Поговорим про performance-тестирование

Андрей Акиньшин, JetBrains

DotNext 2017 Moscow

О чём будем разговаривать

Мы хотим:

О чём будем разговаривать

Мы хотим:

1. Не допустить performance-деградаций

О чём будем разговаривать

Мы хотим:

1. Не допустить performance-деградаций
2. Если допустили, то вовремя об этом узнать

О чём будем разговаривать

Мы хотим:

1. Не допустить performance-деградаций
2. Если допустили, то вовремя об этом узнать
3. Снизить количество ошибок первого рода
   (Проблемы нет, а мы думаем, что есть)

О чём будем разговаривать

Мы хотим:

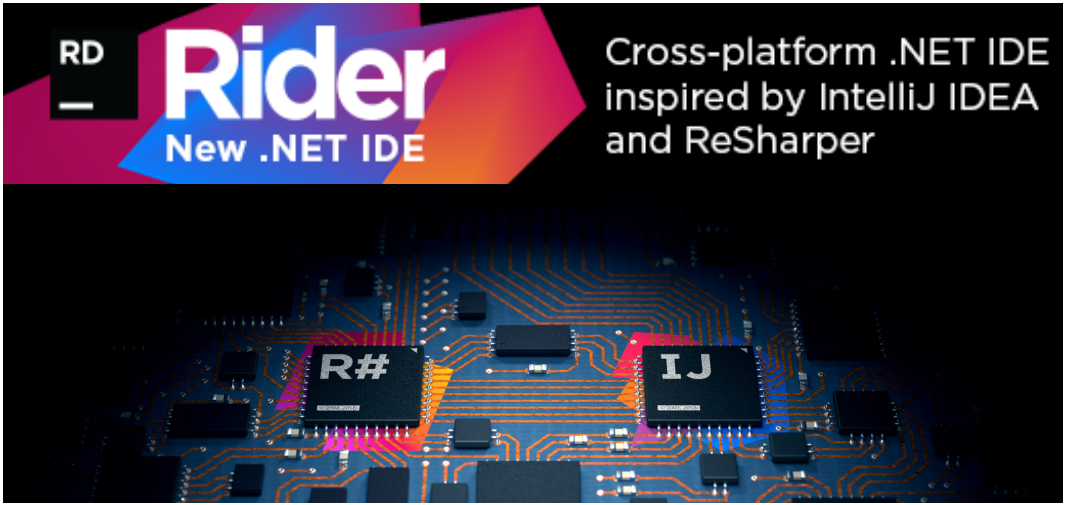
1. Не допустить performance-деградаций
2. Если допустили, то вовремя об этом узнать
3. Снизить количество ошибок первого рода
   (Проблемы нет, а мы думаем, что есть)
4. Снизить количество ошибок второго рода
   (Проблема есть, а мы думаем, что нет)

О чём будем разговаривать

Мы хотим:

1. Не допустить performance-деградаций
2. Если допустили, то вовремя об этом узнать
3. Снизить количество ошибок первого рода *(Проблемы нет, а мы думаем, что есть)*
4. Снизить количество ошибок второго рода *(Проблема есть, а мы думаем, что нет)*
5. Автоматизировать всё, что можно

Источник для вдохновения



Rider

- Файлов: 200'000+
- Строчек кода: 20'000'000+
- Тестов: 100'000+
- Зависимости: IntelliJ IDEA + ReSharper
- Рантаймы внутри: JVM, .NET Framework/Mono
- Целевые ОС: Windows, Linux, macOS

Rider

- Файлов: 200'000+
- Строчек кода: 20'000'000+
- Тестов: 100'000+
- Зависимости: IntelliJ IDEA + ReSharper
- Рантаймы внутри: JVM, .NET Framework/Mono
- Целевые ОС: Windows, Linux, macOS

Хочется, чтобы Rider работал очень быстро...

Заметка 1
Источники performance-данных

Источники performance-данных: крупно

«Специальные» performance-тесты

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт
- Performance-тесты на GUI

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт
- Performance-тесты на GUI
- Нагрузочное тестирование (Яндекс.Танк, JMeter и т.д.)

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт
- Performance-тесты на GUI
- Нагрузочное тестирование (Яндекс.Танк, JMeter и т.д.)
- Автоматизированный поиск точек отказа (capacity planning)

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт
- Performance-тесты на GUI
- Нагрузочное тестирование (Яндекс.Танк, JMeter и т.д.)
- Автоматизированный поиск точек отказа (capacity planning)
- Тестирование асимптотики

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт
- Performance-тесты на GUI
- Нагрузочное тестирование (Яндекс.Танк, JMeter и т.д.)
- Автоматизированный поиск точек отказа (capacity planning)
- Тестирование асимптотики
- ...

Источники performance-данных: крупно

«Специальные» performance-тесты

- Микробенчмарки 
- Холодный и горячий старт
- Performance-тесты на GUI
- Нагрузочное тестирование (Яндекс.Танк, JMeter и т.д.)
- Автоматизированный поиск точек отказа (capacity planning)
- Тестирование асимптотики
- ...

