本日のおすすめテストの作り方

～忍者式テストのテスト抽出アルゴリズム～

ninja-testing.com

RubyStackNews

Independent Ruby & Rails publication — curated technical content

rubystacknews.com

Why sponsor RubyStackNews?

- Highly targeted Ruby / Rails audience
- Readers from US, Europe, and Asia
- Organic traffic from search and developer communities
- No ad networks — direct partnerships only

This article is based on a conference presentation. RubyStackNews turns talks into practical, production-ready insights for working developers.

Sponsorship options

📝 Article Sponsorship
Brand mention inside a technical article
USD 250

📌 Inline Sponsored Block
Highlighted block embedded in an article
USD 100 / week

📎 Sidebar Sponsor
Logo + link site-wide in the sidebar
USD 150 / month

Partner via WhatsApp

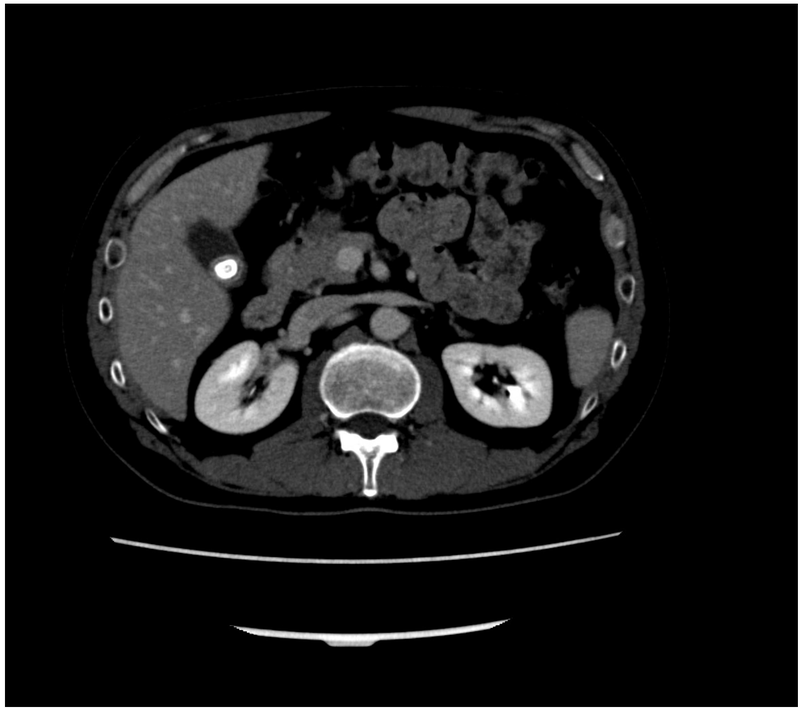
Agenda

- 忍者式テストのおさらい
- 本日のおすすめテストの作り方

忍者式テストについてはこちらのスライドを見てね👉https://speakerdeck.com/mseki/z5

自己紹介

私たちは X 線 CT 装置に搭載するソフトウェアを XP で開発しています



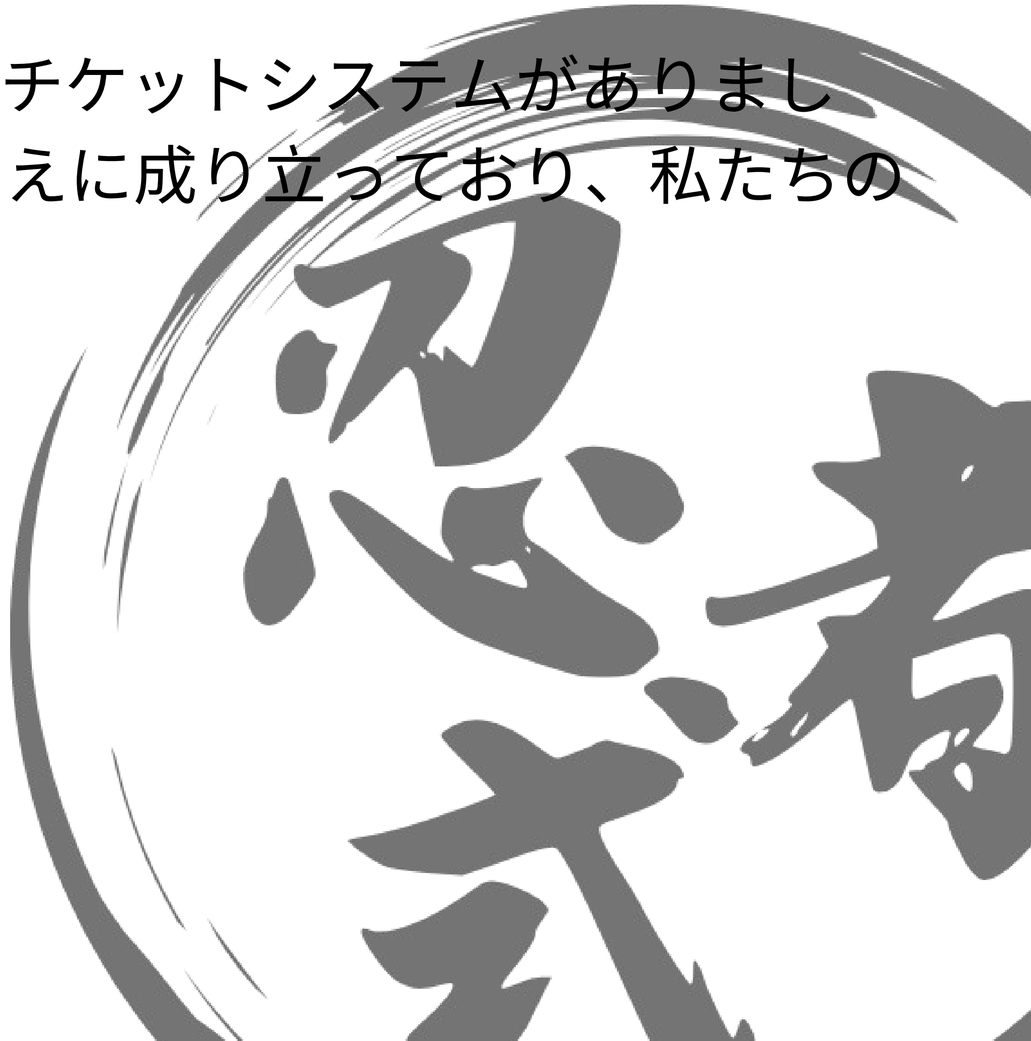
関将俊 / プログラマ

深谷美和 / テスター

この X 線 CT 画像は私たちの開発した装置で撮影した私たちの中身（人間ドックの再検査で撮ってもらったよ！）

RubyWorld Conference によせて

- 私たちのチームの活動の中心には、 Ruby 製のチケットシステムがありました。忍者式テストもこのチケットシステムのうえに成り立っており、私たちの 25 年間は Ruby とともにあったのです。



