

BACKEND TEST AUTOMATION FROM THE SCRATCH

IGOR BALAGUROV

QA AUTOMATION TECH LEAD AT VK

- vk.com/korgan
- linkedin.com/in/ibalagurov
- github.com/ibalagurov



EXPERIENCE

- ▶ VK
 - ▶ Payment systems
 - ▶ Internal Development
 - ▶ Performance testing
- ▶ Startups & Product companies
- ▶ iOS, Android, Web, Backend, Machine learning



PLAN

- ▶ Introduction
- ▶ How to test?
- ▶ How to choose tools?
- ▶ Examples (Python and REST API)
- ▶ What to test in API's next
- ▶ We need go deeper
- ▶ Conclusions



BACKEND TESTING

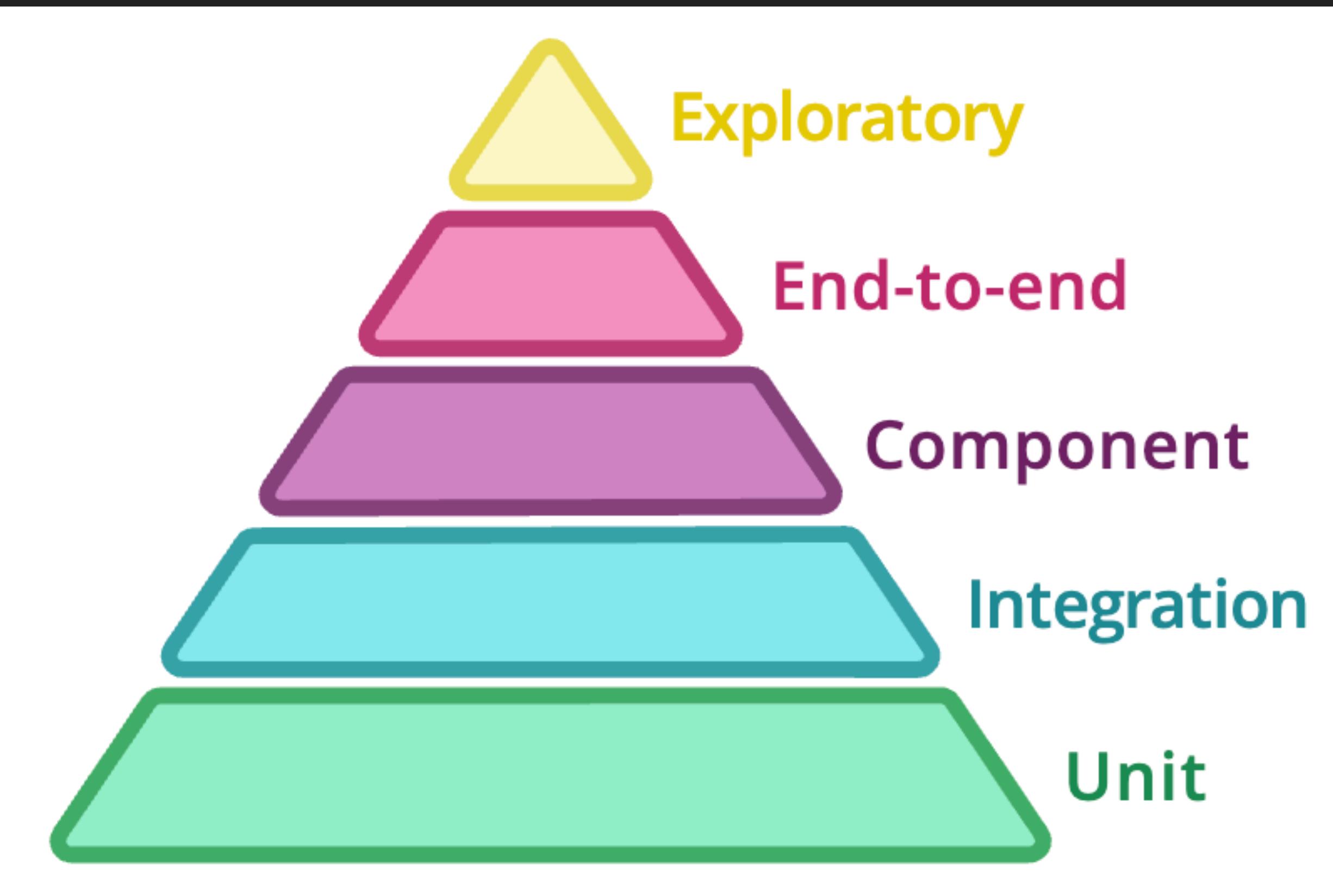
- ▶ Application architecture
- ▶ Different approaches, tools and etc

TEST PYRAMID

- ▶ Too broad conceptions
- ▶ Some layers could be unavailable in your application
- ▶ 3rd parties could change everything

MICROSERVICE TEST PYRAMID

- martinfowler.com
 - [microservice testing](#)
 - [practical test pyramid](#)