«Обычные» тесты

- Все тесты, которые у вас уже есть (>10ms)

Обычные тесты vs. Специальные тесты

Обычные тесты	Специальные тесты

Обычные тесты vs. Специальные тесты

Обычные тесты	Специальные тесты
Случайные машины	Выделенные машины

Обычные тесты vs. Специальные тесты

Обычные тесты	Специальные тесты
Случайные машины	Выделенные машины
Виртуальное окружение	Реальное железо

Обычные тесты vs. Специальные тесты

Обычные тесты	Специальные тесты
Случайные машины	Выделенные машины
Виртуальное окружение	Реальное железо
Легко писать	Сложно писать

Обычные тесты vs. Специальные тесты

Обычные тесты	Специальные тесты
Случайные машины	Выделенные машины
Виртуальное окружение	Реальное железо
Легко писать	Сложно писать
Много и бесплатно	Мало и дорого

Много данных не бывает!

Можно собирать и анализировать:

- Время прохождения теста
- Время прохождения отдельных частей теста
- Железные данные: CPU, Memory, Disk, Network, ...
- Характеристики рантайма (например, GC.CollectionCount)
- По каждой метрике смотрим на: Min/Max/Mean/Median/Q1/Q3/Percentiles/Distribution

Много данных не бывает!

Можно собирать и анализировать:

- Время прохождения теста
- Время прохождения отдельных частей теста
- Железные данные: CPU, Memory, Disk, Network, ...
- Характеристики рантайма (например, GC.CollectionCount)
- По каждой метрике смотрим на: Min/Max/Mean/Median/Q1/Q3/Percentiles/Distribution

Не забываем про проблему множественных сравнений ...

Заметка 2
История

Поучительная история

Однажды, после перехода с Mono 4.9 на Mono 5.2 наши perf-тесты стали падать ...

Поучительная история

Однажды, после перехода с Mono 4.9 на Mono 5.2 наши perf-тесты стали падать ...

```
private readonly IList<object> array = new string[0];

[Benchmark]
public int CountProblem() => array.Count;
```

Поучительная история

Однажды, после перехода с Mono 4.9 на Mono 5.2 наши perf-тесты стали падать ...

```
private readonly IList<object> array = new string[0];

[Benchmark]
public int CountProblem() => array.Count;
```

	Linux	macOS
Mono 4.9	≈5ns	≈5ns
Mono 5.2	≈1400ns	≈2600ns

Заметка 3
Оборона против performance-деградаций

Виды обороны

Виды обороны

- **Merge robot**
  Проверяем производительность на каждый мердж

Виды обороны

- **Merge robot**
  Проверяем производительность на каждый мердж
- **Daily tests**
  Запускаем тесты каждый день

Виды обороны

- **Merge robot**
  Проверяем производительность на каждый мердж
- **Daily tests**
  Запускаем тесты каждый день
- **Retrospective**
  Анализируем исторические данные

Виды обороны

- **Merge robot**
  Проверяем производительность на каждый мердж
- **Daily tests**
  Запускаем тесты каждый день
- **Retrospective**
  Анализируем исторические данные
- **Check points**
  Проверяем опасные изменения

Виды обороны

- **Merge robot**
  Проверяем производительность на каждый мердж
- **Daily tests**
  Запускаем тесты каждый день
- **Retrospective**
  Анализируем исторические данные
- **Check points**
  Проверяем опасные изменения
- **Pre-release**
  Проверяем регрессию перед самым релизом

Сравнение видов обороны

	Обнаружение	Точность	Процесс

Сравнение видов обороны

	Обнаружение	Точность	Процесс
Merge robot	Вовремя	Средняя	Автоматический

Сравнение видов обороны

	Обнаружение	Точность	Процесс
Merge robot	Вовремя	Средняя	Автоматический
Daily tests	Поздно	Хорошая	Полуручной
Retrospective	Слишком поздно	Отличная	Полуручной

Сравнение видов обороны

	Обнаружение	Точность	Процесс
Merge robot	Вовремя	Средняя	Автоматический
Daily tests	Поздно	Хорошая	Полуручной
Retrospective	Слишком поздно	Отличная	Полуручной
Check points	Вовремя	Отличная	Ручной
Pre-release	Зависит	Отличная	Ручной

Сравнение видов обороны

	Обнаружение	Точность	Процесс
Merge robot	Вовремя	Средняя	Автоматический
Daily tests	Поздно	Хорошая	Полуручной
Retrospective	Слишком поздно	Отличная	Полуручной
Check points	Вовремя	Отличная	Ручной
Pre-release	Зависит	Отличная	Ручной

И ещё несколько важных факторов:

- Время прогона
- Количество билд-агентов и их стоимость
- Спектр проблем, которые можно обнаружить

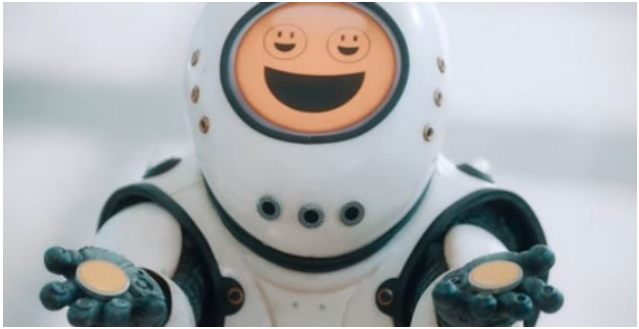
Заметка 4
Когда бить тревогу?