ビジネスの話です！！

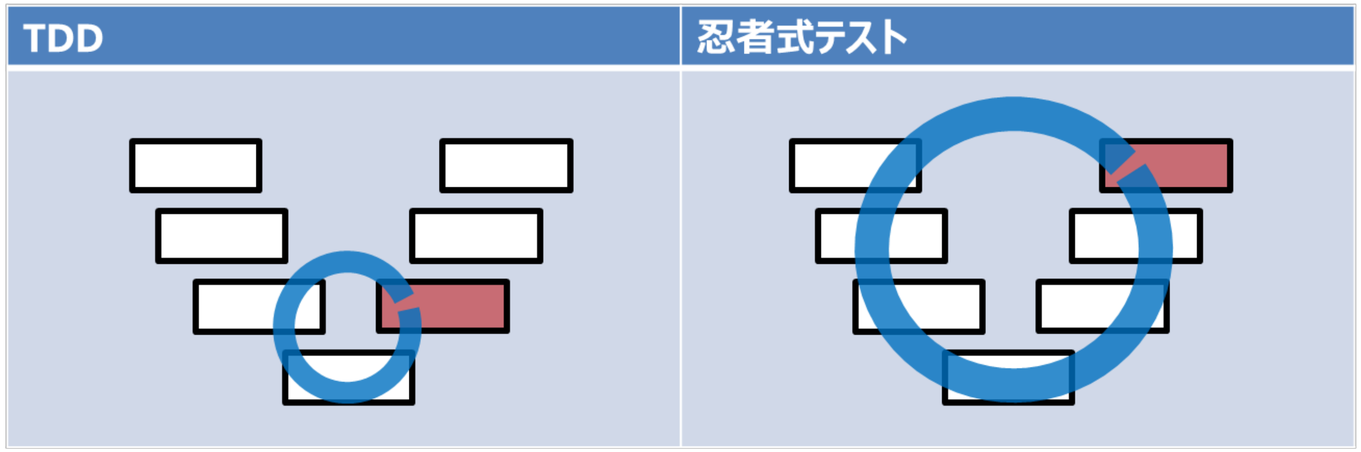
忍者式テストは TDD の自然な拡張

忍者式テスト

- テス
  - x
  - テ
  - ン

➢ テスト駆動開発（ TDD ）を受け入れテストのレベルで行うプラクティス

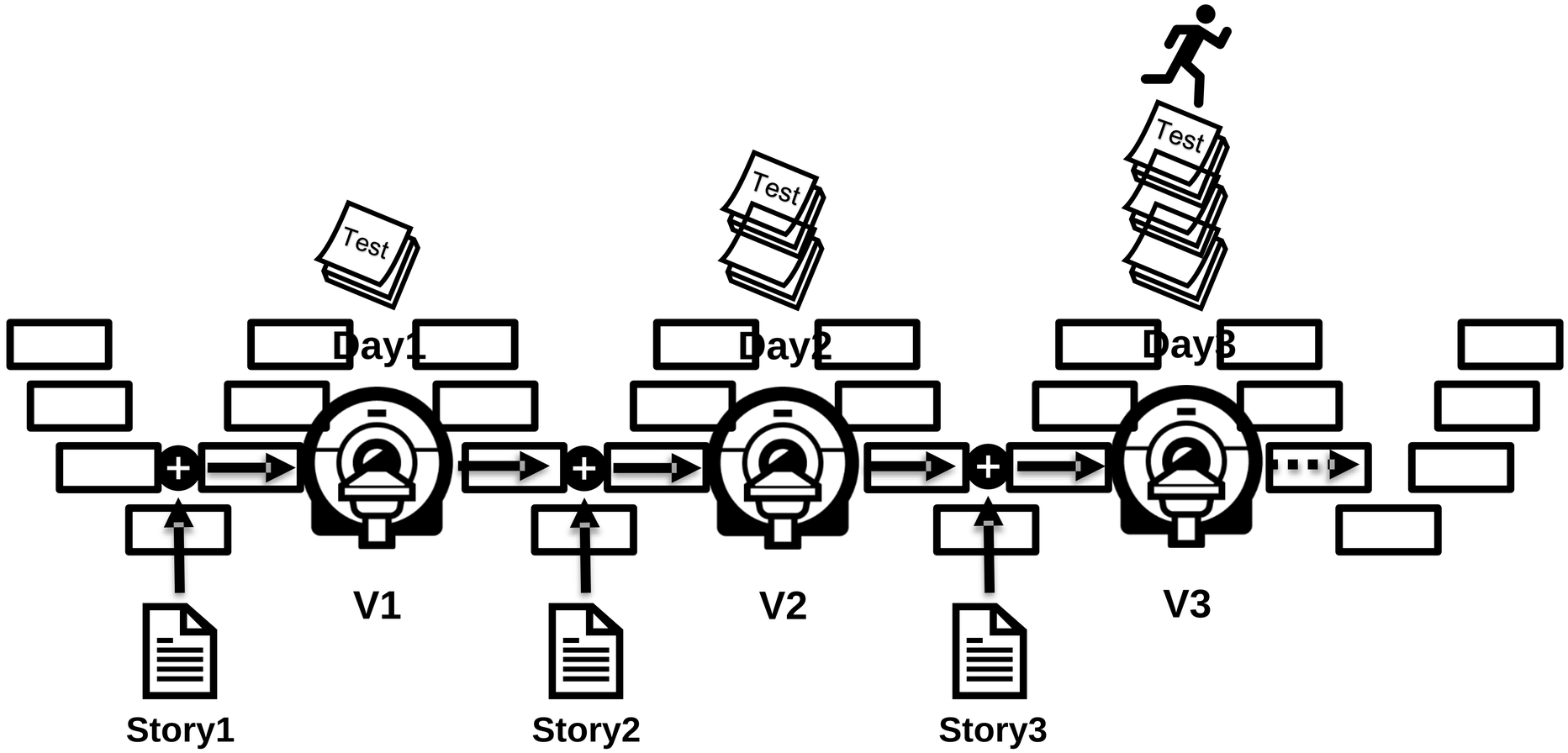
- xUnit を使った TDD のように、受け入れテストからはじめる開発
- テストコードに導かれてプログラムを開発するように、受け入れテストに導かれてストーリーを実現する
- ストーリーが増えるたびに、それまでに作ったすべてのストーリーの受け入れテストをやり直す



