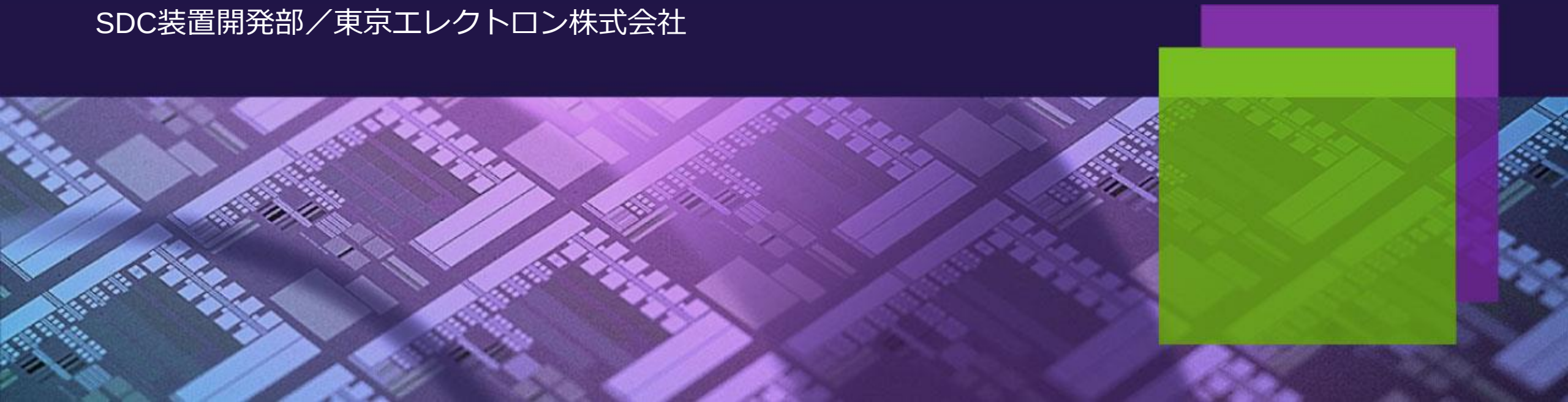
UMLクラス図記法を利用したテストケース仕様書のリライト
– 保守しやすいテストケースの塊 –