Alarm-критерии

Ручной

- Специальный программист, который мониторит тесты 24/7

Alarm-критерии



Автоматизированный

- Абсолютный timeout
- Относительный timeout
- Исторические данные и статистика

Полуавтоматизированный

- Система, которая формирует список performance-проблем
- Специальный программист, который разбирает этот список

Заметка 5
Performance-аномалии

Охота за performance-аномалиями

Охота за performance-аномалиями

- **Performance-пространство** — множество всех performance-метрик тестов на всевозможных конфигурациях.

Охота за performance-аномалиями

- **Performance-пространство** — множество всех performance-метрик тестов на всевозможных конфигурациях.
- **Performance-аномалия** — ситуация, при которой performance-пространство выглядит странно.

Охота за performance-аномалиями

- **Performance-пространство** — множество всех performance-метрик тестов на всевозможных конфигурациях.
- **Performance-аномалия** — ситуация, при которой performance-пространство выглядит странно.
- **Performance-чистота** — отсутствие performance-аномалий

Наведение performance-чистоты позволяет

Наведение performance-чистоты позволяет

- Находить проблемы, на которые не стоят assert'ы

Наведение performance-чистоты позволяет

- Находить проблемы, на которые не стоят assert'ы
- Ускорять билд

Наведение performance-чистоты позволяет

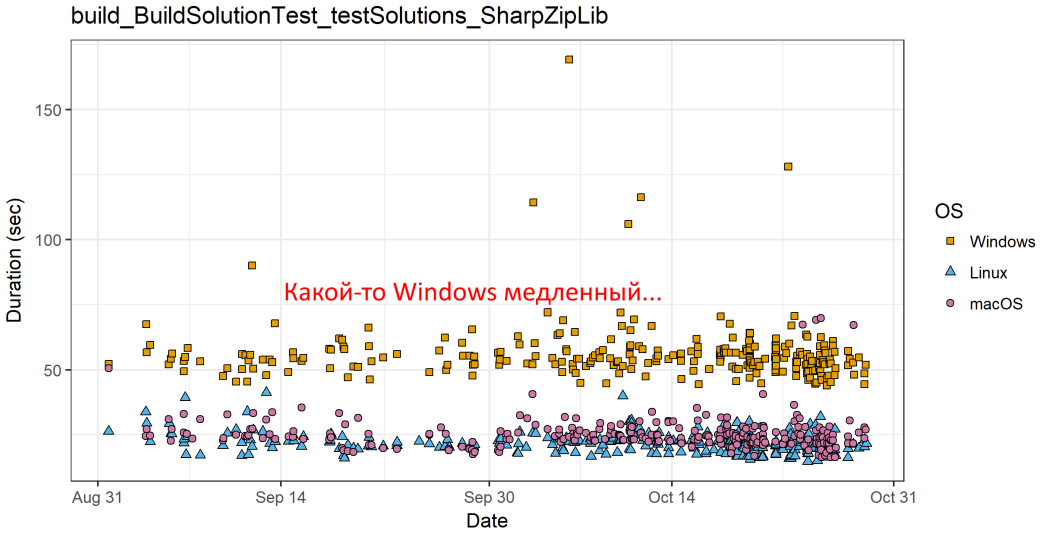
- Находить проблемы, на которые не стоят assert'ы
- Ускорять билд
- Находить flaky-тесты

Наведение performance-чистоты позволяет

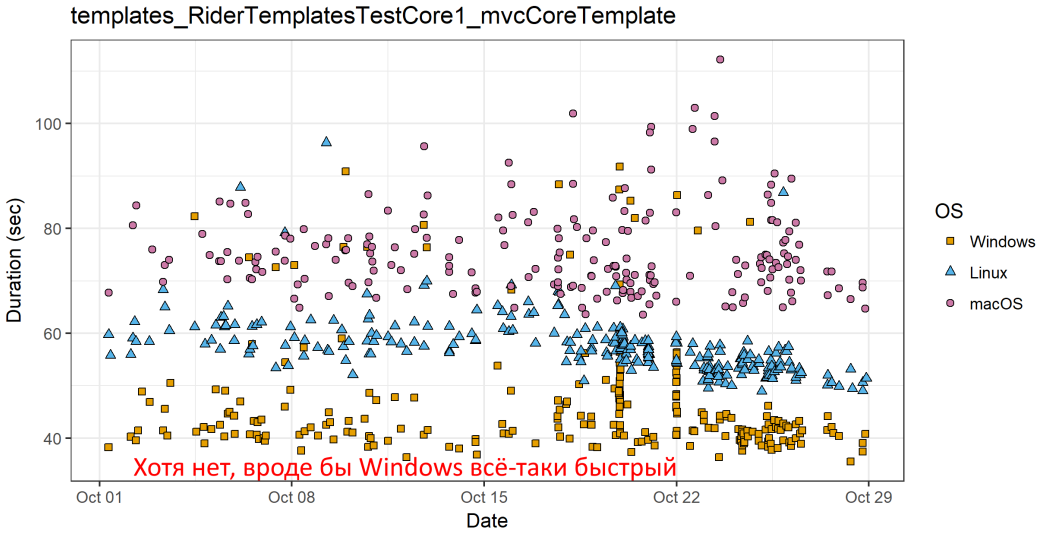
- Находить проблемы, на которые не стоят assert'ы
- Ускорять билд
- Находить flaky-тесты
- Уменьшать количество тестов, которые в будущем помешают найти действительно критичные проблемы