忍者が毎 名前は「テスト」ですがテストだけでなく、ソフトウェア開発全体の活動です。

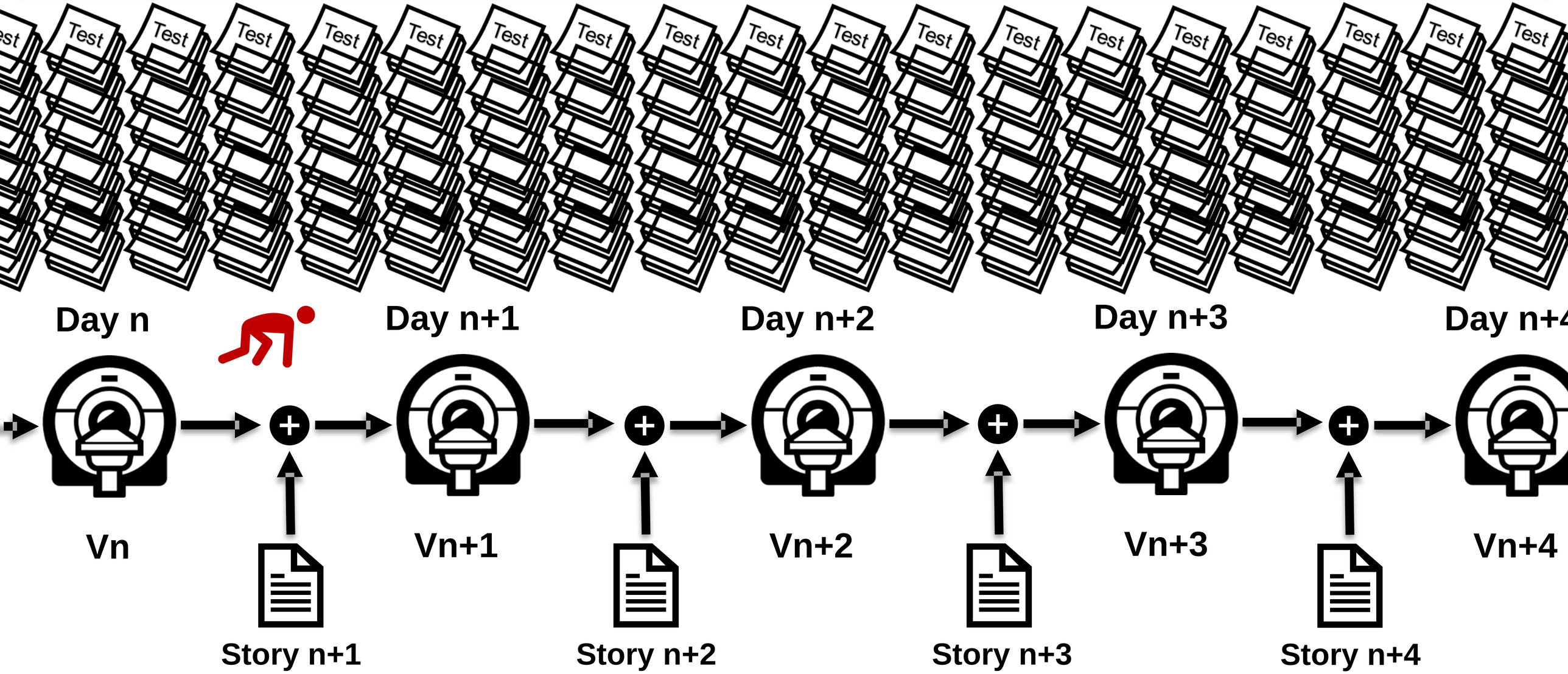
【図解】忍者式テスト

【図解】忍者式テスト



実際の開発では１イテレーションで複数のストーリーを扱ってね

たいへんなことになります



忍者式テストのテストは問題を探すテスト
ソフトウェアのテストとはなにか

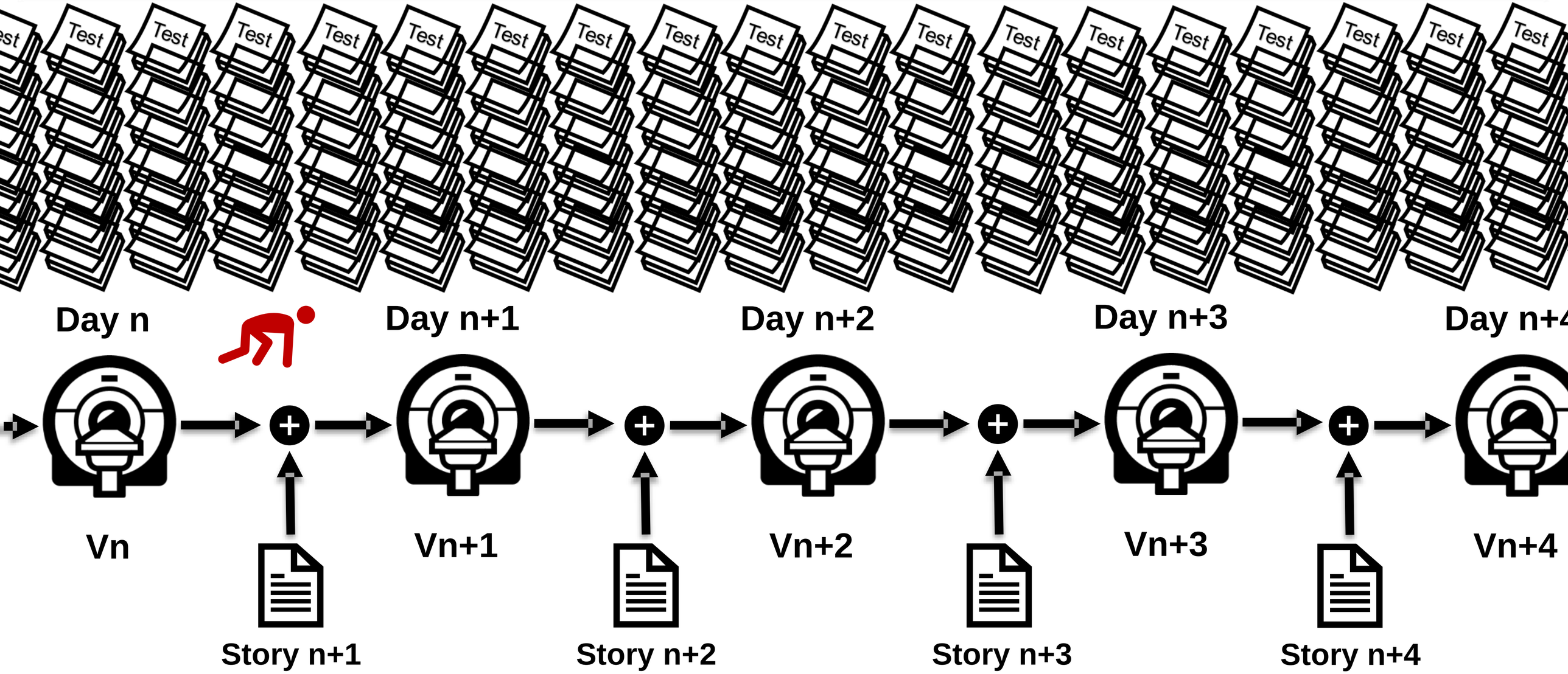
- 気に入らないことを探して、もっと良くするためにテストする
  - うまくいったテスト、というのは新しい問題を見つけられたテスト
  - 失敗したテスト、というのはなにも問題が見つからなかったテスト

エラーを発見する目的でプログラムを実行する過程がテストである.

(ソフトウェアの信頼性 p.194)

問題を見つけるために手動でテストします！！（自動テストでは知らない問題を見つけるのは難しい）

たいへんなことになります



規模と向き合う

- 直近の変更は早く確かめたい／すべてのテストケースを確かめたい

- 一日ではなく、ある期間でテストケースをすべて回す作戦に切り替えた
- まんべんなく、かつ、効果の高いテストケースを抽出するアルゴリズムの開発
  - 新しいストーリー、修正したばかりのチケットのテストケースを最優先
  - 前回のテスト結果がパスしたテストケースの出現頻度を徐々に落とす
  - 開発の状況に応じて機能ごとに出現頻度を調整
  - ある期間で見ると、すべてのテストケースが実行できる
- アルゴリズムにより抽出した今日のテストスイート → 「本日のおすすめテスト」
  - 一日にできそうな量のテストケースしか表示しない（量に圧倒されないようにする）

👋やっと本編がはじまりますよ！

○●● 本日のおすすめテストの作り方

今日はこれ!

