探索的テストってなんですか？

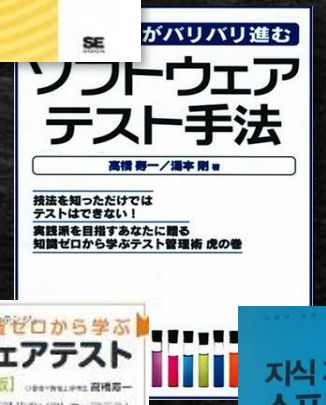
アジャイル時代の効率的なテストを考える

高橋寿一　情報工学博士

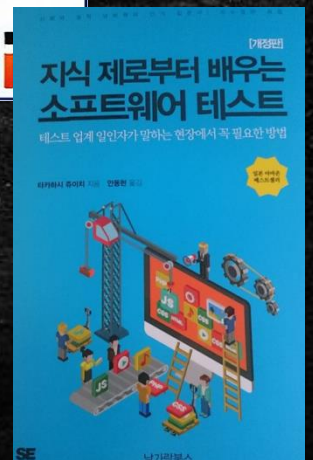
誰？こいつ？

- **知識ゼロから学ぶプロジェクト管理**
  著者：　勝呂暖生
  出版社：　翔泳社

- **現場の仕事がバリバリ進む ソフトウェアテスト手法**
  著者：　高橋寿一、湯本剛
  出版社：　技術評論社



- **知識ゼロから学ぶ ソフトウェアテスト（韓国語版あり）**
  著者：　高橋寿一
  出版社：　翔泳社



経歴

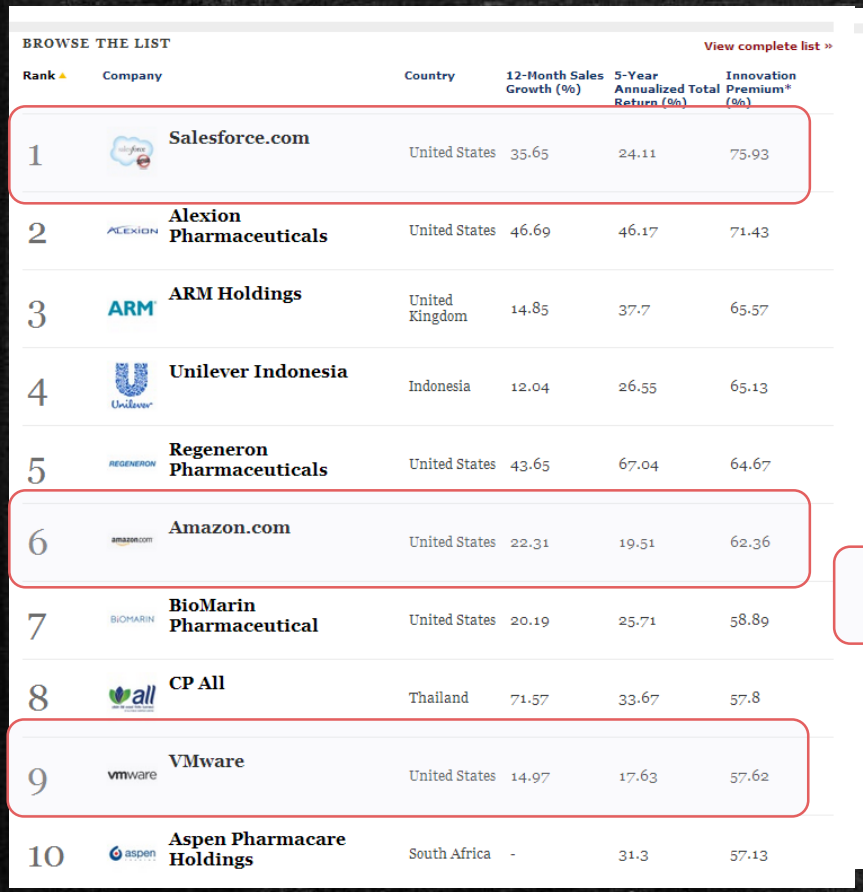
- **学歴**
  - フロリダ工科大学大学院でCem KanerとJames Whittakerに学ぶ
  - 広島市立大学でソフトウェアテストで博士号取得

- **職歴**
  - 米国Microsoft
  - 独SAP
  - 日本の電機会社

ゴール

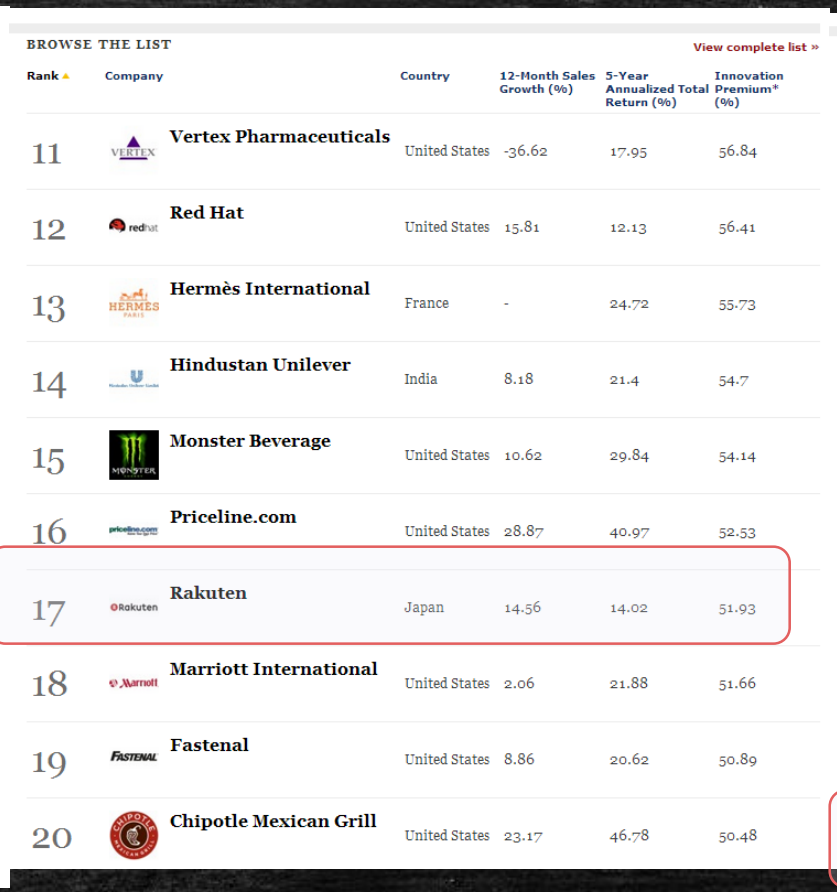
- 全部はテスト出来ない！という認識。
- でもランダムテストでは不安（っていうかマネージャが怒る）、まあまあな探索的テストどうですか？

The World's Most Innovative Companies in 2014



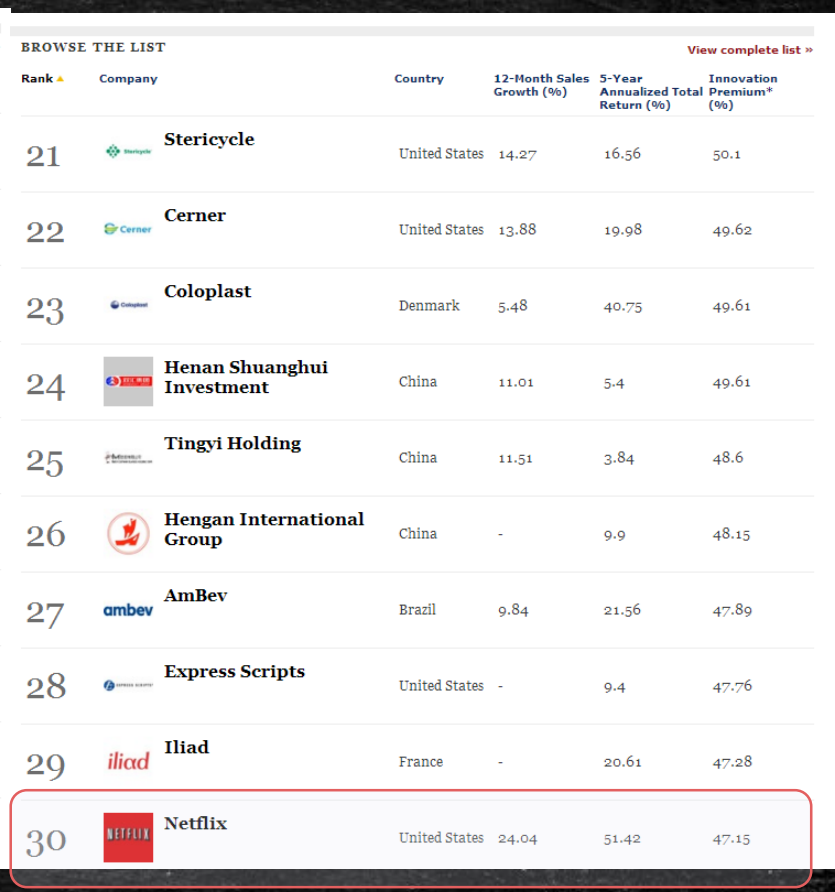
BROWSE THE LIST — View complete list »

Rank	Company	Country	12-Month Sales Growth (%)	5-Year Annualized Total Return (%)	Innovation Premium* (%)
1	Salesforce.com	United States	35.65	24.11	75.93
2	Alexion Pharmaceuticals	United States	46.69	46.17	71.43
3	ARM Holdings	United Kingdom	14.85	37.7	65.57
4	Unilever Indonesia	Indonesia	12.04	26.55	65.13
5	Regeneron Pharmaceuticals	United States	43.65	67.04	64.67
6	Amazon.com	United States	22.31	19.51	62.36
7	BioMarin Pharmaceutical	United States	20.19	25.71	58.89
8	CP All	Thailand	71.57	33.67	57.8
9	VMware	United States	14.97	17.63	57.62
10	Aspen Pharmacare Holdings	South Africa	-	31.3	57.13



BROWSE THE LIST — View complete list »

Rank	Company	Country	12-Month Sales Growth (%)	5-Year Annualized Total Return (%)	Innovation Premium* (%)
11	Vertex Pharmaceuticals	United States	-36.62	17.95	56.84
12	Red Hat	United States	15.81	12.13	56.41
13	Hermès International	France	-	24.72	55.73
14	Hindustan Unilever	India	8.18	21.4	54.7
15	Monster Beverage	United States	10.62	29.84	54.14
16	Priceline.com	United States	28.87	40.97	52.53
17	Rakuten	Japan	14.56	14.02	51.93
18	Marriott International	United States	2.06	21.88	51.66
19	Fastenal	United States	8.86	20.62	50.89
20	Chipotle Mexican Grill	United States	23.17	46.78	50.48



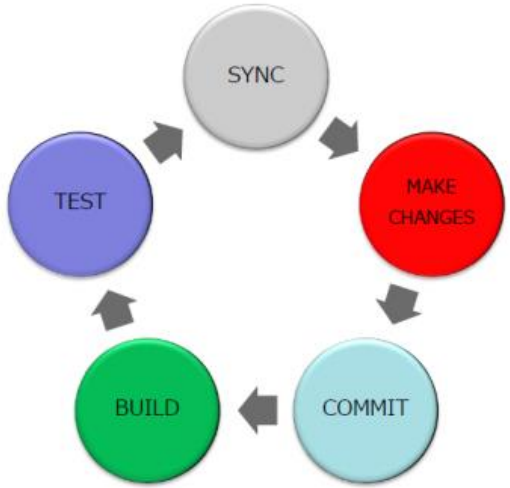
BROWSE THE LIST — View complete list »

Rank	Company	Country	12-Month Sales Growth (%)	5-Year Annualized Total Return (%)	Innovation Premium* (%)
21	Stericycle	United States	14.27	16.56	50.1
22	Cerner	United States	13.88	19.98	49.62
23	Coloplast	Denmark	5.48	40.75	49.61
24	Henan Shuanghui Investment	China	11.01	5.4	49.61
25	Tingyi Holding	China	11.51	3.84	48.6
26	Hengan International Group	China	-	9.9	48.15
27	AmBev	Brazil	9.84	21.56	47.89
28	Express Scripts	United States	-	9.4	47.76
29	Iliad	France	-	20.61	47.28
30	Netflix	United States	24.04	51.42	47.15

*The Innovation Premium is a measure of how much investors have bid up the stock price of a company above the value of its existing business based on expectations of future innovative results (new products, services and markets) . Members of the list must have $10 billion in market capitalization, spend at least 2.5% of revenue on R&D and have seven years of public data.