2020年3月

中岫　信

SDC装置開発部／東京エレクトロン株式会社



目次

1. 自己紹介
2. 背景
3. 課題と取り組み
4. 結果と考察
5. まとめ
6. 今後の課題、展望

背景

装置の初期開発から保守にいたるまで
開発期間が長く、その間に多くの機能追加、
機能改修が行われる

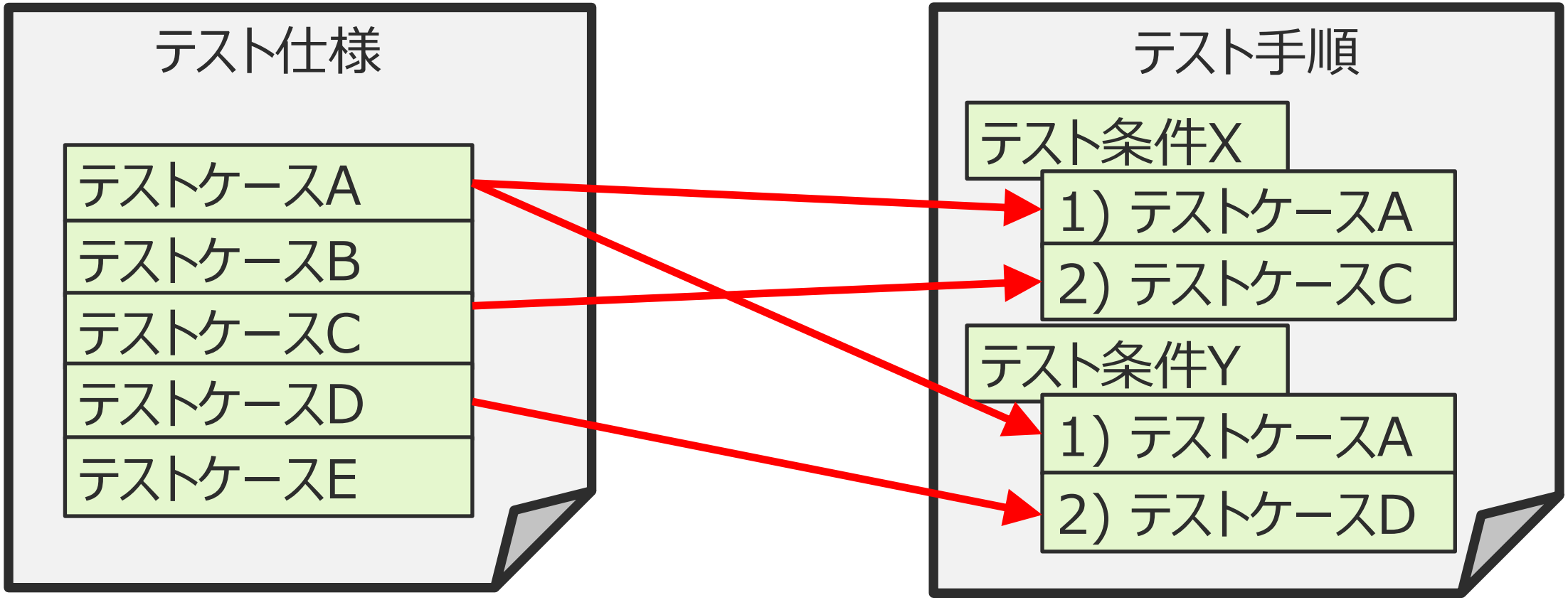
作成者によって書き方が異なっている