- アイデア
- 実装
- チケットシステムへの組み込み

アイデアの背景

- ストリーミング以前の音楽の聴き方
- 全ての音楽ライブラリを持ち歩く！

- 毎日ライブラリが増えるとたいへんなことになる
  - 新しい曲がなかなか聴けない（シャッフル再生）
  - ライブラリが端末の容量を超えてしまう
- スマートプレイリストの工夫で解決してた

これは!

- テストスイートの生成に使えるのでは?
- ある期間で全曲再生するスマートプレイリスト

- ある期間で全部テストするテストスイート

毎日一回再生したい

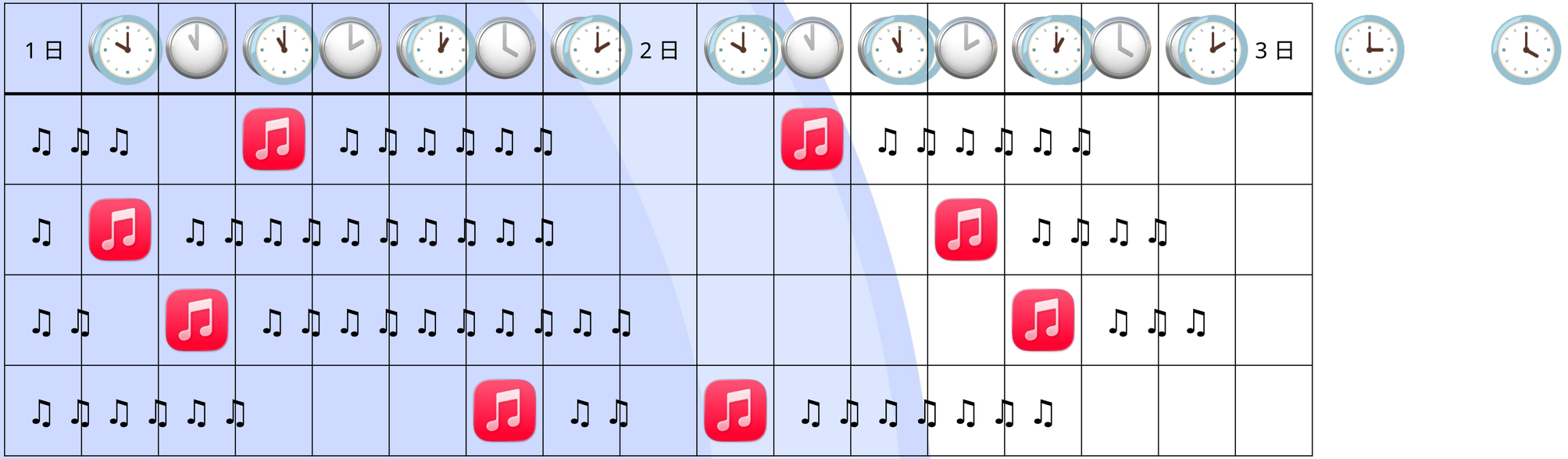
毎日一回再生するプレイリスト

- 最後に再生した日が 1 日以内でない曲

スマートプレイリストは検索条件に従って動的にプレイリストが変化する

毎日一回再生するプレイリスト

- 再生🎵するとリストから消えて、次の日に復活する



1日に同じ曲を何度も聴かない!

毎日一回~~再生~~テストしたい

- 音楽の「再生」を「テスト」に置き換えてみます!

○●● 毎日一回テストするテストスイート

- 最後にテストした日が 1 日以内でないテストを集めれば良い

- 「1 日」を「ある期間」に拡張してみよう

ある期間で全部再生したい

ある期間で全部再生するプレイリスト

- 最後に再生した日が n 日以内でない曲

ある期間で全部再生するプレイリスト

再生したらしばらくお休み

一度聴いた曲はしばらく再生されなくていい感じ！

ある期間で全部テストしたい

- 一度テストしたらある期間お休みするテストスイート
- 期間はチームがこなせるテストの量で調整する
  - ギリギリよりちょっと多いくらいにする
  - その日にテストできなかったら次の日に繰り越し

その日のおすすめが終わらなくても次の日には優先して現れるので気にしない

さらに細かくチューニングしたい

- 新規／修正直後のチケットを最優先
- テストする頻度を調整する
- テーマごとのテストスイート（運用の工夫）

新規／修正直後のチケットを最優先

- 以下は休ませる期間に関係なくテストスイートに入れる
  - 新規＝テストの履歴がない
  - 修正直後＝最後のテスト結果が NG のもの

iTunes にはテスト結果が NG はないので、テストスイート生成用の特別な処理のこと

テストする頻度を調整する

- テストごとに★レートを設定する
- ★★☆☆☆ 自然に毎日さわる機能

（例）起動終了とか

- ★☆☆☆☆ 準備が大変→頻繁にはやりたくない

（例）ディスクフルの用意とか

- ★★★★★ なんとなくあやしいから毎日試したい
  - そういう機能あるでしょ？

レートによって休ませる期間を増減します

テーマごとのテストスイート

- 機能、バージョン、機種など、テーマごとにテストスイートがあるとうれしい
- 関連するテストが続くので集中でき、問題を探索しやすい
- 開発状況に応じて注目点の時間配分を加減できる
- テーマを「テスト分類」と呼びタグで管理してる

実装

```
def idle_period?(item)
  return false unless item.last_result

  period = [300, 90, 28, 14, 7, 1].fetch(item.test_rate)
  item.last_date > Time.now - period * 24 * 60 *60
end

...

list.delete_if { |item|
  idle_period?(item)
}.sort_by { |item|
  [item.last_result ? 1 : 0,
   item.last_date,
   item.name]
}.first(25)
```

並べ替え

休ませるテストの判定をするメソッドと、それを呼ぶ並べ替えのふたつ

休ませるテストの判定

- 最後のテストが設定された期間より新しいものは休み
- 最後の結果が NG のものは休ませない

```
def idle_period?(item)
  return false unless item.last_result

  period = [300, 90, 28, 14, 7, 1].fetch(item.test_rate)
  item.last_date > Time.now - period * 24 * 60 *60
end
```


true だと休みだよ

休ませるテストの判定

- 最後のテストが設定された期間より新しいものは休み
- 最後の結果が NG のものは休ませない

```
def idle_period?(item)
  return false unless item.last_result

  period = [300, 90, 28, 14, 7, 1].fetch(item.test_rate)
  item.last_date > Time.now - period * 24 * 60 *60
end
```


true だと休みだよ

並べ替え

- 休ませるテストを取り除く
- 最後のテストの実施が古い順に並べて N 件選ぶ

```
list.delete_if { |item|
  idle_period?(item)
}.sort_by { |item|
  [item.last_result ? 1 : 0,
   item.last_date,
   item.name]
}.first(25)
```

並べ替え

- 休ませるテストを取り除く
- 最後のテストの実施が古い順に並べて N 件選ぶ

```
list.delete_if { |item|
  idle_period?(item)
}.sort_by { |item|
  [item.last_result ? 1 : 0,
   item.last_date,
   item.name]
}.first(25)
```

並べ替え

- 休ませるテストを取り除く
- 最後のテストの実施が古い順に並べて N 件選ぶ

```
list.delete_if { |item|
  idle_period?(item)
}.sort_by { |item|
  [item.last_result ? 1 : 0, # 最後の結果が NG を優先
   item.last_date,           # 最後の実施日が古いものを優先
   item.name]                # 安定ソートのために名前順
}.first(25)
```

完成!

```
def idle_period?(item)
  return false unless item.last_result

  period = [300, 90, 28, 14, 7, 1].fetch(item.test_rate)
  item.last_date > Time.now - period * 24 * 60 *60
end

...

list.delete_if { |item|
  idle_period?(item)
}.sort_by { |item|
  [item.last_result ? 1 : 0,
   item.last_date,
   item.name]
}.first(25)
```

- あなたのチケットシステムに組み込もう!

