

Como mockar  
estruturas  
complexas em  
testes  
unitários


The image features a light pink background with a series of thin, wavy, concentric lines in a slightly darker shade of pink. These lines originate from the bottom right corner and curve upwards and to the left, creating a sense of motion and depth. The text is positioned on the left side of the image, with the wavy lines appearing behind it.

# Quem sou eu?



- Pedro Fadel, Carioca, 28 anos;
- Desenvolvedor Back-end na Nomo;
- Background de Python;
- Comecei a usar Go em produção em 2016;
- Gago, por isso tenham paciência comigo por favor;
- Amante de música, paçoca e basquete;
- Entusiasta de testes automatizados de todo tipo;

# Agenda de hoje

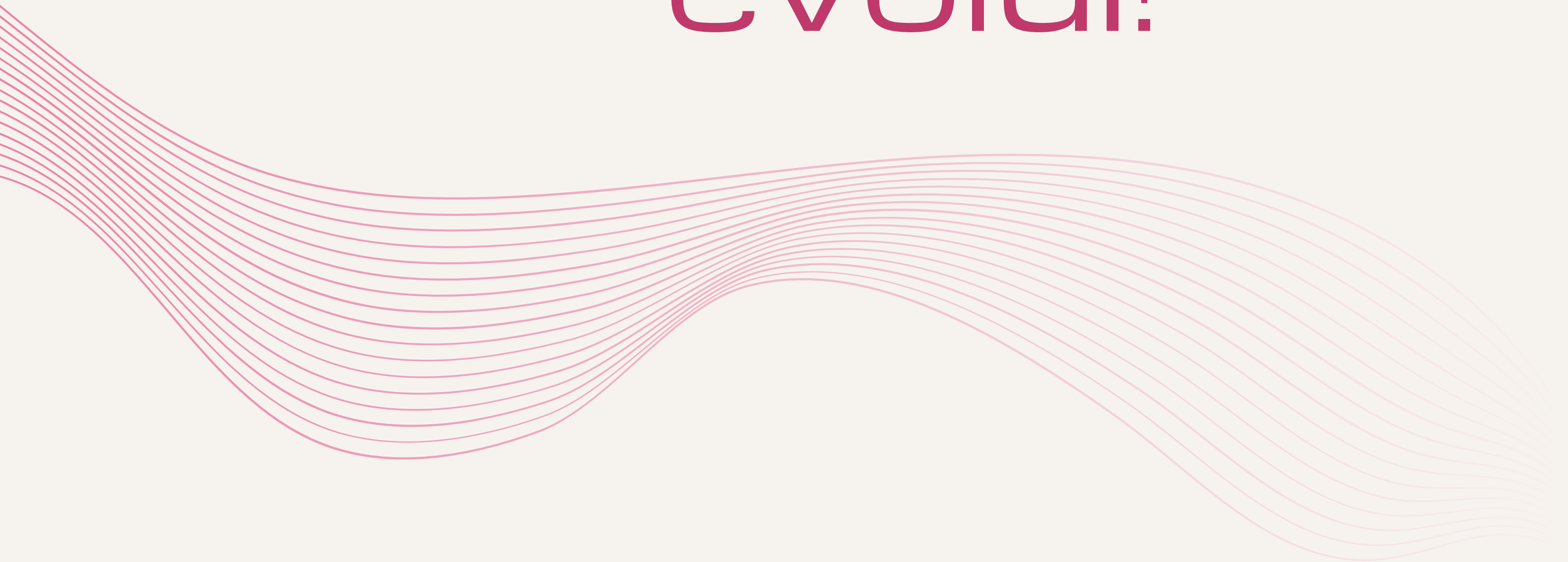
- Por que é importante testar?
  - Quais são os tipos de conflitos no código?
  - O que define um teste unitário?
  - O que define um teste de integração?
  - Por que precisamos de diferentes tipos de testes?
  - Como mockar?
  - Mockery
  - Conclusão e Fontes
  - Dúvidas e Perguntas
- 

Por que é  
importante  
testar?

