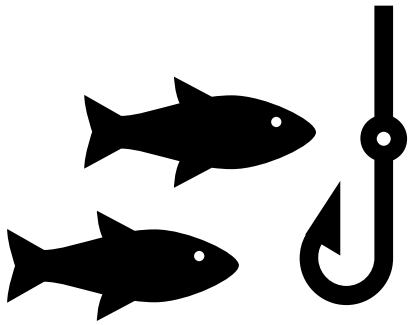
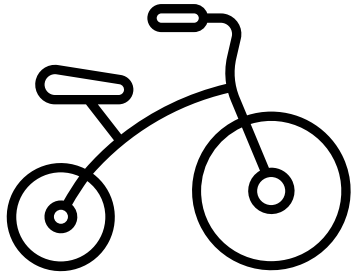
WingArc 1ST
The Data Empowerment Company

E2E自動テストを浸透させるために工夫したこと



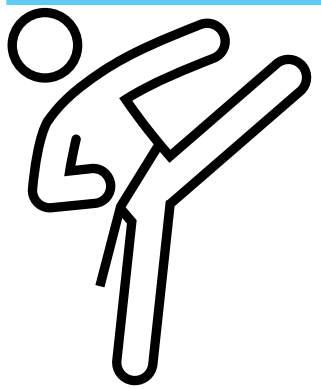


自己紹介

名前　　及川 智之
住まい　北海道札幌市
所属　　ウイングアーク１ｓｔ株式会社
AutomationTeam

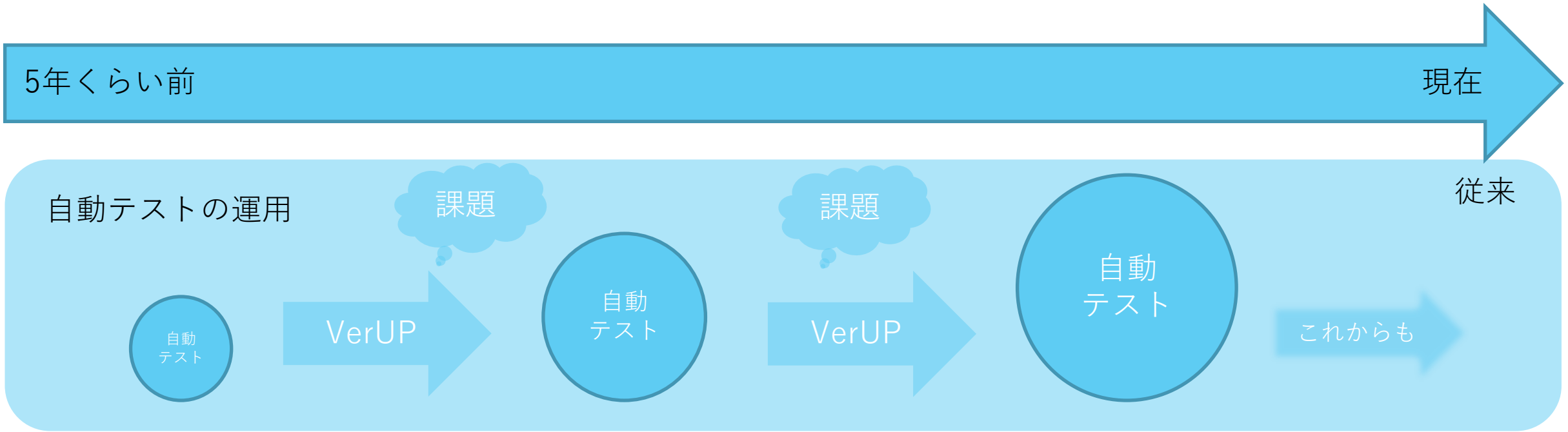
製品グループの枠から外れて
各製品での自動テスト支援
をするチーム

普段は製品のテストをしたり、
自動テスト支援活動してます

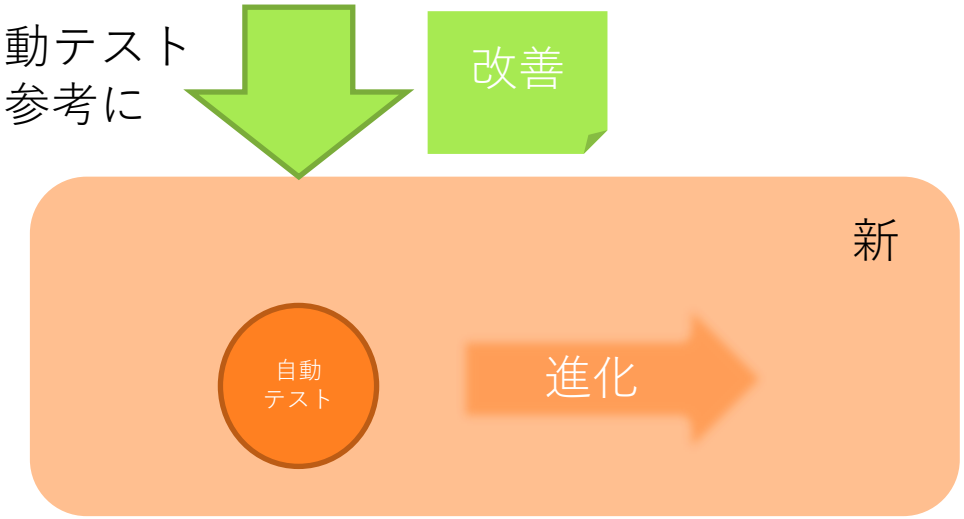


わりとアクティブ趣味

あらすじ



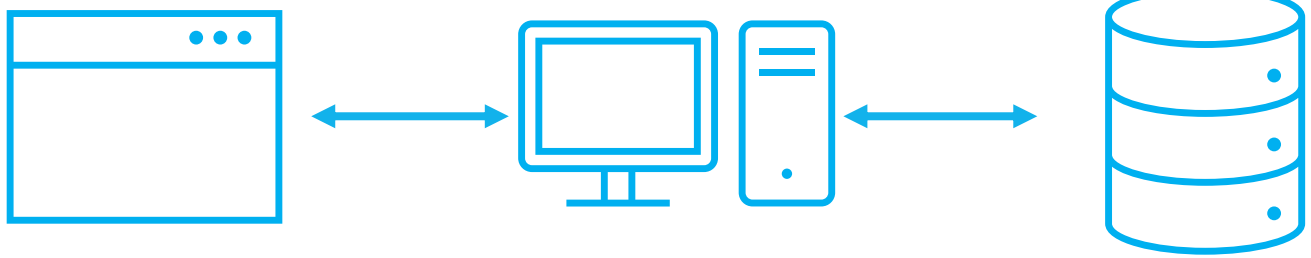
既存システムの自動テスト
フレームワークを参考に
新しく構築



対象となる製品

データを見たり分析したりする製品

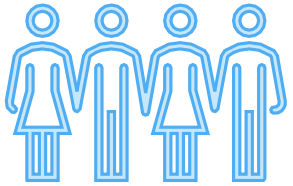
クライアント　　サーバー　　データベース



つくりは全く異なるが同じくデータの表現をする製品で、見た目は全く異なっていても、実行や比較といった枠組みは同じように考えることができる

浸透

ツール作る



品質保証で
使い始める

ケースが増える

開発も使い
始める

早期の品質向上につなぐ

ここが目標

日常的に使われる

頼りにされている

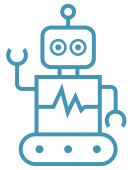
これからも
期待されている

今日のキーワード

コマンドベースのスクリプト	プログラムではなく製品の操作を書く

実行環境作成のワンタッチ化	パッケージマネージャーを使っている

製品バージョン	画面が変わる、操作が変わる、結果が変わる



BVT	定期的に実行されていると安心感が得られる

選択実行	作成中、調査では特定のスクリプトを繰り返し実行する

書きやすいこと。
増やしやすいこと。

コマンドベースのスクリプト	プログラムではなく製品の操作を書く
実行環境作成のワンタッチ化	パッケージマネージャーを使っている
製品バージョン	画面が変わる、操作が変わる、結果が変わる
BVT	定期的に実行されていると安心感が得られる
選択実行	作成中、調査では特定のスクリプトを繰り返し実行する