Заметка 6
Кластеризация

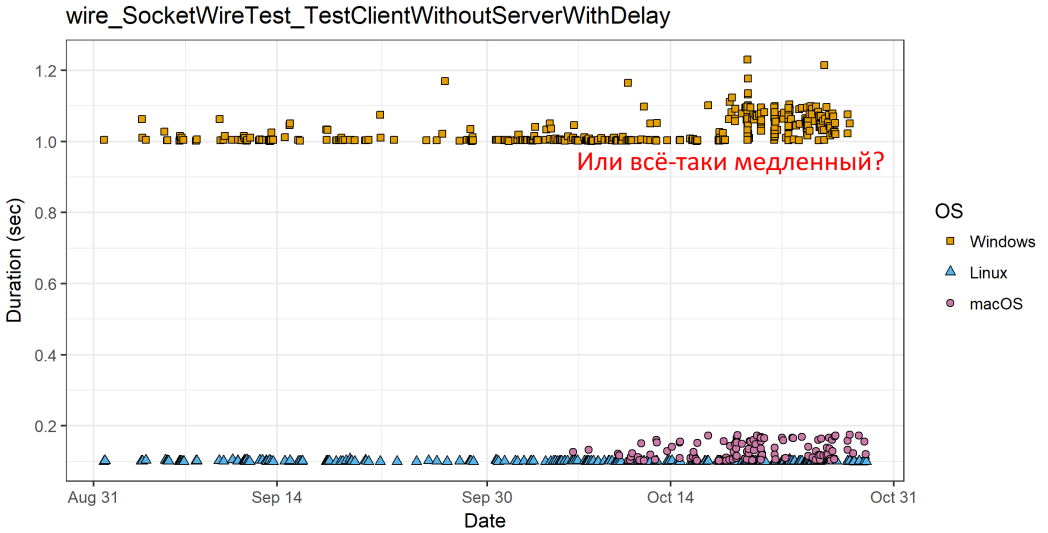
Какая OS самая быстрая?



Какая OS самая быстрая?

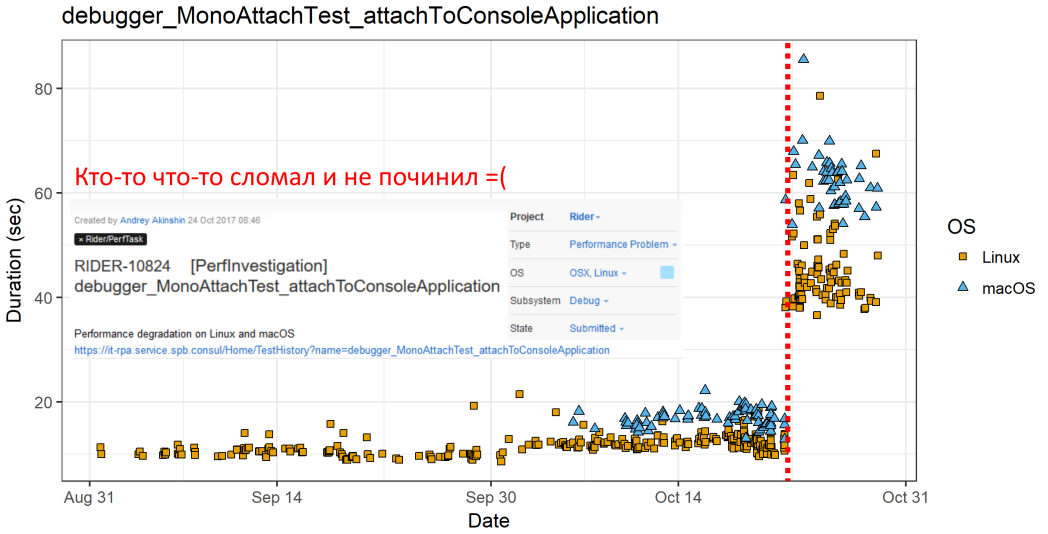


Какая OS самая быстрая?

