IIC2113 - Diseño Detallado de Software

CLASE 5 - Ingeniería inversa y paradigmas de programación

Pontificia Universidad Católica de Chile
2020-2

Índice

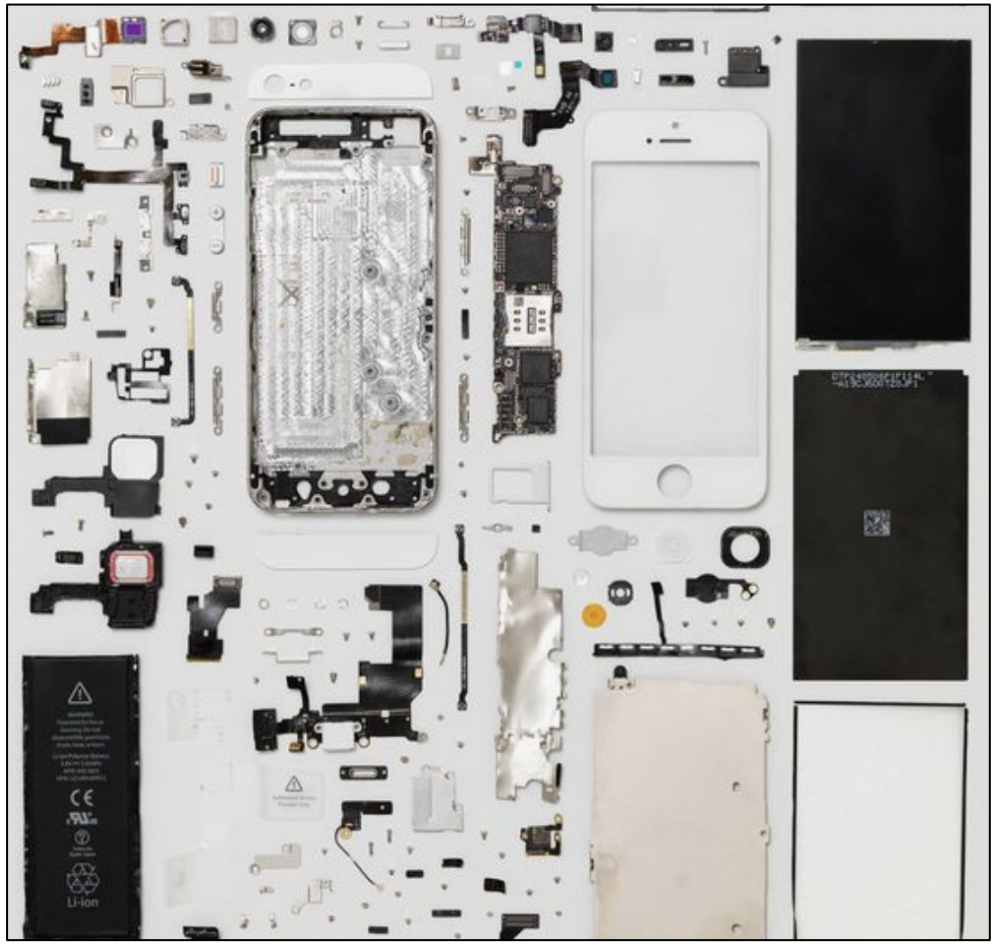
01. Ingeniería Inversa

¿Qué es?

Ingeniería Inversa o retroingeniería

Proceso de analizar la estructura, funcionamiento, características y, en general, los fundamentos técnicos de un sistema o dispositivo ya sea mecánico o electrónico, e incluso un programa computacional. El término se puede aplicar tanto a productos de software, hardware, y hoy en día a todo tipo de productos. Este proceso se realiza porque **no se tiene acceso a las especificaciones ni documentaci´on del producto**.

Se originó como una práctica dada entre empresas competidoras (caso Apple vs Samsung).