テストケースごとの結果の履歴は各自準備してください! レートや分類のタグ付けはオプションだよ

チケットシステムへの組み込み

- RWiki

RWiki

- ERB，dRuby のサンプルとして書いた Wiki
- インメモリ OODB
- 書式は RD （ Ruby Document format ）

チケットの例

- 1 ページが 1 チケットに相当する
- テスト分類、レートも書いてある

XP3-1234

その時刻の天気が記録できる

status

- 種類: story
- イテレーション: 123
- サイン: seki
- 状態: open
- 見積: 1.0 / 0
- テスト分類:
- テストレート:

description

施設ごとに、天気が記録できる。

- 15分単位で指定できる。
- 操業記録のときについでに記録する。
- 一度入れたら、ロット別、工程別のどのビューにも

2019-9-1

操業記録用のイベントのロットIDを拡張して、""のときは

test

- Q: test..
- A: answer..

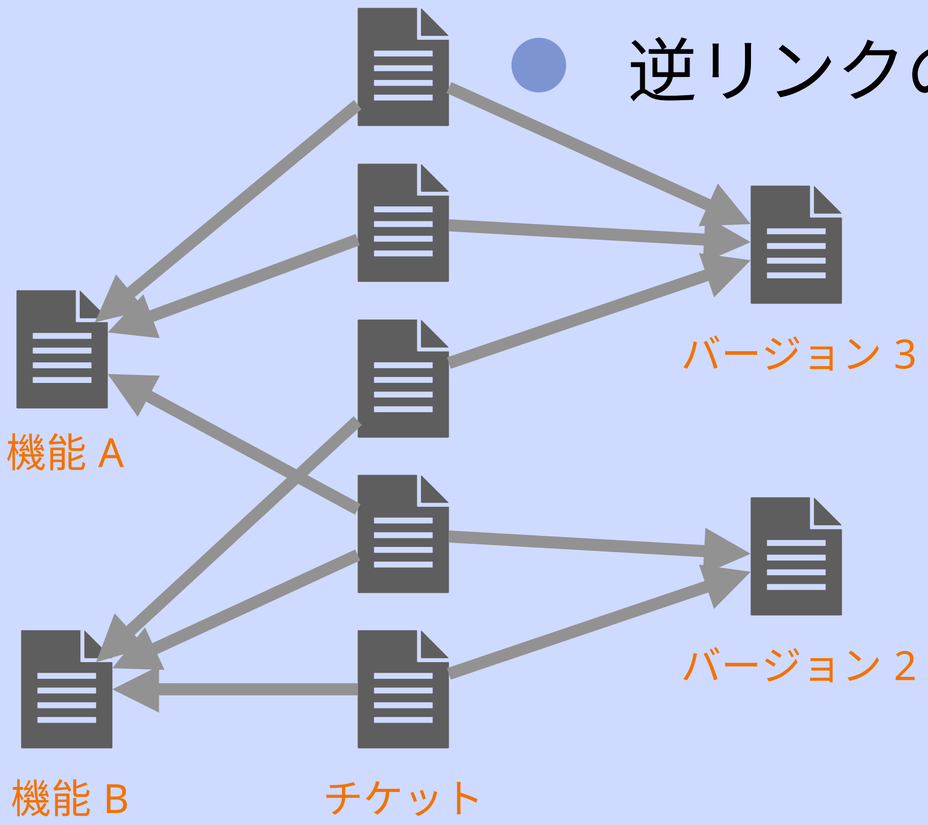
history

- 2019-09-11 rwiki: open

URL: http://localhost:8000/?cmd=view&name=XP3-1
last-modified: Wed, 11 Sep 2019 18:05:53 +0900 (32m)

リンクを利用したタグ付け

- チケットからテスト分類へリンクする
- 逆リンクの集合を使ってテストスイートを作る



テストと履歴

- チケットにはテストが書いてある
- テストした日と結果も記録

test

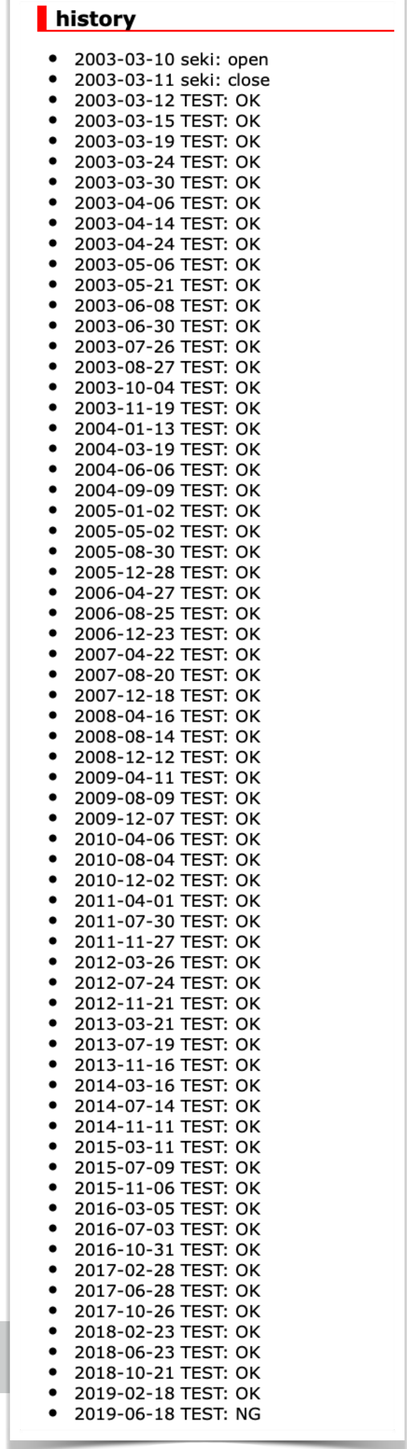
- Q1: あれしてこれする
- A1: こうなるはず
- Q2:
  1. 赤上げる
  2. 白下げる
  3. 赤下げないで白上げる
- A2: つかれる

history

- 2019-09-11 seki: open
- 2019-09-13 TEST: OK
- 2019-09-13 miwa: close
- 2019-09-14 TEST: OK
- 2019-09-15 TEST: NG

おまけ（履歴のすごいやつ）

- 2003 年のチケットが見直されたりする
- たまにテストが NG になる
  - 仕様が陳腐化するケースが多い
  - 新しい仕様が過去の仕様を上書きする
  - 実装の変化によりテストする意味がなくなる
    - テスト通り動くけど意味のなさを検出!