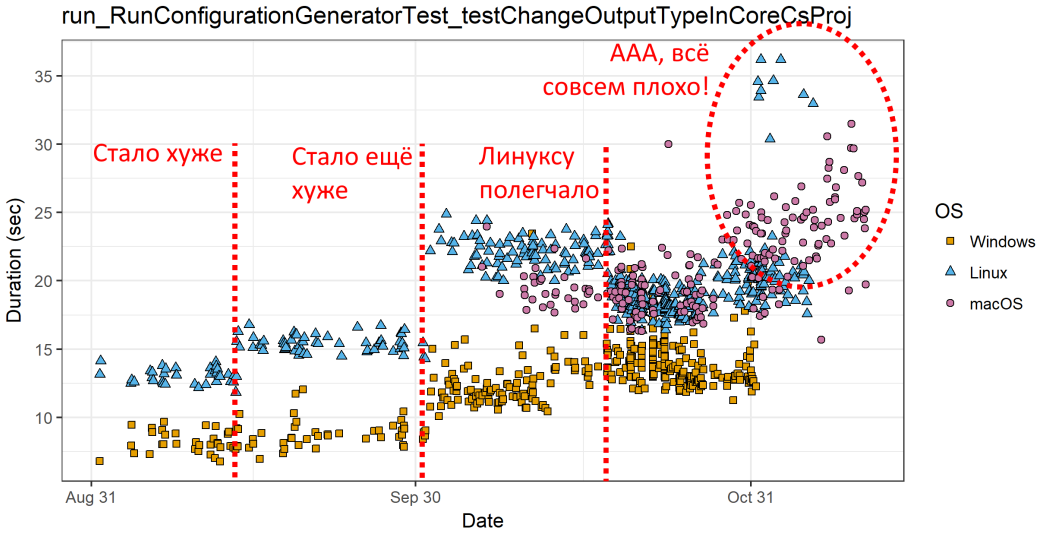


Заметка 7
Деградации: большие и маленькие

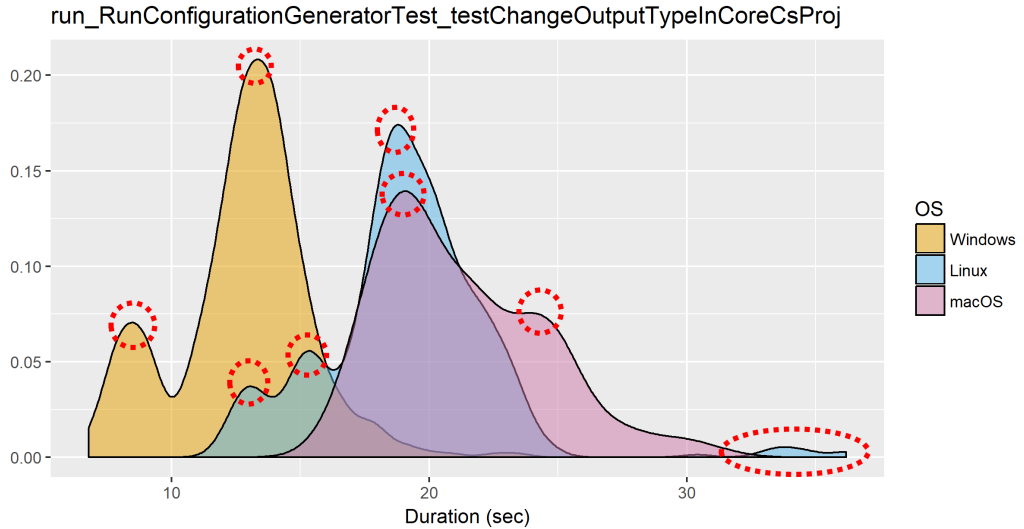
Большая деградация



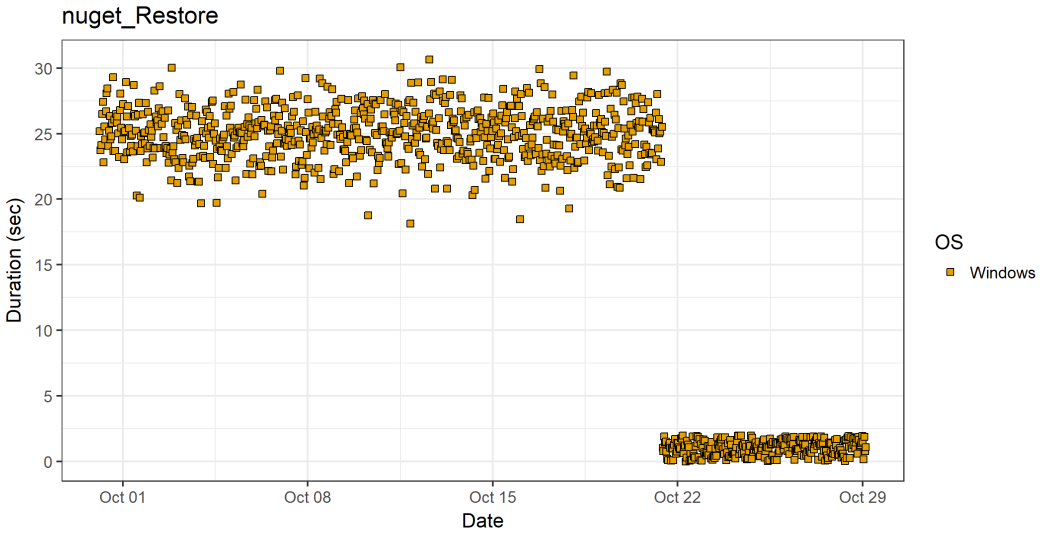
Маленькие деградации



Распределение



Внезапные улучшения



Заметка 8
Распределения странной формы

Плохой код

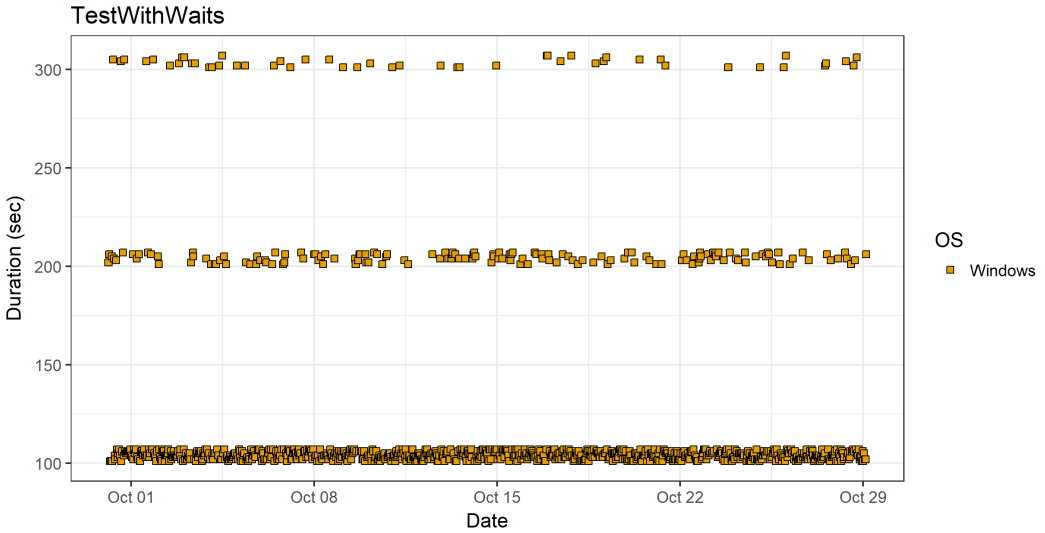
```
while (!isReady) {
  Thread.Sleep(100);
  // Update state
}
```