コマンドベースのスクリプトとは

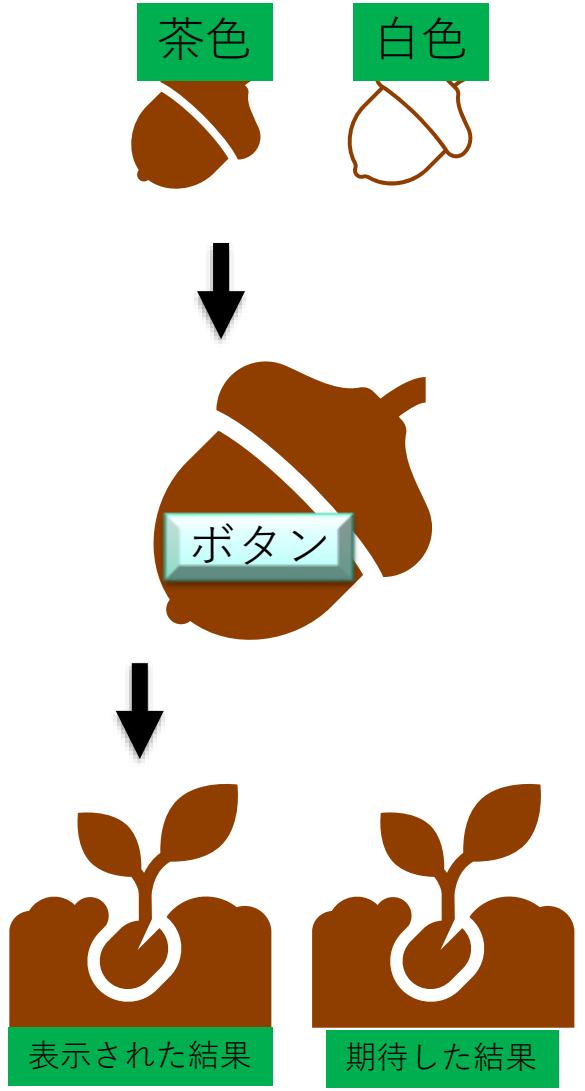
```
1 選ぶ(茶色)
2 押す(ボタン)
3 結果比較()
```

操作を記述する
「選ぶ」とか「押す」
とかがコマンド

茶色を選ぶ

ボタンを押す

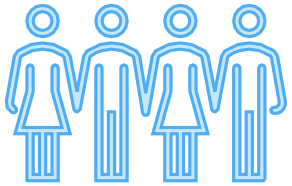
一致すればテスト成功



コマンドベースのスクリプトを使う理由

テスト自動化エンジニア

- 自動化に特化したプログラムを作成できる
- 簡単にテストできる状況を準備する



テストエンジニア（イメージです）

- プログラミングは得意ではない
- コツコツと積み上げるのは得意

プログラミング臭がしないだけではじめやすい

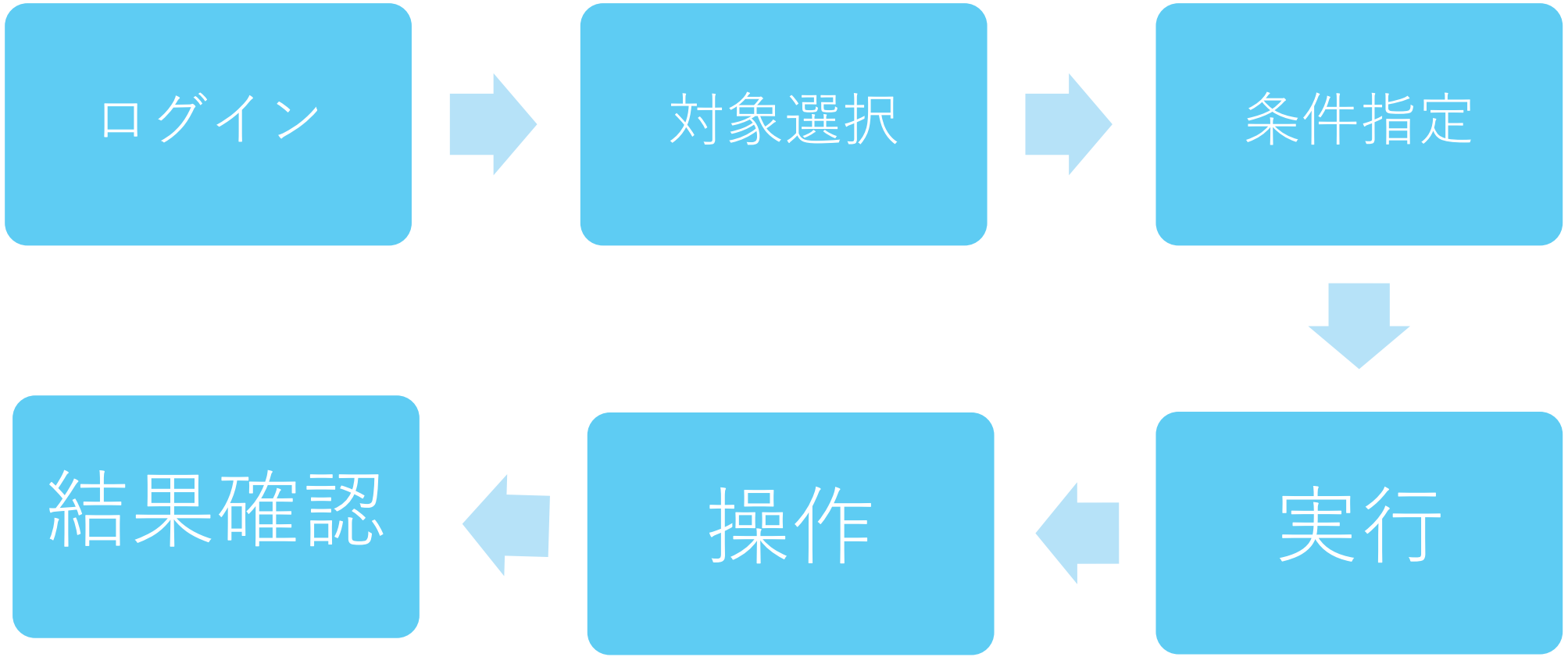
製品開発エンジニア（イメージです）

- テストはめんどくさい（けどやる）
- テストを書くのはもっとめんどくさい（やらない）

テストやるついでならやってくれるかもしれない？

コマンドベースのスクリプトの採用

- 製品操作の基本的な流れ



- これをそのままスクリプトとして書ける

従来と新しいフレームワークでの書式の違い

テキストファイルで記述していたのをJavaコードに変更した

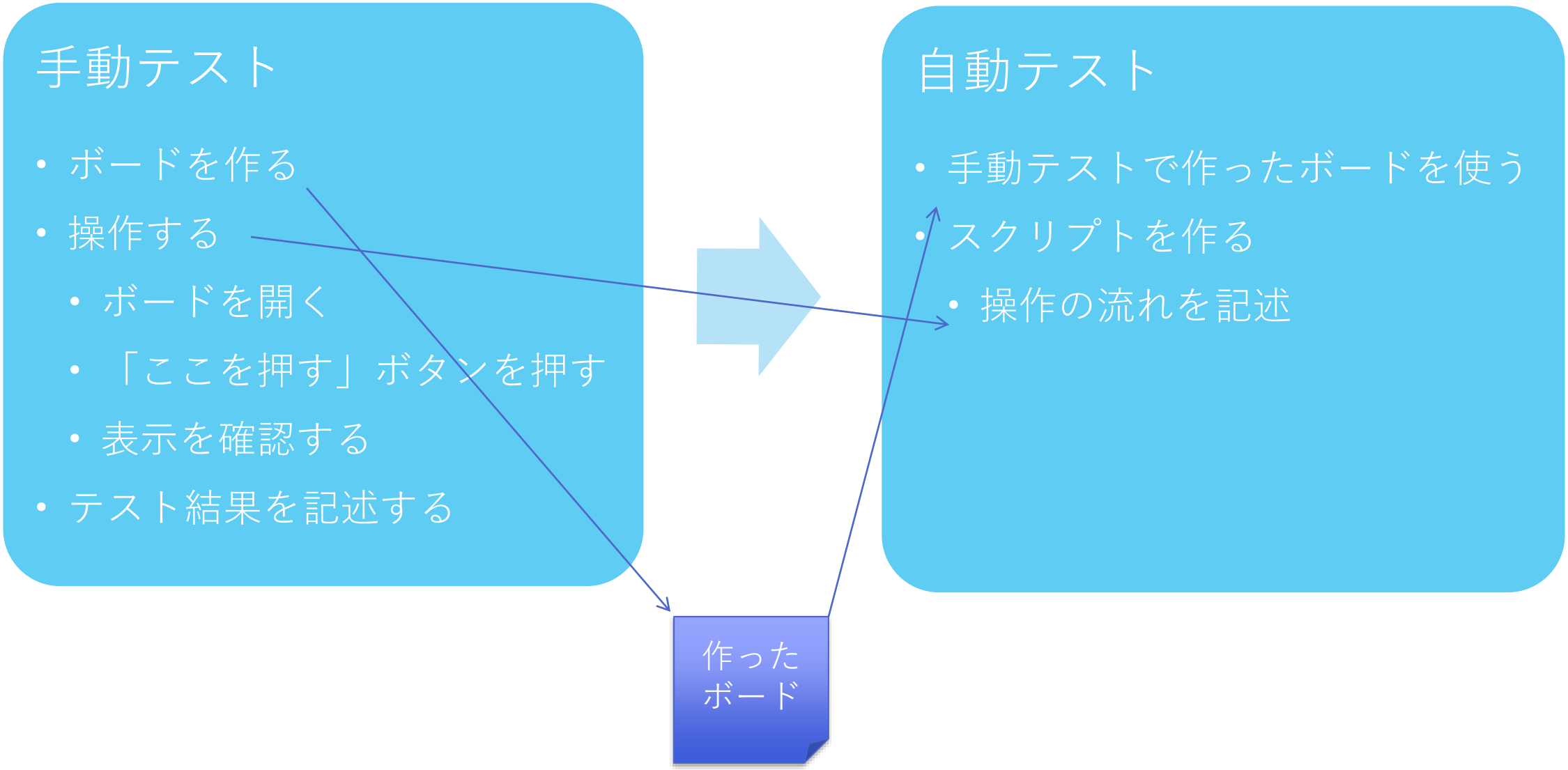
それはなぜなのか？

テキストとJavaでの対比

	テキスト(従来)	Java(新)
使いやすさ	- エディタがあればかけるので気持ちが楽 - コンパイルがいらないので量産しやすい ただし - マニュアルを見ながら作る必要がある	- IDEが補完とjavadoc表示、エラー検出してくれる ただし - 開発環境の準備が必要
構造	- スクリプトエンジンの作りこみが必要 - JUnitとして動かしたい場合はJUnitCoreからカスタマイズする必要があり複雑	- パッケージとクラスとタグでの管理がJUnitの標準機能でできる - 実行構造はJUnit任せなので作りこみがない

新しいフレームワークではIDEの便利さとフレームワーク自体をシンプルにするため、
Javaでテストを書く方法を選択した

手動テストが自動テストになるまで



スクリプトはテストケースそのまま

Neovim

```
 1 @Test
 2 void test() throws Exception {
 3     url("{base_url}/main?mbid=XXX&boardpath=YYY");
 4     ログイン();
 5     ボタン("ここを押す");
 6     wait(FieldType.label,"OK",0);
 7     ボタン("地域");
 8     ボタン("OK");
 9     保存Json(FieldType.text);
10     保存Json(FieldType.label);
11     保存Image();
12 }
```

[無名] [+]