Ingeniería inversa en el software

Es el proceso de analizar un programa con el fin de crear una representación de más alto nivel. Generalmente se aplica de forma interna sobre un ***producto legacy****, por lo que pareciera ser ajeno a la empresa.*

Nivel de extracción

Se refiere a la sofisticación del diseño extraído de un software. A mayor nivel, más información se puede extraer. De mayor a menor nivel:

- Diseño de procesos
- Diseño estructural
- Modelo del sistema
- Relación entre componentes

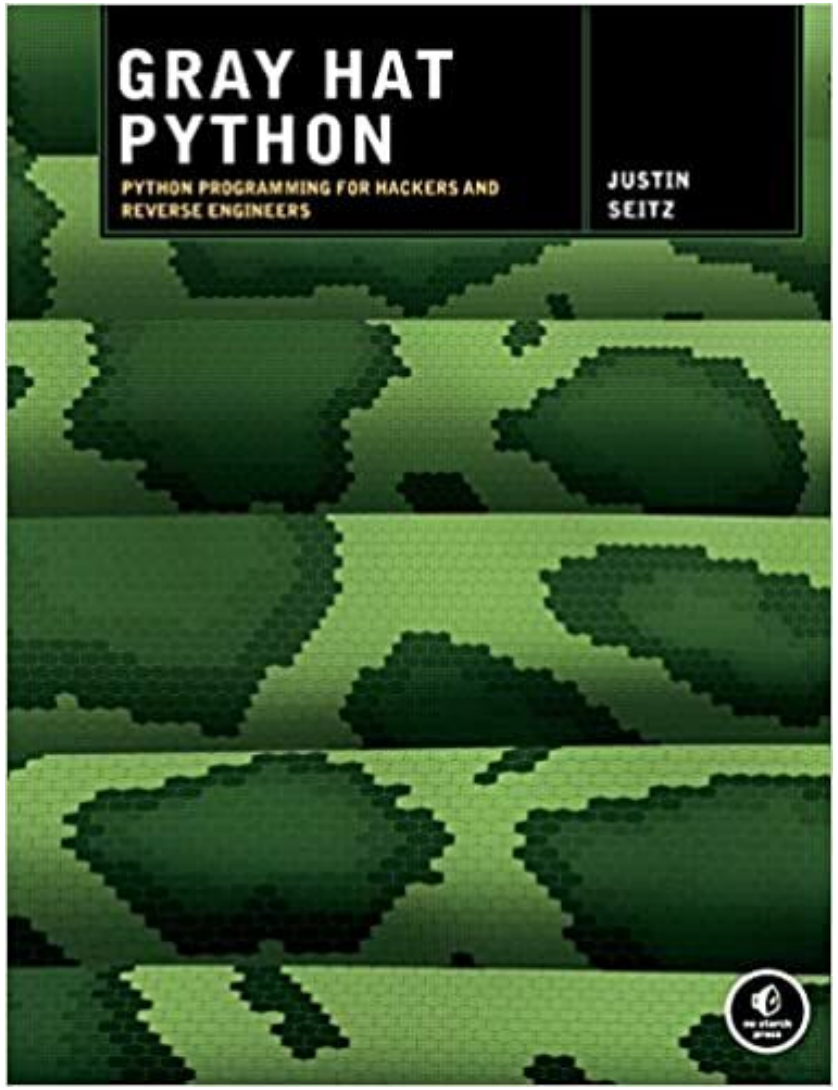
Completitud

Dado un nivel de extracción, se refiere a la cantidad de detalle que se puede obtener. Generalmente es inversamente proporcional al nivel de extracción.