2003 年のチケットのテストが不要になっていることに 2019 年に気づいた例👉

Page から属性を取得する仕組み

- RDtool で RD を変換する過程で DOM 木が生成される。
  DOM 木をパターンマッチして属性リストを作る

```
irb(main):042:0> pp rwiki.page('XP3-1234').prop(:story)
{"summary"=>"その時刻の天気が記録できる\n",
 :summary=>"その時刻の天気が記録できる",
 "種類"=>"story",
 :card_type=>:story,
 "イテレーション"=>"123",
 :iteration=>123,
 "サイン"=>"[xpjug-2019] seki",
 :sign=>"[xpjug-2019] seki",
 "状態"=>"open",
 :status=>:open,
 "見積"=>"1.0 / 0",
 :estimation=>1.0,
 :actual=>0.0,
 "テスト分類"=>"",
 "テストレート"=>"",
 :test=>
  [["Q1: あれしてこれする"],
   ["A1: こうなるはず"],
   ["Q2:<ol>\n" +
    "<li>赤上げる</li>\n" +
    "<li>白下げる</li>\n" +
    "<li>赤下げないで白上げる</li>\n" +
    "</ol>"],
   ["A2: つかれる"]],
 :test_inline=>
```


YAML が登場する前、セ

テストスイート生成ツールの抜粋

```
index = rwiki['test-v1']                    # タグのページを取得
list = index.rev_links.map {|name|          # 逆リンクで map
  rwiki[name].prop(:story)                  # prop を集める
}

# さっき説明した選択と並び替え処理

rwiki['sorted-test-v1'].src = rd            # テストスイートの
                                            # ページを作成
```


ツールは RWiki と別のプロセスだよ

RWiki の Object を外から使う

- dRuby がちゃんと実用になっててすごい!

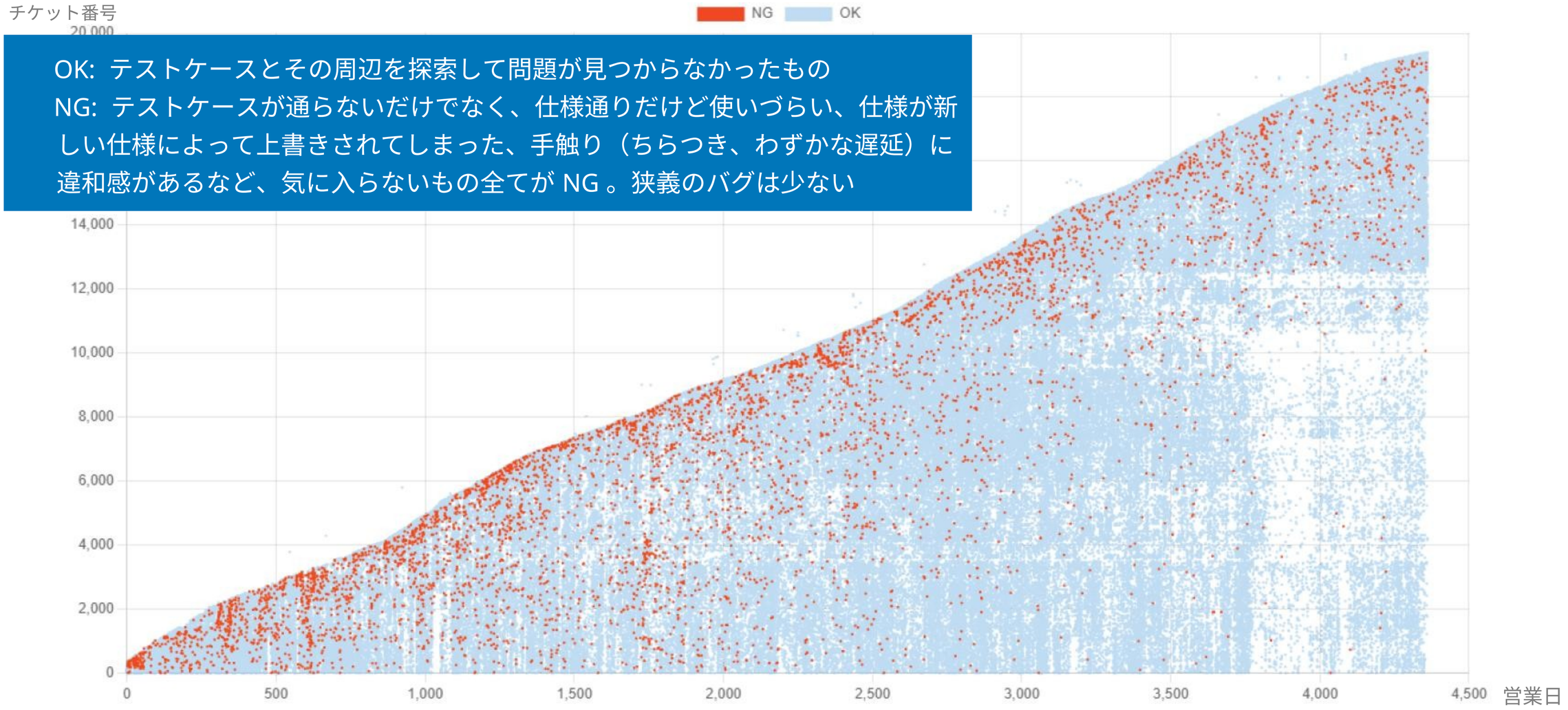
```
index = rwiki['test-v1']                    # タグのページを取得
list = index.rev_links.map {|name|          # 逆リンクで map
  rwiki[name].prop(:story)                  # prop を集める
}

# さっき説明した選択と並び替え処理

rwiki['sorted-test-v1'].src = rd            # テストスイートの
                                            # ページを作成
```