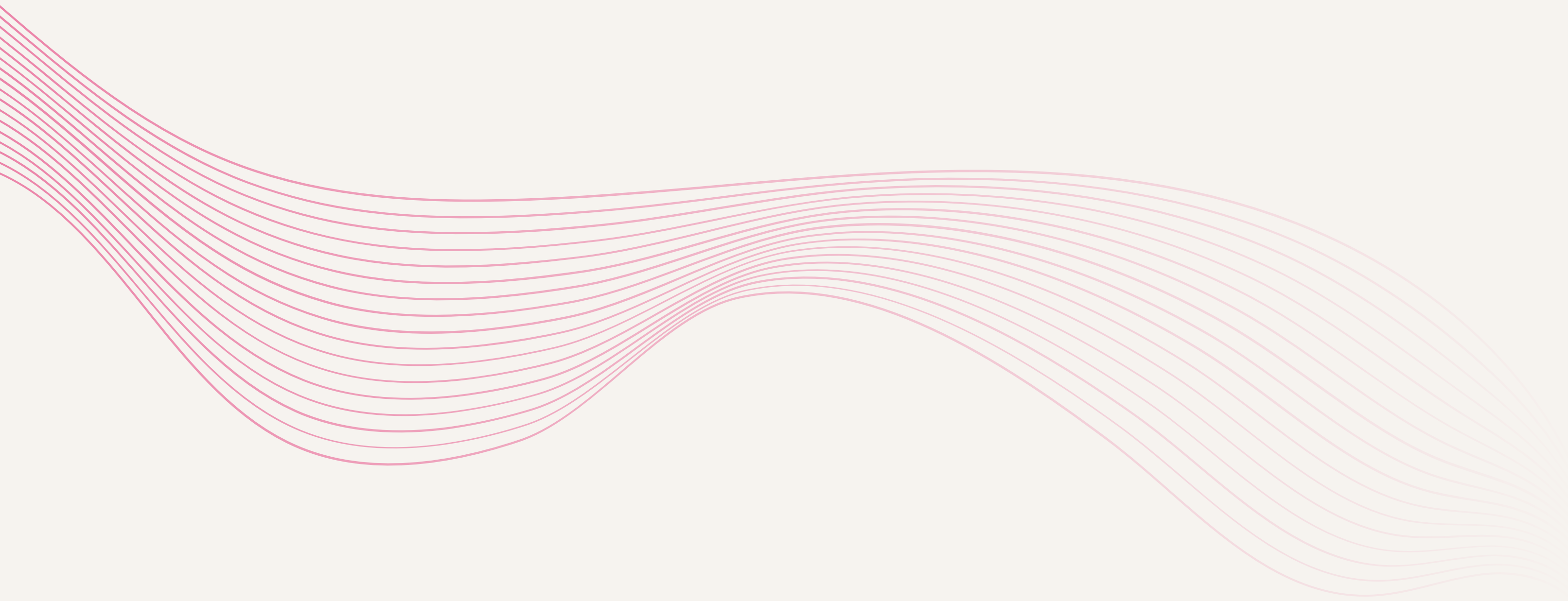




Porque o código  
evolui!



# Mudanças causam conflitos



# Conflicto Físico



# Conflicto Sintático





# Conflito Semântico



# Resumo

1. Conflito físico: Detectável pelo GIT;
  2. Conflito Sintático: Detectável com um processo de CI/build;
  3. Conflito Semântico: Detectável com testes unitários e de integração;
- 
- A series of thin, wavy, light pink lines that originate from the left side of the slide and flow towards the right, creating a decorative background element.

# Exemplo real

"A NASA review board found that the problem was in the software controlling the orbiter's thrusters. The software calculated the force that the thrusters needed to exert in pounds of force. A second piece of code that read this data assumed it was in the metric unit—"newtons per square meter"."

Fonte: <https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/>





# O que define um teste unitário?

É o level de teste de software onde unidades individuais do software são testadas. O objetivo é validar que cada unidade do software funciona como foi planejada. **Uma unidade é a menor parte testável de qualquer software.**

Fonte:

<http://softwaretestingfundamentals.com/unit-testing/>

# O que define um teste de integração?

É o level de teste de software onde **unidades individuais são combinadas e testadas como um grupo**. O objetivo é expor falhas na interação entre as unidades integradas.

Fonte:

<http://softwaretestingfundamentals.com/integration-testing/>



Por que  
precisamos de  
diferentes tipos  
de testes?



E como garantir que  
testes unitários são  
unitários de verdade?



# Mocks!

O ato de "mocking" ou "mockar" é a criação de objetos falsos que simulam o comportamento de objetos reais.

Fonte:

<https://www.typemock.com/what-is-mocking/>

**Como mockar?**

# No Python...

```
>>> from unittest.mock import MagicMock
>>> thing = ProductionClass()
>>> thing.method = MagicMock(return_value=3)
>>> thing.method(3, 4, 5, key='value')
3
>>> thing.method.assert_called_with(3, 4, 5, key='value')
```

# No Go... ???

# Pythonizando Go

Não ficou bom...

```
monkey.PatchInstanceMethod(reflect.TypeOf(cen), "Broadcast", func(_ *gocent.Client, _ []string, _ []byte) (bool, error) {  
    return false, errors.New("test")  
})  
defer monkey.UnpatchAll()
```

Usando a lib ( arquivada desde 2019 ):

<https://github.com/bouk/monkey>



# Use Interfaces!



# Boas práticas

The bigger the interface, the weaker the abstraction.

