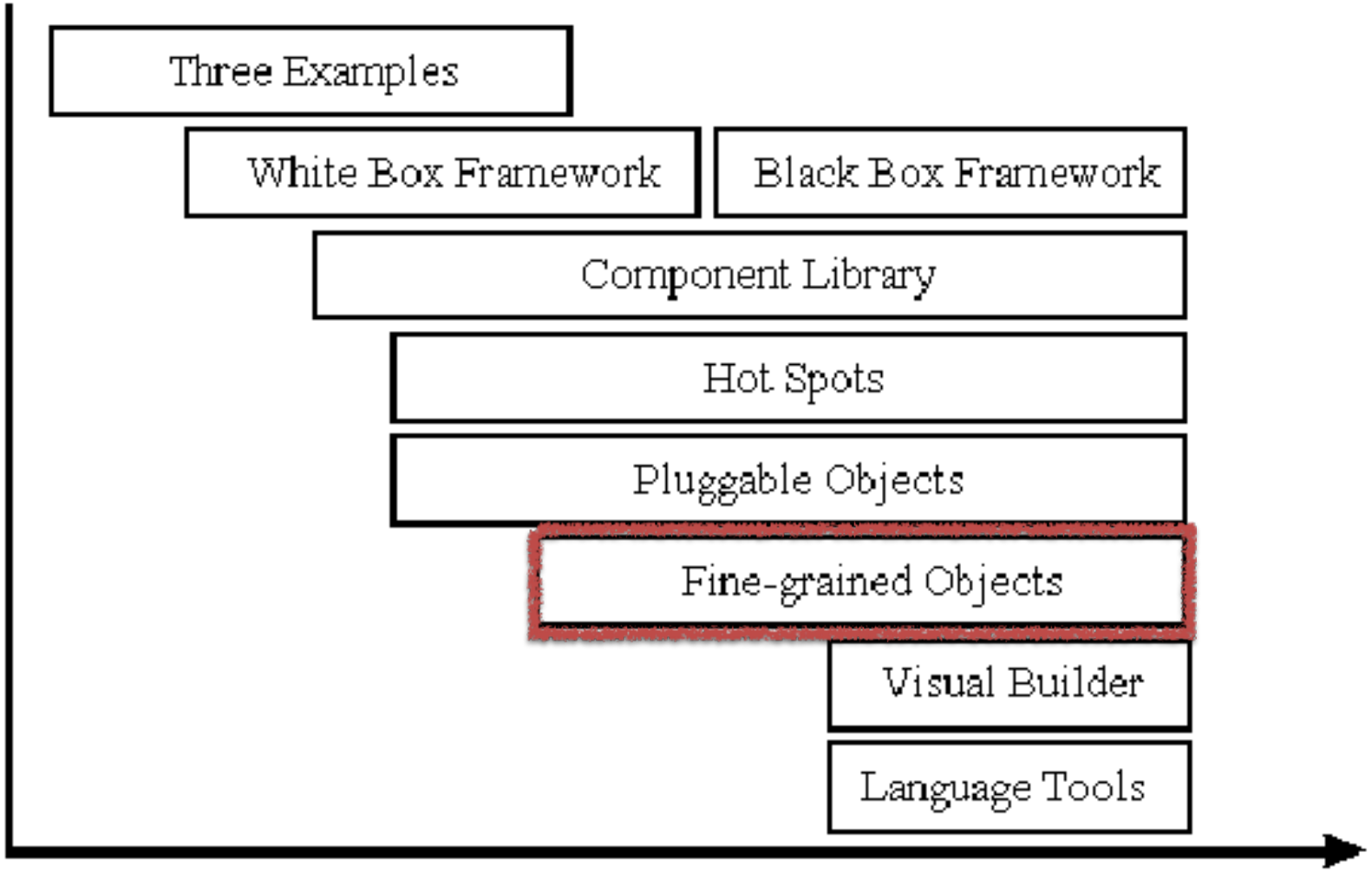


Objetos “enchufables”

- A veces, las clases que creas para distintas aplicaciones difieren de formas triviales (a quien se conectan, que le piden, cuando le avisan, etc.)