製品テストチームに提供時のフレームワークで記述した内容です
現在はチーム内最適化されています

コマンドベースの利点

理解しやすい

- スクリプトがテスト手順になっている

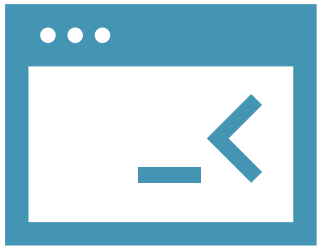
増やしやすい

- 類似テストからの部分修正で動く

生成しやすい

- テスト対象だけ置換
- テスト手順から変換

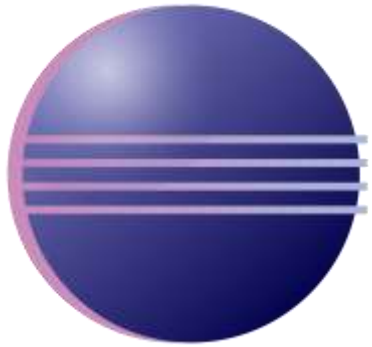
細かいwaitなどはFWで吸収しテスト手順を書くため



実行できないと
始めてもらえない。

コマンドベースのスクリプト	プログラムではなく製品の操作を書く
実行環境作成のワンタッチ化	パッケージマネージャーを使っている
製品バージョン	画面が変わる、操作が変わる、結果が変わる
BVT	定期的に実行されていると安心感が得られる
選択実行	作成中、調査では特定のスクリプトを繰り返し実行する

環境作成の簡単化



- IDEを使ったスクリプト作成をするにあたって開発環境(今回はEclipse)を構成する必要がある

- パッケージマネージャとそれでインストールする各種ツールを使って様々な設定を簡単化
  - 各種ツールのインストールはパッケージマネージャ
  - 各種ライブラリのインストールはMaven
  - 環境の設定やソース、期待値はgit

- パッケージマネージャはChocolateyとScoopを使い分けている

こんな経験ありませんか？

探してダウンロードして
インストールして面倒！

ツール探し

誤った構成

うごかない・・・
バージョンが違う？
エディションが違う？

リポジトリ？なにそれ
改行コード？
文字化ける？
ツール設定？
ライブラリどこ？
起動したら赤い×が？
資料みても設定できない

設定ミス

環境作るまでは待つだけでよくしたい

パッケージマネージャ

- EclipseやGitやJavaなどを集める
- 適切なバージョンとエディションの組み合わせ

Git

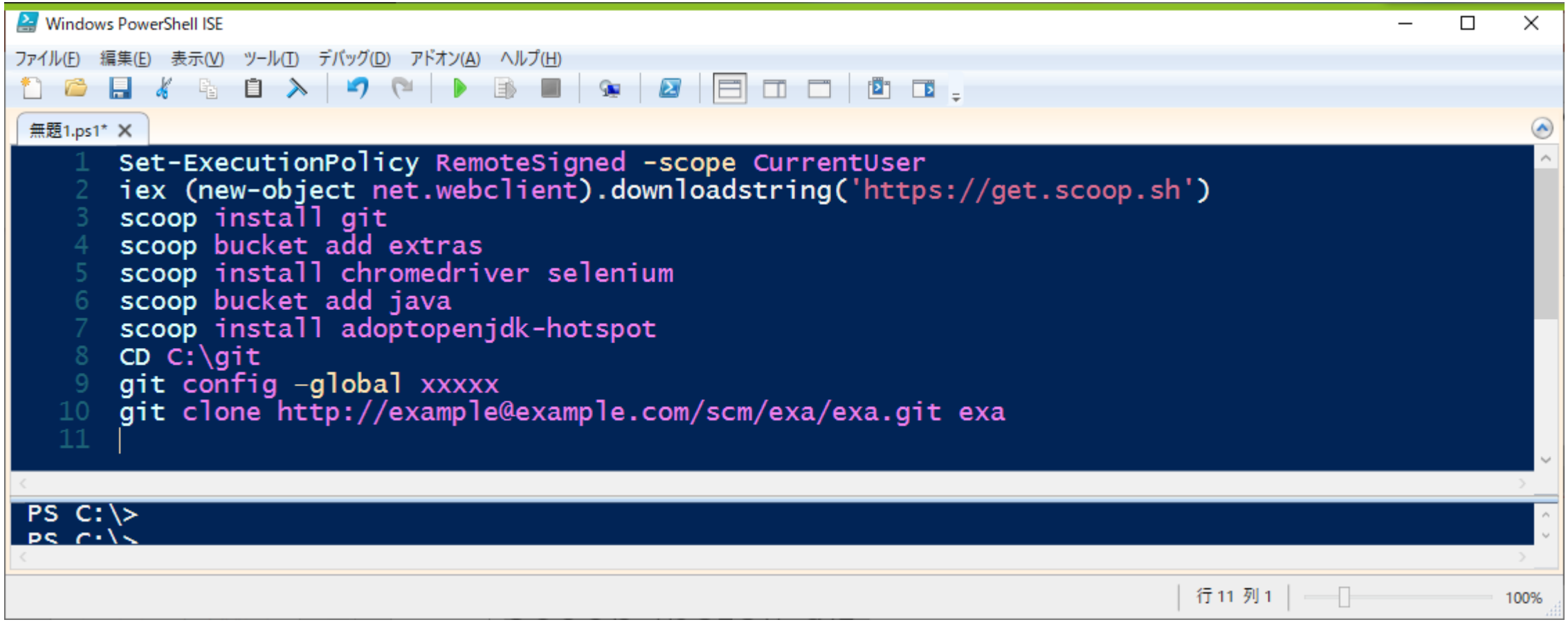
- フレームワークや他者のテストケースを集める
- 起動やファイルの設定もしておく