青いところが RWiki の Object への参照、赤いところが RMI

さて検証の時間です

- 自慢はここまで
- 本日のおすすめテストはうまくいってる?
- 20 年分のテストの記録で確かめてみましょう

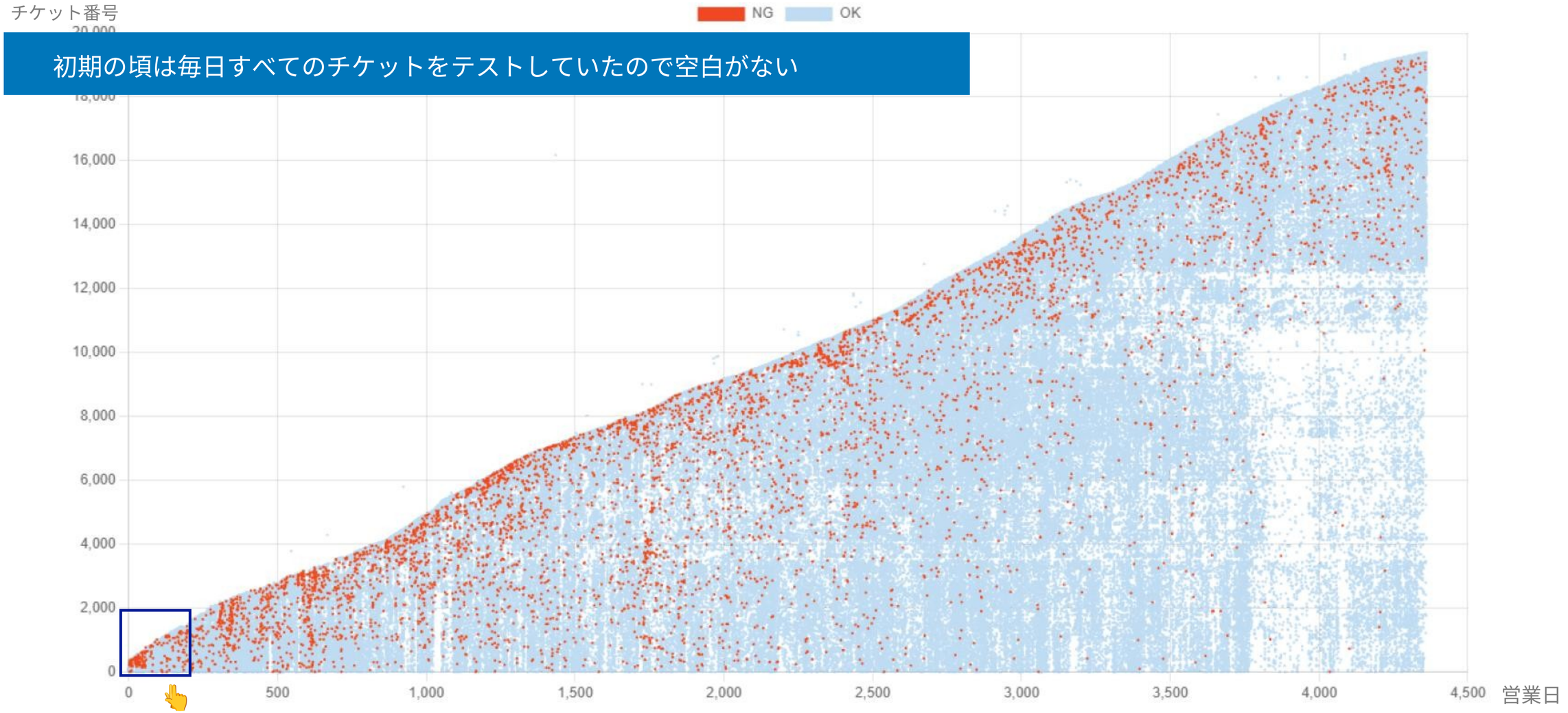
テスト実施の記録（ 20 年分）



OK: テストケースとその周辺を探索して問題が見つからなかったもの
NG: テストケースが通らないだけでなく、仕様通りだけど使いづらい、仕様が新しい仕様によって上書きされてしまった、手触り（ちらつき、わずかな遅延）に違和感があるなど、気に入らないもの全てが NG 。狭義のバグは少ない