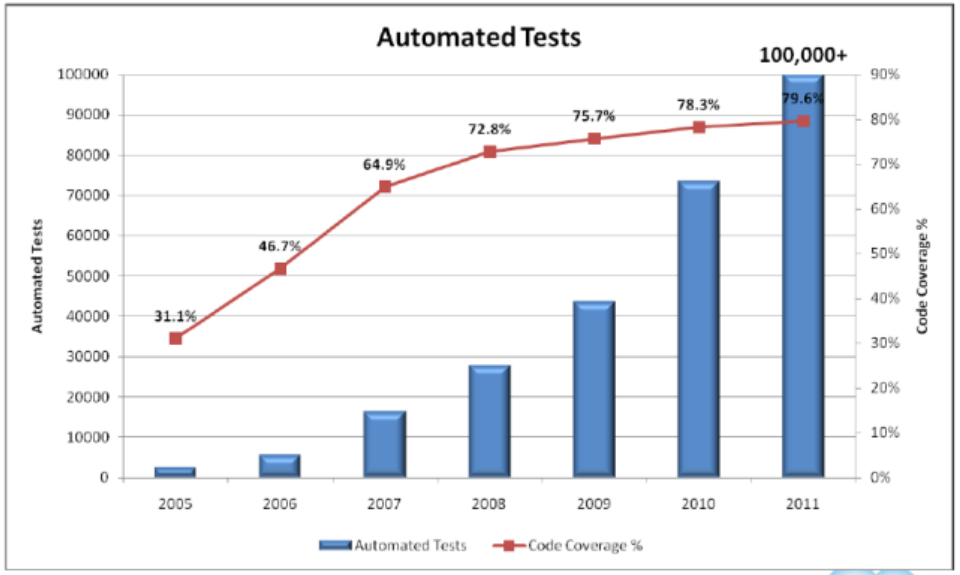
Salesforace

継続的インテグレーション



- XPの開発手法のひとつ
- 小規模な変更を頻繁にレポジトリにコミットを繰り返しながらソフトウェアを構築していく
- 開発ビルド作業の自動化
- 最低1日1回のコミット

自動化テストへの投資



Amazon

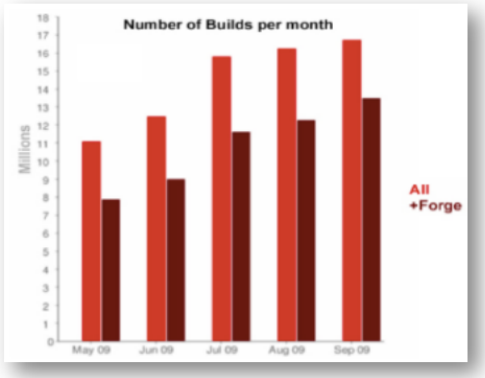
- MAINTAIN A CODE REPOSITORY
- AUTOMATE THE BUILD
- MAKE THE BUILD SELF TESTING
- EVERYONE COMMITS TO THE BASELINE EVERY DAY
- EVERY COMMIT (TO THE BASELINE) SHOULD BE BUILT

Google

Testing at Google

Test & Build System Fast Facts

- Recall, Google's emphasis on speed – 20+ code changes occur per minute
  - Code churn - **50%** of code changes every month
- 65k builds/day
- 20M builds/year



Source: http://googletesting.blogspot.com/2010/04/googles-innovation-factory-and-how.html

1日に65,000回のビルド

Testing at Google

Test & Build System Fast Facts

- 120k test suites in code base
- 7.5M test suites run/day
- 120M individual tests run/day
- Every affect test run after each code change
- Tests run on all major OS/browser pairs

1日に1.2億テストケース実行

コード変更毎にテストケースが走る

ゆっくり手動でテストとかしてませんか？

旧来のテストとは

- 網羅的 – 全機能ちゃんとチェックしたりする。さらにはその機能組み合わせたりする
- お金がかかる – 品質に関わる投資はどんなプロジェクトでも50%を超えている
- 時間がかかる　–　最初のリリースから数か月のテストしてませんか

銀行システム・家電・交通システム・なんか高価なソフトウェア

クラウド・アジャイルの新しいテストとは

- 早い
- 安い
- そこそこの品質

Androidアプリ・Webアプリ・SNS
↓
新しいテストの枠組みが必要

提案する新しいテストのスタイル

- 探索的テスト
  - Kent Beck(XPを作った人)は基本的に**システムテスト**や受け入れテストに関して定義していない。
  - アジャイル・XPで早く開発できるようになったのに、重いテストは誰も受け入れない。それなら**軽い**探索的テストは最適。**Daily release**にも耐える。

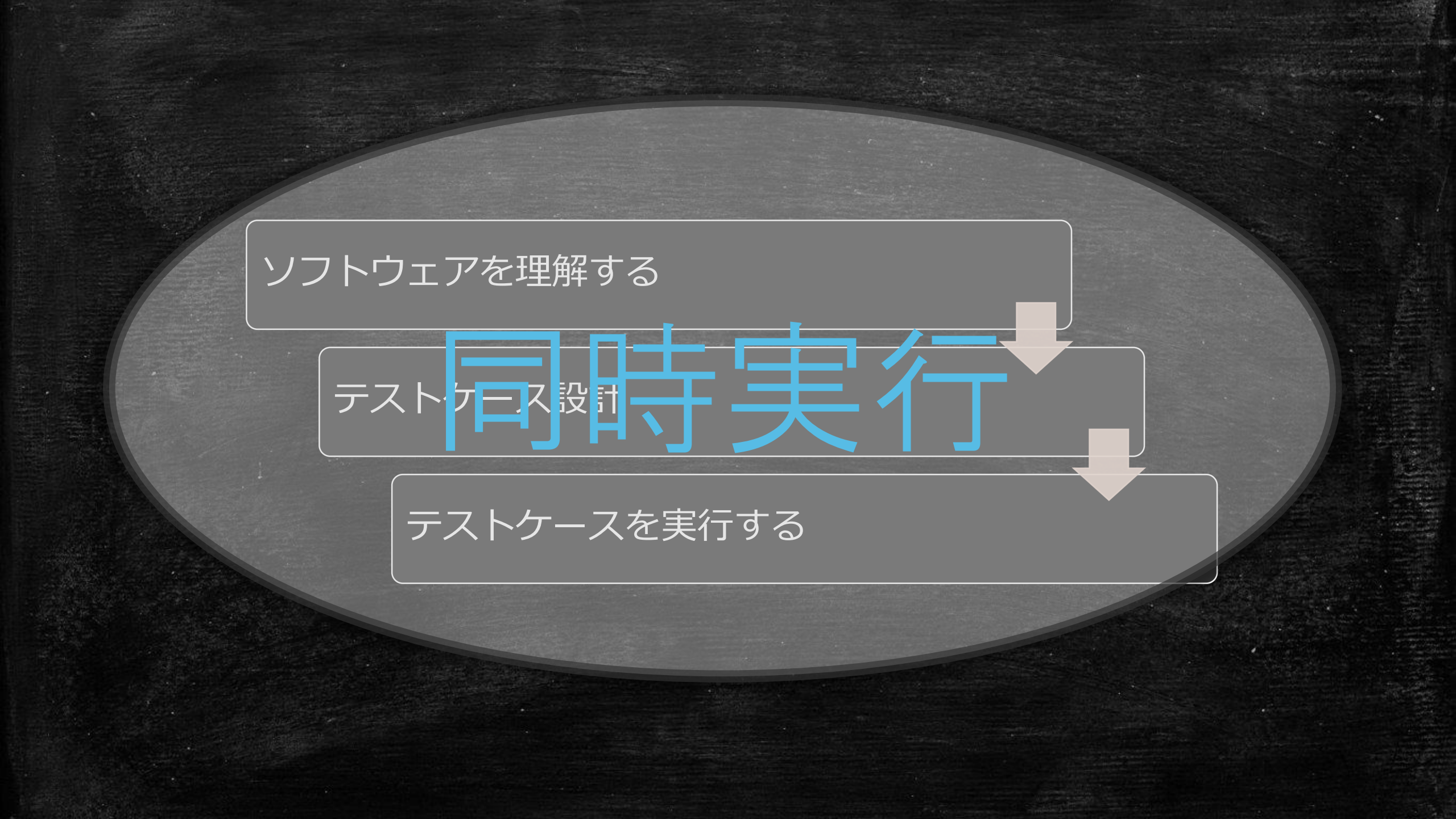
探索的テストの歴史

Exploratory testing has always been performed by **skilled testers**. In the early 1990s, **ad hoc** was too often synonymous with sloppy and careless work. As a result, a group of test methodologists (now calling themselves the Context-Driven School) began using the term "exploratory" seeking to emphasize the dominant thought process involved in unscripted testing, and to begin to develop the practice into a teachable discipline. This new terminology was first published by **Cem Kaner** in his book Testing Computer Software[1] and expanded upon in Lessons Learned in Software Testing.[4] Exploratory testing can be as disciplined as any other intellectual activity.

探索的テストとは？

- ソフトウェアテストの一つの**スタイル**
- 個人に自由意志を持たせるとともに**責任をより明確にする**
- **一個人**のテスト活動である
- **継続的**にテスト活動を洗練させる
- 探索的テストは以下の活動を行う
  - テスト関連の**学習**
  - テスト**設計**
  - テスト**実行**
  - テスト結果を**報告**
- **成熟**したテスト活動
- 上記の活動をプロジェクト期間中**並行**して行う

うーん、よくわからなかったりする．．．


ソフトウェアを理解する
テストケース設計
テストケースを実行する
同時実行

そして

一つ一つの細かいテストケースは書かない!

