Ближайшее будущее мутационного тестирования на Python

Евгений Блинов для Moscow Python Meetup

Обо мне

Делаю разные питоновые штуки

Контакты

Гитхаб: github.com/pomponchik

Линкедин: linkedin.com/in/pomponchik

Личный сайт: pomponchik.org


Hildebrands Deutsche Schokolade
Будущее!
Deutscher Kutter.

L

LL

LLM

3 факта о будущем

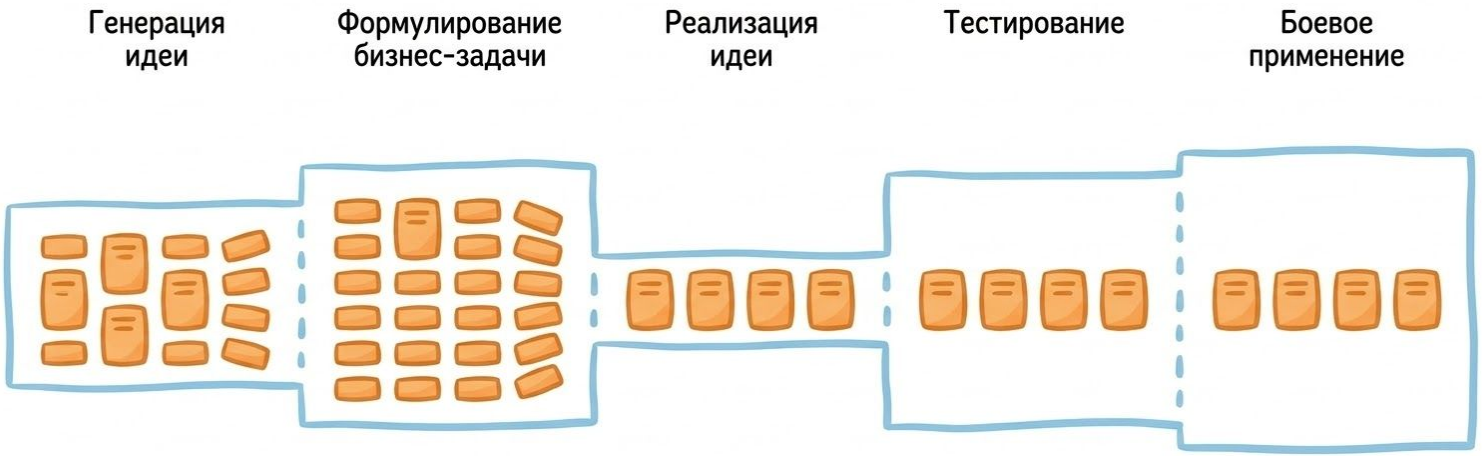
1. Парадокс Джевонса

- Чем дешевле код, тем больше его нужно
- IT будет продолжать расти в капитализации
- Бенефициары перераспределятся
- Какую долю среди них займут “операторы LLM” – вопрос пока открытый

2. Бутылочные горлышки переместятся

- По сути все IT – это фабрики ифов
- Код пишется, чтобы проверять чьи-то гипотезы
- Код – промежуточный артефакт
- Генерация кода – только один из конвееров
- Расширение одного конвеера перемещает ботлнеки в другие

2.1. Нагрузка на тестирование резко возрастает



Тесты – бутылочное горлышко агентского кодинга

- Все используют LLM для генерации кода
- Тесты кажутся легкой целью
- Но программисты по факту все равно перечитывают весь код
- Тесты и их корректность – по-прежнему сложнейшая задача в программировании

> LLM пишет тесты

3. Изменение роли Python

- Если код все равно пишет агент, зачем мне Python, если есть Rust?
- Проще сделать и протестировать прототип на Python, а потом переписать на Rust, чем писать сразу на Rust
- Основная роль Python в будущем – промежуточный язык
- Базовый тест-сьют, вероятно, останется на Python

Вот каким будет будущее

Каким, по твоему, будет будущее?

я не Каким, меня Сашей зовут. да по моему будет будущее

Так а что там про MT?

Главный вопрос тестирования: как протестировать тесты?

Нужна штука, которая будет сомневаться в тестах

- Действительно ли этот тест что-то проверяет?
- А если я код вот так поменяю, тесты упадут?

Картинка по запросу “скептик”

Мутационное тестирование – это автоматизация скептицизма

- Ломаем исходный код (например меняем + на -)
- Запускаем тесты
- Падают – хорошо
- Не падают – плохо
- Повторяем тысячи раз, получаем метрику: процент “убитых мутантов” (MSI)
- ИИ-агент ориентируется на метрику, и при необходимости дописывает недостающие тесты
- Проблема: эквивалентные мутанты

Это единственный способ расширить ботлнек тестирования

Ограничения

- Эта штука не выявляет баги, несоответствие спеке
- Она проверяет только то, что юнит-тесты *что-то* проверяют
- Не заменяет другие виды тестирования
- Все еще дорого по компьюту
- Должно применяться после всех других видов тестирования

Как обстоят дела сейчас?

