Generating Test data for Table driven Tests with different LLMs to evaluate their potential for test automation

xnacly, hlxid

May 23, 2024

DHBW

Inhaltsverzeichnis

Das Problem am Testen

Overengineered FizzBuzz

```
var k = []int{2, 3}
var m = map[int]string{
    k[0]: "Fizz",
    k[1]: "Buzz",
}
func FizzBuzz(n int) string {
    s := ""
    for _, key := range k {
        if n%key == 0 {
            s += m[key]
        }
    }
    if s == "" {
        return strconv.FormatInt(int64(n), 10)
    }
    return s
}
```

Overengineered FizzBuzz testen

```
func TestFizzBuzz(t *testing.T) {
    cases := []struct {
        input    int
        expected string
    }{{2, "Fizz"}, {12, "FizzBuzz"}, {1, "1"}}
    for _, c := range cases {
        t.Run(fmt.Sprint(c), func(t *testing.T) {
            output := FizzBuzz(c.input)
            if output != c.expected {
                t.Errorf("%q != %q\n", c.expected, output)
            }
        })
    }
}
```

Das Problem mit Testen

- Tests in Go Manier $\Rightarrow$ Table Driven
- Coverage ist interessant
- Edge Cases und alle wichtigen Branches testen!
- Testen frisst Zeit, ist redundant $\Rightarrow$ Automatisierung möglich?

Overengineered FizzBuzz weniger ausführlich testen

```
func TestFizzBuzz(t *testing.T) {
    cases := []struct {
        input    int
        expected string
    }{{2, "Fizz"}, {12, "FizzBuzz"}}
    for _, c := range cases {
        t.Run(fmt.Sprint(c), func(t *testing.T) {
            output := FizzBuzz(c.input)
            if output != c.expected {
                t.Errorf("%q != %q\n", c.expected, output)
            }
        })
    }
}
```

Potentielle Lösung

Lösung?

- Von LLMs test cases generieren lassen
- Qualität über `go test -cover` einschätzen (höher → besser)

```
$ go test -v -cover
=== RUN   TestFizzBuzz
=== RUN   TestFizzBuzz/{2_Fizz}
=== RUN   TestFizzBuzz/{12_FizzBuzz}
=== RUN   TestFizzBuzz/{1_1}
--- PASS: TestFizzBuzz (0.00s)
    --- PASS: TestFizzBuzz/{2_Fizz} (0.00s)
    --- PASS: TestFizzBuzz/{12_FizzBuzz} (0.00s)
    --- PASS: TestFizzBuzz/{1_1} (0.00s)
PASS
coverage: 100.0% of statements
ok      example 0.002s
$ go test -v -cover
=== RUN   TestFizzBuzz
=== RUN   TestFizzBuzz/{2_Fizz}
=== RUN   TestFizzBuzz/{12_FizzBuzz}
--- PASS: TestFizzBuzz (0.00s)
    --- PASS: TestFizzBuzz/{2_Fizz} (0.00s)
    --- PASS: TestFizzBuzz/{12_FizzBuzz} (0.00s)
PASS
coverage: 85.7% of statements
ok      example 0.002s
```

Konkretes

Testergebnisse

Model	Function	Score	Failed test cases
GPT-3.5	binomial	90%	2
	md5	97.5%	0
GPT-4o	binomial	100%	0
	md5	96.3%	2
LLAMA 3	binomial	100%	1
	md5	did not compile	
Code LLAMA	binomial	100%	1
	md5	did not compile	
Mixtral	binomial	90.9%	1
	md5	did not compile	

Figure 1: Results of the LLM evaluation for Test Table Generation

Ist die Lösung eine Lösung?

Ist die Lösung eine Lösung?

- Je komplexer das Test Target, desto schlechter der Score
- Schnelles Auftreten von fehlerhaften Tests
- Bei nicht-trivialen Funktionen schnell Compiler Fehler
- Zeitgewinn gleich 0, weil Dev Tests checken und reparieren muss

Fragen?

Fragen

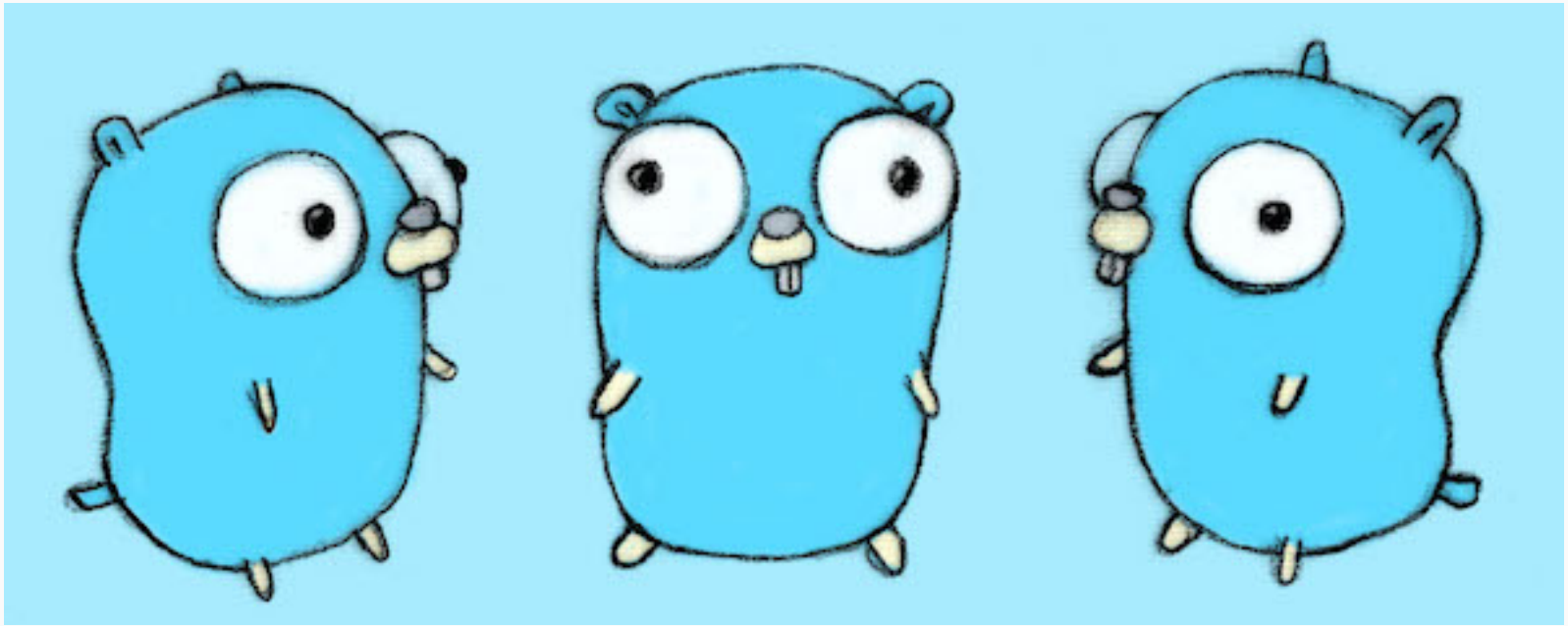

Figure 2: Quelle: `https://go.dev/blog/gopher`