Ingeniería inversa en el software

Ventajas:

- Reducir la complejidad del sistema
- Recuperar o actualizar información
- Identificar el alcance de un producto
- Facilitar la reutilización
- Análisis de vulnerabilidades

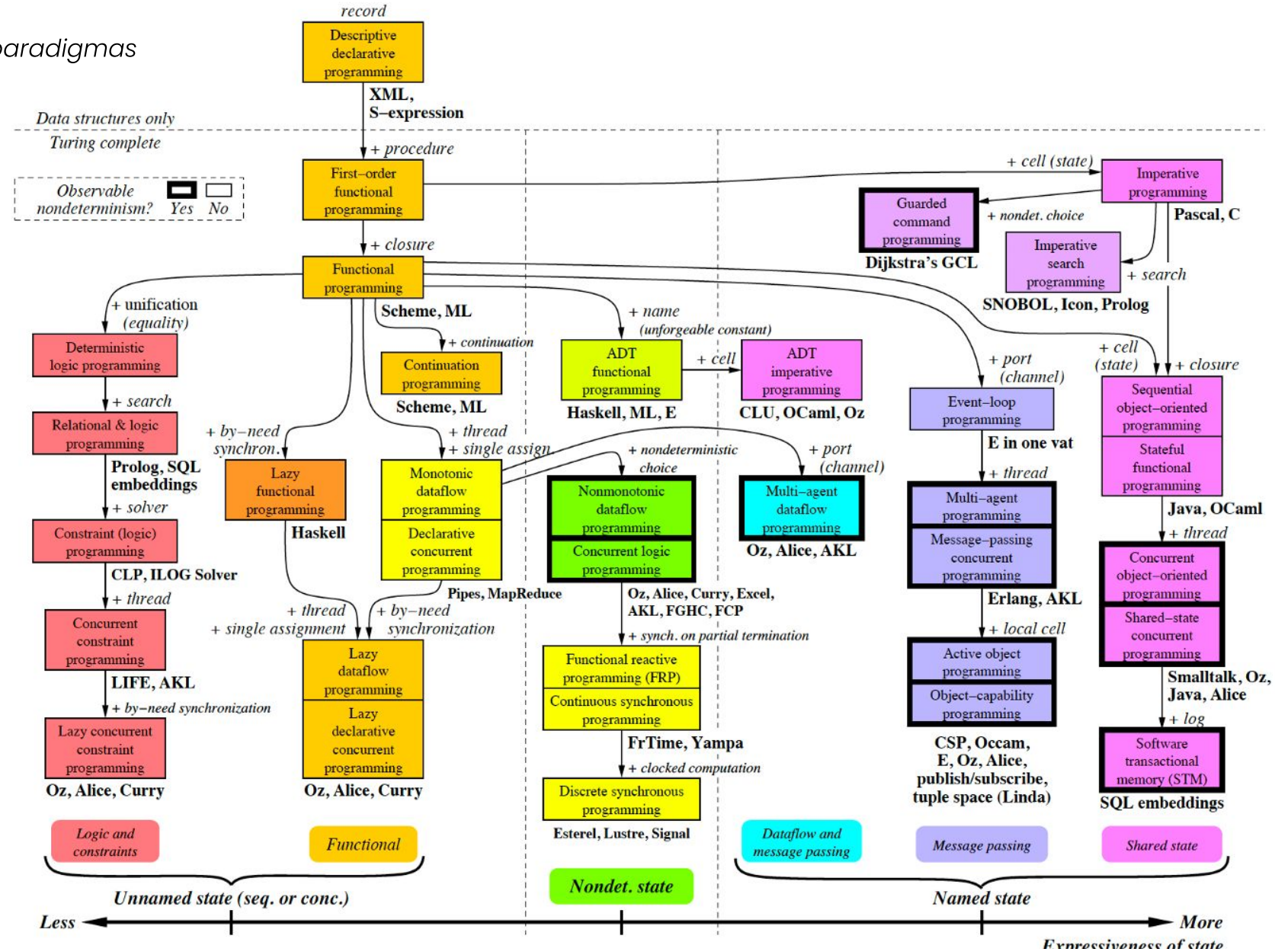


https://www.amazon.com/Gray-Hat-Python-Programming-Engineers/dp/1593271921

Ejemplo de Ingeniería inversa sobre app de pokemon go

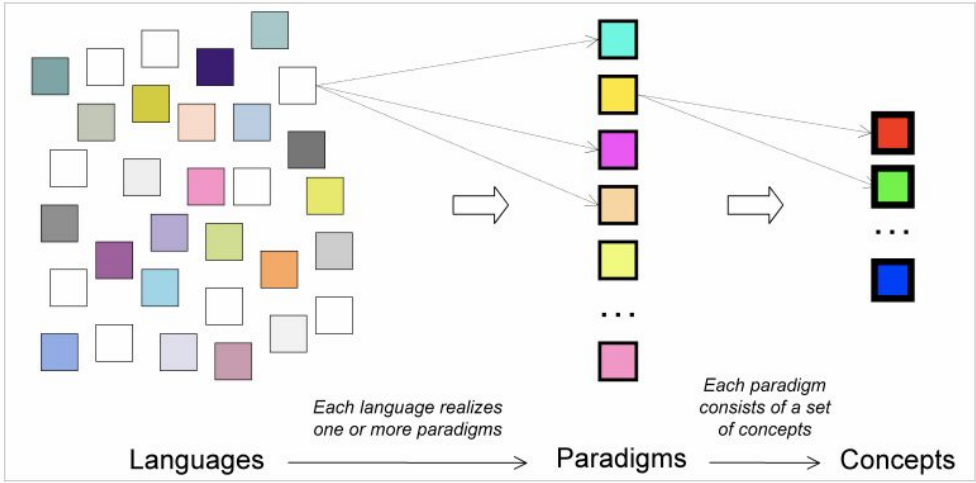
https://www.fabernovel.com/en/article/tech-en/unbundling-pokemon-go-2

02. Paradigmas de programación



"Programming Paradigms for Dummies," https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf

¿Qué son los paradigmas de programación?



Son una forma de clasificar los lenguajes de programación en base a sus funcionalidades. Se pueden clasificar en base a sus modelos de ejecución, a cómo está organizado el código o sobre el estilo/gramática. Los más comunes son:

- **Imperativa**: el programador indica cómo cambian los estados.
  - Programación por **procedimientos** (procedural).
  - Orientado a **objetos** (OOP).
- **Declarativa**: el programador declara propiedades del resultado, pero no los computa.
  - Programación **Funcional**.
  - Programación **Lógica**.
  - Programación **Reactiva**.
  - **Optimización** (mathematical programming)

1. Programación por procedimientos

*Puede darse a alto o bajo nivel. Cuando se aplica como técnica a alto nivel se le conoce como **programación funcional.***

```haskell
solveRPN :: (Num a, Read a) => String -> a
solveRPN = head . foldl foldingFunction [] . words
    where   foldingFunction (x:y:ys) "*" = (x * y):ys
            foldingFunction (x:y:ys) "+" = (x + y):ys
            foldingFunction (x:y:ys) "-" = (y - x):ys
            foldingFunction xs numberString = read numberString:xs
```