NG の点はよく見えるように強調しています

テスト実施の記録（ 20 年分）

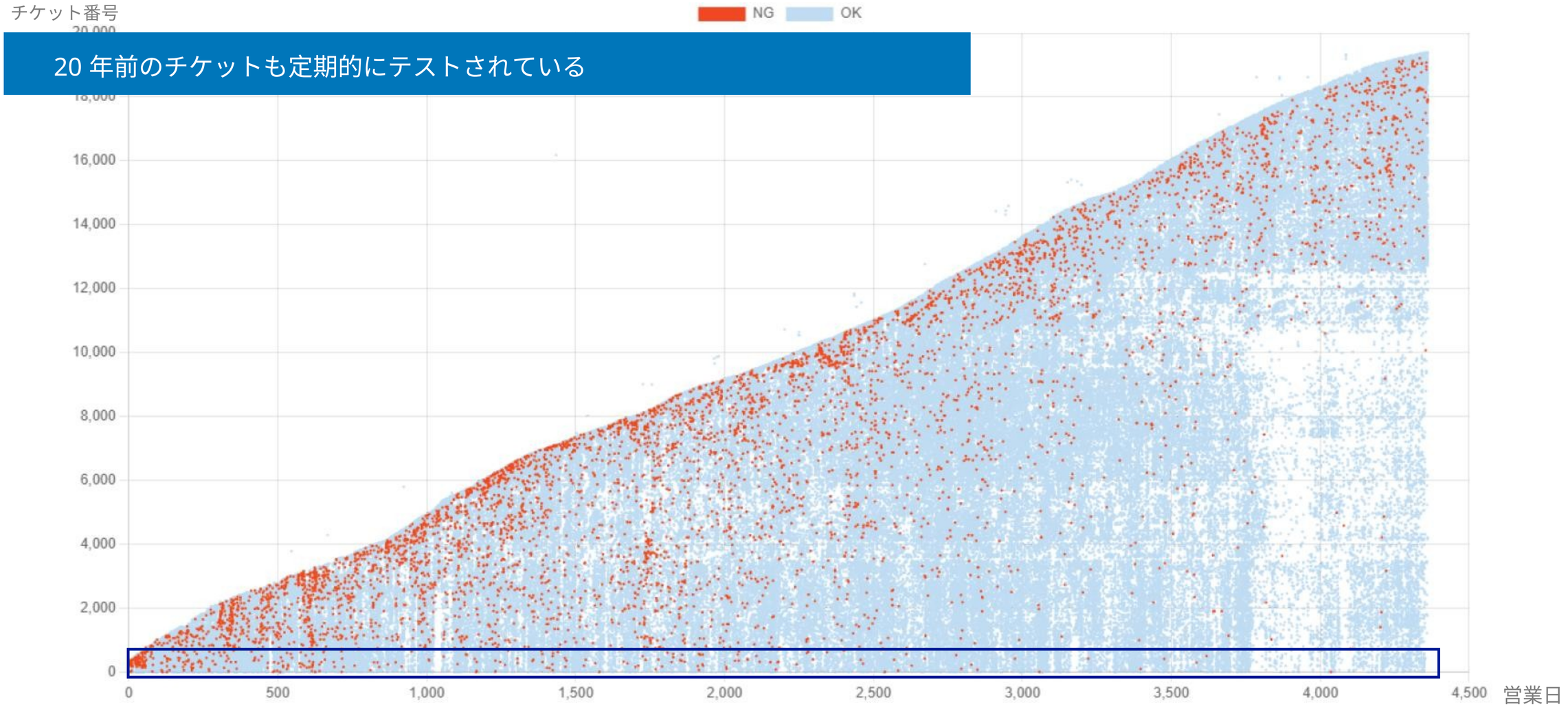


この辺りから「本日のおすすめテスト」を運用

NG の点はよく見えるように強調しています

テスト実施の記録（ 20 年分）

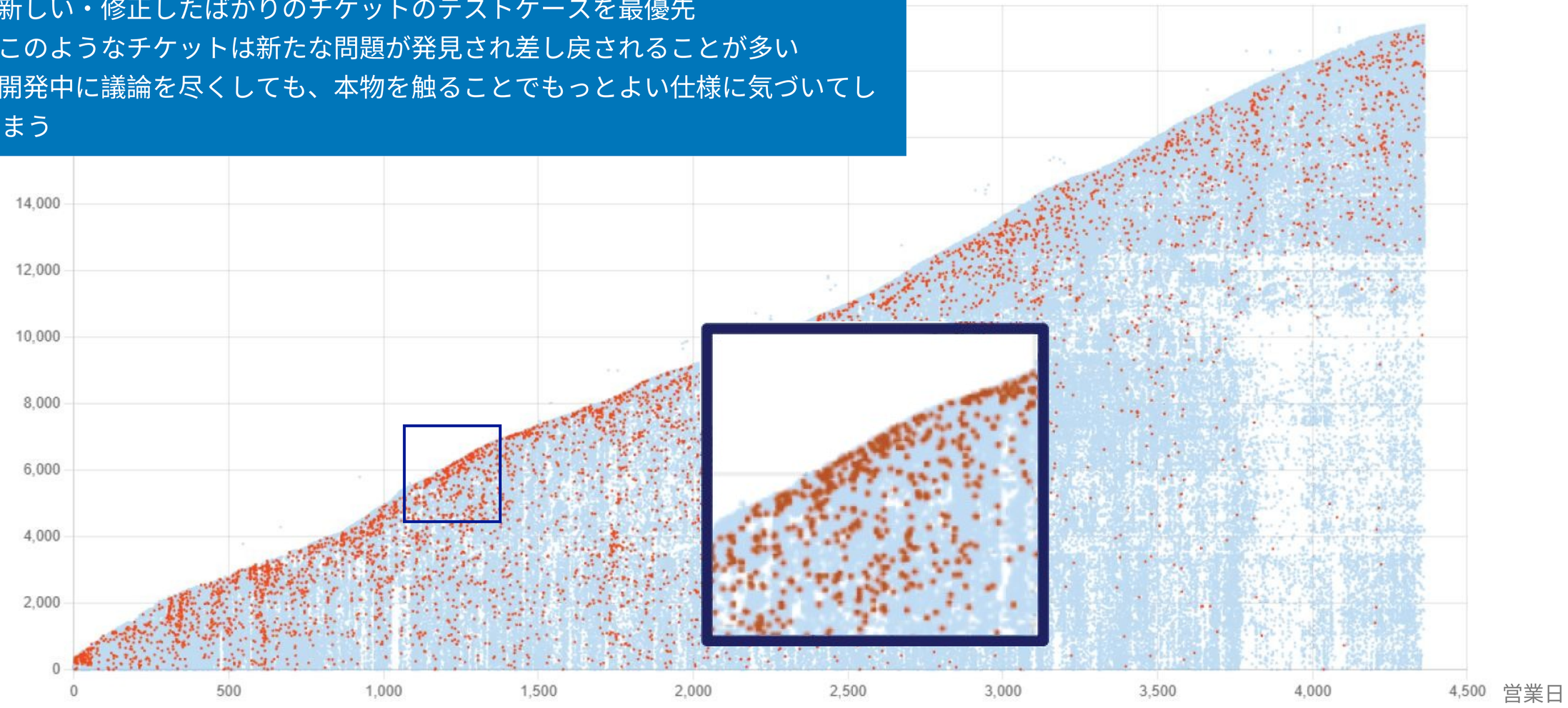
20 年前のチケットも定期的にテストされている



NG の点はよく見えるように強調しています

テスト実施の記録（ 20 年分）

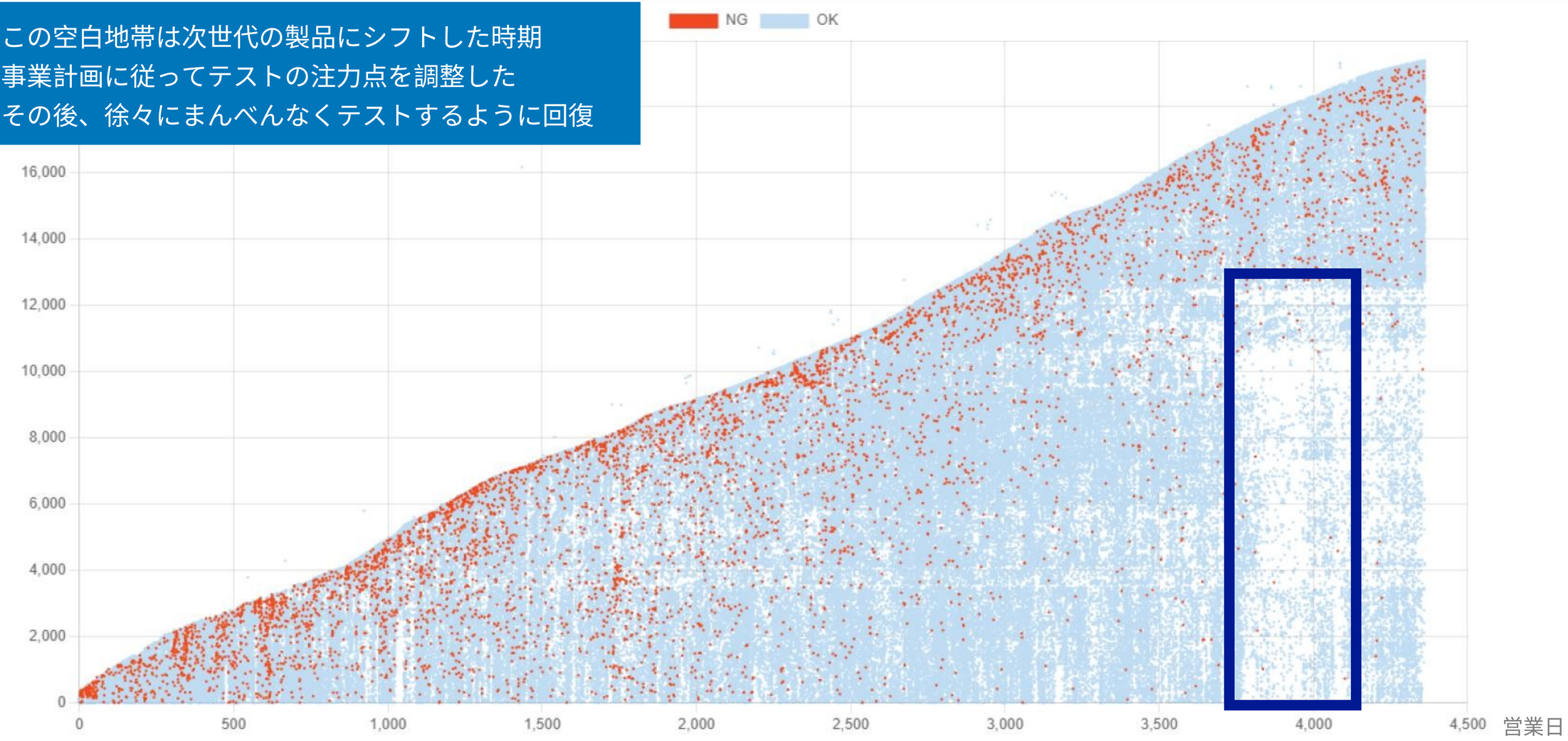
新しい・修正したばかりのチケットのテストケースを最優先
このようなチケットは新たな問題が発見され差し戻されることが多い
開発中に議論を尽くしても、本物を触ることでもっとよい仕様に気づいてしまう



NG の点はよく見えるように強調しています

テスト実施の記録（ 20 年分）

この空白地帯は次世代の製品にシフトした時期
事業計画に従ってテストの注力点を調整した
その後、徐々にまんべんなくテストするように回復



NG の点はよく見えるように強調しています

うまくできてるの?

- ✦ うまくできてた! 👏

本日のおすすめテストの作り方

ー Episode MATSUE ー

Ninja testing ZOZ5 Anniversary

忍者式テストを支える「本日のおすすめテストの作り方」を自慢しました🥷　コールドスリープ前の最後の講演でした