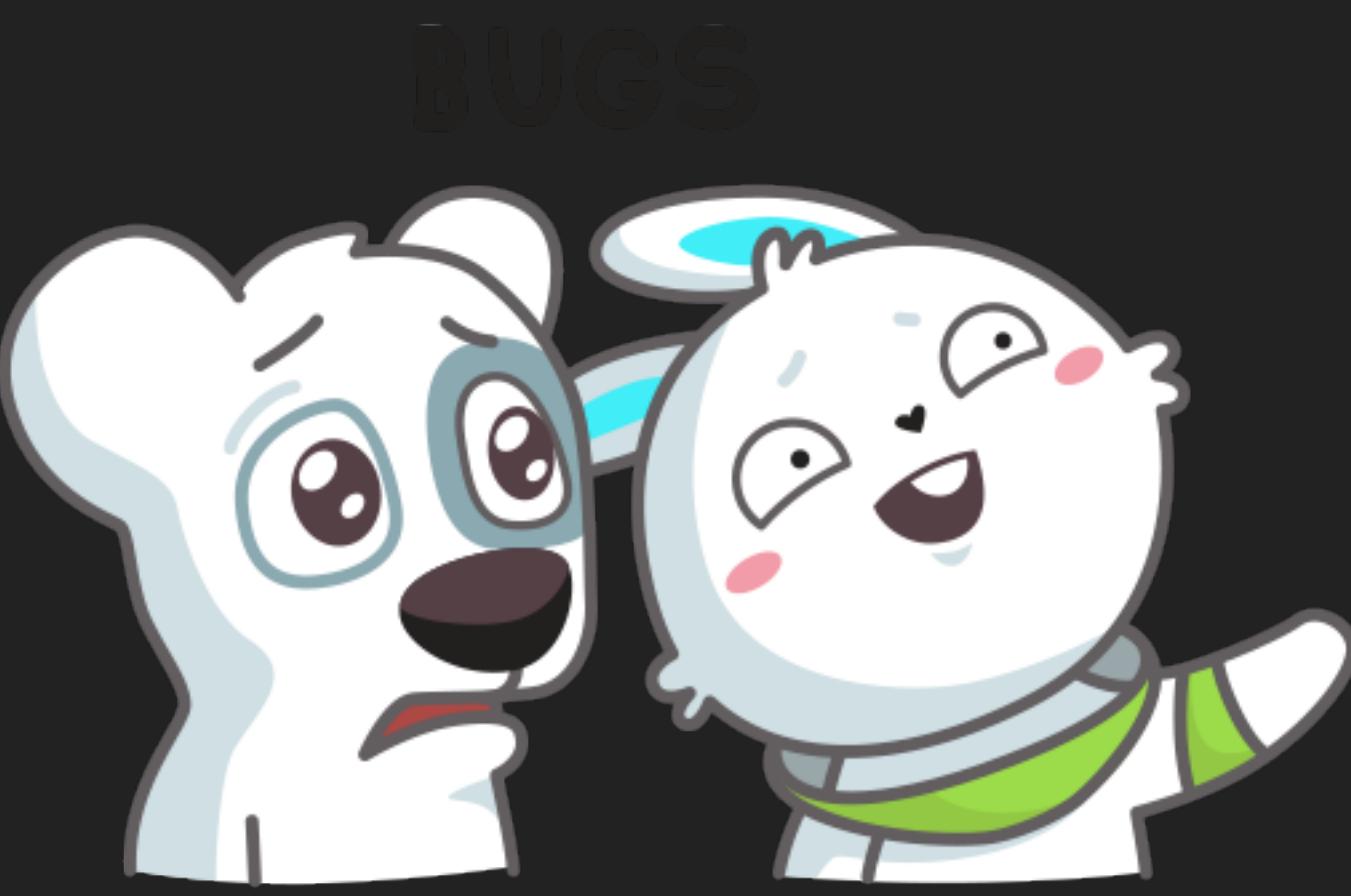


TEST PYRAMID

- ▶ Solve your problems

PROTOCOLS

- ▶ HTTP or not
- ▶ Text and binary
- ▶ Synchronous and asynchronous
- ▶ REST, GraphQL, MQ, gRPC, Thrift, DB



BUGS
EVERYWHERE

PROTOCOLS

- ▶ Interface
 - ▶ Input and output data
- ▶ Built for programs
 - ▶ Fit for automation

EDGE CASES

- ▶ Configuration
- ▶ BigData
- ▶ Machine Learning

PROTOCOLS

- ▶ Interface
 - ▶ Input and output data
- ▶ How to test?

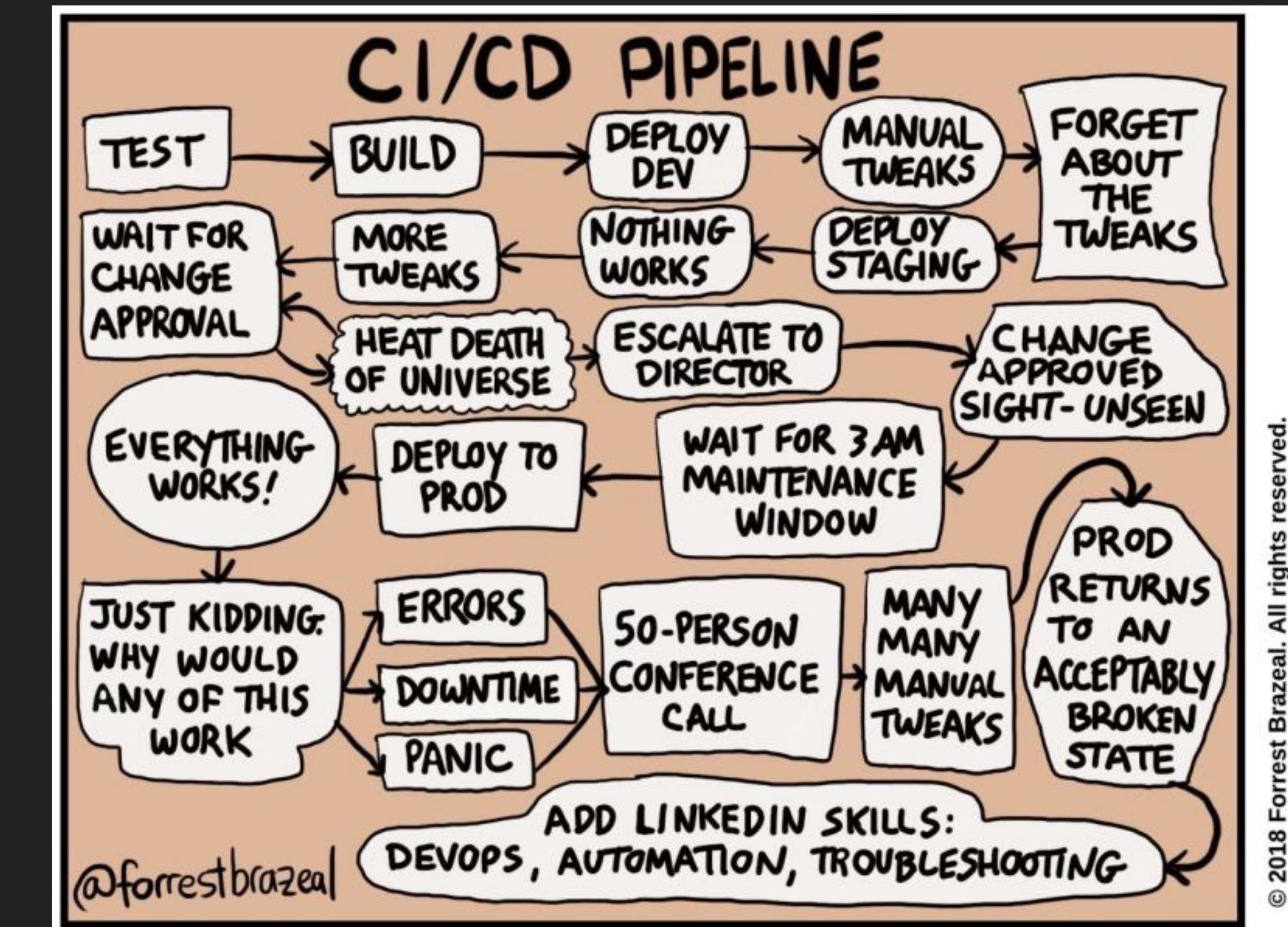
MANUALLY

- ▶ Requirements / Documentation
 - ▶ Usability / UX
 - ▶ Testability
- ▶ Exploratory
 - ▶ Security (Penetration) testing
- ▶ Acceptance



AUTOMATION

- ▶ CI / CD
 - ▶ Fast feedback
 - ▶ Reduce Time2Market
 - ▶ Frequent releases
 - ▶ Close to development
 - ▶ UI tests preconditions



TOOLS FOR AUTOMATION

- ▶ Same language with development?
 - ▶ Tools for testing
 - ▶ Community (StackOverflow ❤)
 - ▶ Learning curve
 - ▶ Development performance
 - ▶ Code performance



SHORT PROJECT

- ▶ Manual
- ▶ Postman
- ▶ Hello world automation

SHORT PROJECT

- ▶ Problems
 - ▶ Project became bigger
 - ▶ Throw away old approaches



FEATURE TEAMS

- ▶ Same repositories
- ▶ Same technologies?

FEATURE TEAMS

- ▶ Problems
 - ▶ Full stack QA Engineers?
 - ▶ Huge QA Team inside each feature team

BIG PROJECT + SMALL QA TEAM

- ▶ Developers write almost all automated tests
- ▶ Small dedicated QA team
 - ▶ Convenient language and tools for QA
 - ▶ Same language?