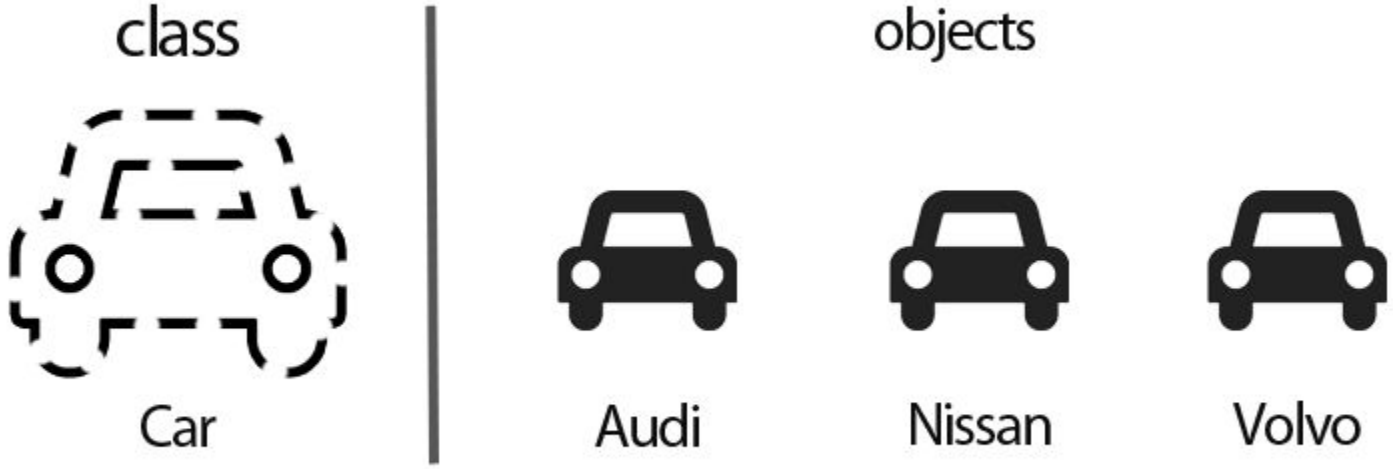
Ejemplo en Haskell http://learnyouahaskell.com/functionally-solving-problems

Lenguajes puramente funcionales: Haskell, Miranda, Erlang.

Lenguajes que soportan procedimientos y funciones: C++, C#, StarBasic, Pascal, Python, Java, Javascript, Kotlin,Visual Basic .NET.

2. Prog. Orientada a Objetos

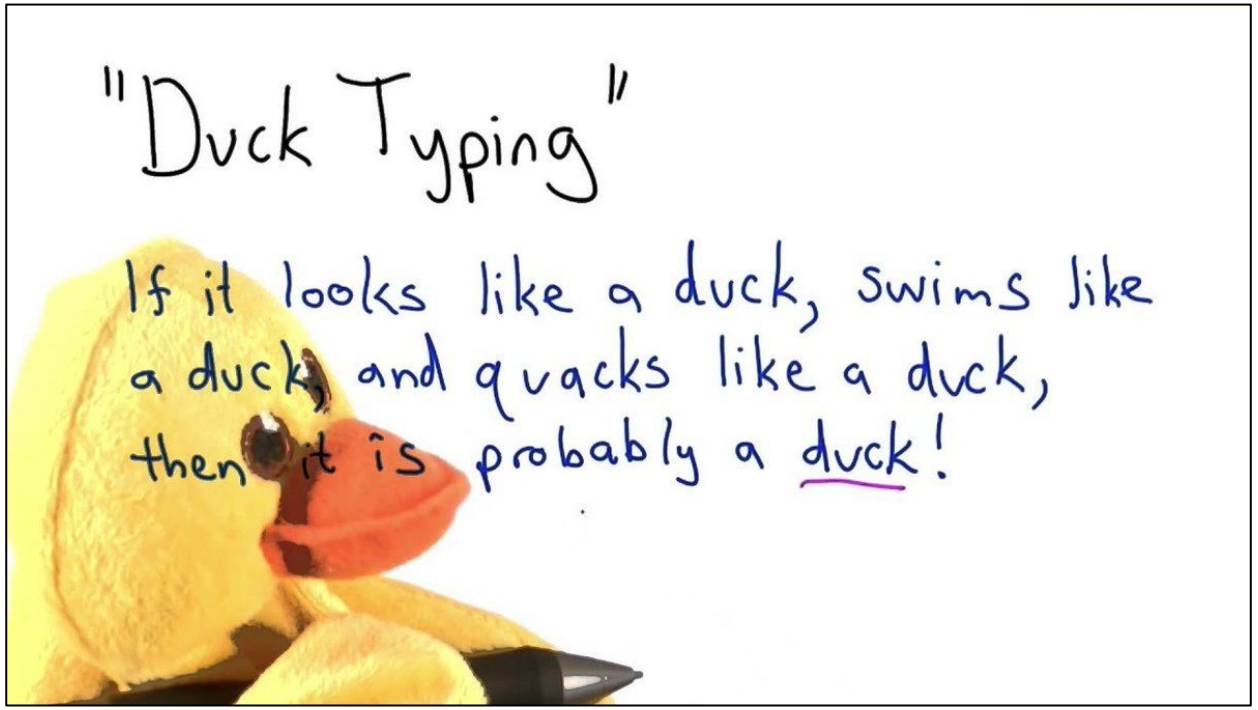
*Los objetos emulan una instancia real del modelo del problema que se está resolviendo. Se basa en **herencia**, **cohesión**, **abstracción**, **polimorfismo**, **acoplamiento** y **encapsulamiento**.*



