2.2 Metodología en el diseño del lenguaje

PyLP 361075 Carlos Eduardo Sánchez Torres

24 de febrero de 2021

La metodología, cuyo significado aplica en el desarrollo de software moderno, es un proceso marcado por unas disciplinas (requerimientos, diseño, implementación, pruebas y despliegue) y conjunto de buenas prácticas a su alrededor de manera iterativa e incremental -versionamiento- para cumplir ciertas expectativas marcadas por su filosofía o mantra, guiada por personas experimentadas: evitando la crisis del software. Nos centraremos en la toma de decisiones o diseño en la primeras iteraciones para crear un lenguaje usable: legible, escribible, confiable...

Para entender las decisiones de los líderes de equipos técnicos, entendamos el trabajo previo -estado del arte-: la dialéctica -tesis, contratesis, síntesis- crea el devenir de la técnica, llevándolo hacia el absoluto, una historia arete, más cercana al lenguaje humano. El primer lenguaje de alto nivel sobre circuitos electrónicos -computadoras modernas- fue «Plankalkül» creado por Konrad Zuse en 1945 cuya motivación fue proveer formalización para cualquier procedimiento computacional, el siguiente fue «Flow Diagrams» por Herman H. Goldstine y John von Neumann en 1946, cuyo objetivo fue representar algoritmos de una manera a un nivel más alto que el lenguaje máquina, y el tercero «Composition» por Haskell B. Curry en 1948 logró estructurar programas -funciones puras- [7], y así sucesivamente proveían capas de abstracción con tendencia a ser cercana a los lenguajes humanos -naturalmente, el inglés-, sus decisiones se fundamentan en simplificar la creación de software -sea inteligente o no- para un dominio y mejorar el rendimiento del mismo desde su contexto dado -alcance, presupuesto y tiempo-, inclinándose sobre uno u otro.

Casos de estudio

Go, lenguaje compilado y de propósito general inspirado en C, liberado como «open source» en 2009 por Google, ha sido liderado por Robert Griesemer, Rob Pike y Ken Thompson, cuyo mantra es simplficar -evitar lo verboso- el desarrollo de aplicaciones para servidor multi-núcleo repensando la orientación a objetos de C++ y Java. Nació de la síntesis de los lenguajes interpretados, simplificadores y lentos -Python, JavaScript, Ruby, ...- contra los lenguajes rápidos y verbosos como C++ y Java. Los hijos de C y Unix dominaron el mercado, así como la orientación a objetos desde Simula. Java más limpio que C++, fue elegido para ser enseñado principalmente en muchas universidades, expandiendo su uso. Ellos se usan porque funcionan bien en proyectos de amplia escala. Lo malo consiste en su falta de adaptación en los tiempos que corren: grandes tiempos de compilación, complicaciones de entorno de trabajo y su pobre adaptación de servidores en la nube con multi-núcleos [3] [4].

Listing 1: «Hello World» en Go.

```
package main
import "fmt"
func main() {
    fmt.Println("Hello World")
}
```

Listing 2: Concurrencia inspirada por comunicación secuencial de procesos de Tony Hoare

```
package main

import (
        "fmt"
        "time"
)

func say(s string) {
        for i := 0; i < 5; i++ {
                time.Sleep(100 * time.Millisecond)
                fmt.Println(s)
        }
}

func main() {
        go say("world")
        say("hello")
}
```

Listing 3: En Go decidieron eliminar la herencia. Y en su lugar usar completamente la composición.

```
type Figure struct{
    name string
}

func (figure Figure) getName() string {
    return figure.name
}

type Circle struct{
    Figure
}

type Square struct{
    Figure
}
```

Wolfram Language, lenguaje comercial y simbólico, diseñado por Stephen Wolfram y creado en 1988, cuyo mantra es ser un lenguaje inteligente buscando usar todo el conocimiento científico para científicos [5] [6]. Nació de la síntesis como lenguaje computacional, de la suma de los lenguajes de propósito general -C, Lisp y C++- más entidades inyectadas -base de datos-: añadiendo semántica al código científico. Se puede afirmar que Wolfram Language es un lenguaje de programación por ser Turing Completo: es posible implementar regla 110 [8] [9].

Listing 4: «Hello World» en Wolfram Language.

```
"Hello World"
```

Listing 5: Una lista de los planetas. Donde el lenguaje inyecta el conocimiento.

```
EntityList[EntityClass["Planet", All]]
```

Listing 6: Proporción entre la masa de Jupiter y la Tierra.

```
Entity["Planet", "Jupiter"]["Mass"]/Entity["Planet", "Earth"]["Mass"]
```

Referencias

[1] Tucker, A. B., & Noonan, R. (2007). Programming languages: Principles and paradigms. In Programming languages: Principles and paradigms. New York: McGraw-Hill Higher Education.

[2] Sebesta, R. W. (2016). Concepts of programming languages. In Concepts of programming languages. Upper Saddle River: Pearson.

[3] 2016. Frequently Asked Questions (FAQ) - The Go Programming Language. Golang.org. Retrieved February 23, 2021 from https://golang.org/doc/faq#go_or_golang

[4] Rob Pike. 2010. Another Go at Language Design. Retrieved from https://web.stanford.edu/class/ee380/Abstracts/100428-pike-stanford.pdf

[5] 2019. Why Wolfram Tech Isn't Open Source—A Dozen Reasons—Wolfram Blog. Wolfram.com. Retrieved February 24, 2021 from https://blog.wolfram.com/2019/04/02/why-wolfram-tech-isnt-open-source-a-dozen-reasons/

[6] 2019. What We've Built Is a Computational Language (and That's Very Important!)—Stephen Wolfram Writings. Stephenwolfram.com. Retrieved February 23, 2021 from https://writings.stephenwolfram.com/2019/05/what-weve-built-is-a-computational-language-and-thats-very-important/

[7] Knuth, Donald E.; Pardo, Luis Trabb. Early development of programming languages. Encyclopedia of Computer Science and Technology. Marcel Dekker. 7: 419–493.

[8] Matthew Cook. Universality in Elementary Cellular Automata. Retrieved February 24, 2021 from https://wpmedia.wolfram.com/uploads/sites/13/2018/02/15-1-1.pdf

[9] 2021. Rule 110. Wolfram.com. Retrieved February 24, 2021 from https://mathworld.wolfram.com/Rule110.html