- Diseña clases adaptables
  - Se las puede configurar con mensajes para enviar, bloques a ejecutar, u otra parametrización trivial.
  - Ejemplos: BlockPluggableClipboardMonitor, …

Objetos pequeños

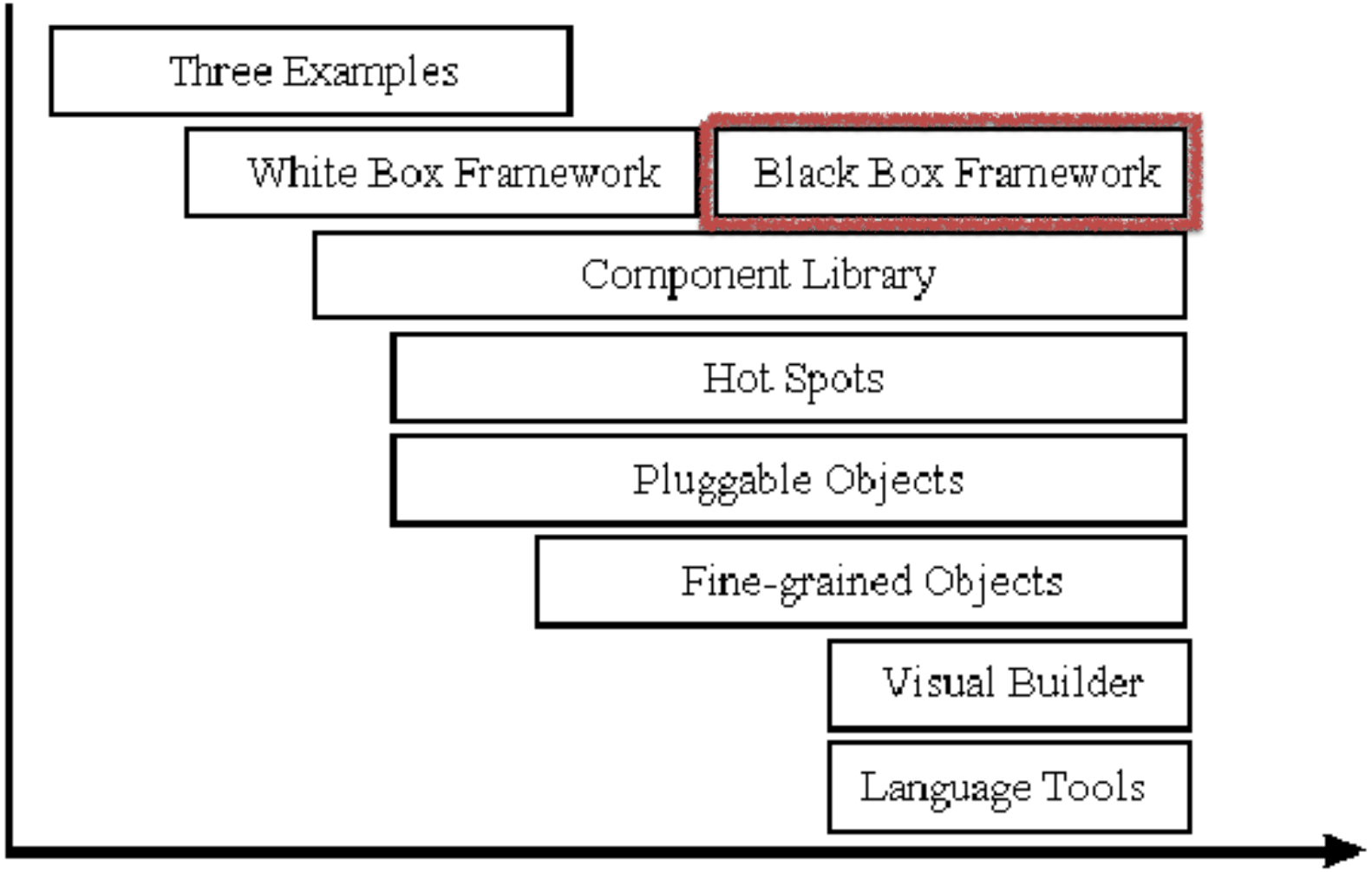


Objetos chiquitos

- Estas encapsulando tus variabilidades, refactorizando para que aparezcan nuevas clases ¿hasta donde llegas?