Maven

- Javaのライブラリを適切な順序と依存で設定する
- Eclipseを起動したらすぐ使える、すぐ書ける

すぐ始められるならちょっとやってみようかな、という気持ちにもなる

こんなPowershellをつらつらと



プログラミング
になじみがない

これらのツール
の存在知らない

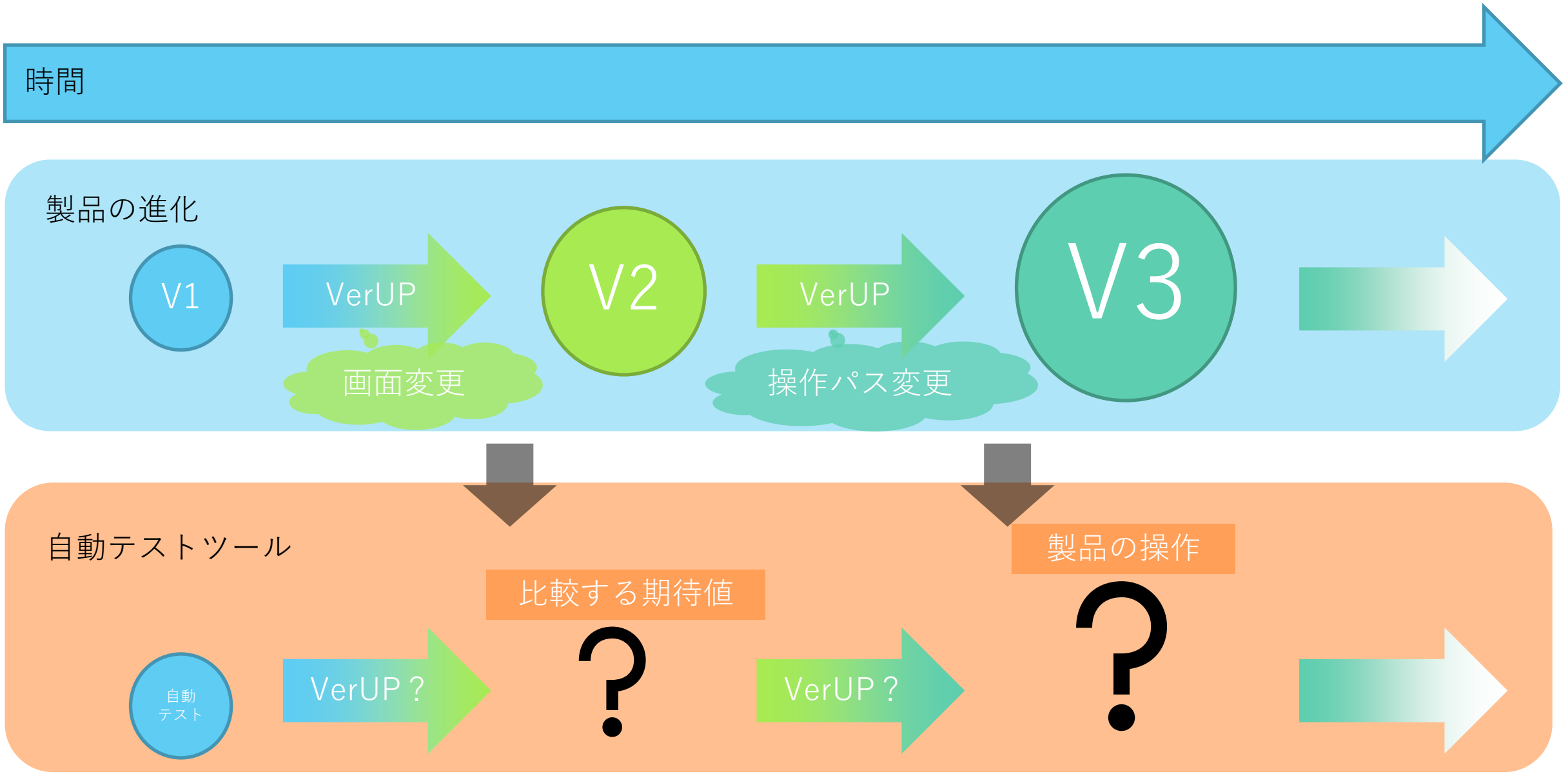
便利！

ほかのひらめきや効率化
のあしがかりになるかも

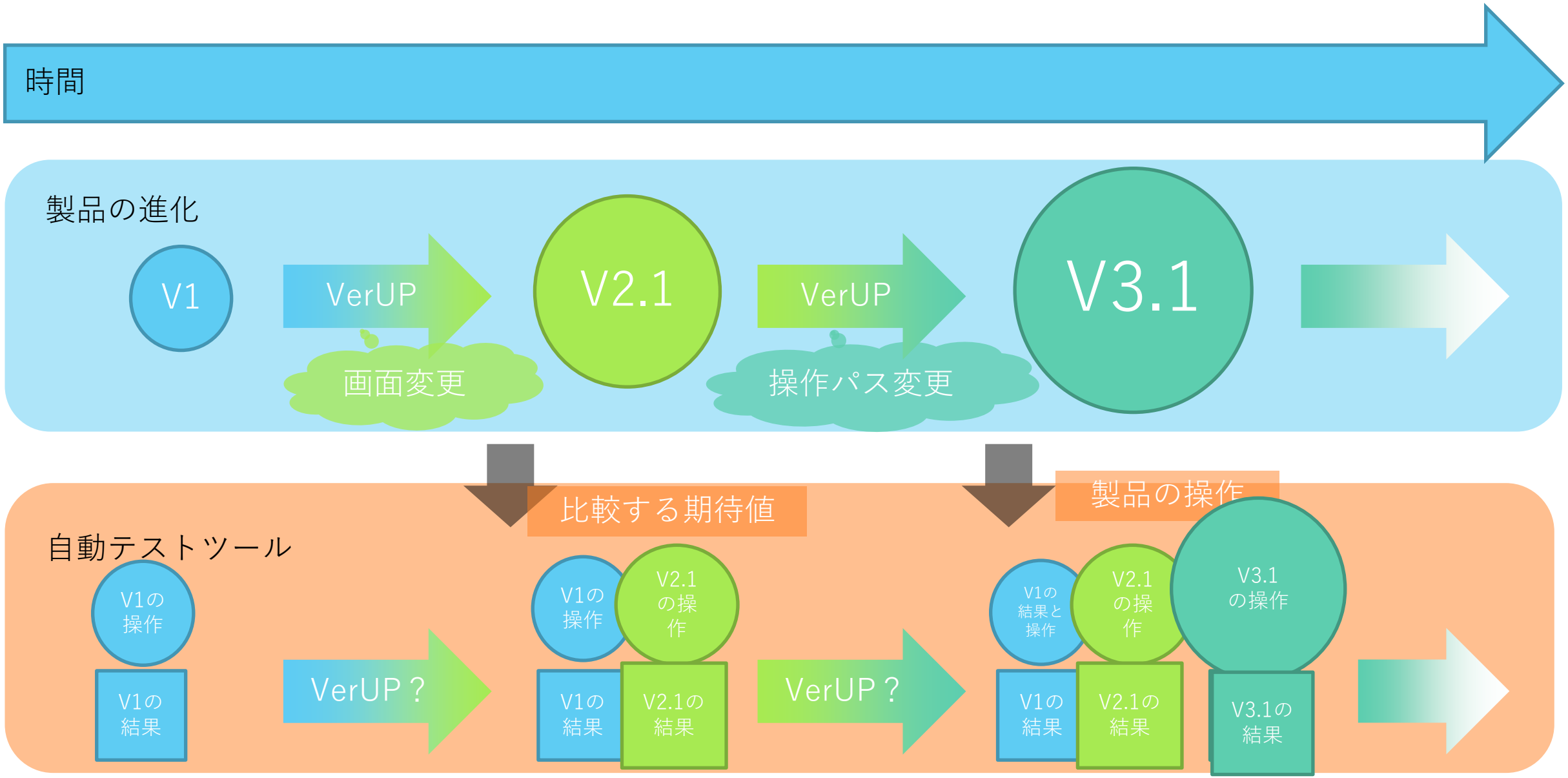
検証対象の製品バージョンを
意識しないと爆発する。

コマンドベースのスクリプト	プログラムではなく製品の操作を書く
実行環境作成のワンタッチ化	パッケージマネージャーを使っている
製品バージョン	**画面が変わる、操作が変わる、結果が変わる**
BVT	定期的に実行されていると安心感が得られる
選択実行	作成中、調査では特定のスクリプトを繰り返し実行する