Плохой код

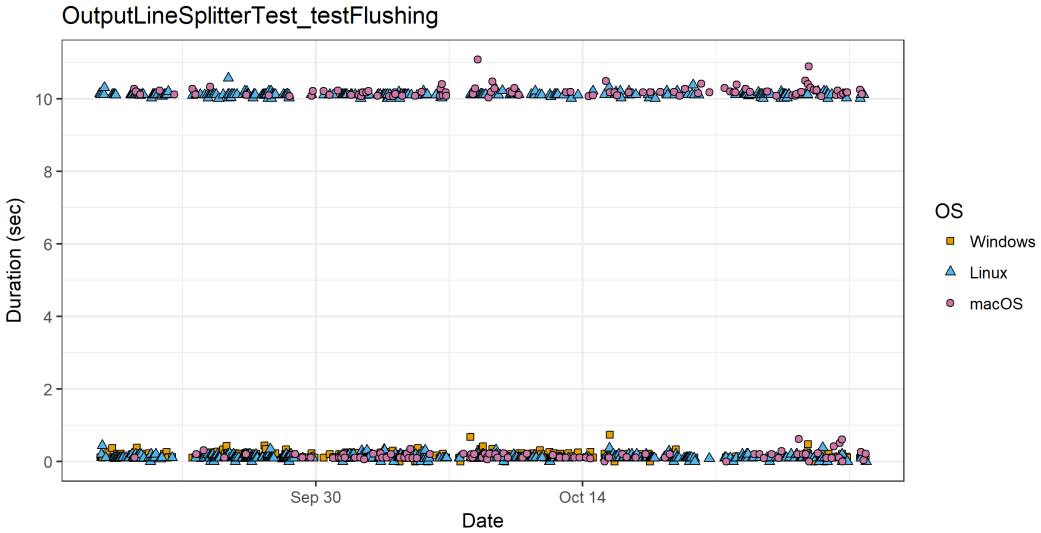
```
while (!isReady) {
  Thread.Sleep(100);
  // Update state
}
```

`Thread.Sleep` — источник *постоянных* проблем

Мультимодальное распределение



Бимодальное распределение



Заглядываем внутрь

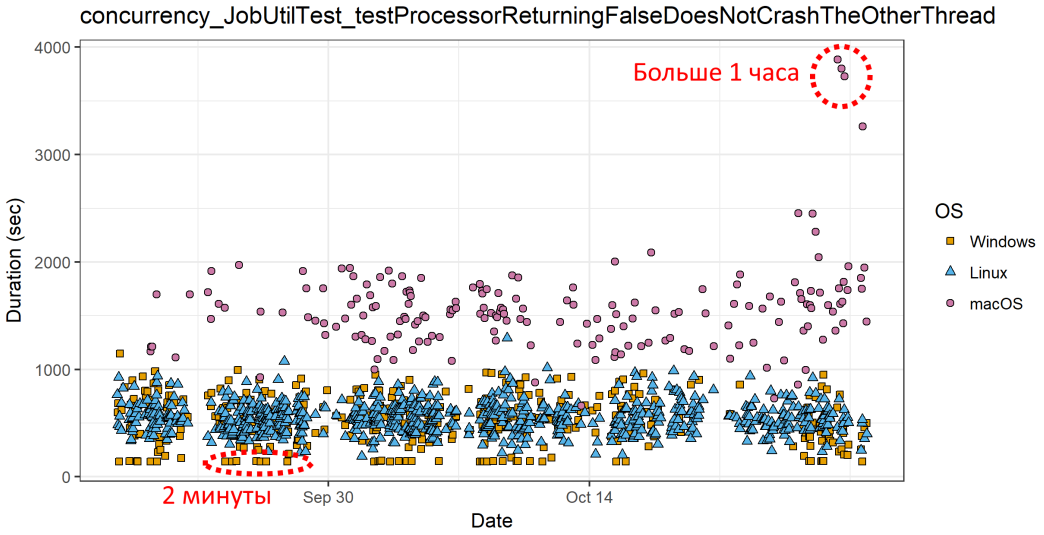
```
// OutputLineSplitterTest_testFlushing
while (!isFinished.get()) {
  mySplitter.process(StringUtil.repeat("A", 100), each);
  i++;
  if (i % 10 == 0) {
    written.release();
    try {
      if (!read.tryAcquire(10, TimeUnit.SECONDS))
        throw new TimeoutException();
    }
    catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
}
```