Общий стейт:

- Вообще идея мутационного тестирования старая, из 70-х
- Крупные компании типа Google и Meta внедряют в продакшен массовые проверки с помощью огромных мутационных систем
- Однако опенсорсный тулинг *для питона* все еще не очень
- В некоторых языках все обстоит гораздо лучше (например, JS, C#, Java)

Google

- На 2021 год: активно используется более чем 24 000 инженеров, проверяются все PRы
- Система встроена в продакшен
- Все крутится вокруг дешевых инкрементальных проверок

research.google/pubs/practical-mutation-testing-at-scale-a-view-from-google/

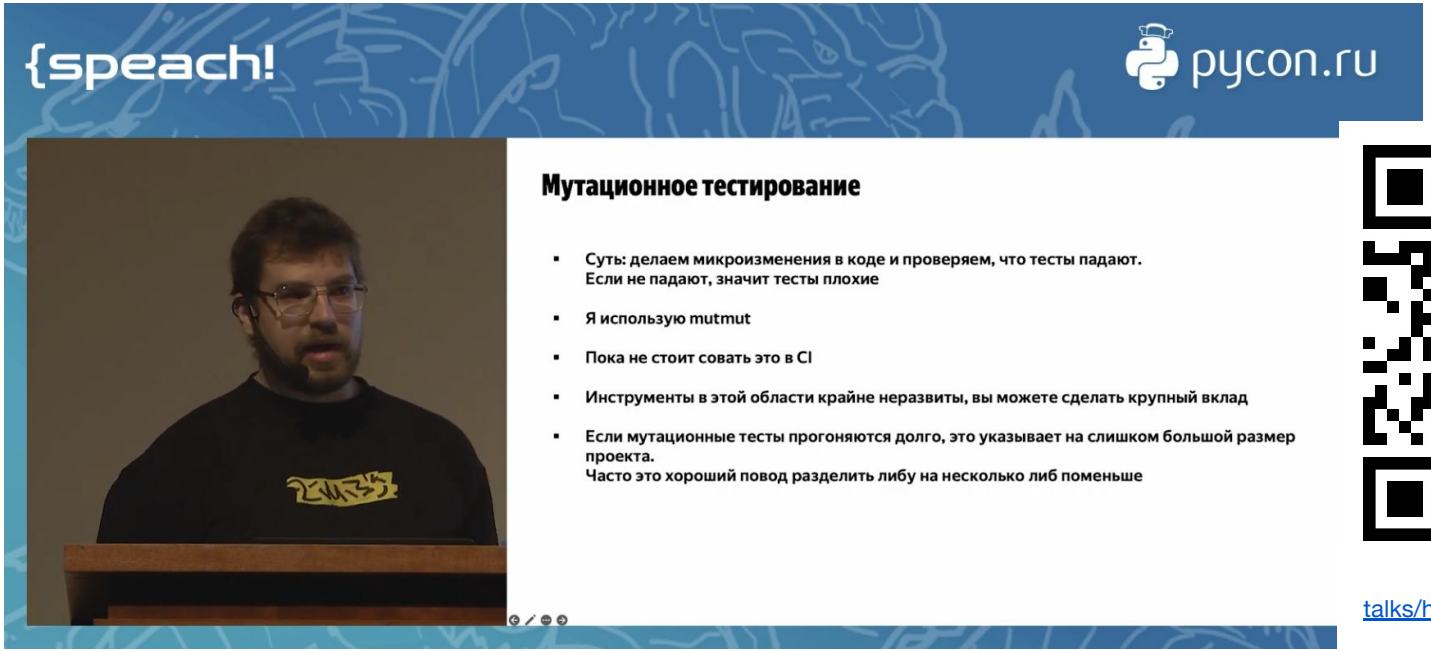
Meta (запрещено в РФ)

- Активно мутируют
- Интегрируют с LLM: нейронка как ломает код, так и пишет тесты

engineering.fb.com/2025/09/30/security/llms-are-the-key-to-mutation-testing-and-better-compliance/

А питонячий опенсорс?

2024: PyCon



pomponchik.org/talks/how-to-test-libraries-in-python/

Пока не очень

Нужно, чтобы одновременно:

- Удобно
- Параллелизм из коробки (между машинами!)
- Сэндбоксинг (изолированные среды)
- Много мутаций, поддержка LLM-мутаций
- “Пригодность к продакшену”

Я не нашел решения, где есть сразу все, и решил его создать

Новый фреймворк

Главные фичи

- Плагинная система – легко добавлять мутации
- Параллелизация из коробки
- Сендбоксинг: запуск кода в изолированных средах
- Минимум настройки
- Адаптация к повседневной работе и продакшену (возможность встроить в CI)

Принципы разработки

- Максимальная модульность, опора на собственную плагинную систему
- Разбиение проекта на подлибы
- Вертикальная интеграция
- Внимание к деталям

Пример компонента: pristan

```python
from pristan import slot

@slot
def some_slot(a, b) -> dict[str, int]:
    ...

@some_slot.plugin
def plugin_1(a, b) -> int:
    return a + b

@some_slot.plugin
def plugin_2(a, b) -> int:
    return a + b + 1

print(some_slot(1, 2))
#> {'plugin_1': 3, 'plugin_2': 4}
```

github.com/mutating/pristan

Пример компонента: metacode

- Многие тулзы используют машиночитаемые комментарии
- Мне это нужно, чтобы мьютить мутации
- Я заметил, что нет общего стандарта вокруг этого и предложил свой

github.com/mutating/metacode

Когда?

Скоро!

(мне можно писать по поводу закрытого тестирования)

Роадмап:

- Фреймворк – весна 2026
- Агенты, покрывающие код тестами
- Комплексное решение для создания кода, соответствующего спеке

Следите за анонсами!

Ссылки:

- Гитхаб: github.com/mutating

За красивую тему для презентации отдельная благодарность компании Evrone