バージョン



複数の製品バージョンをサポート



バージョンアップへの耐性を付けておこう

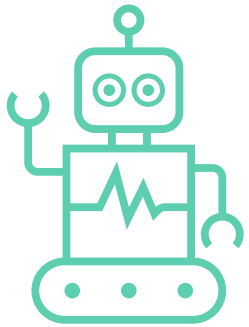
バージョンの対象

スクリプト

選択実行
で対処

結果

分離



ツール

クラス構成

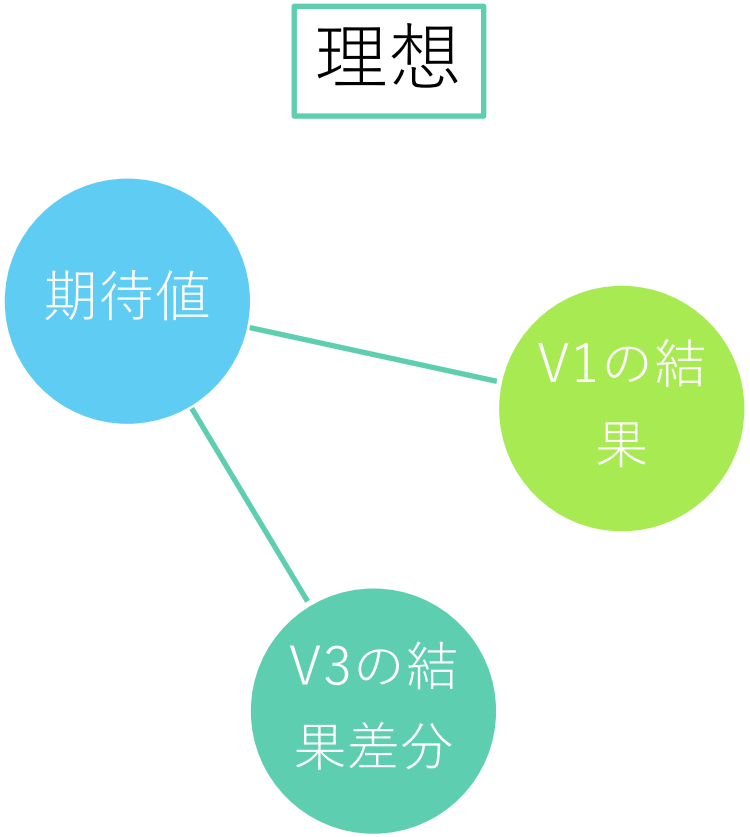
製品

バージョン
検出

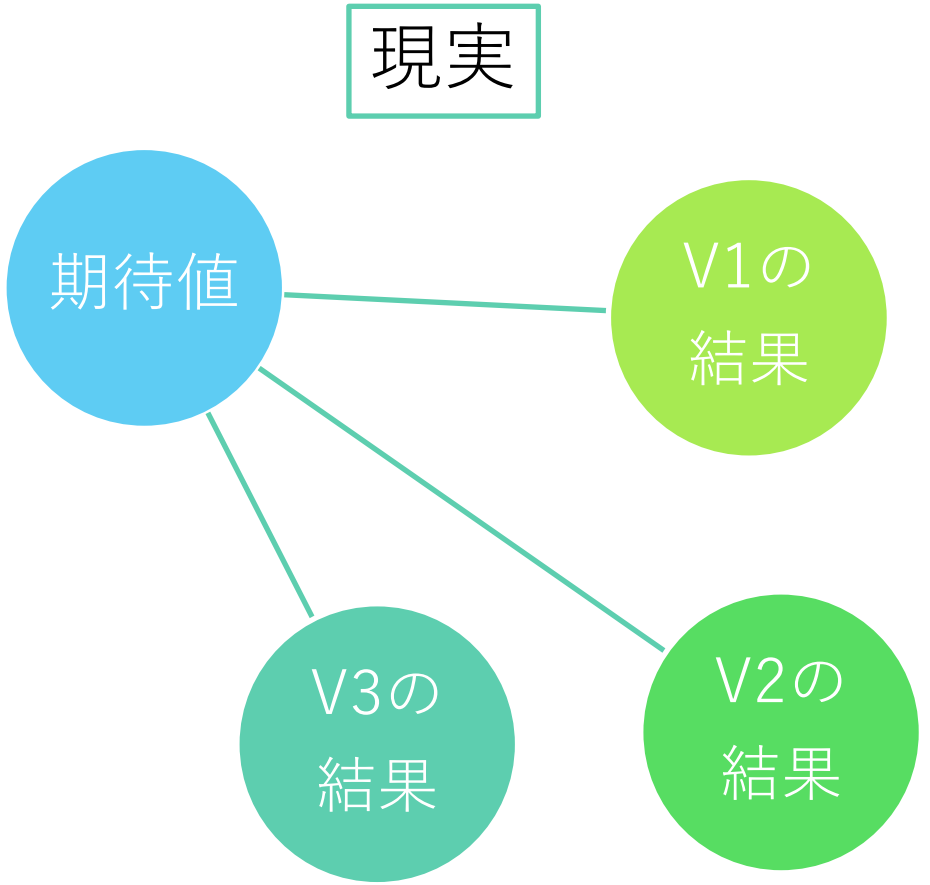
スクリプトのバージョン

- 後述

期待値のバージョンの理想と現実



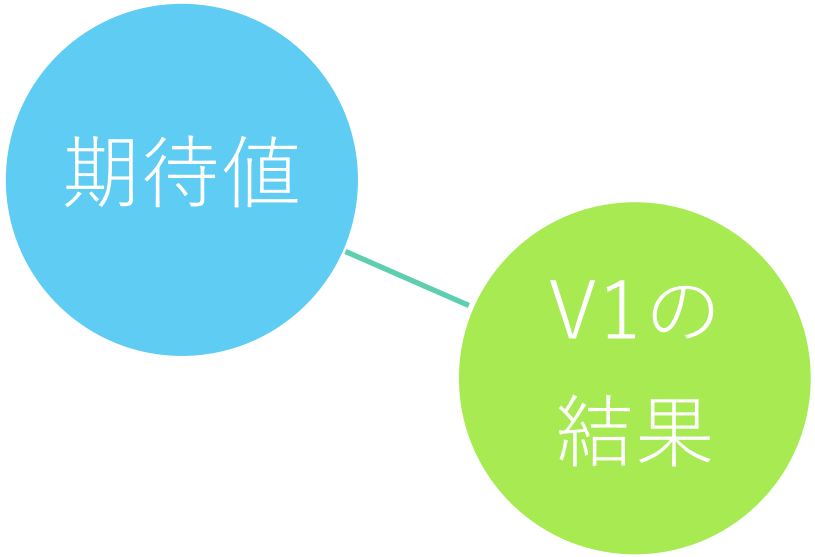
ファイルサイズ増えるし
変わった分だけ管理したい



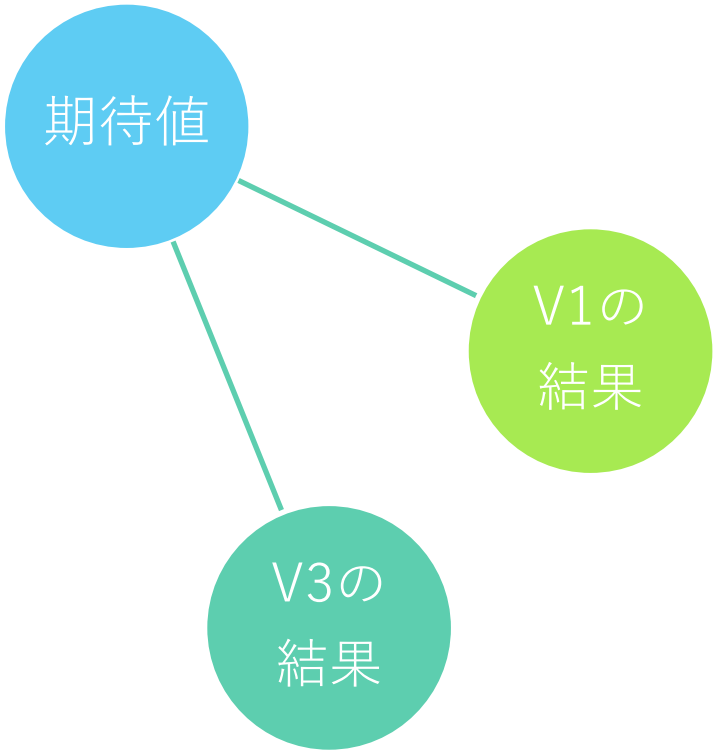
全部のバージョン結果違った

差分管理でトレードオフしていたリスク

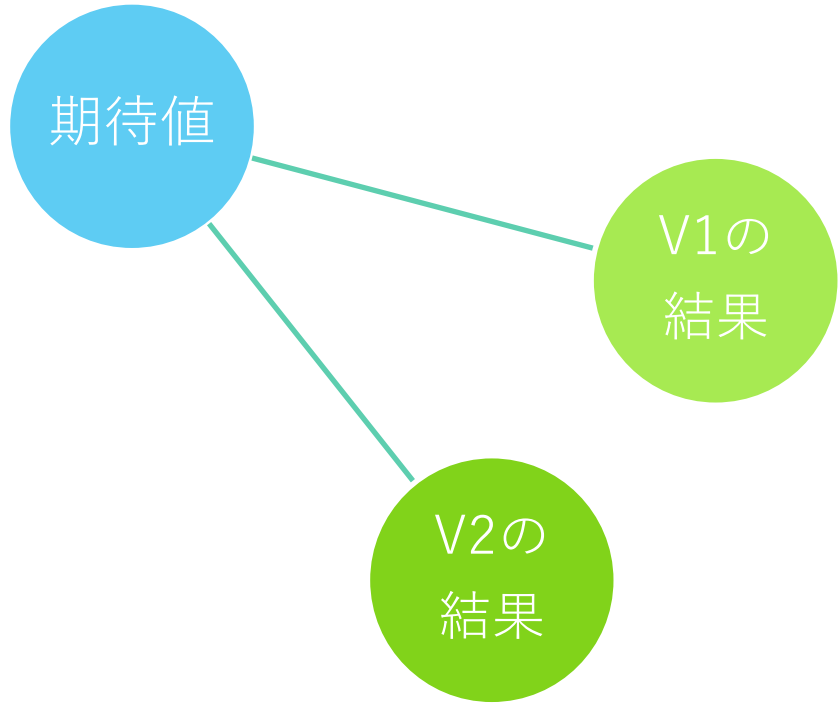
V1~3で結果が共通



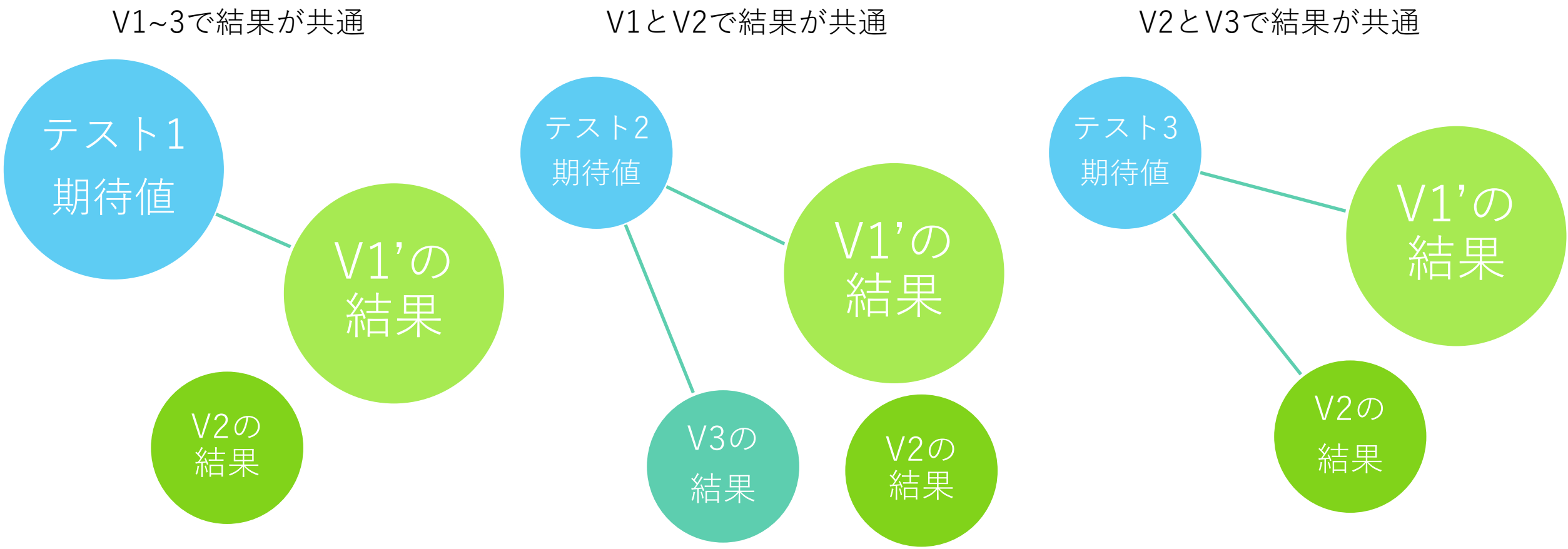