Lenguajes puramente OOP: Smalltalk.

Lenguajes que soportan OOP: C++, C#, Python, Java, Javascript, Swift,Visual Basic .NET, Ruby.

2. Programación Orientada a Objetos - Duck typing



Duck typing hace referencia a los programas escritos en algún lenguaje orientado a objetos principalmente, en que los objetos pasados a una función o método soporta todo tipo de atributos en tiempo de ejecución.

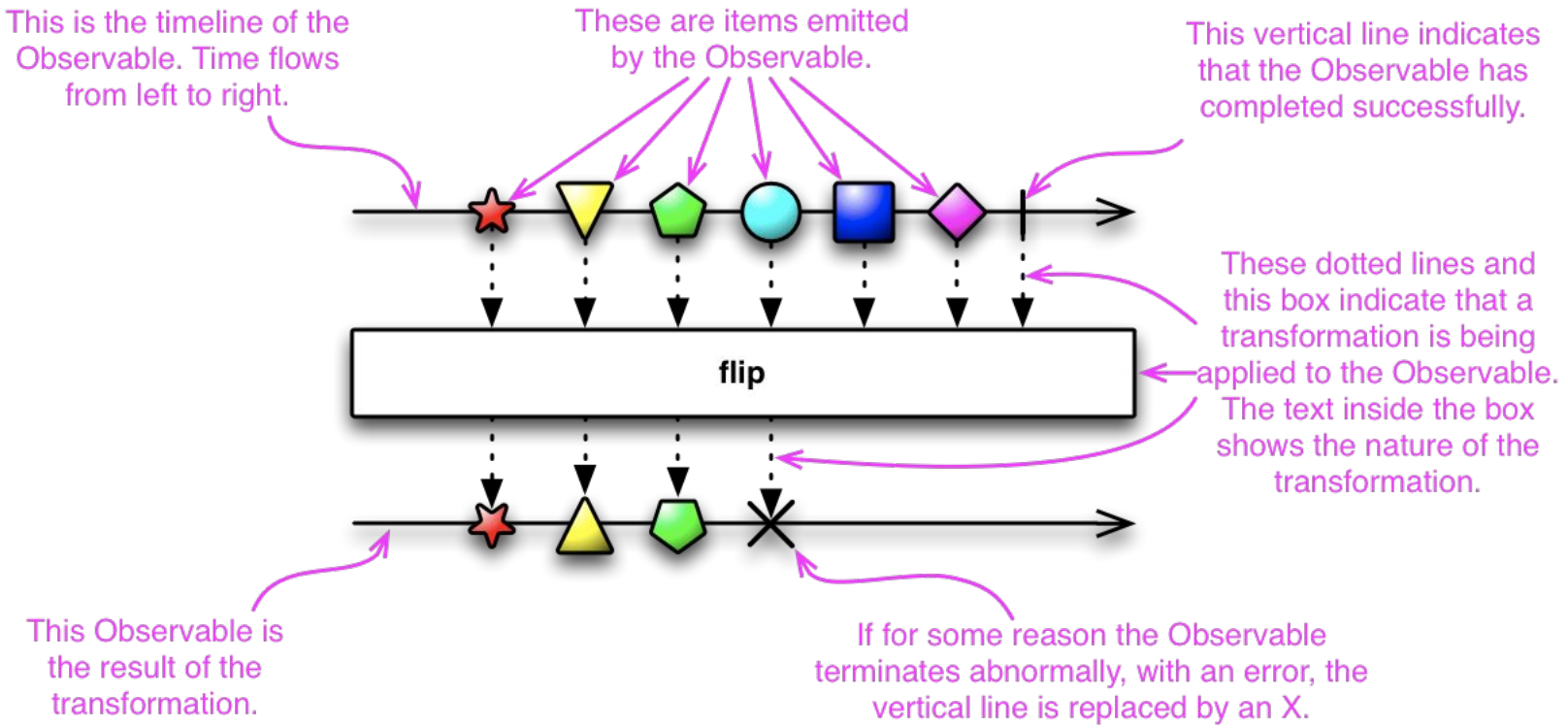
A diferencia del **tipado estático**, duck typing es dinámico y solo verifica el tipo en el momento de ser accedido.