BIG PROJECT + SMALL QA TEAM

- ▶ Problems
 - ▶ Frequent releases
 - ▶ Update tests for new logic
 - ▶ Revert?

BIG PROJECT + BIG QA TEAM

- ▶ Same technologies

BIG PROJECT + BIG QA TEAM

- ▶ Problems
 - ▶ Your own frameworks, tools and etc
 - ▶ Feature toggles, AB tests

COMPANY TYPES

- ▶ Challenges

LANGUAGES FOR AUTOMATION (IMHO)

- ▶ C#: few tools for testing
- ▶ Java: a lot of tools
- ▶ Python: easy tools integration, wide community
- ▶ JS: a lot of tools, complex tools integration
- ▶ Ruby: cool tools, but few candidates
- ▶ Go/Kotlin: a new hope

PYTHON AND REST API

- ▶ Approaches are language agnostic
- ▶ REST API



TEST RUNNER

- ▶ Parametrization
- ▶ Parallel execution
- ▶ Rerun failures
- ▶ Soft asserts
- ▶ Ability to extend

PYTEST

- ▶ Good community
- ▶ Nested fixtures (`setup+teardown`)
- ▶ Hook based Plugin Architecture
 - ▶ Fast tests development
 - ▶ Easy to extend
 - ▶ Complex tests, different protocols, many integrations and etc.

HTTP CLIENT

```
1 import requests
2
3 response = requests.post(
4     url="https://your-site.com",
5     json={"a": 1}
6 )
7
8 response.request.headers["content-type"]
9
10 # 'application/json'
```

REQUESTS

- ▶ Good community
- ▶ Dead simple client
- ▶ API standard

TESTS

- ▶ Risk based scenarios
- ▶ Privacy (role based model)
- ▶ Input / Output equivalence classes and boundary values
- ▶ API methods combinations
- ▶ Based on test coverage lower level tests

EXTEND THEM

- ▶ Random data generation
- ▶ Complex data validation (Response schemas)
- ▶ Base security checks
- ▶ Property based testing

HIDDEN CORNER CASES

- ▶ Pesticide Paradox:
 - ▶ Hardcode values in test data
 - ▶ Inability to variate data across all tests



APPROACH

- ▶ Randomization in getting boundary values and inside equivalence classes
- ▶ Randomization helps to test:
 - ▶ Combinations
 - ▶ Localization
 - ▶ New checks
- ▶ Seed for reproducible tests

TOOLS

- ▶ Momezis
- ▶ Faker

EXAMPLES: NEGATIVE CASES

```
1 import mimesis
2
3 generate = mimesis.Generic(locale="en", seed=100500)
4
5 nulls = ["null", "nan", "nil", "none"]
6 random_word = generate.text.word()
7 common = [random_word, *nulls]
8
9 invalid_currency = generate.choice(common)
10 # "none"
```

RANDOMIZATION: PROS & CONS

- ▶ Reusable values
- ▶ It could be used as standalone data generator solution
- ▶ Combinations
- ▶ ...

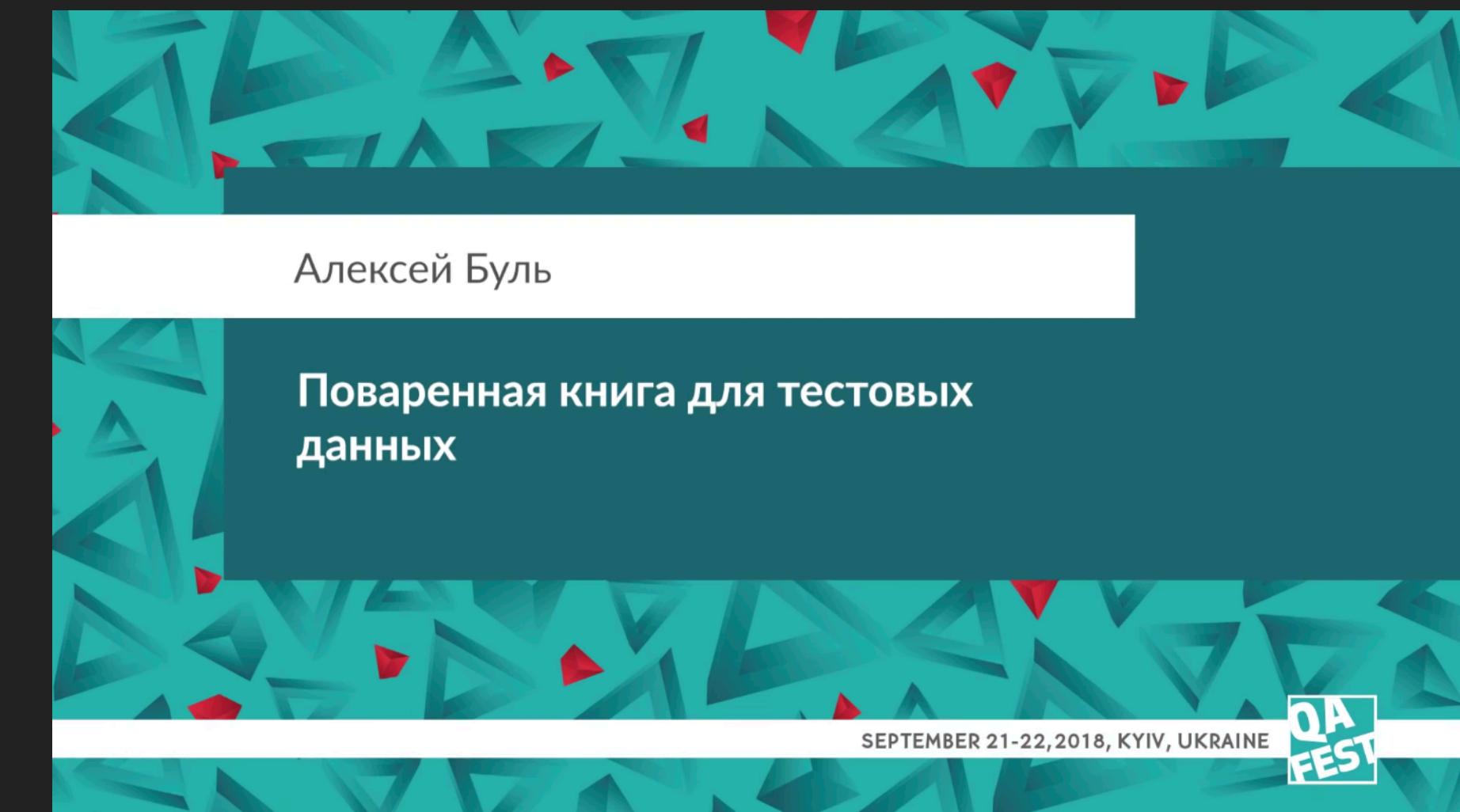
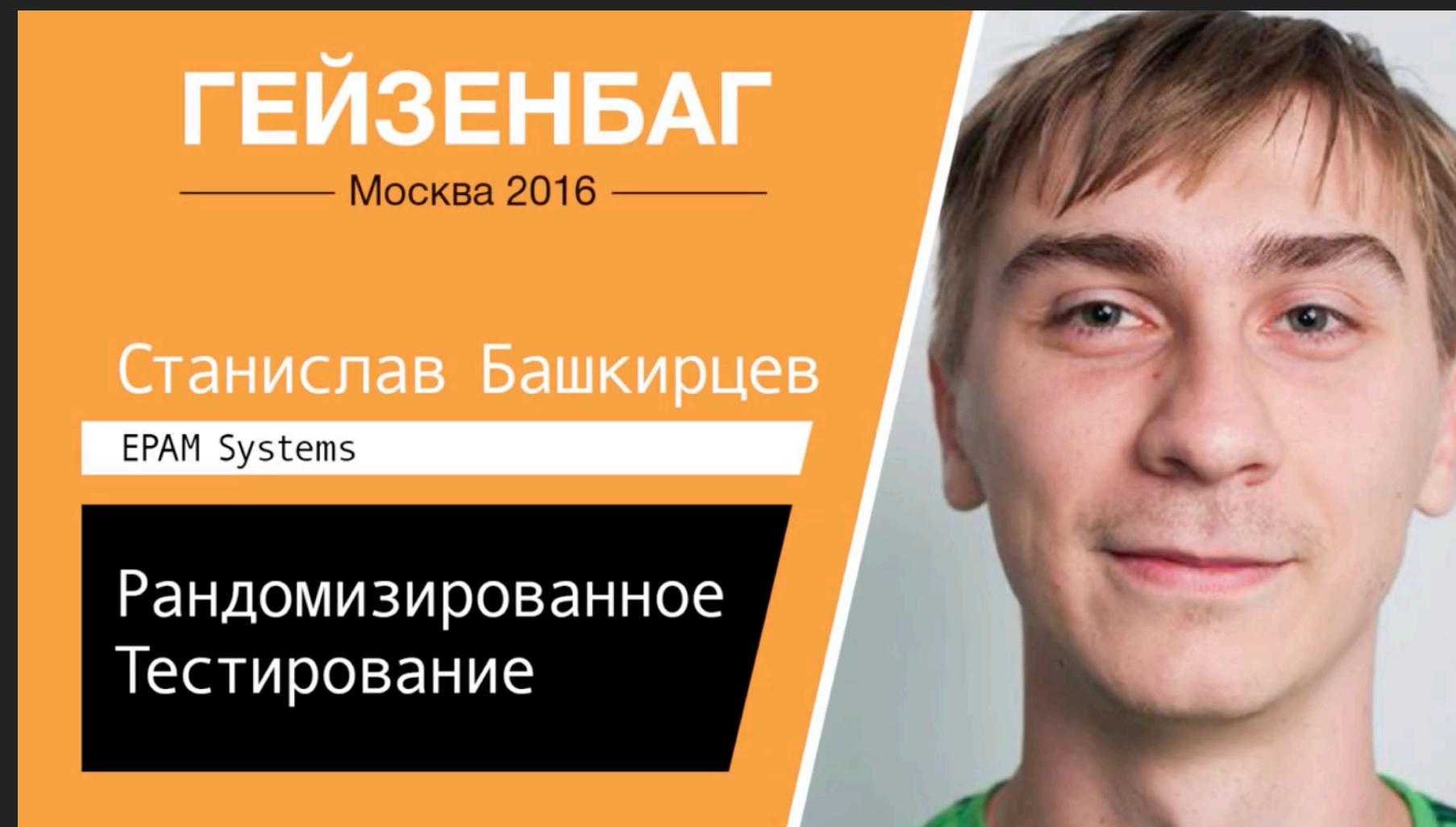
RANDOMIZATION: PROS & CONS