そんな暇があったら実際にテストしようぜ！

旧来のテストケースベースのテストをネガティブに言うと

- テストケースドキュメントなんて**時間かかる**しメンドウだから書かない！そんな時間があったらテストする時間にあてたほうがいい（テストケースの作成）

- だいたい製品テストなんてどこにバグが出るかわからないし、**触ってみていろいろテストすると**わかるじゃん！（テストケースの実行）

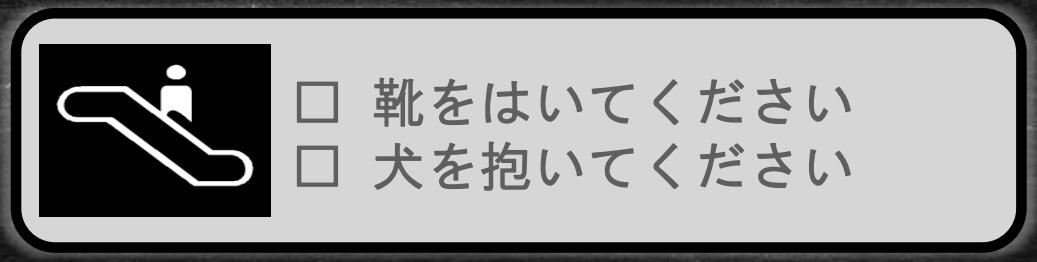
- ばかの一つ覚えのように繰り返しテストやったって意味ねーよ。開発者の修正とか、製品の癖とか考えながら**スキルのあるテスト担当者**がテストするからたくさん回帰テストのバグが見つかるんだよ（回帰テスト）

テストケースベースのテスト

- テスト設計・ケース作成を早い段階で行う
- テストリリースが出てきて、その早い段階で作ったケースを実行する
- 同じテストケースをたくさん実行する

テストケースベースのテストの限界

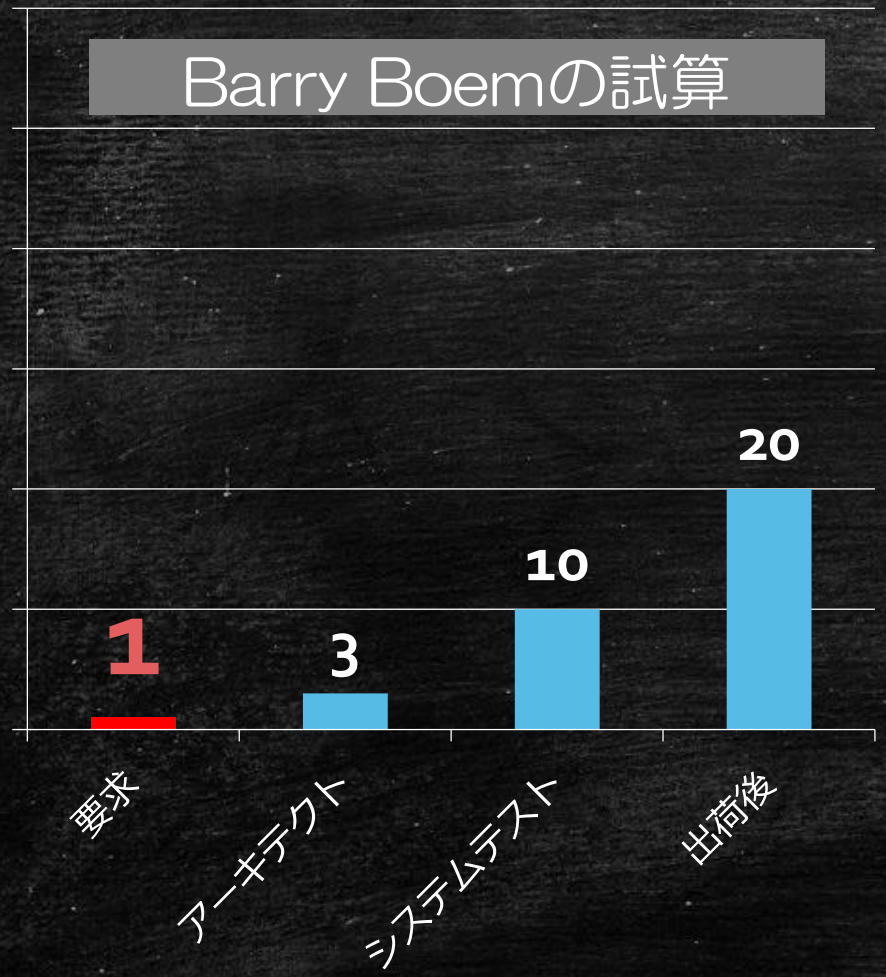
開発初期のあいまいな要求仕様や設計仕様に
致命的に対応できない

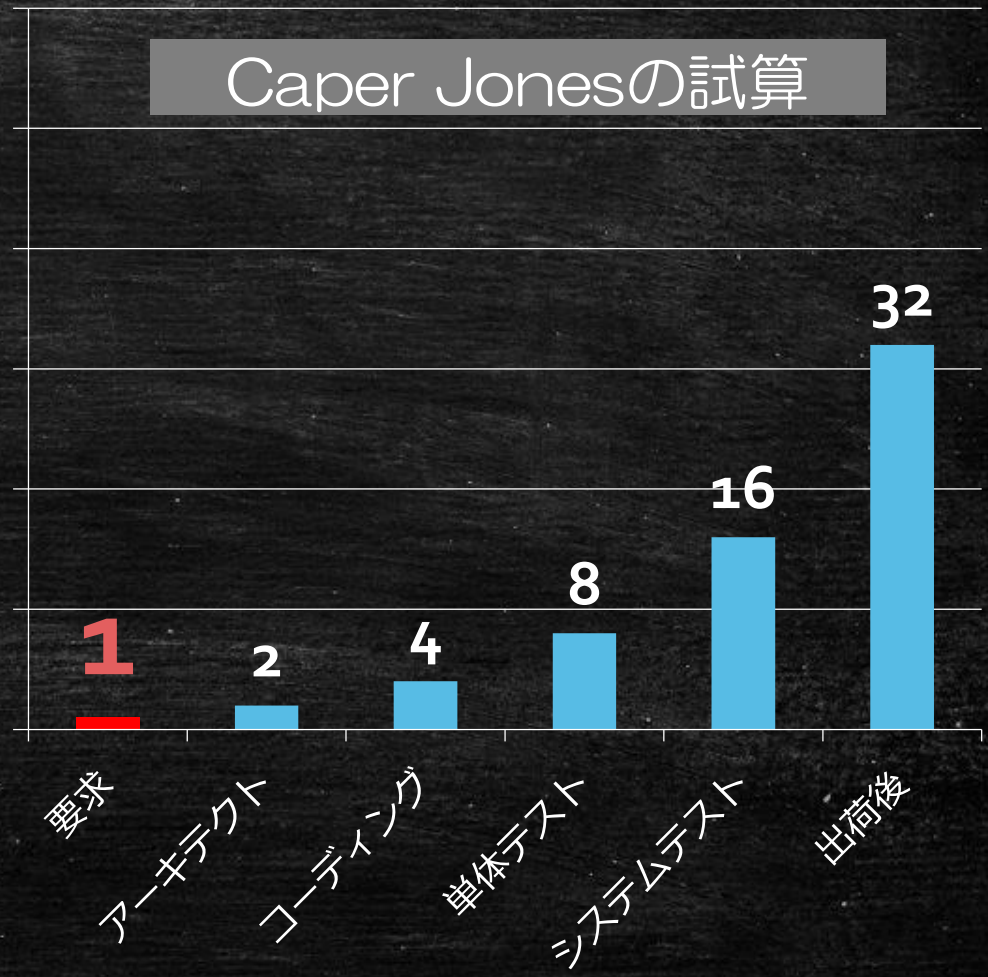


$$\forall x \cdot (\mathrm{OnEscalator}(x) \rightarrow \exists y \cdot (\mathrm{PairOfShoes}(y) \wedge \mathrm{IsWearing}(x, y)))$$