V1とV2で結果が共通



V2とV3で結果が共通

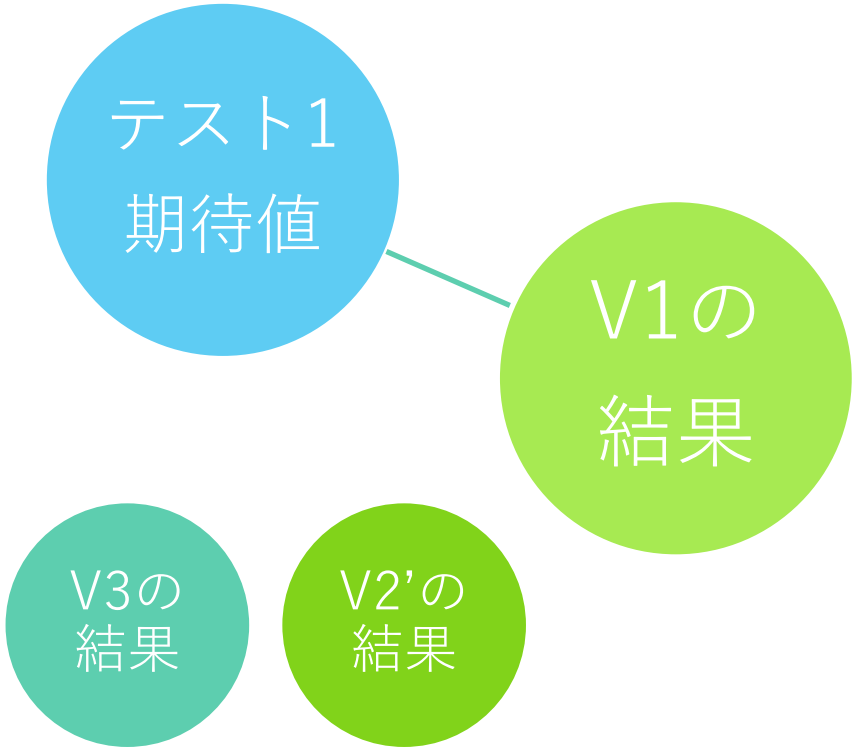


V1の結果が変わった

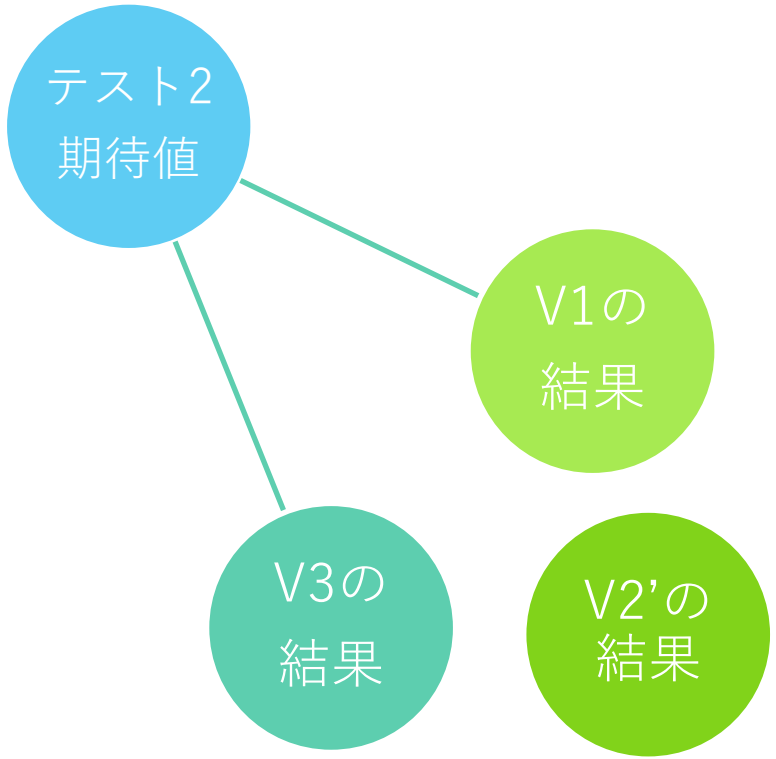


V2の結果が変わった

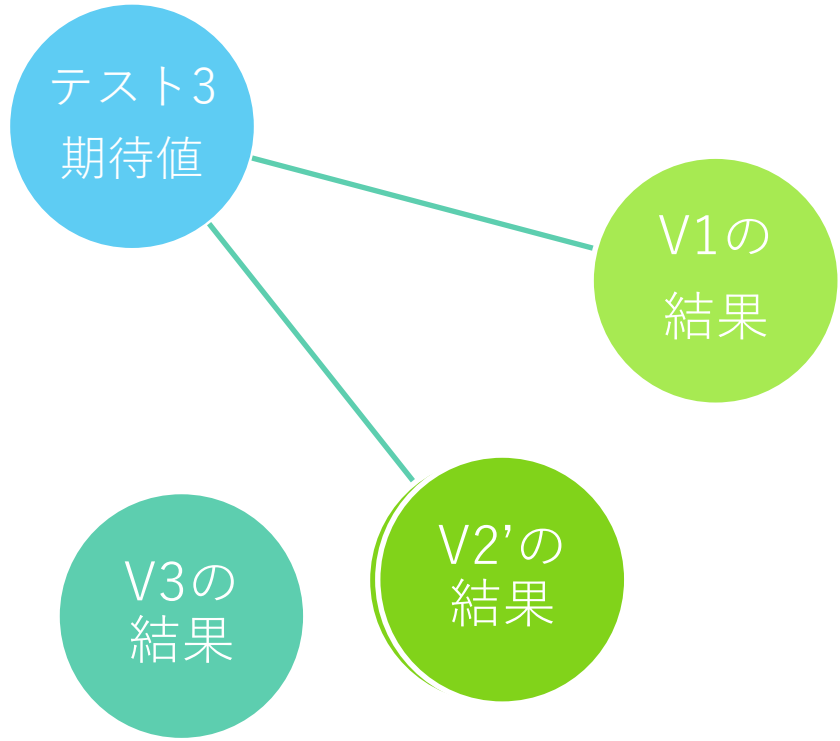
V1~3で結果が共通



V1とV2で結果が共通



V2とV3で結果が共通



そのときどきで修正が変わる

- 事故が起こりそうな予感がしませんか？

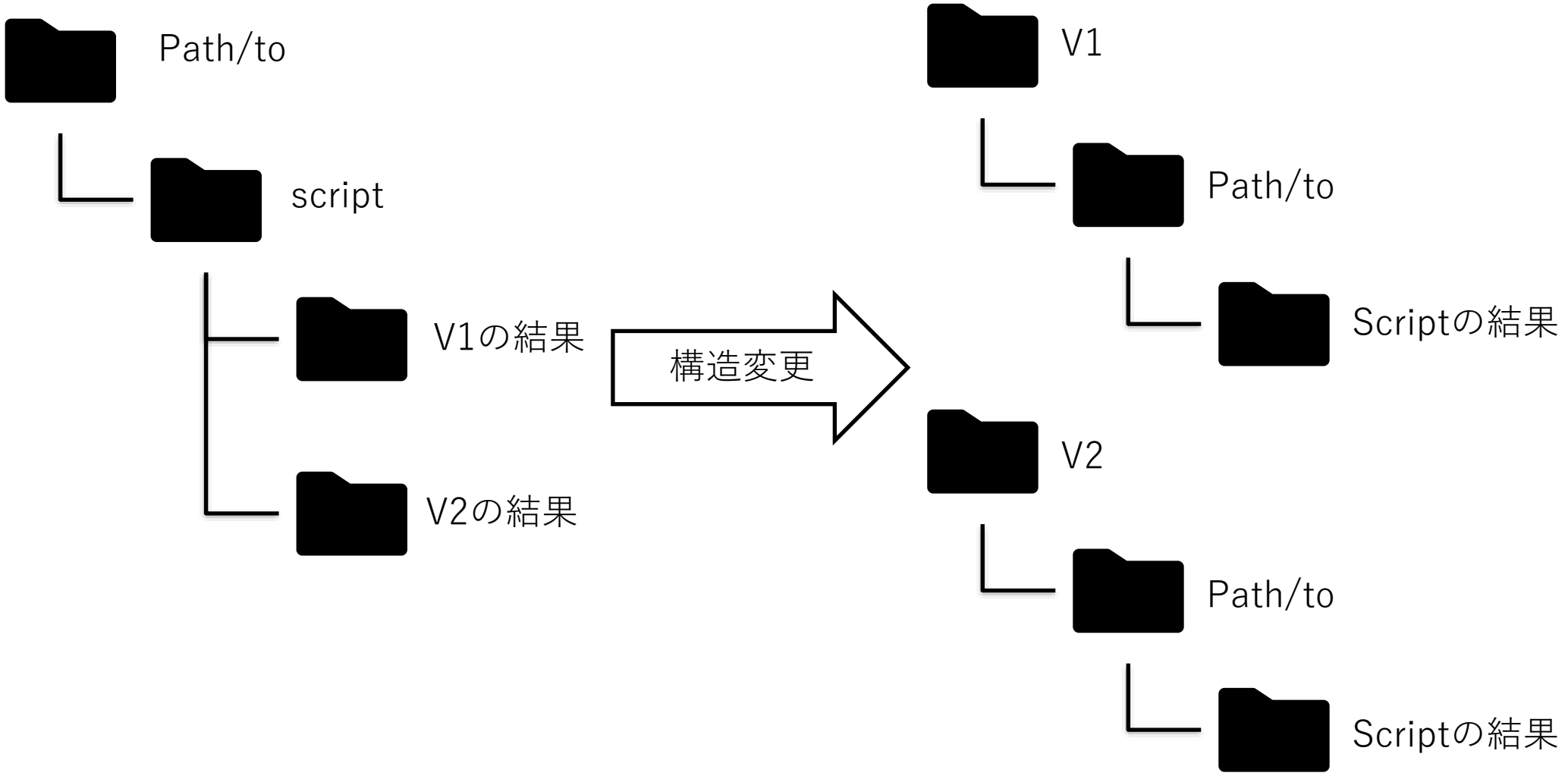
誤リプレイス

間違った結果で上書き

間違ったバージョンで上書き

発生した！

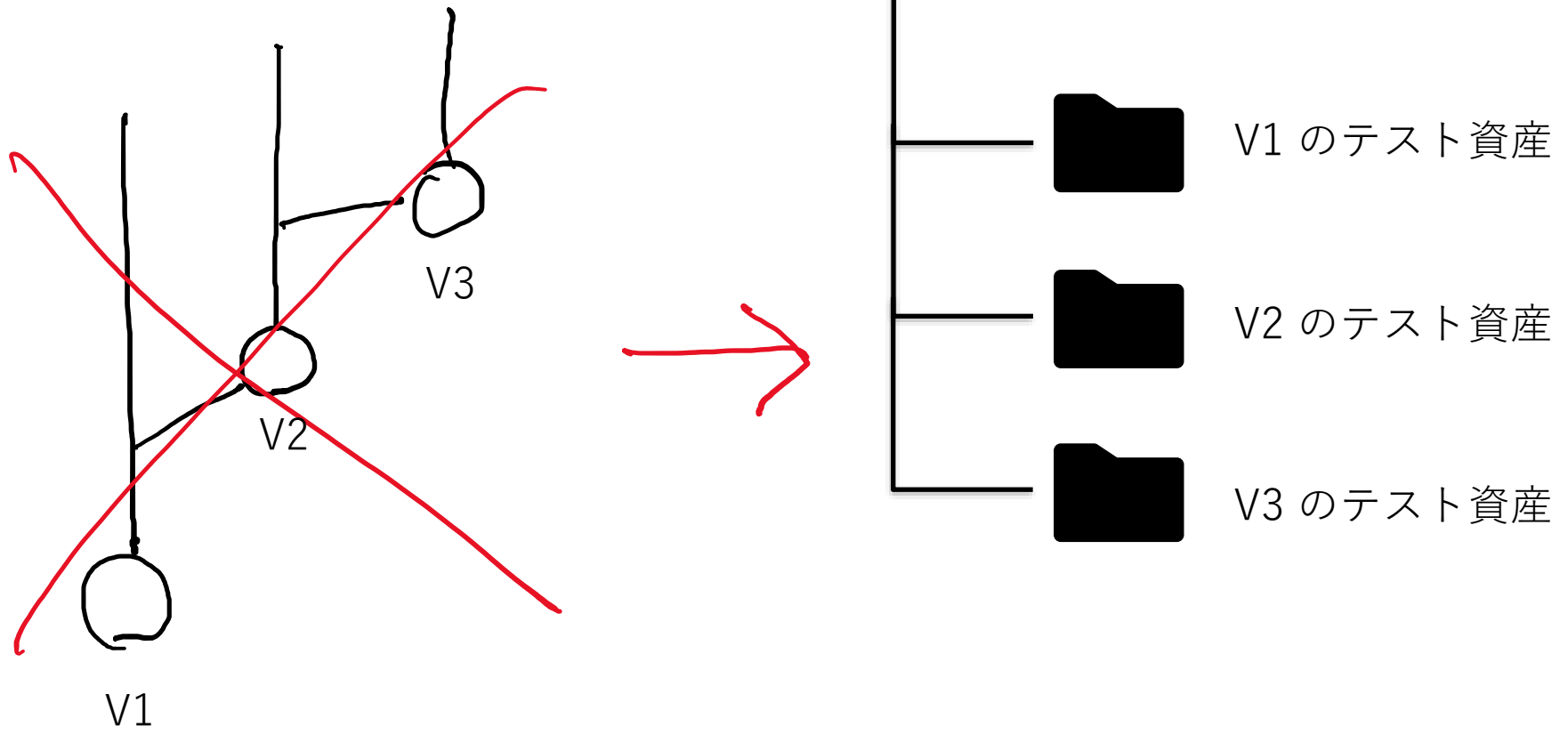
テスト期待値のバージョンの構造を変更して管理を楽にした



gitを使ってますが

git

ブランチではなくフォルダで管理



バージョンの管理方法

製品の変更を全部に適用は面倒