- ▶ Reusable values
- ▶ It could be used as standalone data generator solution
- ▶ Combinations
- ▶ ...
- ▶ Sometimes tests fail → Knowledge mining
- ▶ You need the seed to reproduce test



WHAT'S NEXT?

- [Mimesis documentation](#)
- [Articles \(habr\)](#)
- [Heisenbug 2016 Moscow: Randomized testing](#)
- [QA Fest 2018: Поваренная книга для тестовых данных](#)



COMPLEX DATA VALIDATION

- ▶ 3rd-parties data
- ▶ Sensitive data:
 - ▶ Any new field is a risk
- ▶ Contract:
 - ▶ Any new format type is a risk

APPROACH

- ▶ Data validation, schemas
 - ▶ Body contains only predefined fields, in appropriate format
 - ▶ No additional information: in any case, in any place
 - ▶ There is no any sensitive data



TOOLS

- ▶ Cerberus, Pydantic, jsonschema
- ▶ Convenient API: no need in additional conversion
- ▶ All errors in one check
- ▶ Required / Unknown fields
- ▶ Custom rules (any, all, none, dependencies)

EXAMPLES

```
1 wallet_balance = {  
2     "data": {  
3         "type": "dict",  
4         "required": True,  
5         "require_all": True,  
6         "schema": {  
7             "balance": {  
8                 "type": "number", "min": 0, "max": 60000  
9             },  
10            "userId": {"type": "integer", "min": 1},  
11        }  
12    }  
13 }
```

EXAMPLES

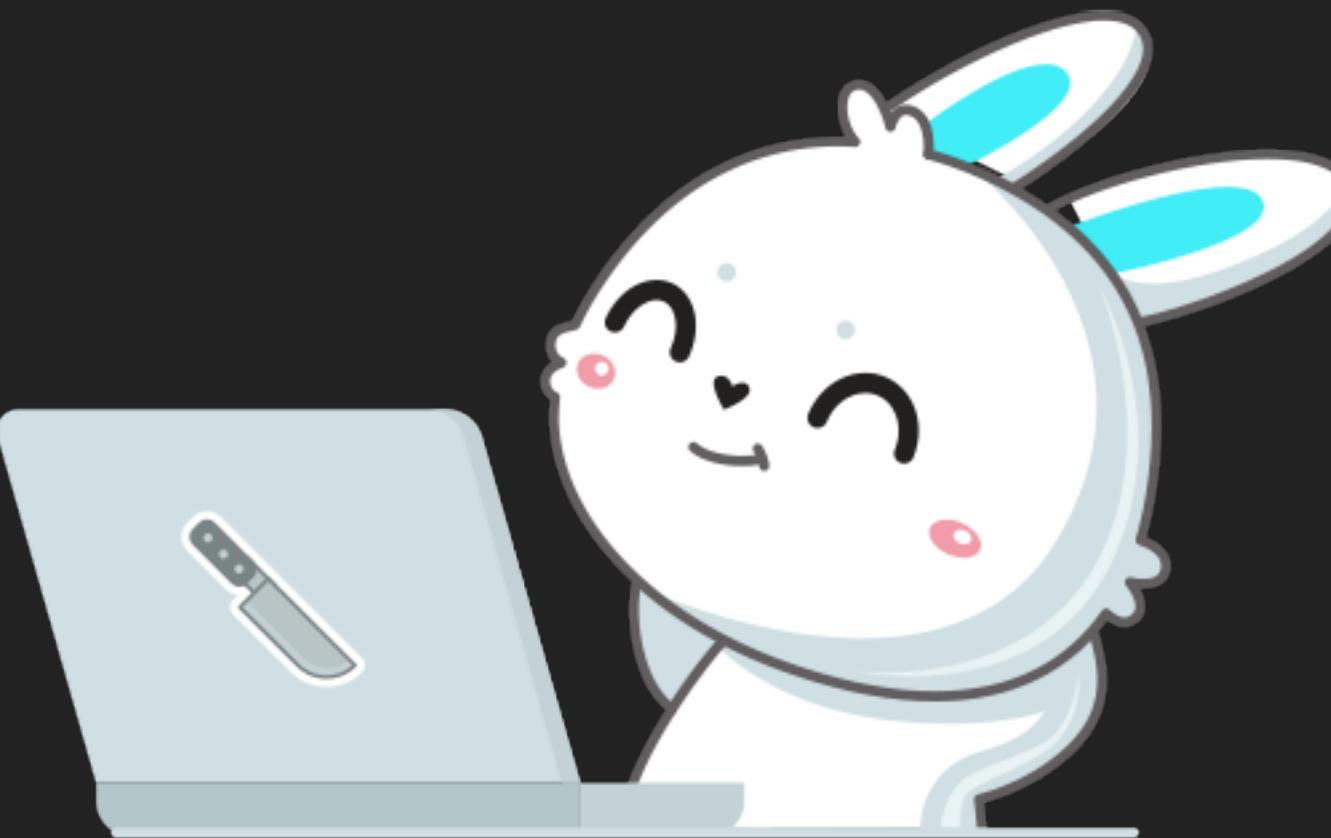
```
1 data_to_validate = response.json()
2 # {"data": {"userId": 0, "balance": -0.1}}
3
4 schema_validator.validate(data_to_validate))
5 # False
6
7 schema_validator.errors
8 # {'data':[{
9 #     'balance': ['min value is 0'],
10 #     'userId': ['min value is 1']
11 # }]}
```

