

ruleguard workshop

GopherCon Russia 2021

Сверчки детектед

Unexpected noises imminent



Какой у нас план?

Какой у нас план?

До



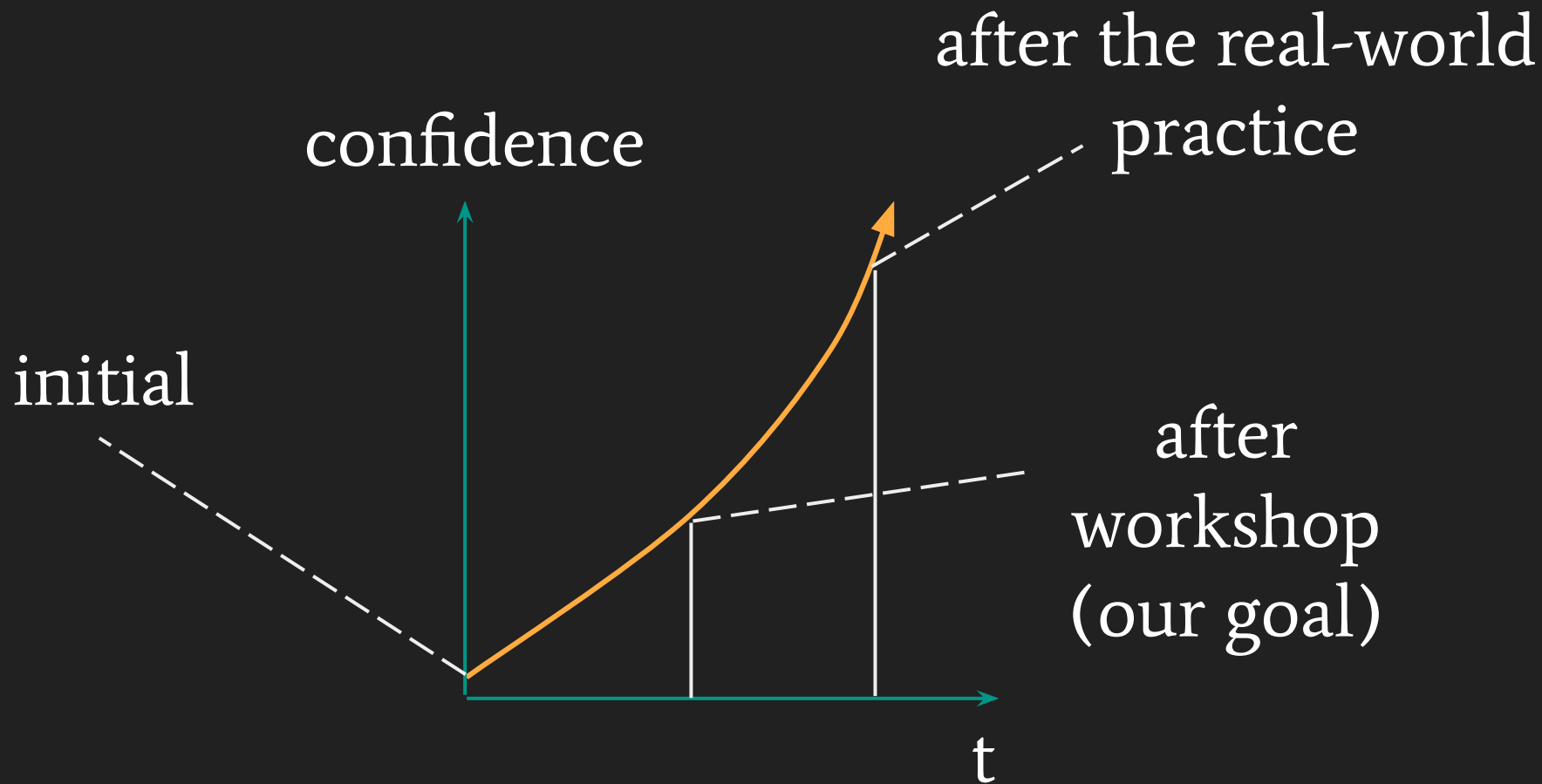
Какой у нас план?

До



После





github.com/quasilyte/gophercon2021-ruleguard

Репозиторий проекта со всеми материалами



Что такое ruleguard?



Старый талисман проекта



Новый талисман проекта

Что такое ruleguard?

- Инструмент для рефакторинга

Что такое ruleguard?

- Инструмент для рефакторинга
- Инструмент для структурного поиска кода

Что такое ruleguard?