Lenguajes que soportan Duck Typing: Python, Ruby.

3. Programación Reactiva

Declarativo orientado al manejo asíncrono de streams de datos y la propagación de los cambios que introducen. Facilita la comunicación entre múltiples threads.

Se basa en una adaptación del patrón **Observer**.



En general define:

- Observable
- Observer
- Schedulers
- Backpressure

Programación multiparadigma

En general los lenguajes de alto nivel tienden a **soportar más de un paradigma.** De hecho, a pesar de que un lenguaje en su core no soporte algún paradigma, suelen existir librerías y/o frameworks que permiten a distintos lenguajes soportarlo. Por ejemplo hoy en día muchos lenguajes soportan programación reactiva gracias a librerías.

Entonces da lo mismo cuál lenguaje use, si todo se puede importar → **No**. Dependerá de:

- El tipo de **implementación** del lenguaje.
- El tipo de **soporte** que entrega al paradigma que necesitas.
- Al final los **lenguajes de propósito general** tienen sus ventajas y desventajas.

Domain Specific Languages (DSL)

Si nos enfocamos en un modelo en particular, pueden existir lenguajes que representen de mejor y única manera ese modelo. Un **Lenguaje Específico de Dominio** (DSL) es un lenguaje de programación o especificación orientado a resolver problemáticas en un dominio muy acotado.

- **DSL Interno** (Embedded DSL): Se escriben dentro de un lenguaje existente que lo contiene.
  - Rake y Rspec son DSL internos de Ruby.
  - .NET también ofrece distintos tipos de DSL y herramientas de Extensibility.
  - Es común escribir DSL internos para problemas particulares, con tal de tener interfaces fluidas.
- **DSL Externo**: Tienen su propio lenguaje, pero necesitan ser **interpretados** por otros.
  - SQL, HTML, CSS, Sass, XML, UML, Latex.

Domain Specific Languages (DSL)

Ventajas frente a los lenguajes de propósito general:

- Tienen mayor legibilidad
- Tienen menor verbosidad
- Su uso implica menor probabilidad de error
- Pueden ser analizados y usados por especialistas del dominio sin experiencia en programación

Desventajas:

- Overhead extra para aprender
- Aumenta la complejidad
    - Diseñar un DSL interno puede ser complejo

Clase 5 - ing. inversa y paradigmas

Bibliografía

- Pressman, R. S. (2009). Software Engineering: A Practitioner's Approach
- Domain Specific Languages. by Martin Fowler, with Rebecca Parsons. 2010.

IIC2113 - Diseño Detallado de Software

Fernanda Sepúlveda - mfsepulveda@uc.cl

Pontificia Universidad Católica de Chile
2020-2