DATA VALIDATION: PROS & CONS

- ▶ Useful with 3rd parties
- ▶ If you want to know about:
 - ▶ Every new field
 - ▶ Any new format
- ▶ ...

DATA VALIDATION: PROS & CONS

- ▶ Useful with 3rd parties
- ▶ If you want to know about:
 - ▶ Every new field
 - ▶ Any new format
- ▶ ...
- ▶ Sometimes tests fail → Knowledge mining
- ▶ You need to describe your schemas



WHAT'S NEXT?

- PiterPy 2018: Cerberus, or Data Validation for Humans
- Cerberus documentation



BASE SECURITY TESTS

- ▶ Security risks
- ▶ Sensitive data
- ▶ Separate security department:
 - ▶ It is not so iterative, as it could be

APPROACH

- ▶ OWASP API Security:
 - ▶ Privacy / Permissions
 - ▶ Wrong tokens / hashes / signs / private keys
 - ▶ Invalid values (overflow attempts, injections)
 - ▶ Rate & Resource limits (flood control)
 - ▶ Race conditions

APPROACH

- ▶ Negative actors → privacy
- ▶ Negative values → resources limits
- ▶ Wrong headers → tokens / hashes
- ▶ It's not a problem to extend tests



APPROACH

- ▶ Check every request param
 - ▶ Overflow attempts
 - ▶ Injections
- ▶ Check concurrent responses
 - ▶ Rate limits
 - ▶ Race conditions

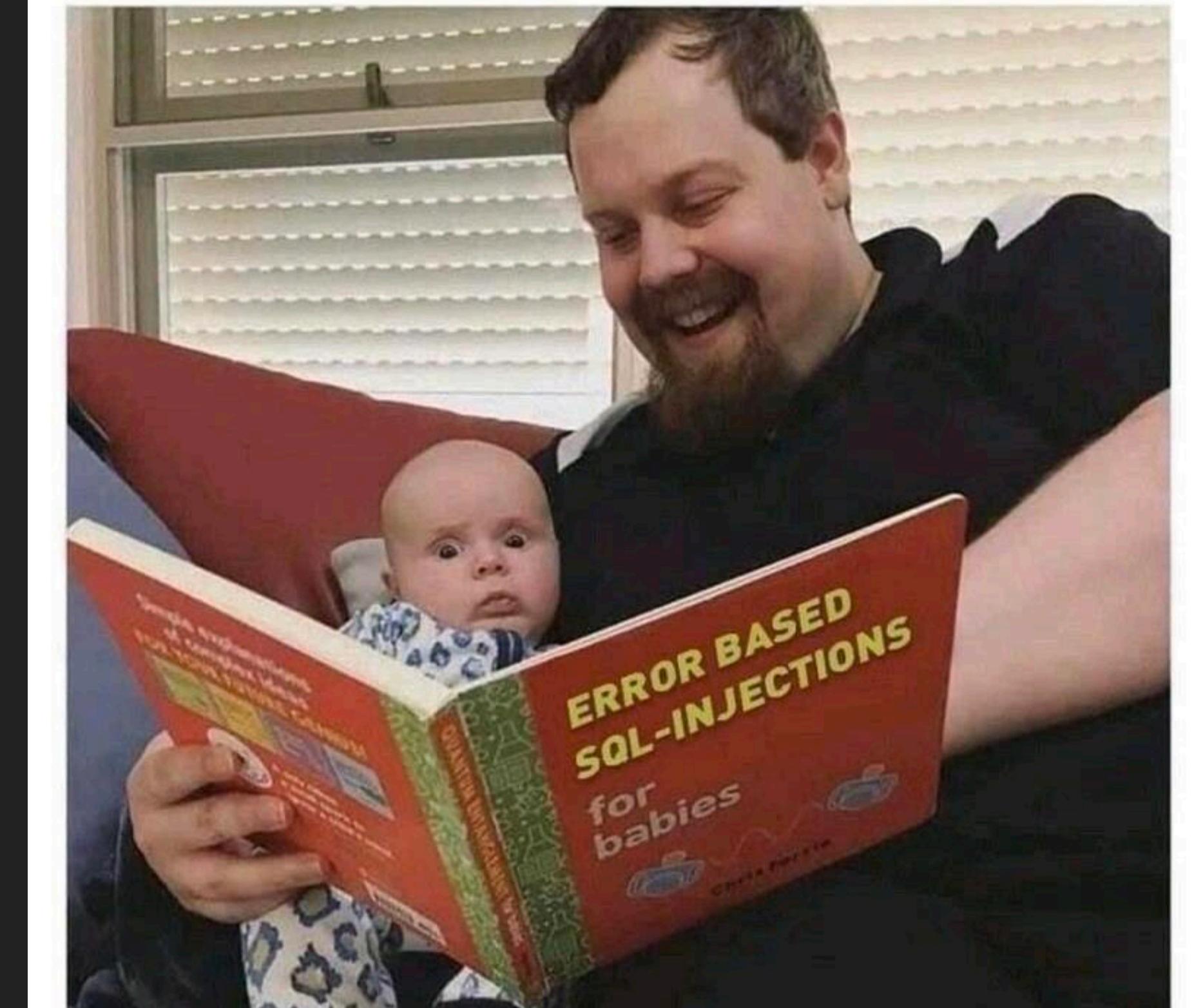
EXAMPLES: NEGATIVE CASES

- ▶ Nulls
- ▶ Invalid format values
- ▶ Non-existent values
- ▶ ...