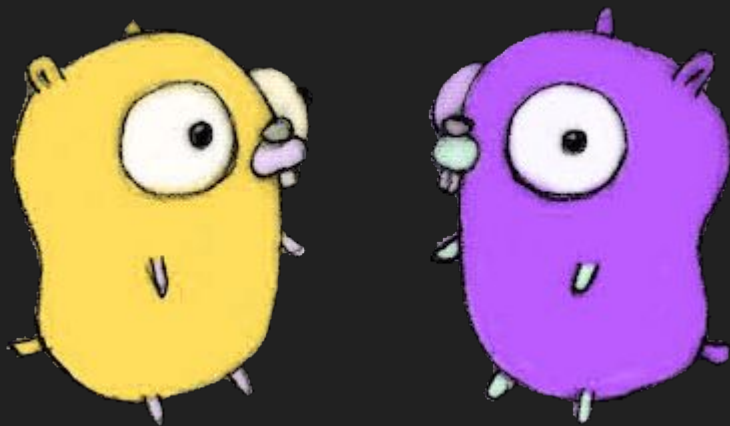
- Инструмент для рефакторинга
- Инструмент для структурного поиска кода
- Написан на Go, для Go

Что такое ruleguard?

- Инструмент для рефакторинга
- Инструмент для структурного поиска кода
- Написан на Go, для Go
- **Расширяемый** статический анализатор

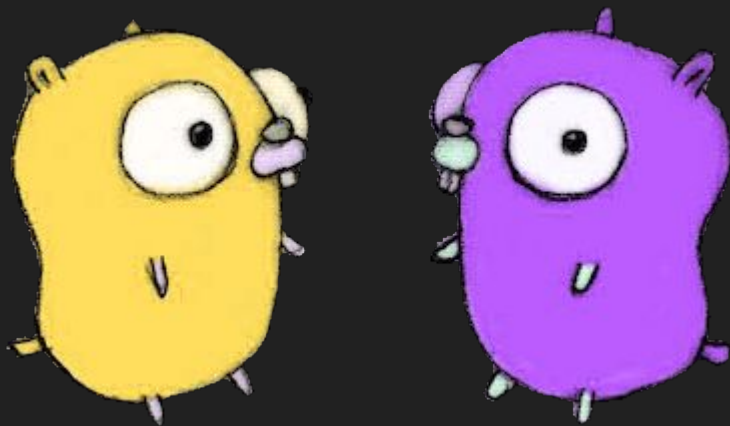
Концепция рулгарда

Из разговора с одной из конференций



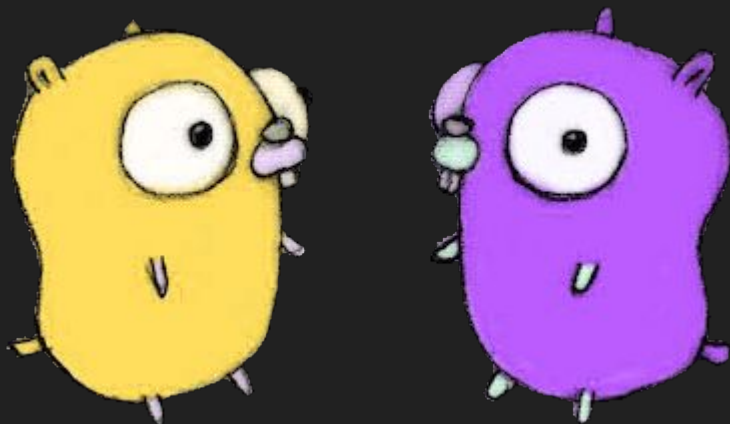
Из разговора с одной из конференций

gogrep?



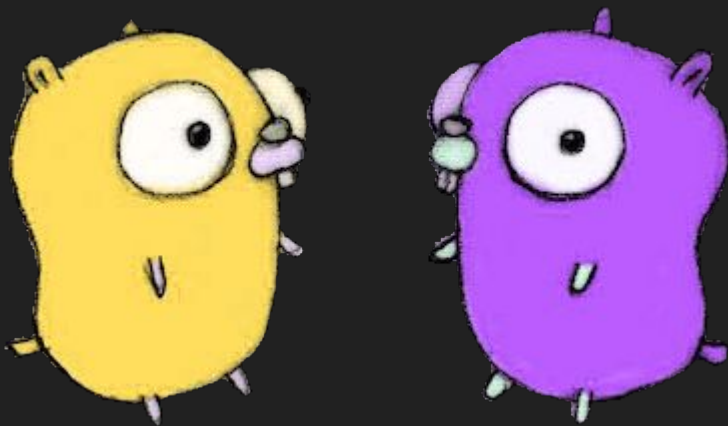
Из разговора с одной из конференций

static analysis?



Из разговора с одной из конференций

gogrep-based static
analysis!