要求仕様は間違いというより、
記述しきれてないものがバグとなる

不具合の原因発生工程と発見工程のずれによる損失

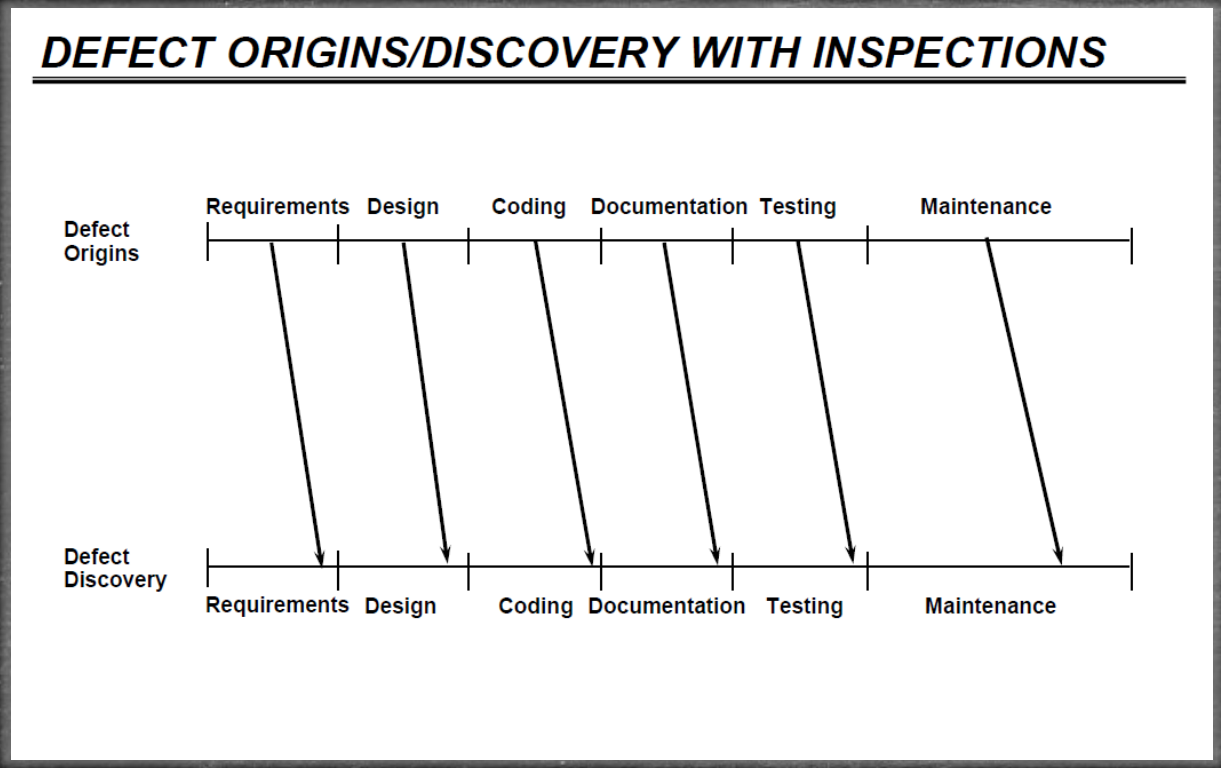


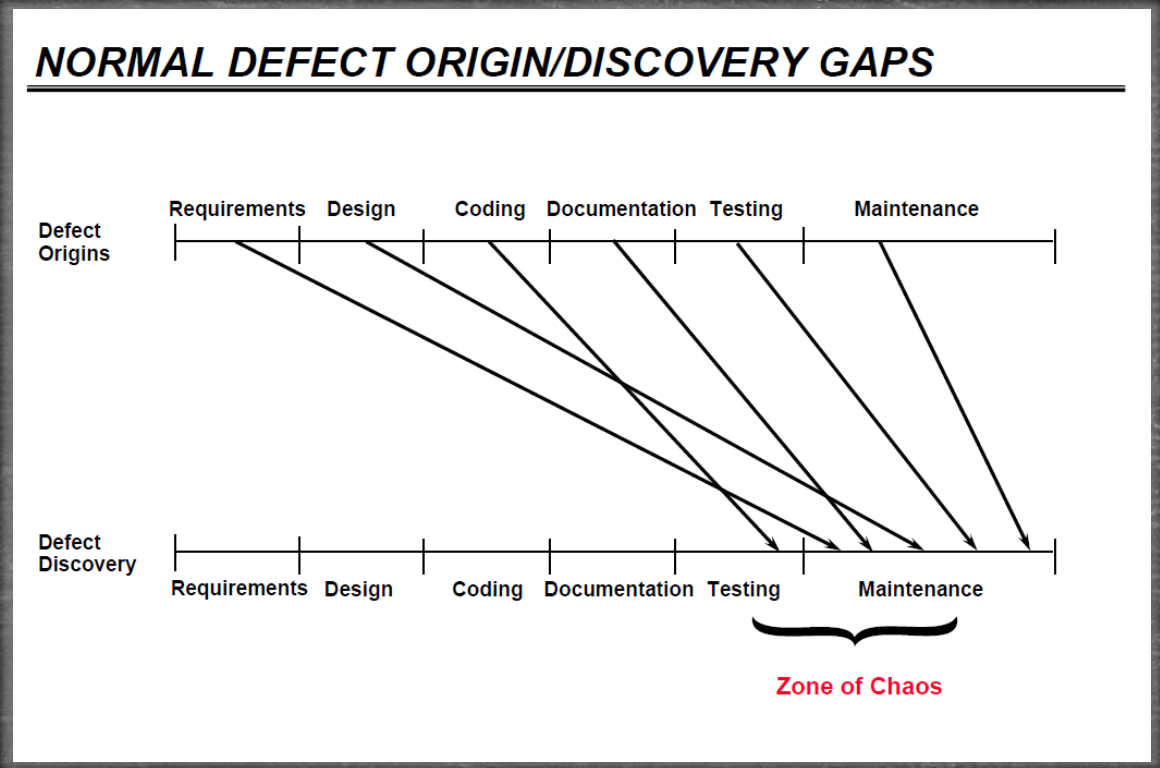


"Barry Boem and Victor Basil, "Software Defect Reduction Top 10 List", *IEEE Computer*, January 2001.

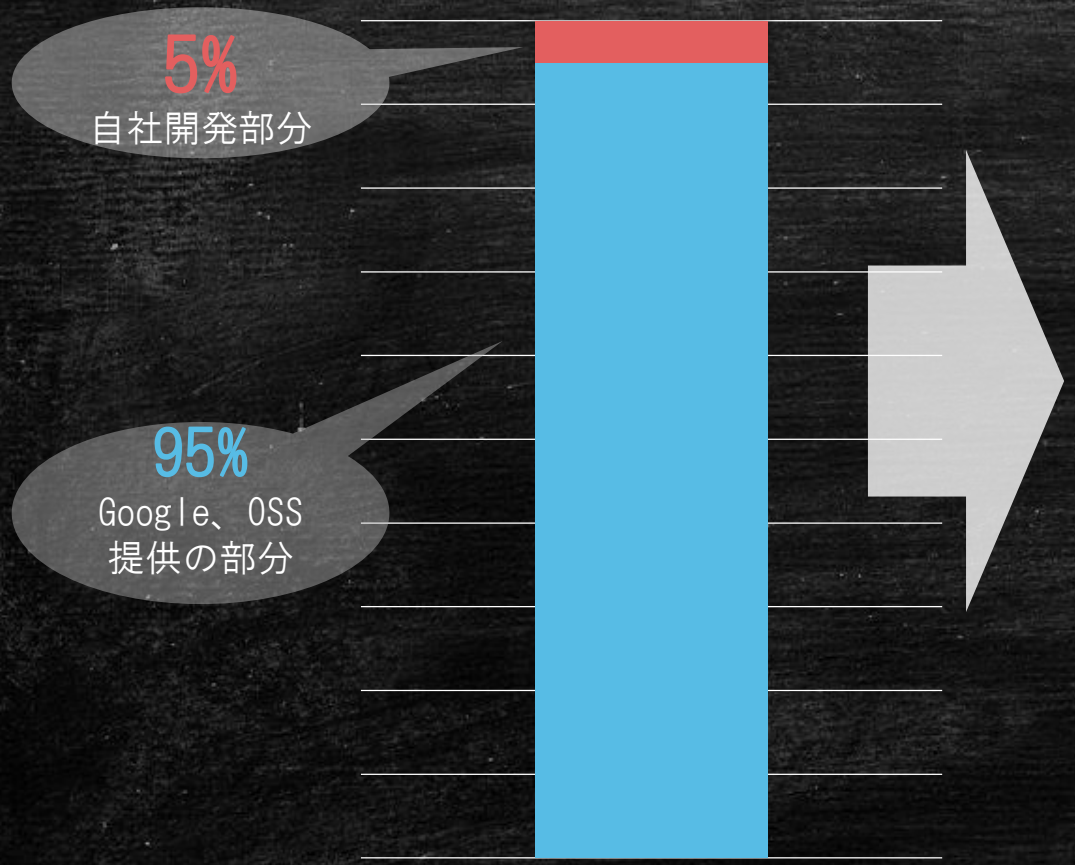
Caper Jones, Japan Software Testing Symposium, 2008

要求仕様のバグをテストで見つけてませんか？
更に進んで、要求仕様のバグを見つけずに出荷！





要求仕様のバグをテストで見つけてませんか？
更に進んで、要求仕様のバグを見つけずに出荷！
Androidベース製品の開発



- ５%の開発でさえＯＳＳ・インナーソース利用
- 開発者のやることは実は少ない（開発スピードが上がってしまう）
- テストとはほとんどが派生（すでにやりきれないテストケースの歴史）や、他の人や会社が作ったソフトウェアの組み合わせのためのテスト。旧来のテストでテストできますか？

探索的テストのやり方

V1.0
Designed by James Bach, Testing Consultant
Satisfice, Inc.

08/26/99 3:53 PM
james@satisfice.com
http://www.satisfice.com

General Functionality and Stability Test Procedure

for Certified for Microsoft Windows Logo

Desktop Applications Edition

This document describes the procedure for testing the functionality and stability of a software application (hereafter referred to as "the product") for the purpose of certifying it for Windows 2000. This procedure is one part of the Windows 2000 compatibility certification process described in *Certified for Microsoft Windows Test Plan*.

This procedure employs an exploratory approach to testing, which means that the test cases are not defined in advance, but rather are defined and executed on the fly, while you learn about the product. We chose the exploratory approach because it is the best way to test a product quickly when starting from scratch.

This document consists of five sections:

- Introduction to Exploratory Testing
- Working with Functions
- Testing Functionality and Stability
- Reading and Using this Procedure
- Test Procedure

The first three parts explain the background and concepts involved in the test procedure. The fourth section gives advice about getting up to speed with the procedure. The fifth section contains the procedure itself.

This document is designed to be duplex printed (two sides on each page). For that reason, pages 2, 10, and 12 are intentionally blank.

探索的テストのPass/Fail基準

定義	Pass 基準	Fail 基準
機能性	各々の主機能がテストされ、ユーザが望む操作ができ、かつそのアウトプットが正しいものである。	1 つ以上の主機能の実装の不備があり、ユーザがその目的を達成できない
	不正な操作をしたとしても、その後の操作に支障をきたさずソフトウェアが正常に動作する。	不正なオペレーションをした後、ソフトウェアが正常に動作しない。
安定性	対象ソフトウェアは Windows OS を不安定にさせない。	Windows OS が機能不全に陥ることがある
	対象ソフトウェアはハングアップやクラッシュやデータの損失をしない。	ハングアップやクラッシュやデータの損失が起こる
	主機能が操作不可能や、オペレーションの阻害が起こらない	操作不可能や、オペレーションの阻害が起こる

テストプロセス

- 製品の目的を明確にする
  (Identify the purpose of the product)
- 機能を明確にする
  (Identify functions)
- 弱いエリアを叩く
  (Identify areas of potential instability.)
- テスト実行及びバグを記録
  (Test each function and record problems.)
- 上記を継続的に実行する
  (Design and record a consistency verification test.)

テストプロセス

- 製品の目的を明確にする
  (Identify the purpose of the product)
- **機能を明確にする**
  (Identify functions)
- 弱いエリアを叩く
  Identify areas of potential instability.
- テスト実行及びバグを記録
  (Test each function and record problems.)
- 上記を継続的に実行する
  (Design and record a consistency verification test.)

機能を明確にする (Identify functions)

- 製品をウオークスルー(Walk through)する。製品がどういった機能を有しているか

- 重要機能のリストアップ
  - オンラインヘルプ
  - ツールバー
  - 右クリック、ダブルクリック
  - メニュー

以上を通して基本機能をテストするとも言える

テストプロセス

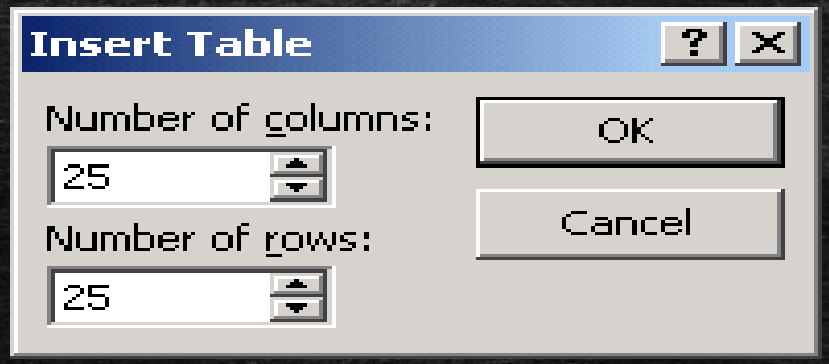
- 製品の目的を明確にする
  (Identify the purpose of the product)
- 機能を明確にする
  (Identify functions)
- **弱いエリアを叩く**
  (Identify areas of potential instability)
- テスト実行及びバグを記録
  (Test each function and record problems.)
- 上記を継続的に実行する
  (Design and record a consistency verification test.)

弱いエリアを叩く
(Identify areas of potential instability)