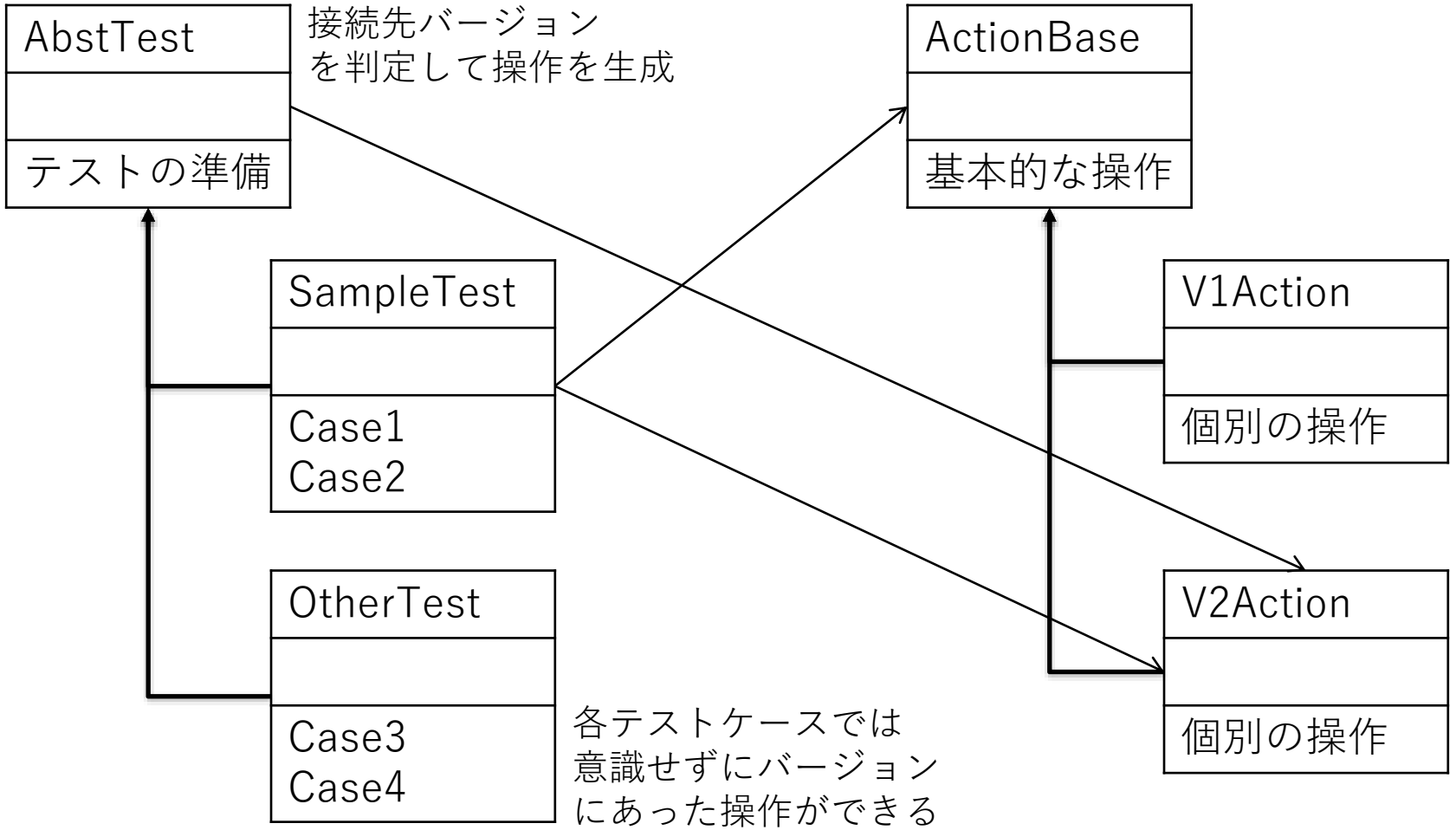
フレームワークの修正を全部に適用するのは面倒

複数バージョンテストで自動判別したほうが間違えない

過去版にテスト追加するごとに各ブランチコミット面倒

という感想をもとに製品バージョンはブランチではない管理としています

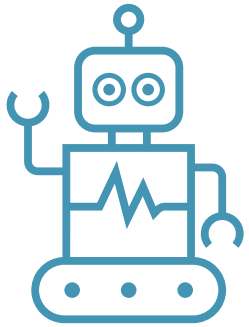
テストケースからはバージョンを意識させない作り



操作のバージョン

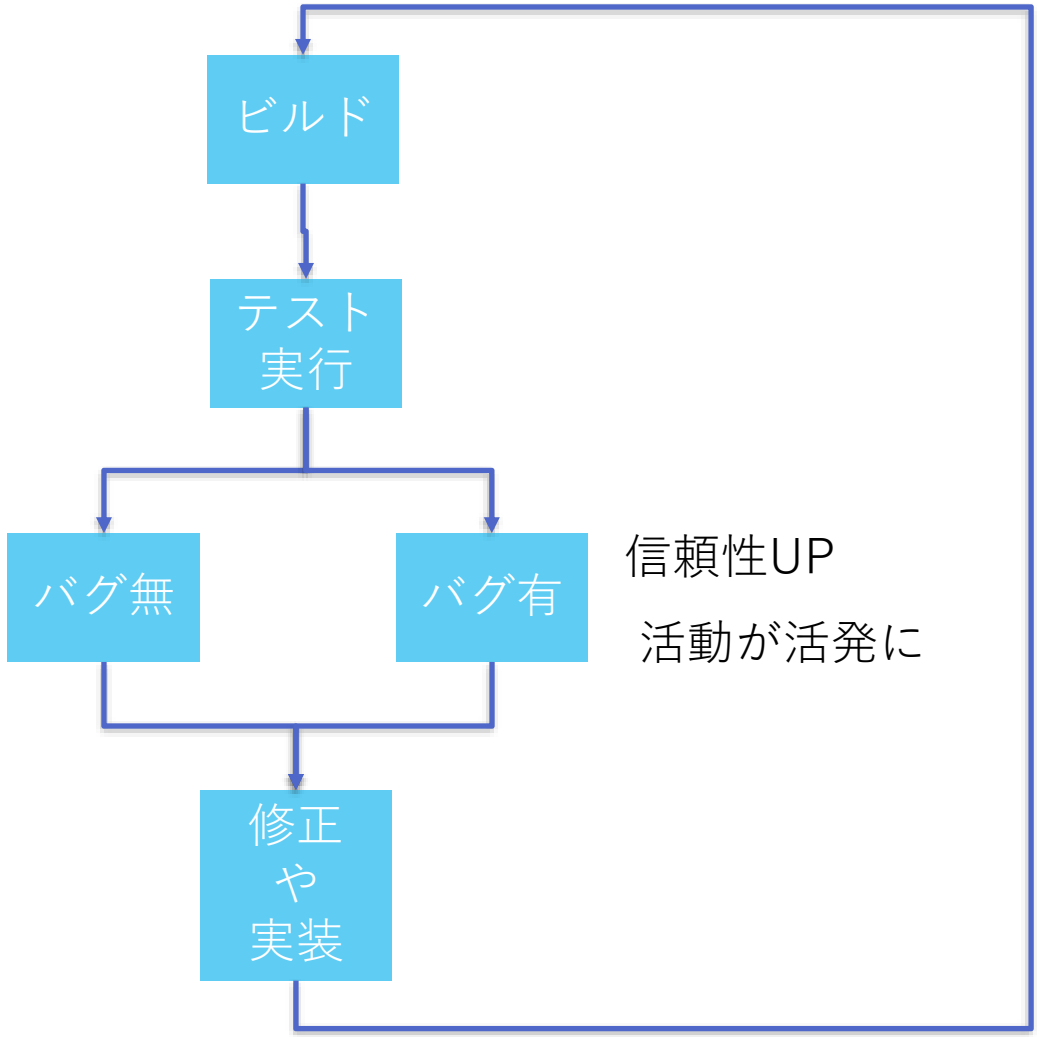
テストスクリプトを書くメンバーが面倒なことを意識しなくてもいいというところが普及のカギであると思っています

定期的な実行が大事。



コマンドベースのスクリプト	プログラムではなく製品の操作を書く
実行環境作成のワンタッチ化	パッケージマネージャーを使っている
製品バージョン	画面が変わる、操作が変わる、結果が変わる
BVT	**定期的に実行されていると安心感が得られる**
選択実行	作成中、調査では特定のスクリプトを繰り返し実行する