- Si hay un objeto que combina dos aspectos que podrían variar por separado, rómpelo en dos. Sigue hasta que no quede ninguno para romper o lo que resultase de romperlos ya no tenga sentido en el dominio.
  - Vas a tener muchos objetos.
  - Finalmente vas a tener que proveer herramientas para manejar esa complejidad, pero vas en camino a black box.

Black box

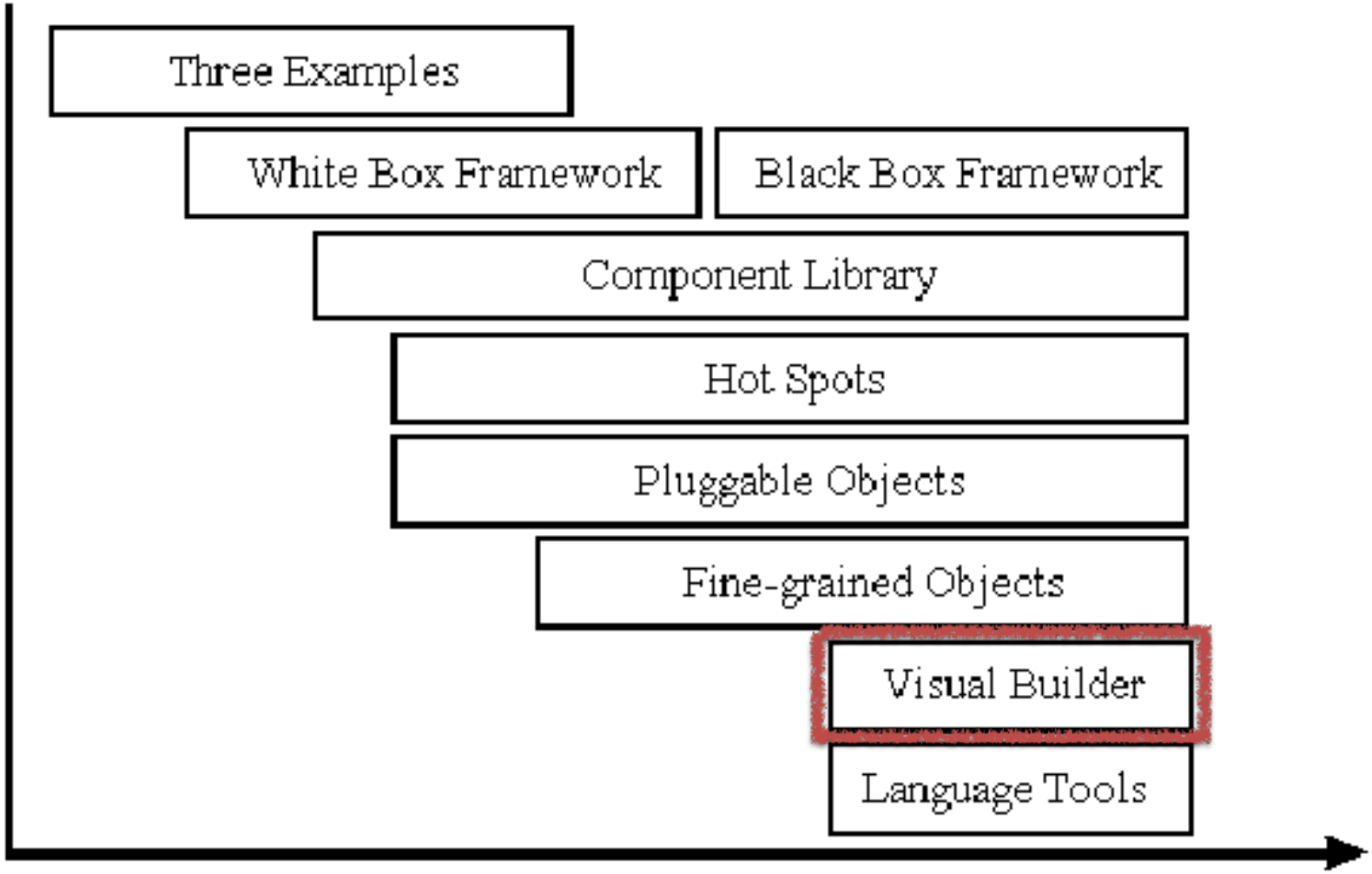

Three Examples
White Box Framework
Black Box Framework
Component Library
Hot Spots
Pluggable Objects
Fine-grained Objects
Visual Builder
Language Tools