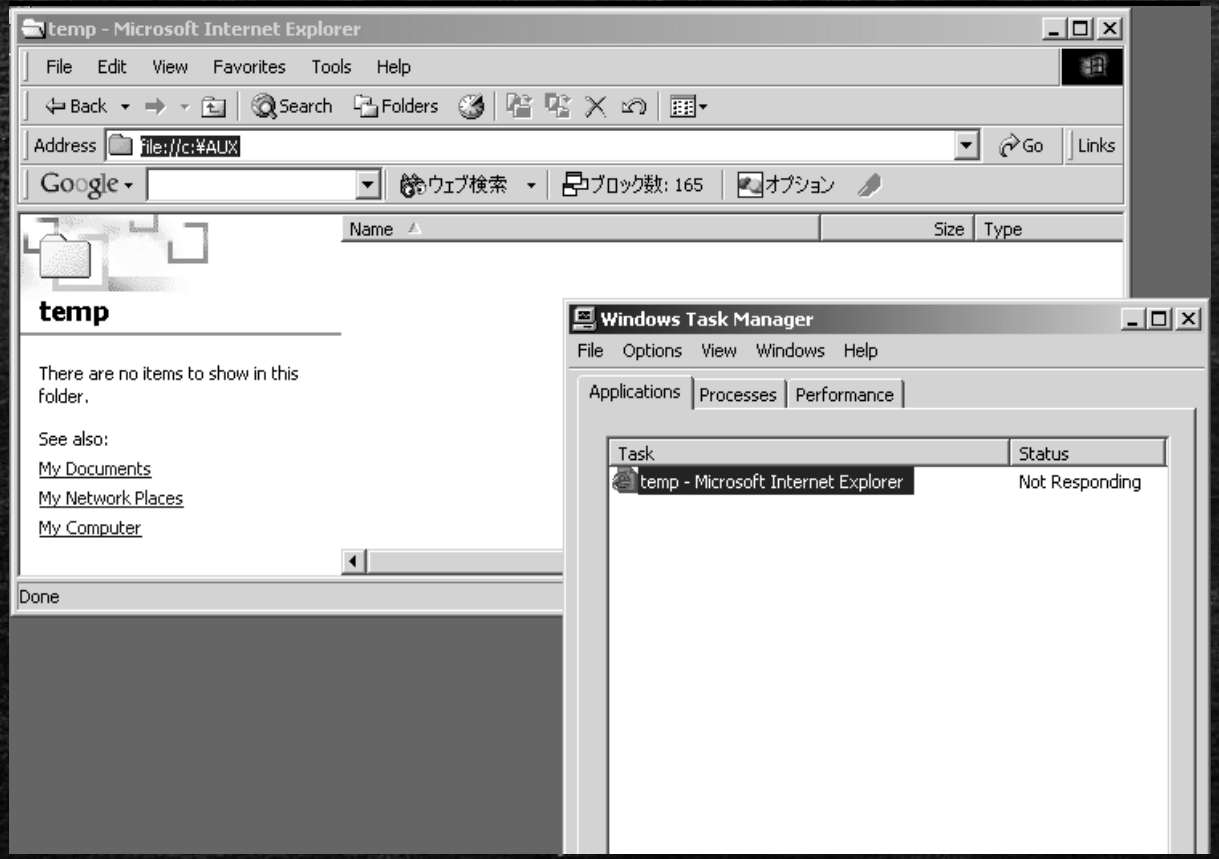
- データの交換が発生する機能（オブジェクトの埋め込み、ファイル変換）
- 他のソフトウェアとイベントを共有機能 (目覚まし機能、メール送信)
- あまり使わないと思われる複雑に組み合わさったデータ入力をハンドルする機能
- ファイルをネットワーク越しにオープンさせる機能
- エラーや例外処理からの復帰機能
- オペレーティングシステムとデータを交換させる機能（初期値）
- オペレーティングシステムにサイズが依存する機能（メモリ）

弱いエリアを叩く考え方

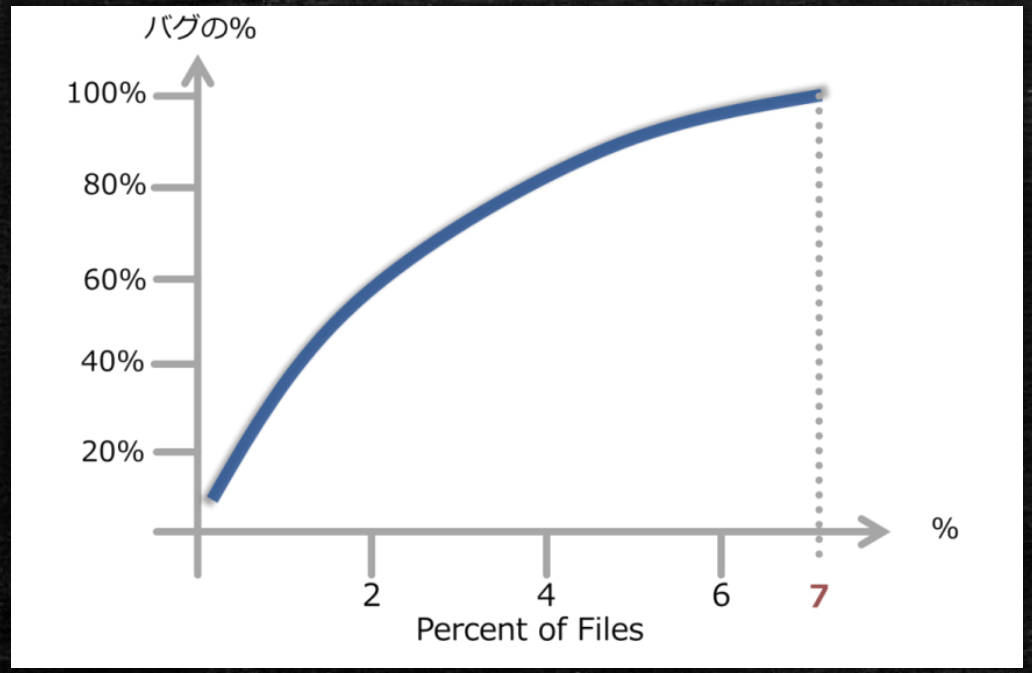
James Whittakerのアイディア

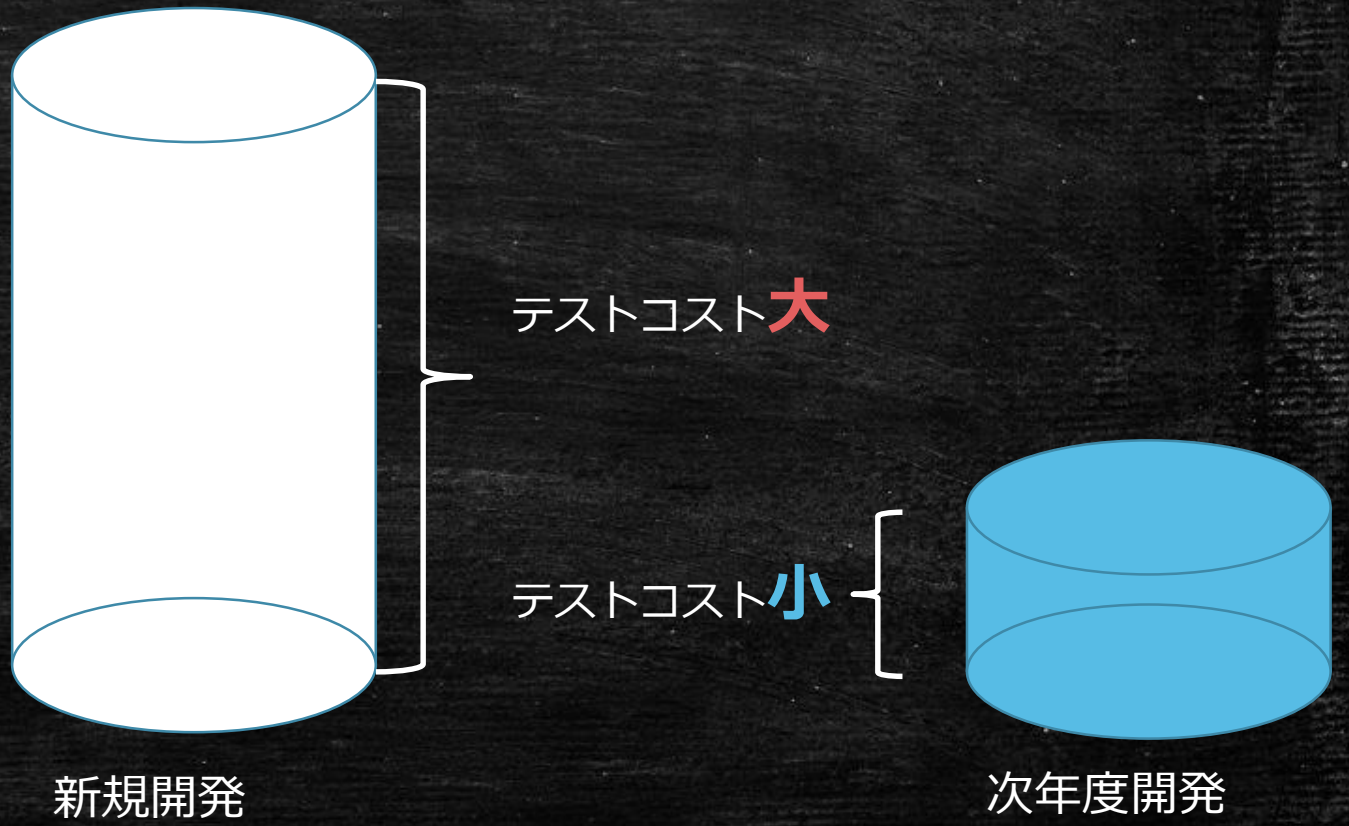

Insert Table
Number of columns:
25
Number of rows:
25
OK
Cancel

File://c:¥aux


temp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Folders
Address file://c:¥AUX
Go Links
Google ウェブ検索 ブロック数: 165 オプション
Name Size Type
temp
There are no items to show in this folder.
See also:
My Documents
My Network Places
My Computer
Done
Windows Task Manager
File Options View Windows Help
Applications Processes Performance
Task Status
temp - Microsoft Internet Explorer Not Responding

理論的には継続開発では 7％(hot Spot)のファイルから すべてのバグが出る





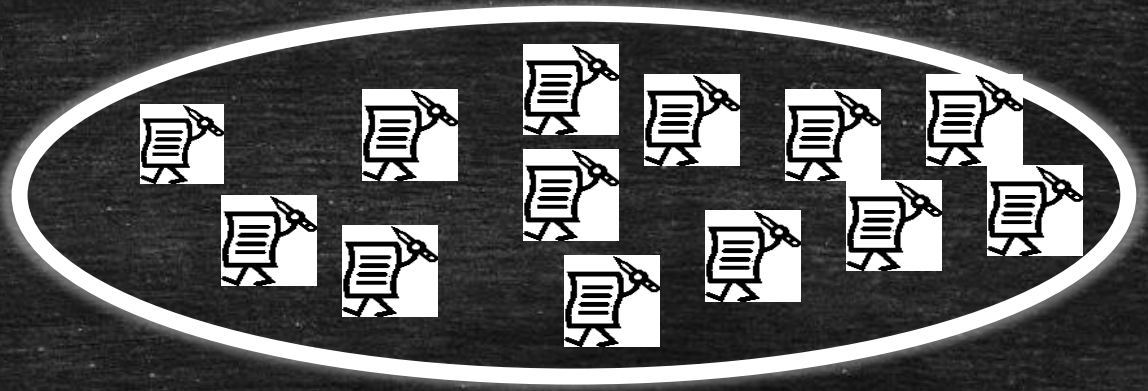
Googleの使っているHot Spotアルゴリズム

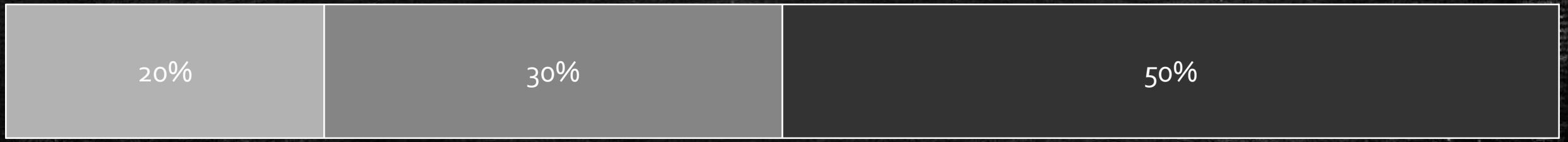
$$\text{Score} = \sum_{i=0}^{n} \frac{1}{1 + e^{-12ti + 12}}$$