## type Writer

```
type Writer interface {  
    Write(p []byte) (n int, err error)  
}
```

## type Reader

```
type Reader interface {  
    Read(p []byte) (n int, err error)  
}
```

Como lidar com legado?

# Legado real

```
type CaseCreatable interface {  
→   GetRequester() string  
→   SetData(key string, data interface{})  
→   SetFullData(data map[string]interface{})  
→   GetData(key string) interface{}  
→   SetCaseNumber(*string)  
→   GetCaseNumber() *string  
→   SetCustomField(key string, data interface{})  
→   GetCustomField(key string) interface{}  
→   Send() error  
  
→   SetSignature(sig *Signature)  
→ }  
}
```

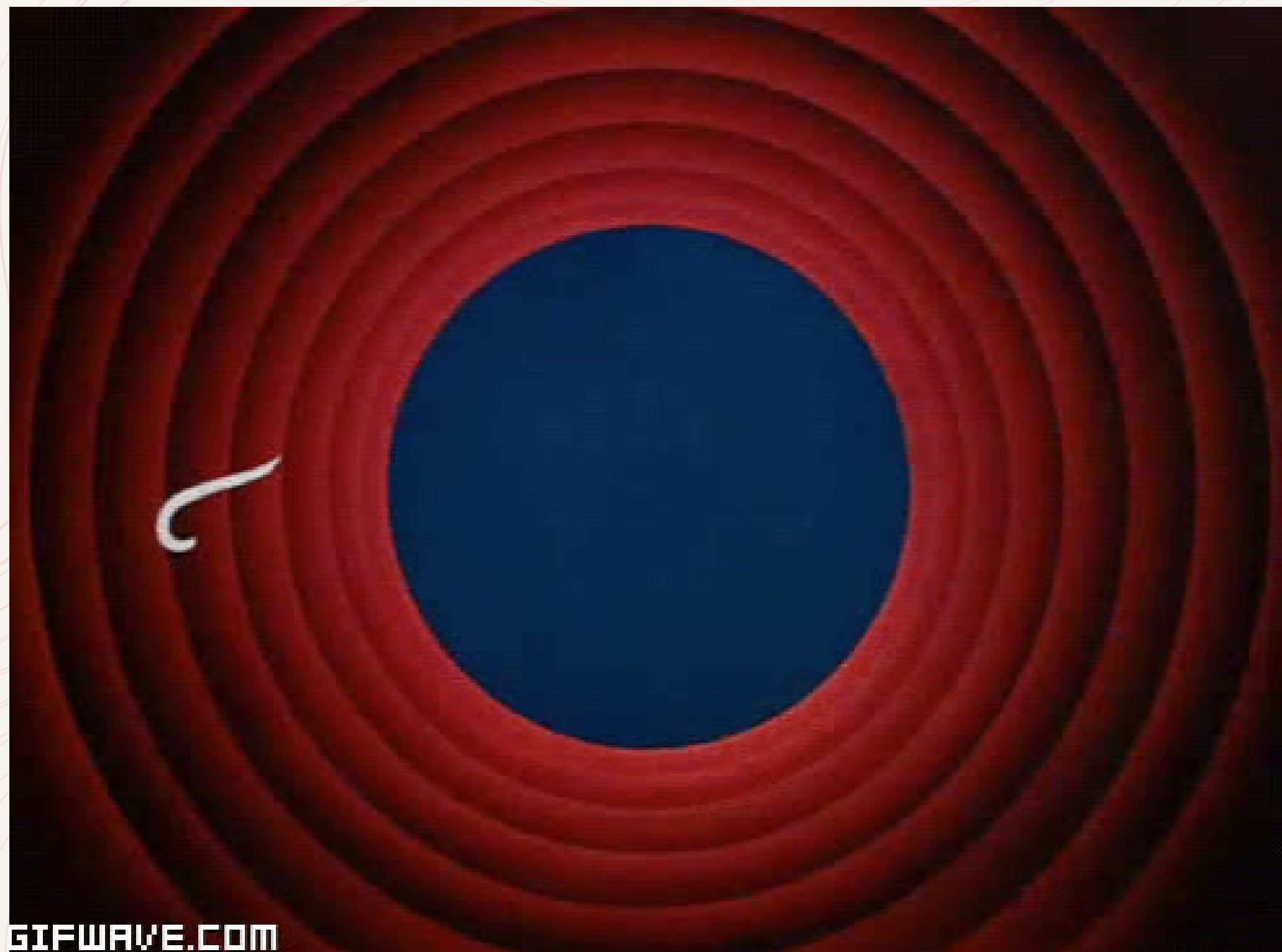
# Apresentando Mockery

<https://github.com/vektra/mockery>

```
mockery --name=<interface> --with-expecter
```



# Conclusão





# Perguntas