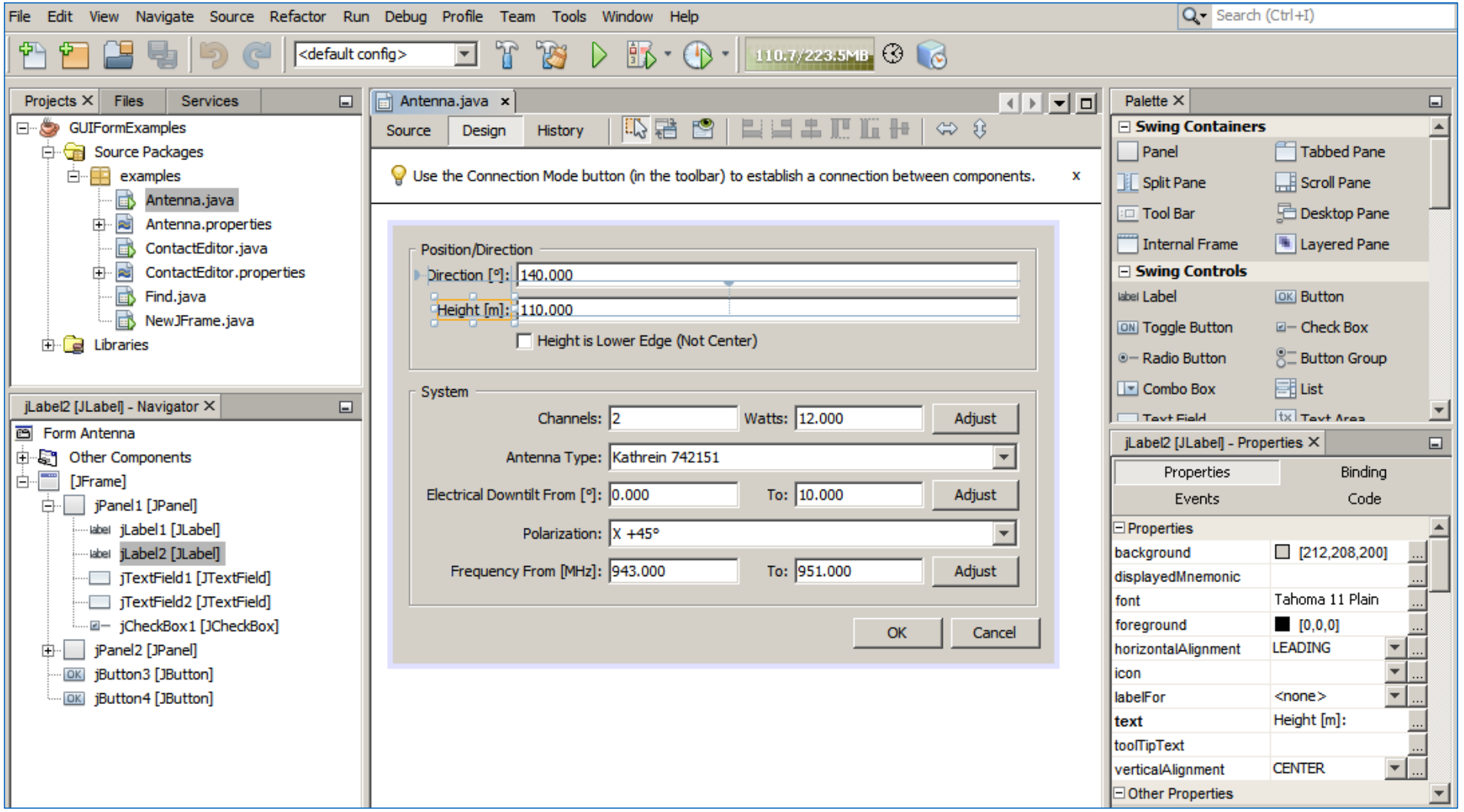
Black box

- ¿Herencia o composición? (otra vez…)
  - Ahora ya maduro tu framework
  - Tienes objetos chiquitos
  - Tienes una librería de componentes
  - El los hotspots/frozenspots están maduros

- Favorece composición a herencia – convierte herencia en composición (sobretodo, cerca de los hotpots)