８０％のバグは全体の２０％のモジュール
から発見され、
５０％のモジュールにはバグは含まれていない

Barry Boehm and Victor R. Basili

テストケース郡

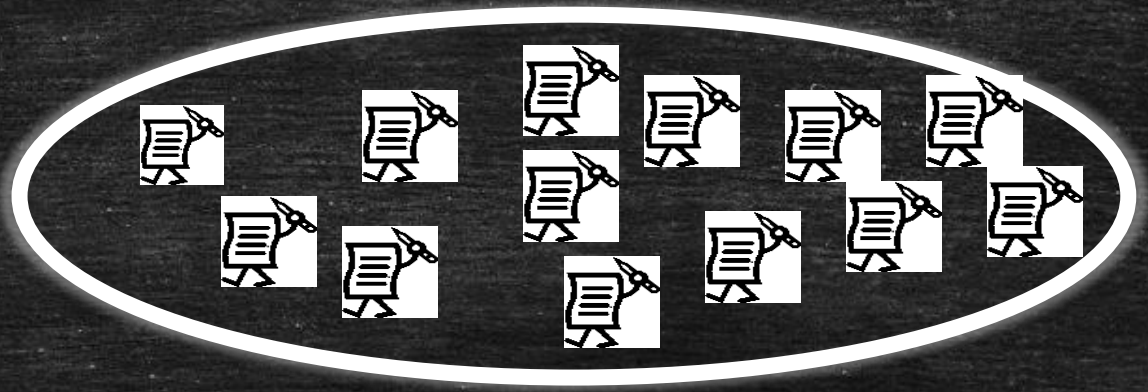


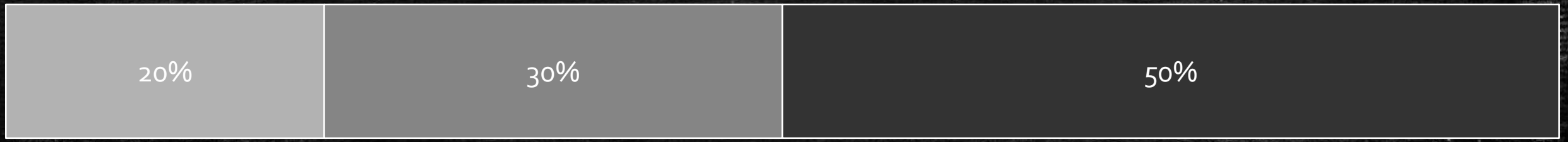


よくないテストケースの配置

テストケースが無駄に増えるし
無駄なテスト実行時間が増える

テストケース郡





よいテストケースの配置

テストケース数が適切

テスト実行時間が適切

ランダムテストと探索的テストの違い
弱い（バグの出る）エリアを効率的に叩ける

- ランダムテストの問題点
  - ランダムなので、単位時間あたりのバグの発見数が少ない。そういう意味ではいままでのテストも同様に単位時間あたりのバグ発見数が少ない。
- 探索的テストの利点
  - 基本的には弱い場所（Hot Sport）をたたいているので、単位時間のバグ発見率が良い。

探索的テスト＋Hot Spotの単体テストは
最高の組み合わせ！

5 年後はコード網羅を見ながらの探索的テスト？

探索的テストはまったく文章を書かないわけではない！各プロセスについてタスクシートを書く！

タスクの記述(Task Description)	**なにをやろうとしている**のかを記述
探索アプローチガイド(Heuristics)	このガイドラインなり、アイディアが探索的テストをどのように進めていくかを記述する。やるべきタスクを羅列するのではなく、どういう考え方でテストするかを記述する。例えば動詞を全部羅列するとかもアイディアです。
結果(Results)	**どういう品質**のものがリリース可能か。
You can say you’re done when	**どのように終わらせる**
FAQ	

