Go como Lenguaje de Diseño:
Reducción de la Complejidad Accidental en Sistemas Concurrentes

AUTORES Valentín Torassa Colombero
Santiago Enrique Roatta · María Eugenia Casco
CAETI – Universidad Abierta Interamericana

PROBLEMA

El desarrollo de sistemas concurrentes introduce una **complejidad accidental** que no proviene del dominio del problema, sino de las herramientas utilizadas para implementarlo.
En la práctica, esta complejidad se manifiesta como:

- Estados compartidos difíciles de razonar
- Ejecuciones no deterministas
- Errores tardíos y difíciles de reproducir
- Fallos con impacto en mantenibilidad y seguridad

HIPÓTESIS

El diseño del lenguaje de programación **influye directamente** en la complejidad del software.

Go fue diseñado deliberadamente para **reducir la complejidad accidental,** limitando el espacio de diseño posible y favoreciendo modelos mentales simples y explícitos.

¿QUÉ ES COMPLEJIDAD ACCIDENTAL?

- Complejidad esencial: propia del dominio
- Complejidad accidental: introducida por el lenguaje, las abstracciones y las herramientas

GO COMO LENGUAJE DE DISEÑO

Go adopta decisiones de diseño opinionadas:

- Lenguaje pequeño y sintácticamente simple
- Pocas formas de hacer lo mismo
- Reglas claras y predecibles
- Restricciones deliberadas para evitar diseños frágiles

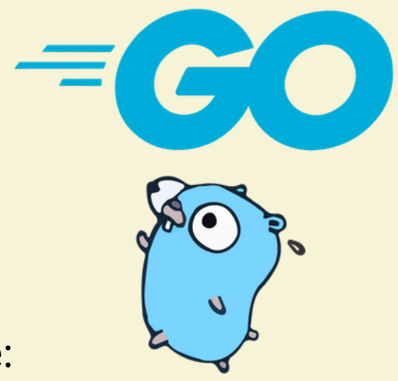
“When in doubt, leave it out”
— Ken Thompson

CONCURRENCIA EXPLÍCITA

Go se basa en goroutines y channels, inspirados en CSP.

- Comunicación explícita
- Menor dependencia de memoria compartida
- Ciclo de vida visible de las tareas
- Cancelación estructurada

```go
func worker(ctx context.Context,
  in <-chan int, out chan<- int) {
  for {
    select {
    case <-ctx.Done():
      return
    case v := <-in:
      out <- v * 2
    }
  }
}
```



COMPOSICIÓN SOBRE HERENCIA

Go elimina la herencia clásica y promueve:

- Composición explícita
- Interfaces implícitas
- Bajo acoplamiento estructural

TOOLING COMO PARTE DEL LENGUAJE

Go integra herramientas como parte del ecosistema:

- gofmt
- go test
- go vet
- race detector

El tooling refuerza las decisiones de diseño y reduce la variabilidad entre equipos.

CONCLUSIÓN

Go puede entenderse como un lenguaje de diseño que reduce la complejidad accidental mediante restricciones deliberadas. Al limitar el espacio de diseño y priorizar la explicitud, facilita el razonamiento humano y favorece la construcción de sistemas concurrentes más mantenibles y seguros.

IMPACTO

El enfoque de Go produce efectos concretos:

- Menor complejidad accidental
- Código más legible y mantenible
- Mejor razonamiento en equipos grandes
- Reducción indirecta de fallos con impacto en seguridad

REFERENCIAS

- F. P. Brooks — No Silver Bullet
- Jon Bodner — Learning Go
- Rob Pike — Go at Google
- C. A. R. Hoare — Communicating Sequential Processes