EXAMPLES: NEGATIVE CASES

- ▶ Nulls
- ▶ Invalid format values
- ▶ Non-existent values
- ▶ ...
- ▶ Overflow attempts
- ▶ Injections

When I become a dad ❤️



EXAMPLES: NEGATIVE CASES

```
1 import mimesis
2
3 generate = mimesis.Generic(locale="en", seed=100500)
4
5 nulls = ["null", "nan", "nil", "none"]
6 random_word = generate.text.word()
7 common = [random_word, *nulls]
8
9 invalid_currency = generate.choice(common)
10 # "none"
```

EXAMPLES: NEGATIVE CASES

```
1 overflow_int = "9" * 100
2
3 my_cool_injections = [ " ' OR 1 == 1" ]
4
```

EXAMPLES: NEGATIVE CASES

```
1 import mimesis
2
3 generate = mimesis.Generic(locale="en", seed=100500)
4
5 nulls = ["null", "nan", "nil", "none"]
6 my_cool_injections = ['' OR 1 == 1']
7 overflow_int = "9" * 100
8 common = [overflow_int, *nulls, *my_cool_injections]
9
10 invalid_currency = generate.choice(common)
11 # "none"
```

APPROACH

- ▶ Check every request param
 - ▶ Overflow attempts
 - ▶ Injections
- ▶ Check concurrent responses
 - ▶ Rate limits
 - ▶ Race conditions

APPROACH

- ▶ Check concurrent responses
 - ▶ Rate limits
 - ▶ Race conditions

TOOLS

- ▶ Standard library:
 - ▶ concurrent.futures
 - ▶ threading
 - ▶ asyncio
 - ▶ and etc.

EXAMPLES

```
1 from functools import partial
2 from src import vk_pay, test_data, parallel
3
4 transaction_id = test_data.create_transaction()
5
6 api_method = partial(
7     func=vk_pay.confirm_transaction,
8     transaction_id=transaction_id
9 )
```

EXAMPLES

```
1 def test_race_condition():
2     responses = parallel.execute(
3         func=api_method, times=10
4     )
5
6     ok_responses = [
7         response.status_code == 200
8         for response in responses
9     ]
10
11    assert len(ok_responses) == 1
```

EXAMPLES

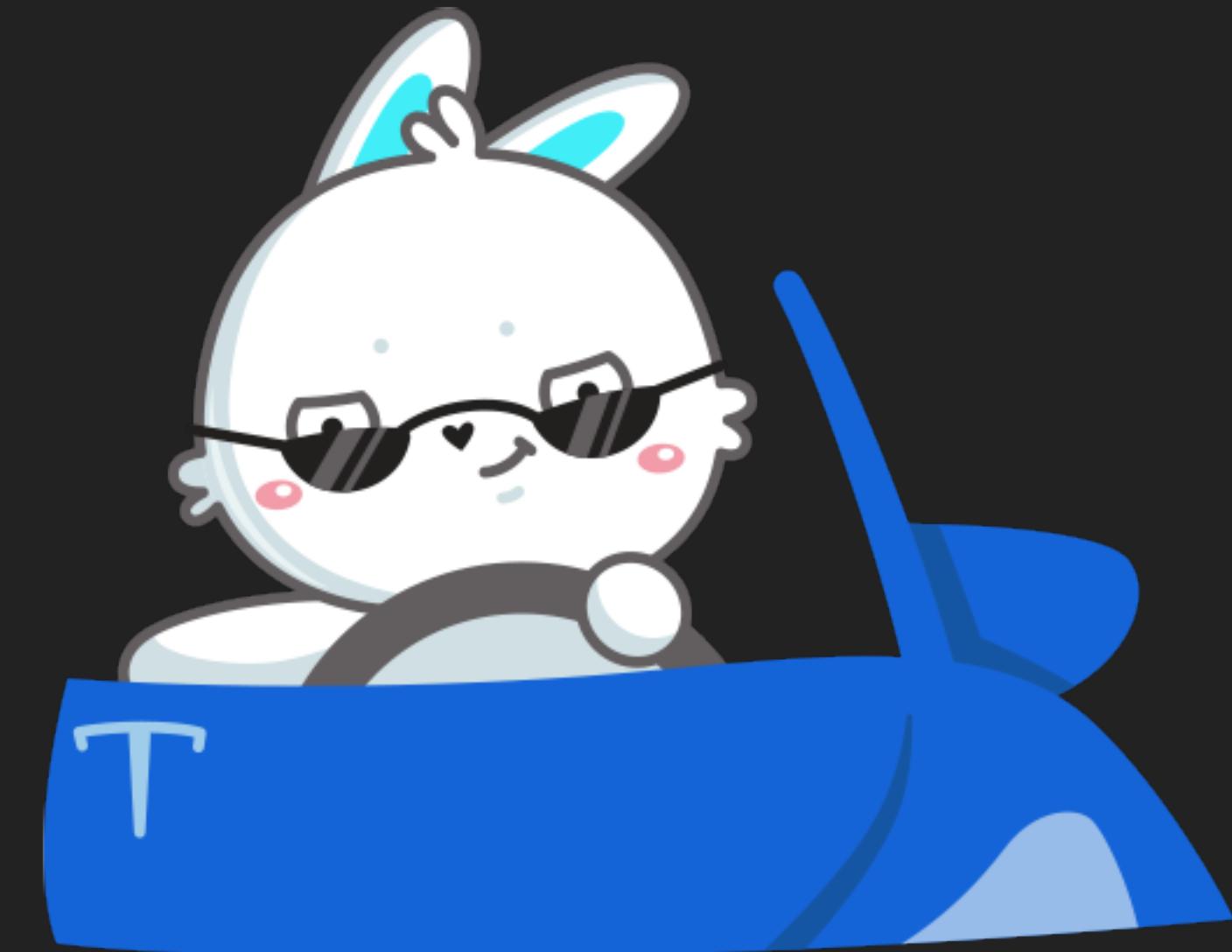
```
1 from concurrent.futures import ThreadPoolExecutor
2
3
4 def execute(func, times=1):
5     pool = ThreadPoolExecutor(max_workers=times)
6     futures = [pool.submit(func) for _ in range(times)]
7     return [f.result() for f in futures]
```

PROS & CONS

- ▶ If you want to prevent or find earlier:
 - ▶ Security issues
 - ▶ ...