探索的テストの利点・欠点

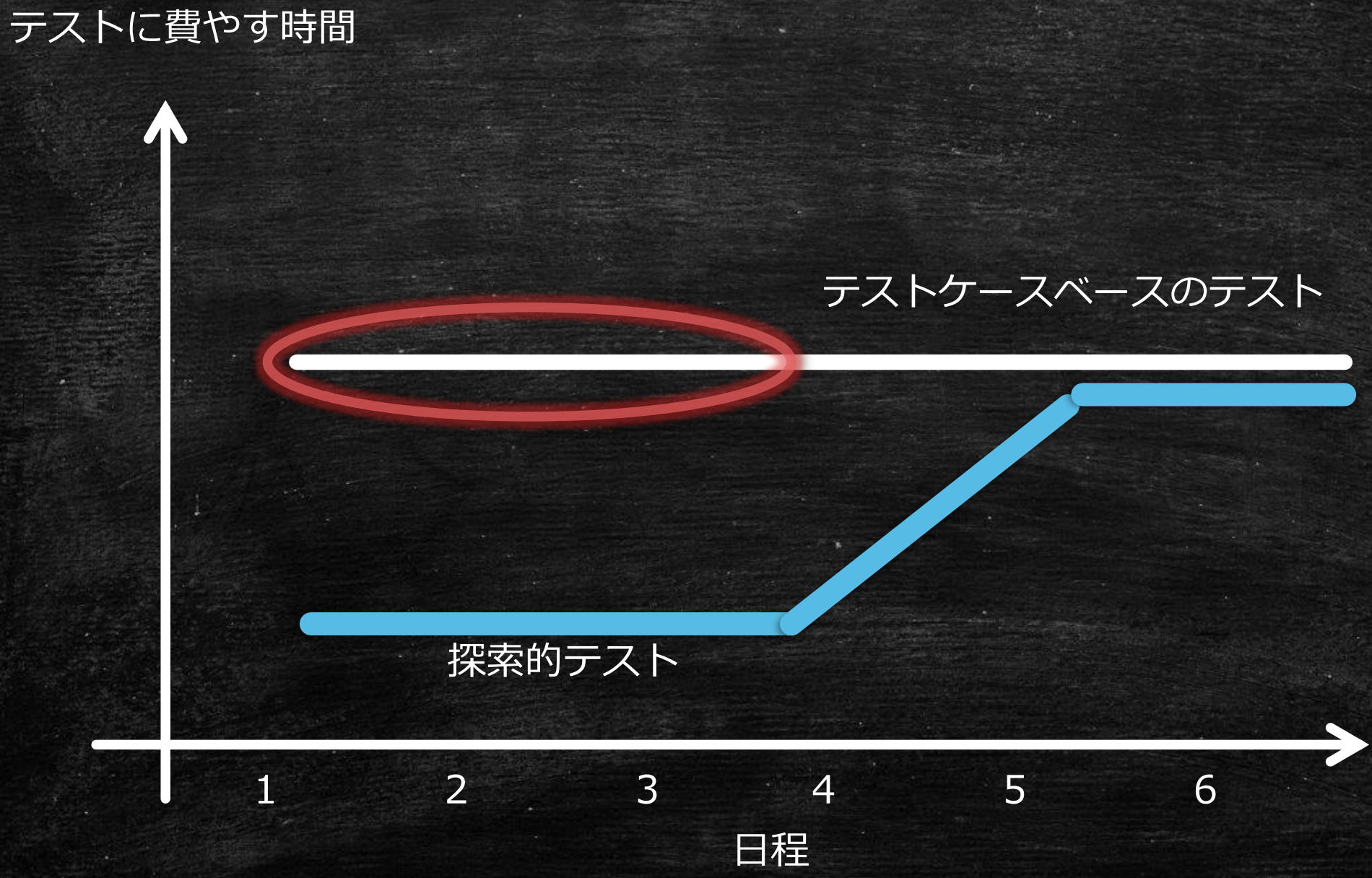
探索的テストの利点・欠点

- 利点
  - 抜群に投資効果が高い
  - 早くバグを見つけられる
  - たくさんのバグを見つけられる

- 欠点：**理論的にはない**。悪くてもテストケースベースのテストと同等

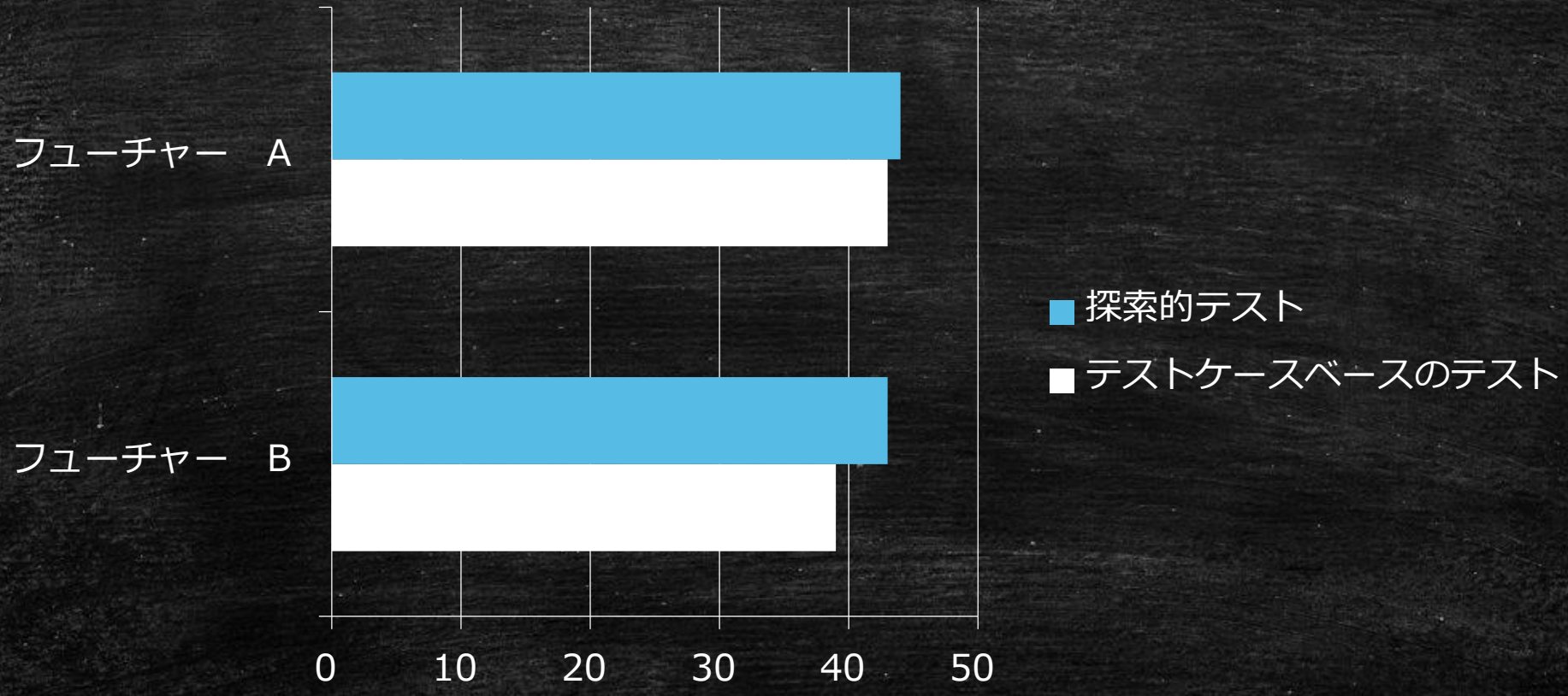
探索的テストの利点・欠点

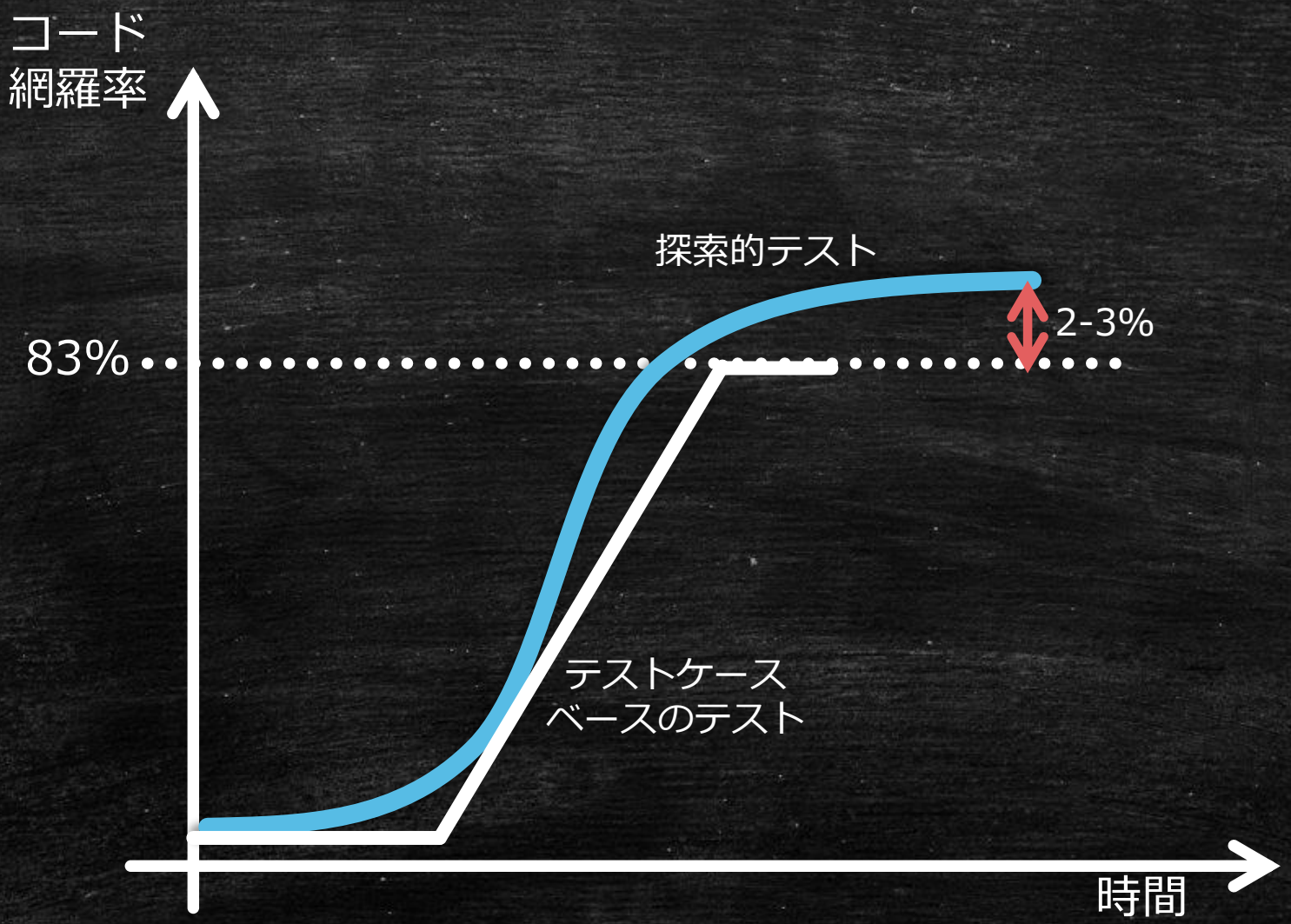
- 利点
  - **抜群に投資効果が高い**
  - 早くバグを見つけられる
  - たくさんのバグを見つけられる

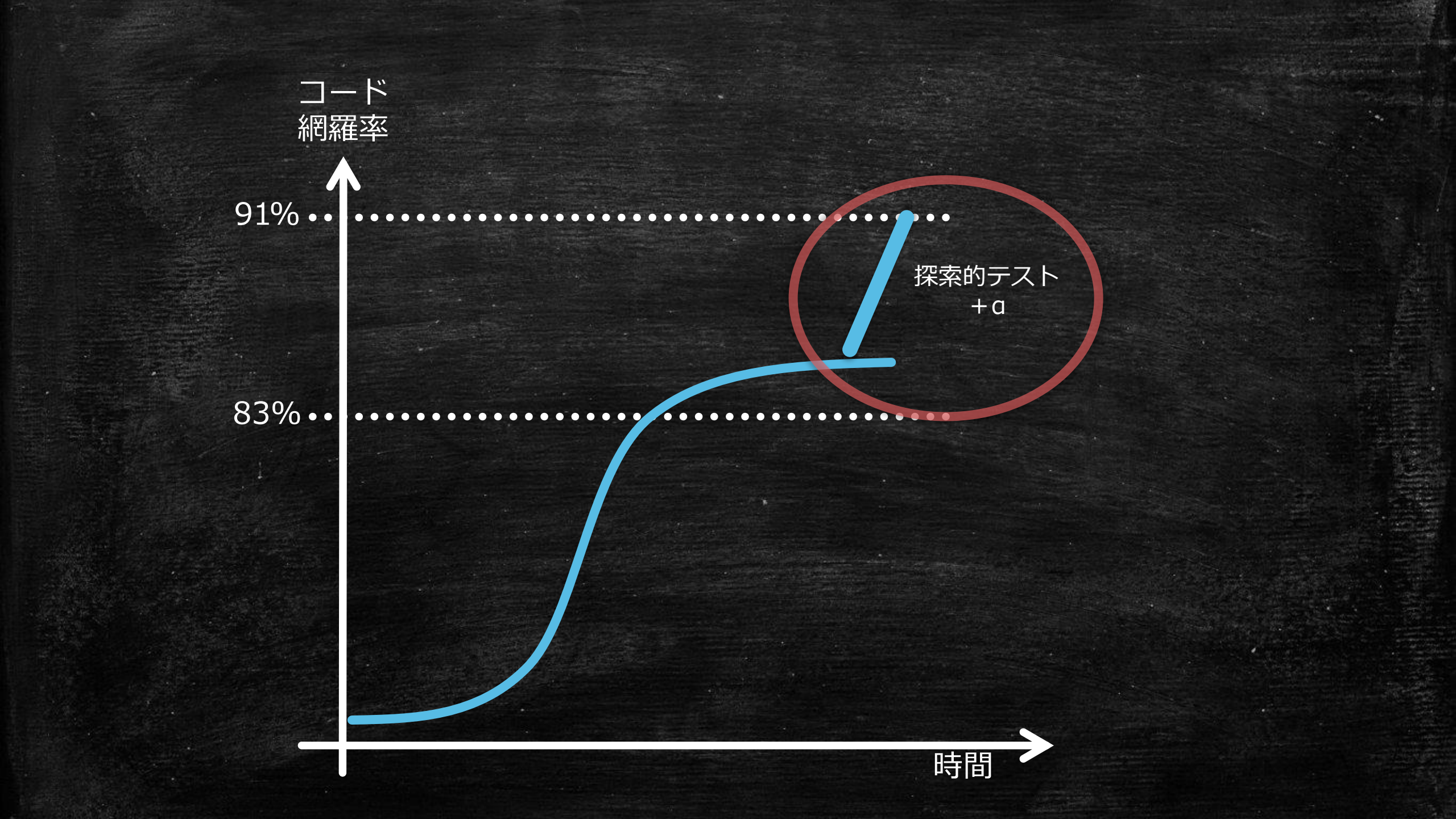

テストに費やす時間
テストケースベースのテスト
探索的テスト
1
2
3
4
5
6
日程

探索的テストの利点・欠点

- 利点
  - 抜群に投資効果が高い
  - 早くバグを見つけられる
  - **たくさんのバグを見つけられる**


フューチャー　A
フューチャー　B
0
10
20
30
40
50
探索的テスト
テストケースベースのテスト

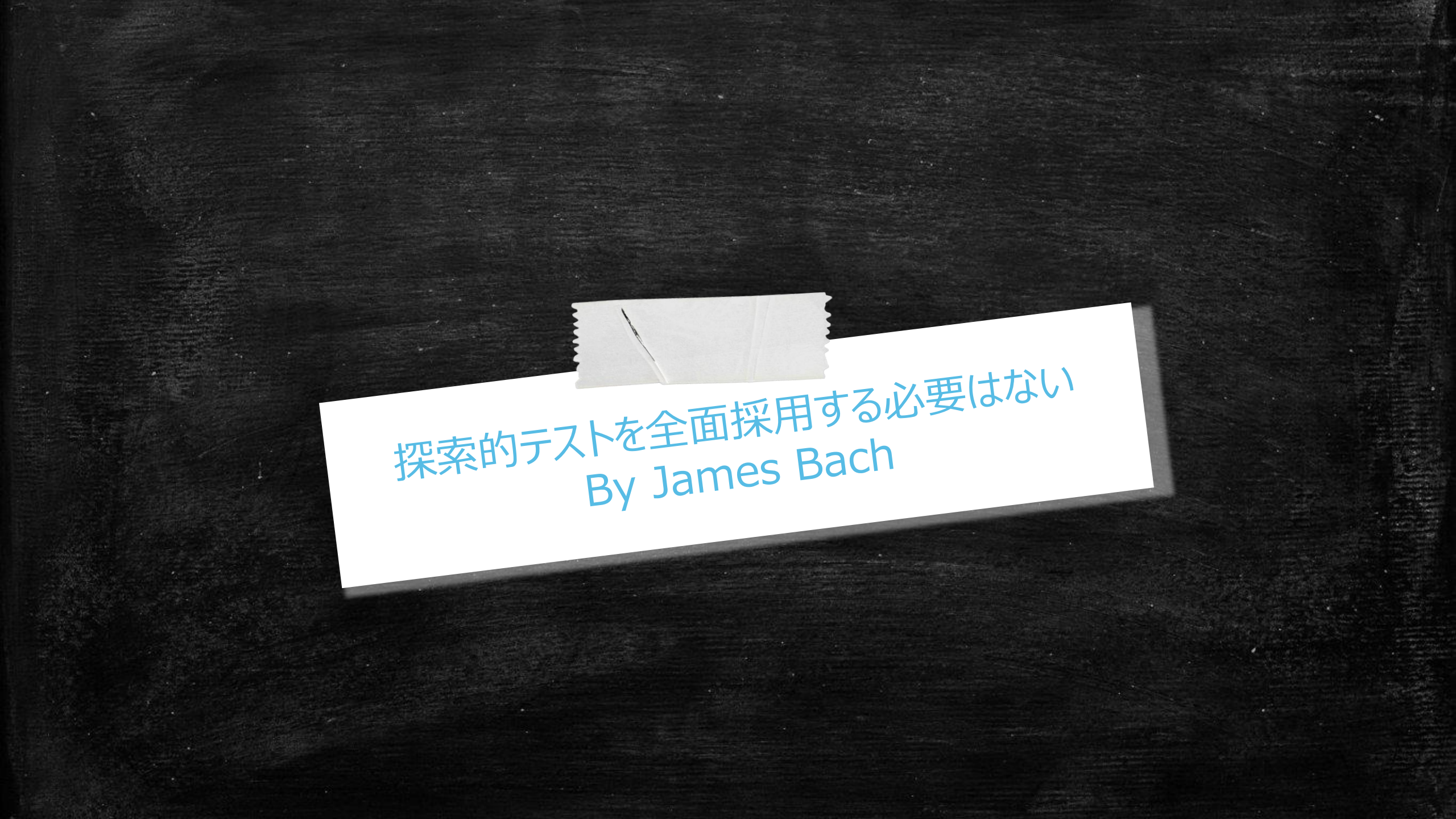

コード
網羅率
探索的テスト
2-3%
83%
テストケース
ベースのテスト
時間


コード
網羅率
91%
83%
探索的テスト
+α
時間

探索的テストの注意点

探索的テストの注意点

- 探索的テストに100%頼らない
  - テストケースベースのテストに100%頼らないというのと同じ
- 探索的テストはトレーニングを十分受けたテスト担当者によってなされる
- 非機能テストを探索的テストでアプローチしないかも．．．