Концепция

A special Go-compatible format

Not just gogrep anymore



Реализация, эволюция

Для чего он подходит лучше всего?

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич
- Рекомендация лучших альтернатив

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич
- Рекомендация лучших альтернатив
- Прототипирование диагностик

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич
- Рекомендация лучших альтернатив
- Прототипирование диагностик

Вы *сами* делаете этот линтер.

Он создан для того, чтобы его расширяли.

Сравнение с традиционными линтерами

Динамическое расширение

Рулгард расширяется динамически. После добавления новых диагностик не нужна перекомпиляция.

Декларативное описание диагностик

Вы описываете шаблоны, фильтры и сообщения для предупреждений. Ничего лишнего.

Остальное за вас делает рулгард.

Авто-интеграция в golangci-lint


Ваши правила автоматически становятся запускаемыми
через все любимый golangci-lint.

Ближайшие аналоги:

CodeQL и Semgrep

Ближайшие аналоги:


[CodeQL] и Semgrep



Высокий порог входа для написания правил.
Правила пишем на своеобразном специализированном языке
программирования.

Ближайшие аналоги:

CodeQL и **[Semgrep]**



Значительно хуже поддерживает Go.
Правила приходится “программировать” в YAML.

А у нас в ruleguard:

* Правила на Go *

* Установка правил через Go модули *

* Отличная поддержка Go *

Курс молодого бойца

Особо важные пакеты из stdlib

Особо важные пакеты из stdlib

- `go/ast` - чтобы понимать AST матчинг

Особо важные пакеты из stdlib

- `go/ast` - чтобы понимать AST матчинг
- `go/types` - для понимания фильтров

Особо важные пакеты из stdlib

- `go/ast` - чтобы понимать AST матчинг
- `go/types` - для понимания фильтров

В общем-то и всё.

Остальное инкапсулирует рулгард.

Но есть нюанс, который стоит повторить

[go/ast] vs go/types

Типы **go/ast** указывают категорию синтаксического элемента (например, “цикл”).

[go/ast] vs [go/types]

Типы **go/types** указывают на тип выражения или декларации, которое выводится не только исходя из синтаксической структуры.

`int(15)`

AST type = `CallExpr`

Expression type = `int`

```
Node.Is("CallExpr") && Type.Is("int")
```

Пример фильтра, демонстрирующий важность этого
разделения

AST паттерны

AST паттерны

```
if err != nil {  
    return err  
}
```

Это AST паттерн.

AST паттерны

```
if err != nil {  
    return err  
}
```

Пробелы игнорируются, поэтому это матч:

```
if err!=nil { return nil }
```

Применяем gogrep

```
$ cd $GOROOT
```

```
$ pattern='if err != nil { return err }'
```

```
$ gogrep -x "$pattern" ./src/encoding/...
```

Такие паттерны понимает и gogrep,
и ruleguard

Применяем gogrep

```
/go/src/encoding/csv/writer.go:103:5: if err != nil { return err; }  
/go/src/encoding/csv/writer.go:138:3: if err != nil { return err; }  
/go/src/encoding/gob/encoder.go:235:2: if err != nil { return err; }  
/go/src/encoding/json/decode.go:102:2: if err != nil { return err; }  
/go/src/encoding/json/stream.go:64:2: if err != nil { return err; }  
/go/src/encoding/json/stream.go:207:2: if err != nil { return err; }  
/go/src/encoding/json/stream.go:226:3: if err != nil { return err; }  
/go/src/encoding/json/stream.go:313:3: if err != nil { return err; }  
/go/src/encoding/json/stream.go:323:3: if err != nil { return err; }  
/go/src/encoding/xml/marshal.go:163:2: if err != nil { return err; }
```

AST паттерны (продолжение)

```
if $err != nil {  
    return $err  
}
```

И это тоже AST паттерн.

Но уже с переменными (\$).

AST паттерны (продолжение)

```
if $err != nil {  
    return $err  
}
```

Это матч:

```
if err2 != nil { return err2 }
```

AST паттерны (продолжение)

```
if $err != nil {  
    return $err  
}
```

И это тоже матч:

```
if err != nil { return err }
```

Переменные шаблона ($\$<name>$)
захватывают *любой* AST-элемент

$\$x$ matches ``14.6``

$\$x$ matches ``x + y``

$\$x$ matches ``f(g(), 2)``

$\$x$ matches ``if cond {...}``

...

AST паттерны (продолжение)

```
if err != nil { $*_ }
```

А здесь мы используем `$*_` для матчинга любого тела внутри блока.

AST паттерны требуют изучения, но когда
вы попробуете их в деле, вам
понравится.

Справку по терминологии ищи в
“`notes/terminology.md`”
внутри репозитория

А теперь приступим!