Editores visuales



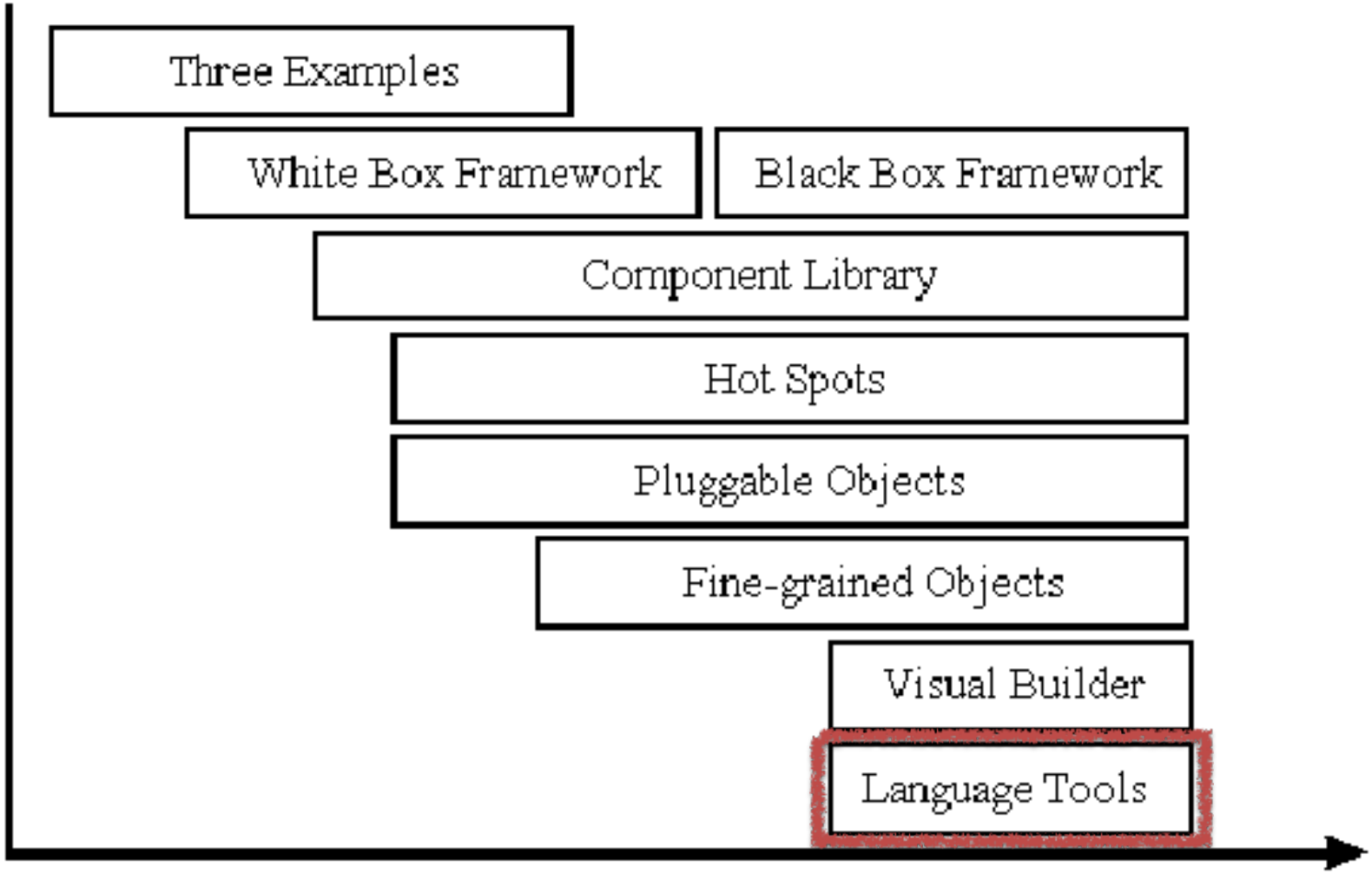

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
110.7/223.5MB
Projects
Files
Services
GUIFormExamples
Source Packages
examples
Antenna.java
Antenna.properties
ContactEditor.java
ContactEditor.properties
Find.java
NewJFrame.java
Libraries
jLabel2 [JLabel] - Navigator
Form Antenna
Other Components
[JFrame]
jPanel1 [JPanel]
jLabel1 [JLabel]
jLabel2 [JLabel]
jTextField1 [JTextField]
jTextField2 [JTextField]
jCheckBox1 [JCheckBox]
jPanel2 [JPanel]
jButton3 [JButton]
jButton4 [JButton]
Antenna.java
Source
Design
History
Use the Connection Mode button (in the toolbar) to establish a connection between components.
Position/Direction
Direction [°]: 140.000
Height [m]: 110.000
Height is Lower Edge (Not Center)
System
Channels: 2
Watts: 12.000
Adjust
Antenna Type: Kathrein 742151
Electrical Downtilt From [°]: 0.000
To: 10.000
Adjust
Polarization: X +45°
Frequency From [MHz]: 943.000
To: 951.000
Adjust
OK
Cancel
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
jLabel2 [JLabel] - Properties
Properties
Binding
Events
Code
Properties
background [212,208,200]
displayedMnemonic
font Tahoma 11 Plain
foreground [0,0,0]
horizontalAlignment LEADING
icon
labelFor <none>
text Height [m]:
toolTipText
verticalAlignment CENTER
Other Properties