探索的テストを全面採用する必要はない
By James Bach

探索的テストの注意点

- 探索的テストに100%頼らない
  - テストケースベースのテストに100%頼らないというのと同じ
  - **１つのテスト手法に頼らない**
  - **所詮テストで見つけられるバグは５０%以下**

QA活動の種類 (Activity)	レンジ
カジュアルなデザインレビュー Informal design review	25%-40%
フォーマルデザインレビュー Formal design inspection	45%-65%
インフォーマルなコードレビュー Informal code reviews	20%-35%
フォーマルコードインスペクション Informal code inspection	45%-70%
モデル化やプロトタイプの作成 Modeling and prototyping.	35%-80%
個人的なコードチェック Personal desk-checking of code	20%-60%
ユニットテスト Unit test	15%-50%
新機能テスト New function (component) test	20%-35%
統合テスト Integration test	25%-40%
回帰テスト Regression test	15%-30%
システムテスト System test	25%-55%
小規模のベータテスト（10サイト以下） Low-volume beta test(<10 site)	25%-40%
大規模のベータテスト（1000サイト以上） Low-volume beta test(> 1000)	60%-75%

探索的テストの注意点

- 探索的テストに100%頼らない
  - テストケースベースのテストに100%頼らないというのと同じ
- **探索的テストはトレーニングを十分受けたテスト担当者によってなされる**
- 非機能テストを探索的テストでアプローチしない

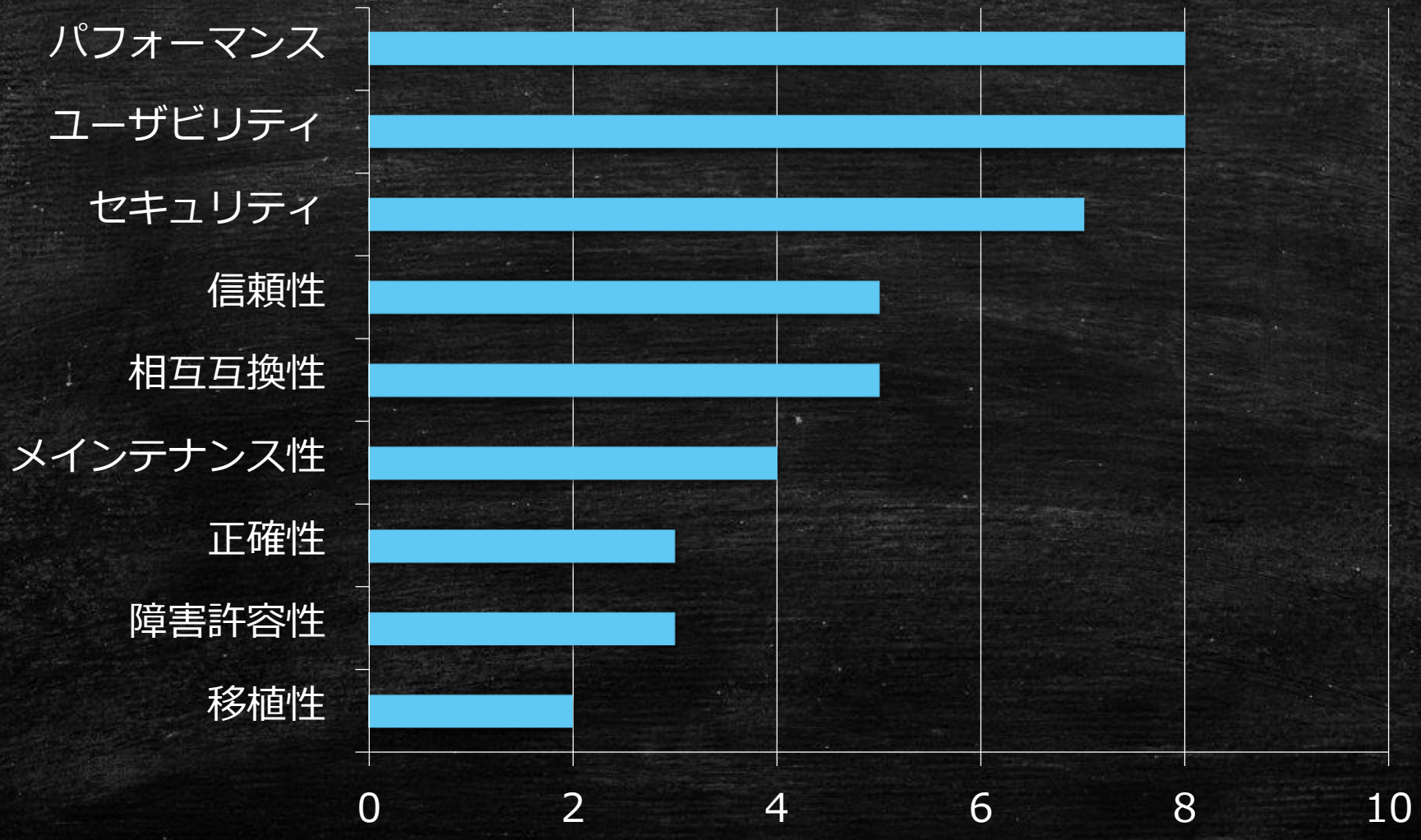
探索的テストはトレーニングを十分受けたテスト担当者によってなされる

- Microsoftの実験
  - 事前にバグを入れ込んだソフトウェアにおいて、トレーニングを受けていないテスト担当者は総バグの**10%**しかみつけられなかったが、トレーニングを受けたテスト担当者は**65%**のバグを見つけた。

探索的テストの注意点

- 探索的テストに100%頼らない
  - テストケースベースのテストに100%頼らないというのと同じ
- 探索的テストはトレーニングを十分受けたテスト担当者によってなされる
- **非機能テストを探索的テストでアプローチはしないかも．．．**

非機能要求の重要性


パフォーマンス
ユーザビリティ
セキュリティ
信頼性
相互互換性
メインテナンス性
正確性
障害許容性
移植性
0
2
4
6
8
10

非機能要求の探索的テスト

- パフォーマンステスト
- ユーザビリティテスト
- セキュリティテスト
- 信頼性テスト



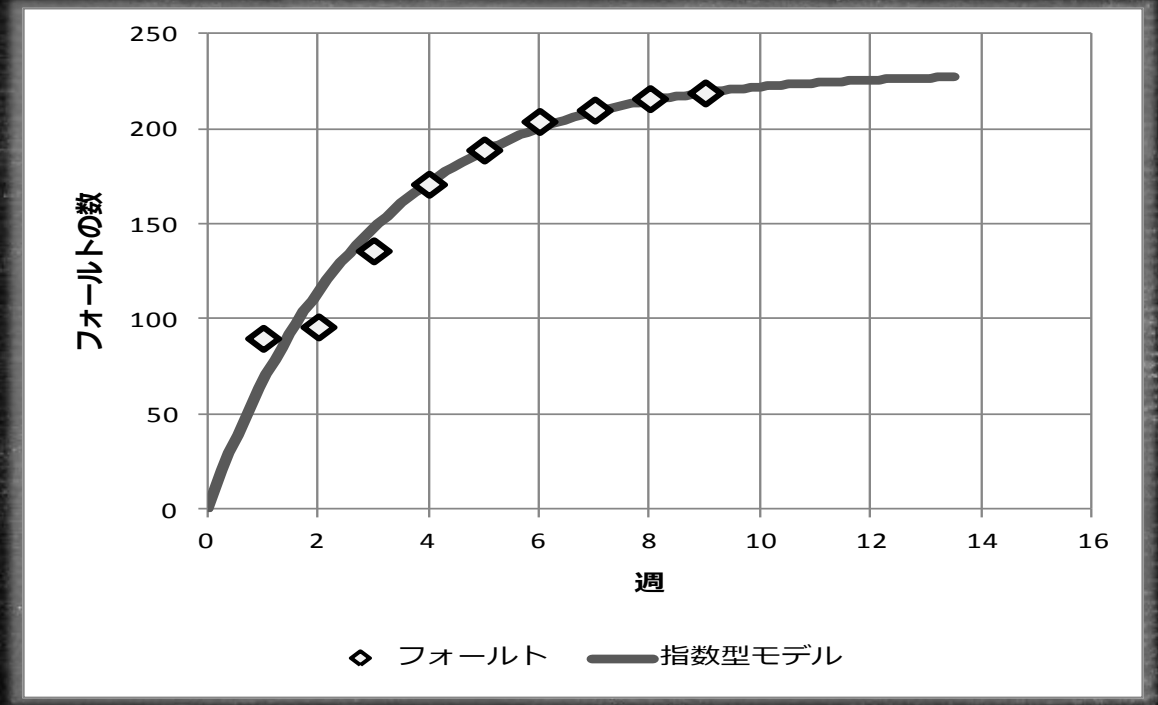

ユーザビリティテスト

タイプ	探索的テスト	テストケーステスト	探索的テスト/テストケーステスト（%）
ドキュメンテーション Documentation	8	4	200%
ＧＵＩ	70	49	143%
矛盾点Inconsistency	5	3	167%
機能欠如 Missing function	98	96	102%
パフォーマンス Performance	39	41	95%
技術的バグ Technical defect	54	66	95%
ユーザビリティ Usability	19	5	380%
間違った振る舞い Wrong behavior	263	239	110%

信頼度成長曲線

$$m(t) = a(1 - e^{-bt})$$

テスト実行時間(週)	発見されたバグの数
1	90
2	63
3	44
4	31
5	22
6	15
7	11
8	7
9	0



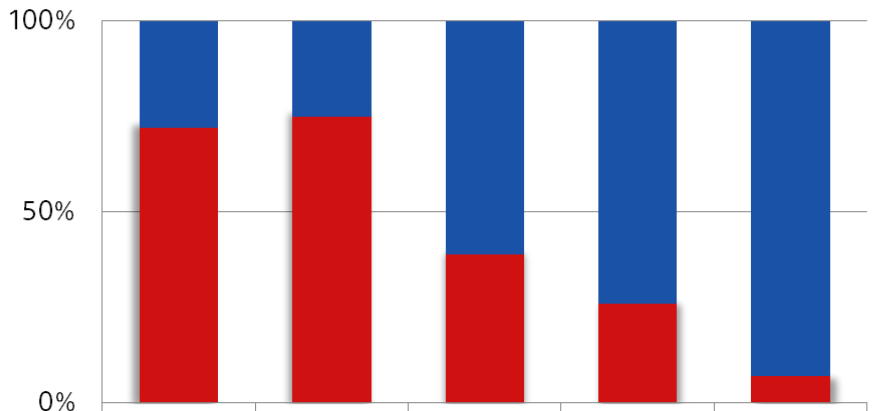
探索的テストでの人材の問題

Google

We certainly hire based on development skill as well—Google has traditionally hired folks with a **computer science degree** for either **development or testing**, and our test candidates are also asked development questions and **coding questions** in our interviews. We believe that those core skills are really crucial for doing this.

Microsoft
上流品質を作りこめるエンジニアが必要

MicrosoftのSTE（テストだけの人），SDET(コードを書くテスター)の割合



	1999	2001	2003	2005	2007
SDET(コードを書くテストな人)	28%	25%	61%	74%	93%
STE(テストだけな人)	72%	75%	39%	26%	7%

end