バグがなくても安心感を得られる



安心感

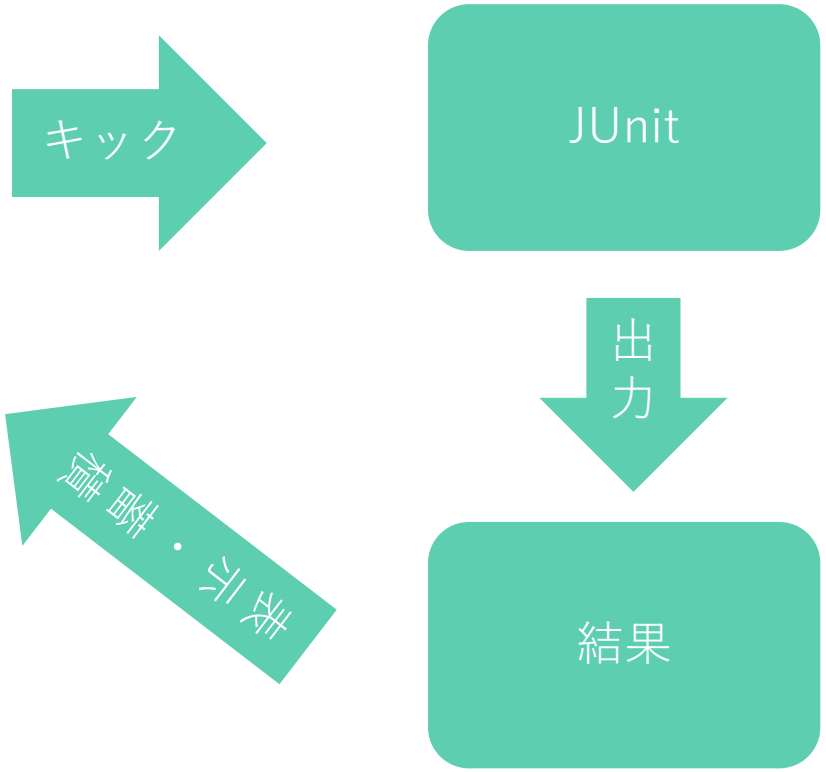
もっと回したい

信頼性UP

活動が活発に

月ごと、週ごと、日ごと、ビルドごとと頻度を増やすと安心感が高まります

バグが見つけられる実感を得られると、カバー範囲を広げたくなります

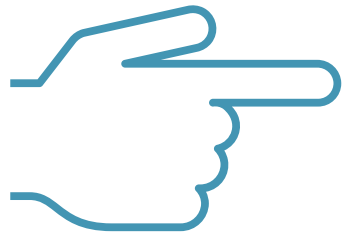


こんな感じの流れを
想定したツール選定
（や作りこみ）

じつは

- 従来テストフレームワークはBVTしてない
- 網羅率優先でやっていたら大きくなった
- いまは別のフレームワーク(APIレイヤー)でBVTしてる

必要なテストだけを実行。



コマンドベースのスクリプト	プログラムではなく製品の操作を書く
実行環境作成のワンタッチ化	パッケージマネージャーを使っている
製品バージョン	画面が変わる、操作が変わる、結果が変わる
BVT	定期的に実行されていると安心感が得られる
選択実行	作成中、調査では特定のスクリプトを繰り返し実行する

自動テストはいつも全部流すわけではない

大きすぎるテストセットの一部だけ実施

のちの分散実行

作成中のテストセットだけを実施

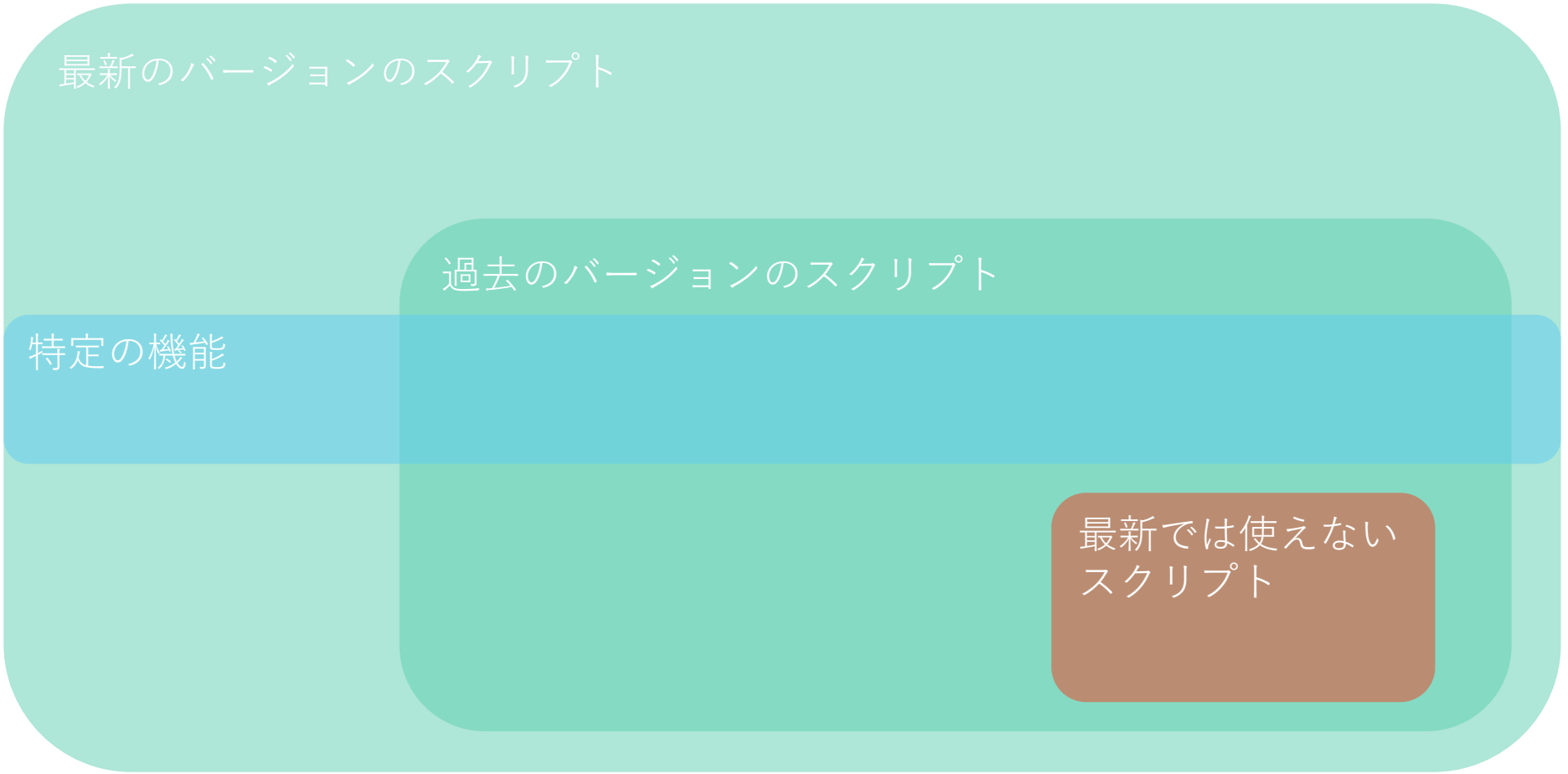
エラーや失敗になったテストセットだけを実施

できないと調査時間が

必要なバージョンのテストを実施

バージョン耐性の考慮

後述、と言っていたスクリプトのバージョン



各範囲や軸をタグやパッケージで分類することで必要なテストだけを実行する

テストランナーフレームワークによる違い

	テキスト(従来)	JUnit(新)
選択して実行	自作ツールを利用してスクリプトやディレクトリ単位で実行	Eclipseを利用してメソッドやクラス単位で実行
グループを作って実行	テキストとして保存しておく タグも利用可 オートタグなどの仕組みも構築しやすい	プリセットをEclipseに登録 タグによるグループ化
コマンドでの実行	作成済みテキストを指定	Maven testで上記の指定が可能

実行ツールを作成すれば自由度高く表現も好みにできるがメンテナンス(と覚悟)は必要
JUnitの範疇で使えば何も作りこまなくても実質やりたいことはできる

まとめ

信頼・安心・実績の積み上げ

バグ見つけて信頼性

小さくても定期実行

頻度上げて安心感

とっつきにくさ対策

コマンド化

環境ワンタッチ化

失敗時/巨大化時の考慮

調査時の個別実行

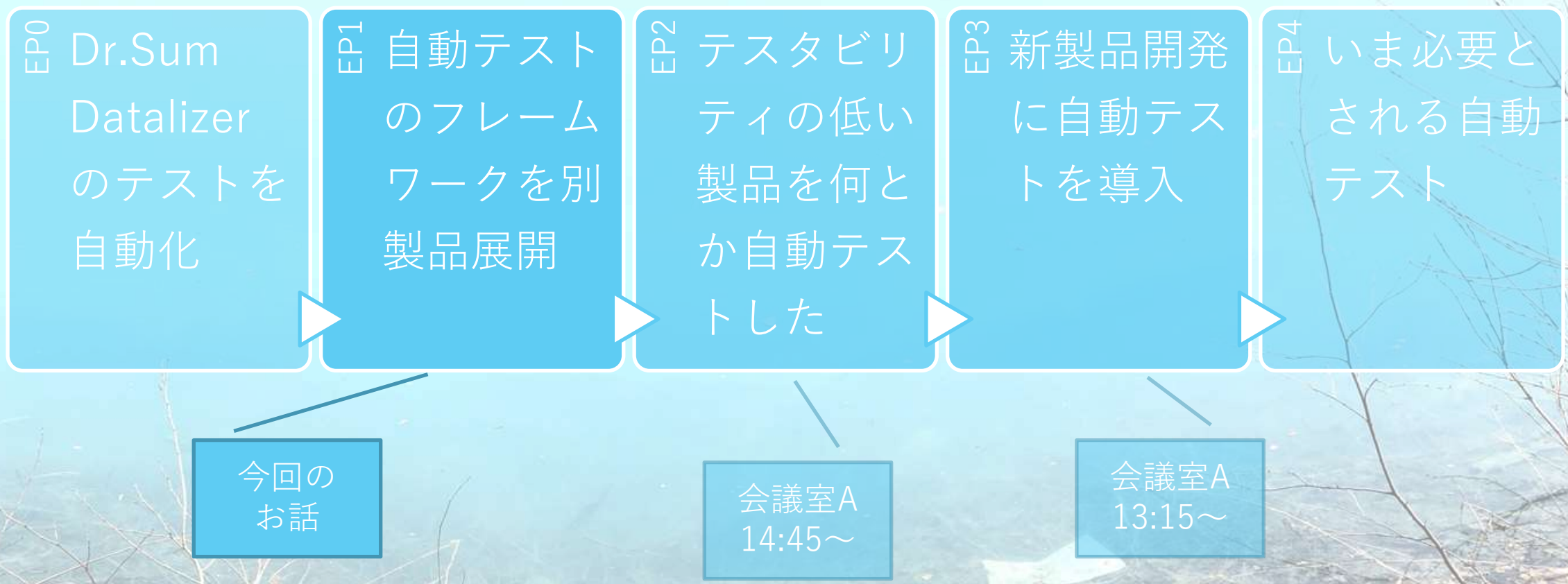
分散実行のための準備

VerUP耐性

クラス構成

ディレクトリ構成

ウイングアークの品質部門のAutomationTeamの歩み



Automationを含むさまざまな改善活動を行っています

「**medium wingarc**」で検索してみてください

The Data Empowerment Company

データに価値を、企業にイノベーションを。