Просто странное распределение

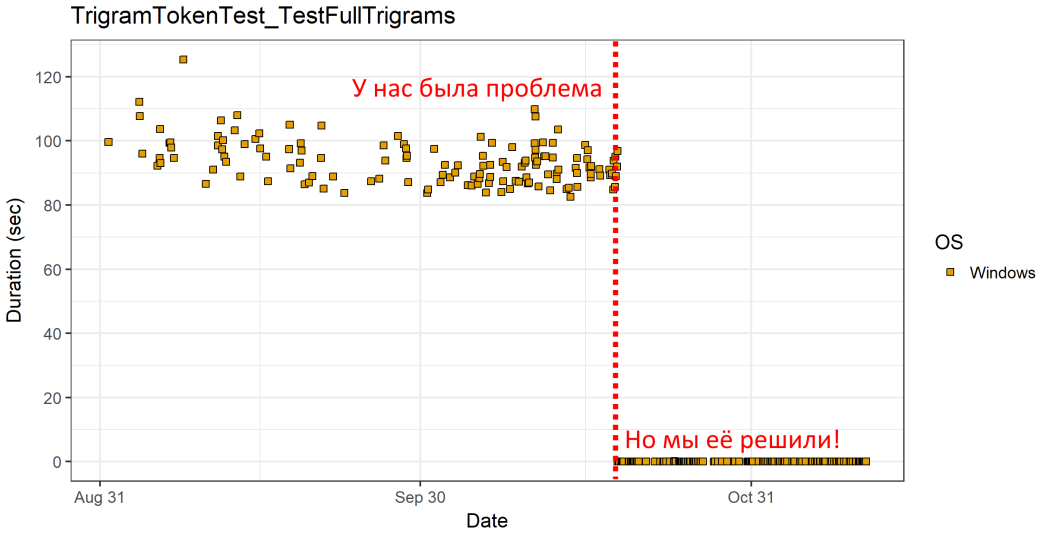
По мотивам экспериментов с .NET Core

Заметка 9
Большая дисперсия

Очень большая дисперсия



Подозрительная дисперсия



Смотрим тест

```
[Test]
public void TestFullTrigrams()
{
  const int start = 256;
  const int limit = 512;
  for (var a = start; a <= limit; ++a)
    for (var b = start; b <= limit; ++b)
      for (var c = start; c <= limit; ++c)
        TestTrigramToken((char)a, (char)b, (char)c);
}
```

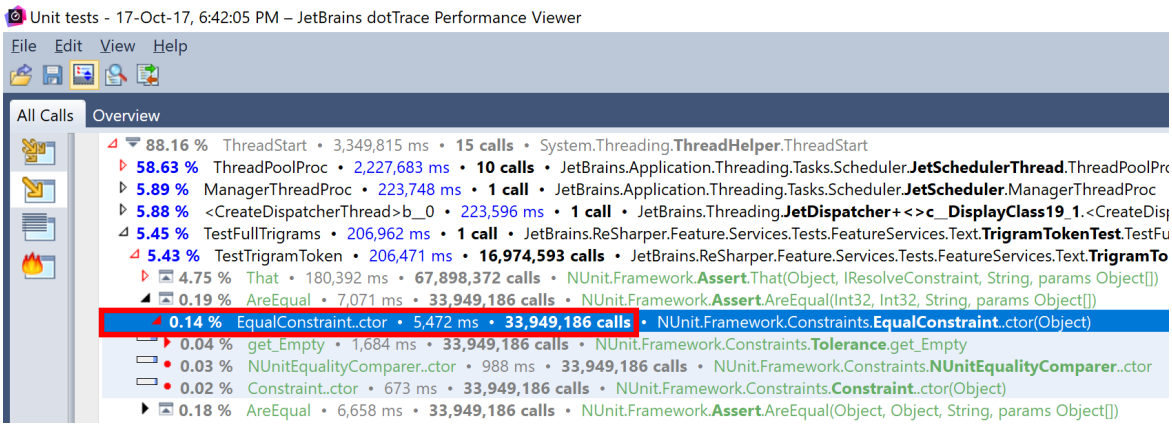
Смотрим тест

```
private static void TestTrigramToken(char a, char b, char c)
{
  var testedTrigram = new TrigramToken(a, b, c);
  var testedTrigramHash = (int)testedTrigram;
  var expectedIsShort = ((a | b | c) >> 8 == 0);

  Assert.AreEqual(expectedIsShort, testedTrigram.IsShort());

  Assert.AreEqual(c,
    TrigramToken.ExtractLastCharacter(testedTrigramHash));
}
```

Смотрим снэпшот



Смотрим исходники

```
// NUnit 2.6.4
public static void AreEqual(int expected, int actual)
{
    Assert.That(actual, Is.EqualTo(expected), null, null);
}

public static void AreEqual(object expected, object actual)
{
    Assert.That(actual, Is.EqualTo(expected), null, null);
}

public static EqualConstraint EqualTo(object expected)
{
    return new EqualConstraint(expected);
}
```