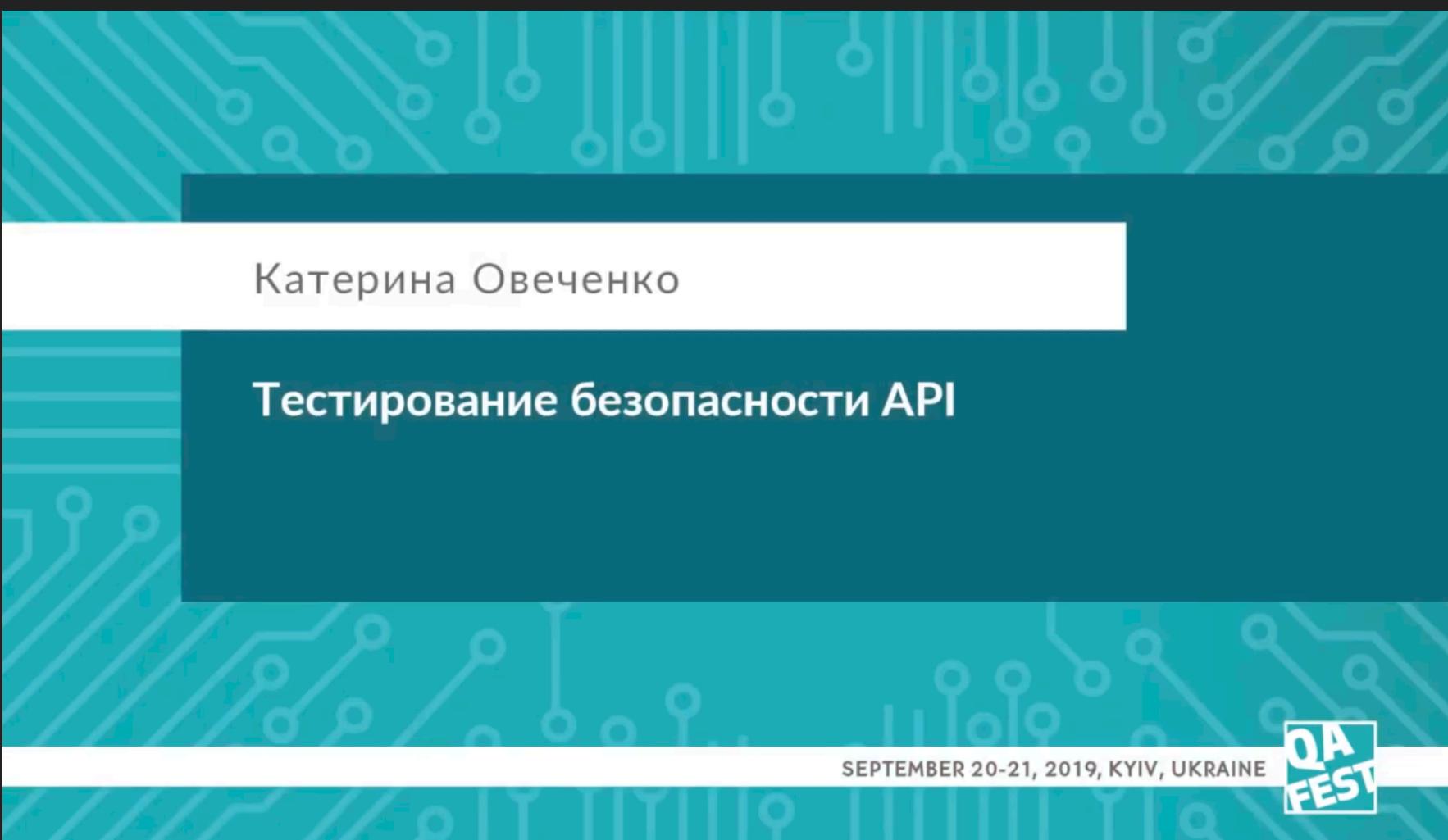
PROS & CONS

- ▶ If you want to prevent or find earlier:
 - ▶ Security issues
 - ▶ ...
- ▶ Sometimes tests fail → Knowledge mining
- ▶ Learning and investigation time



WHAT'S NEXT?

- OWASP API Security
- QA Fest 2019: Security API Testing



APPROACH

- ▶ API Schema (Swagger, GraphQL, Something custom)
- ▶ Parsing
- ▶ Generate data for each type
- ▶ Fuzzing (send requests)
 - ▶ Each method
 - ▶ Each field
 - ▶ Check responses

APPROACH AND TOOLS

- ▶ Parsing schema
 - ▶ Swagger
- ▶ Property-based testing
 - ▶ Hypothesis

TOOLS

- ▶ Schemathesis (hypothesis-jsonschema)
- ▶ Swagger 2.0 / OpenAPI 3.0
- ▶ GraphQL
 - ▶ Full-featured support - in progress
- ▶ Requests
- ▶ Pytest (parametrization)

EXAMPLES: CONNECT TO SWAGGER SCHEMA

```
1 import schemathesis
2
3 schema = schemathesis.from_uri(
4     "https://my-cool-service.com/v2/api-docs"
5 )
6
7
8 @schema.parametrize()
9 def test_no_server_errors(case):
10     ...
```

EXAMPLES: QUICK START FROM DOCUMENTATION

```
1 @schema.parametrize()
2 def test_no_server_errors(case):
3     # `requests` will make an appropriate call
4     response = case.call()
5     # You could use built-in checks
6     # case.validate_response(response)
7     # Or assert the response manually
8     assert response.status_code < 500
```

EXAMPLES: REAL WORLD

```
1 1 @pytest.mark.parametrize("user_role", [...])
2 2 @schema.parametrize()
3 3 def test_no_server_errors(case, arrange_user, user_role):
4 4     user = arrange_user(role=user_role)
5 5     client = http_client.auth_session(user=user)
6
7 7     response = case.call(
8 8         headers=client.headers, cookies=client.cookies
9 9     )
10
11 11     assert response.status_code < 500, "Server error"
```