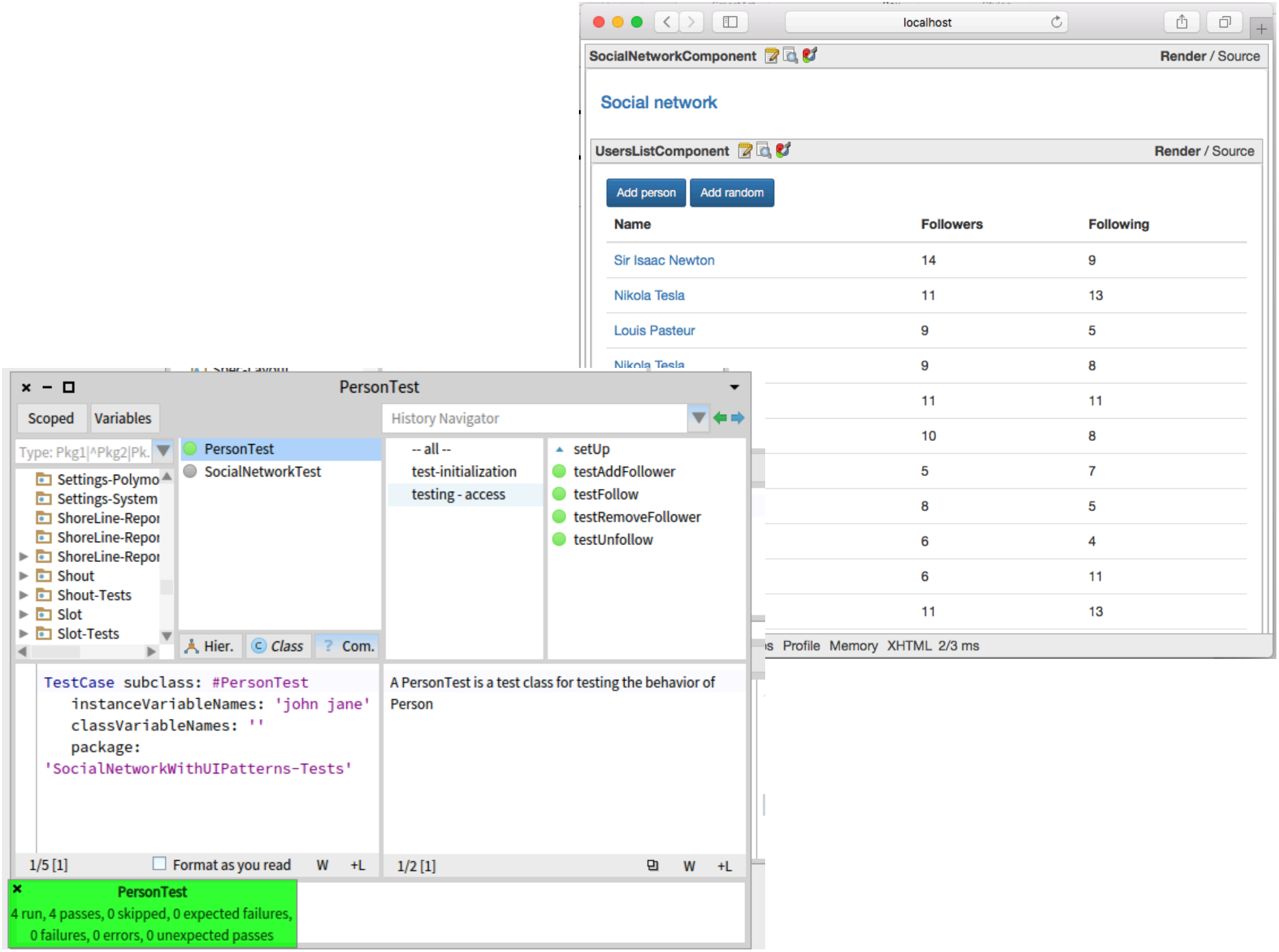
Editores visuales

- Los pasos de composición/conexión de componentes son muy similares de aplicación a aplicación. Y normalmente complejos (con muchos objetos).

- Provee herramientas visuales que permitan especificar los objetos de tu aplicación y conectarlos.
  - Ellas van a generar código por vos.
  - El patrón Builder puede ser de utilidad

Herramientas específicas




localhost
SocialNetworkComponent
Render / Source
Social network
UsersListComponent
Render / Source
Add person
Add random
Name
Followers
Following
Sir Isaac Newton 14 9
Nikola Tesla 11 13
Louis Pasteur 9 5
9 8
11 11
10 8
5 7
8 5
6 4
6 11
11 13
Profile Memory XHTML 2/3 ms
PersonTest
Scoped
Variables
History Navigator
Type: Pkg1|^Pkg2|Pk.
Settings-Polymo
Settings-System
ShoreLine-Repo
ShoreLine-Repo
ShoreLine-Repo
Shout
Shout-Tests
Slot
Slot-Tests
PersonTest
SocialNetworkTest
-- all --
test-initialization
testing - access
setUp
testAddFollower
testFollow
testRemoveFollower
testUnfollow
Hier.
Class
Com.
TestCase subclass: #PersonTest
instanceVariableNames: 'john jane'
classVariableNames: ''
package:
'SocialNetworkWithUIPatterns-Tests'
A PersonTest is a test class for testing the behavior of Person
1/5 [1]
Format as you read
W
+L
1/2 [1]
W
+L
PersonTest
4 run, 4 passes, 0 skipped, 0 expected failures,
0 failures, 0 errors, 0 unexpected passes

Herramientas específicas

- Ahora que tus objetos se crean con el builder, hay un gap grande entre lo que tu piensas de una aplicación y lo que el inspector y el debugger te muestran.

- Crea inspectores y debuggers a medida
  - Esconden todos los objetos que no hacen a la idea de la aplicación
  - Esconden aquellos objetos que raramente se rompen
  - Te permiten efectuar acciones interesantes, complejas y repetitivas.
  - Tienen una puerta para acceder a las herramientas originales (las que muestran todo)