Заметка 10
Когда аномалия есть, а проблемы нету

Ложные друзья performance-инженера

Ложные друзья performance-инженера

- Изменения в тесте или порядке тестов

Ложные друзья performance-инженера

- Изменения в тесте или порядке тестов
- Изменения в агенте или окружении

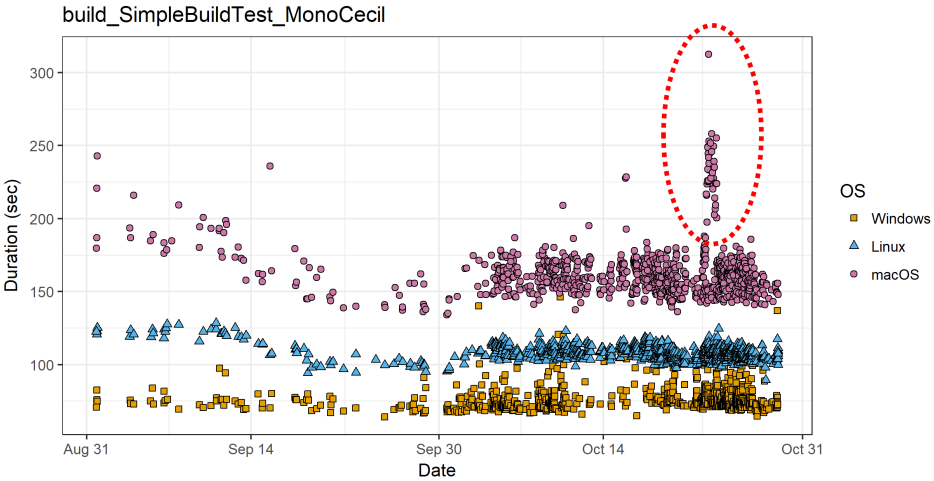
Ложные друзья performance-инженера

- Изменения в тесте или порядке тестов
- Изменения в агенте или окружении
- Изменения в tradeoff-ax

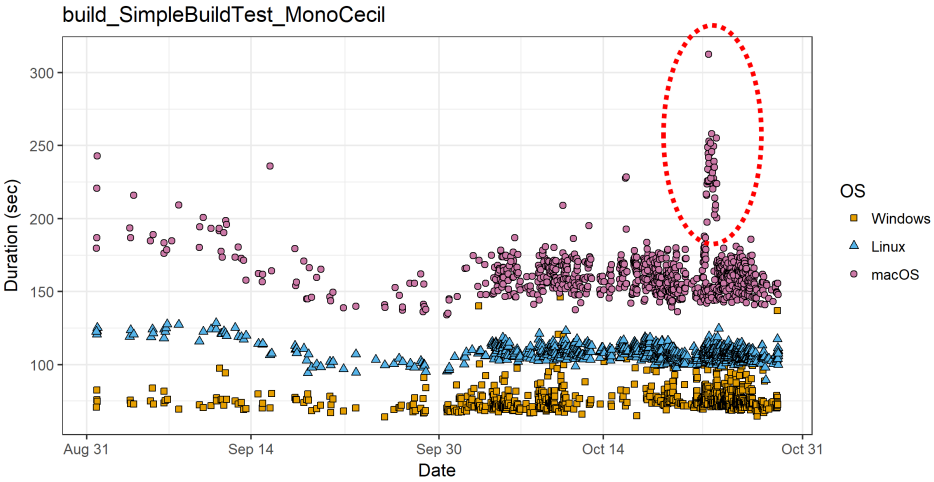
Ложные друзья performance-инженера

- Изменения в тесте или порядке тестов
- Изменения в агенте или окружении
- Изменения в tradeoff-ax
- Да и вообще, любые изменения

Найди деградацию!



Найди деградацию!



А её нету, просто macOS-агентам было плохо. . .

Заметка 11
Performance Driven Development (PDD)

Performance Driven Development (PDD)

Performance Driven Development (PDD)

1. Пишем тест

Performance Driven Development (PDD)

1. Пишем тест
2. Собираем performance-метрики и задаём performance-требования

Performance Driven Development (PDD)

1. Пишем тест
2. Собираем performance-метрики и задаём performance-требования
3. Проверяем, что тест красный

Performance Driven Development (PDD)

1. Пишем тест
2. Собираем performance-метрики и задаём performance-требования
3. Проверяем, что тест красный
4. Делаем рефакторинг и оптимизации

Performance Driven Development (PDD)

1. Пишем тест
2. Собираем performance-метрики и задаём performance-требования
3. Проверяем, что тест красный
4. Делаем рефакторинг и оптимизации
5. Проверяем, что тест зелёный
   (И все остальные тесты тоже)

Performance Driven Development (PDD)

1. Пишем тест
2. Собираем performance-метрики и задаём performance-требования
3. Проверяем, что тест красный
4. Делаем рефакторинг и оптимизации
5. Проверяем, что тест зелёный
   (И все остальные тесты тоже)
6. Возвращаемся к шагу 1

Заметка 12
Performance Culture

Performance Culture

Заметка 13
Заключительные мысли

Мораль

Мораль

- Performance-тестирование — увлекательно и полезно, но сложно.

Мораль

- Performance-тестирование — увлекательно и полезно, но сложно.
- Тестировать performance можно и нужно.

Мораль

- Performance-тестирование — увлекательно и полезно, но сложно.
- Тестировать performance можно и нужно.
- Соблюдайте performance-чистоту!

Мораль

- Performance-тестирование — увлекательно и полезно, но сложно.
- Тестировать performance можно и нужно.
- Соблюдайте performance-чистоту!
- Будьте performance-культурными!

Вопросы?

Андрей Акиньшин
http://aakinshin.net
https://github.com/AndreyAkinshin
https://twitter.com/andrey_akinshin
andrey.akinshin@gmail.com