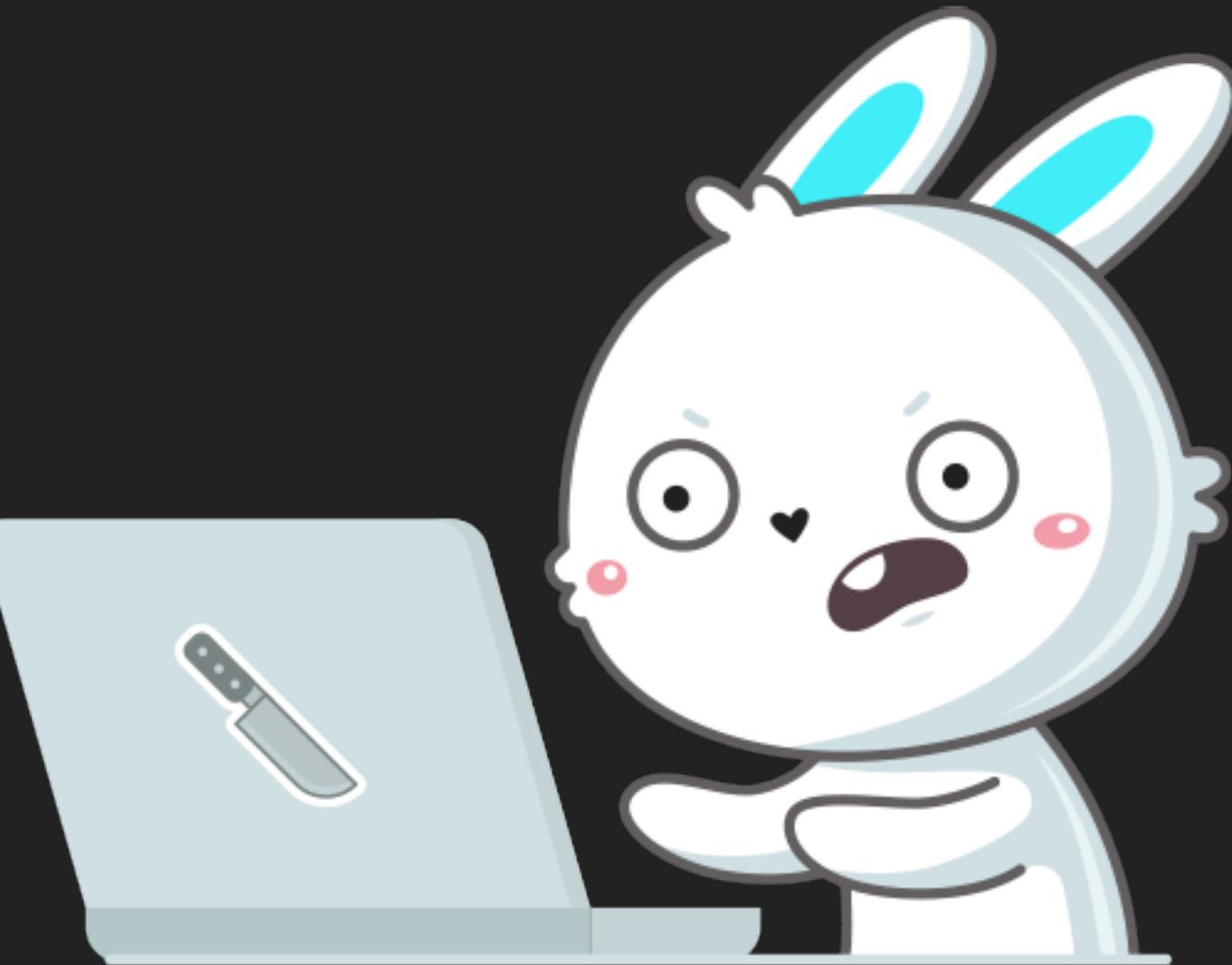
PROS

- ▶ Rapid start testing for new services:
 - ▶ Schema → few minutes → several 5xx errors
- ▶ Declarative approach:
 - ▶ Describe rules that should be validated
- ▶ Lazy test automation:
 - ▶ No need to write tests on every new API method



CONS

- ▶ Send all required fields and data that fits types
- ▶ Require up-to-date swagger schema
 - ▶ Could fail on schema parsing
- ▶ Pytest-xdist parallel issues
- ▶ Many layers → issues require time
- ▶ Many dependencies



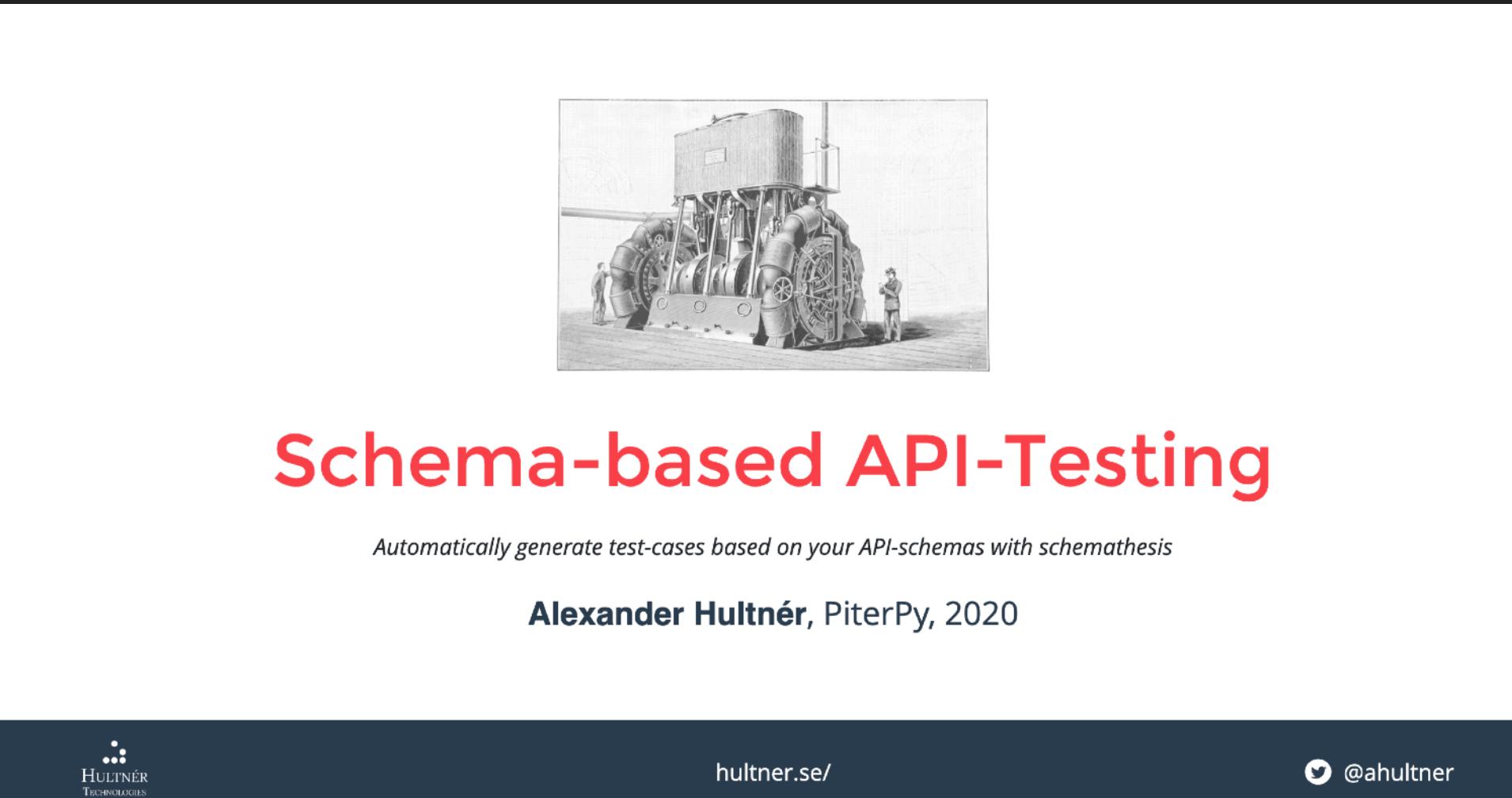
WHAT'S NEXT?

- Introduction to property-based testing (Hypothesis)
- Choosing properties for property-based testing (F#)
- Hypothesis documentation
- Heisenbug 2019 Piter: Property testing: Strategic automation for devs and SDETs



WHAT'S NEXT?

- PiterPy 2020: Schema-based API-Testing – Automatically generate test-cases with schemathesis
- Progress report and plan (2020-10-05)
- Habr (Russian)
- Medium (English)



REMEMBER ALL

- ▶ Architecture of your application
 - ▶ Approaches and tools
- ▶ Common approaches (Risk based scenarios)

REMEMBER ALL

- ▶ Next approaches
 - ▶ Random data generation
 - ▶ Data validation based on schemas
 - ▶ Base security checks
 - ▶ Property based testing

QUALITY IS ABOUT WHOLE DEVELOPMENT PROCESS

- ▶ Shift-left & white box
 - ▶ Code review (at least unit tests) & code coverage analysis
 - ▶ Static code analysis, code formatters
 - ▶ Testability is one of software architecture quality attributes
- ▶ Shift-right & SRE
 - ▶ AB-testing, Canary releases, Feature toggles

THANK YOU!!!

vk.com

presentation

linkedin.com