テストケースを保守することが
困難になっている。

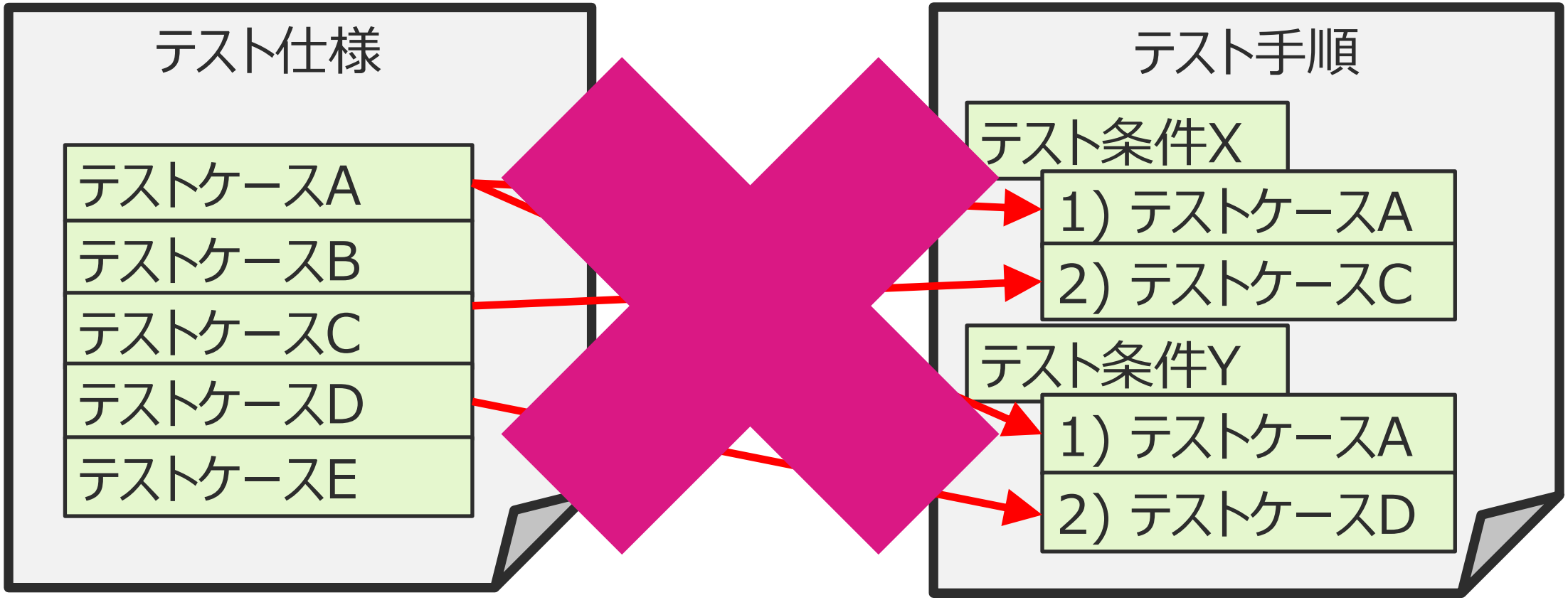
テストケースの取捨選択も
困難にする。

保守フェースに入ると…

理想としては…
テスト仕様とテスト実行手順は
区別して扱わなければいけない

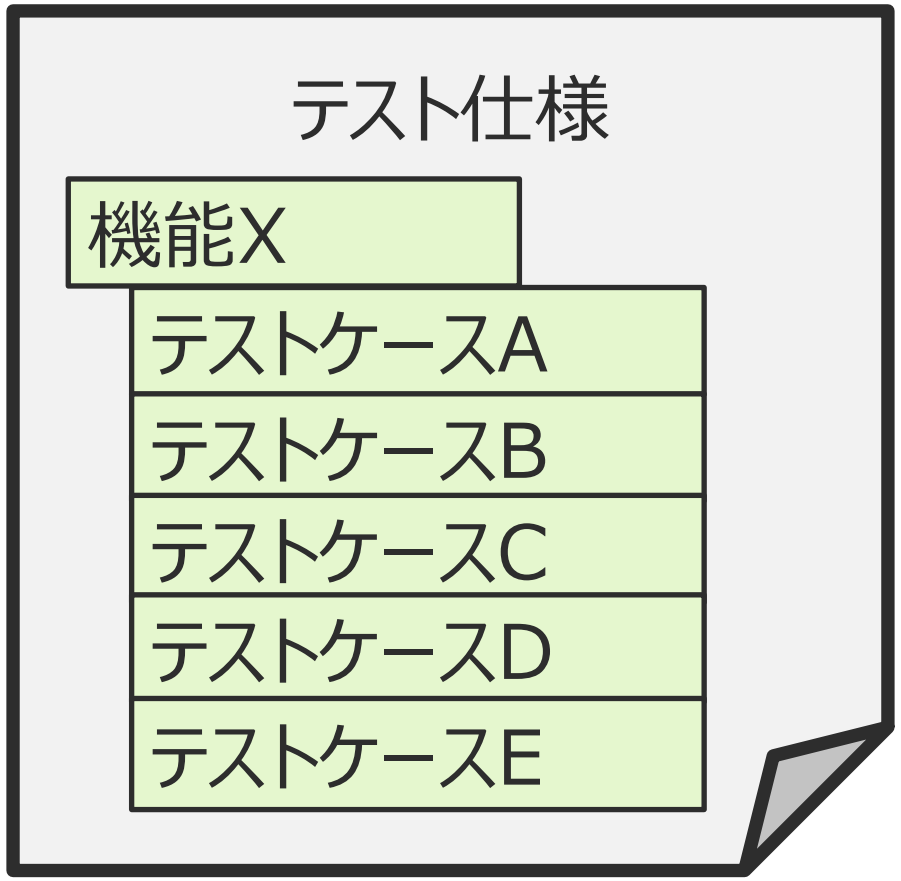


現実は…
テスト仕様とテスト実行手順は
区別されていない

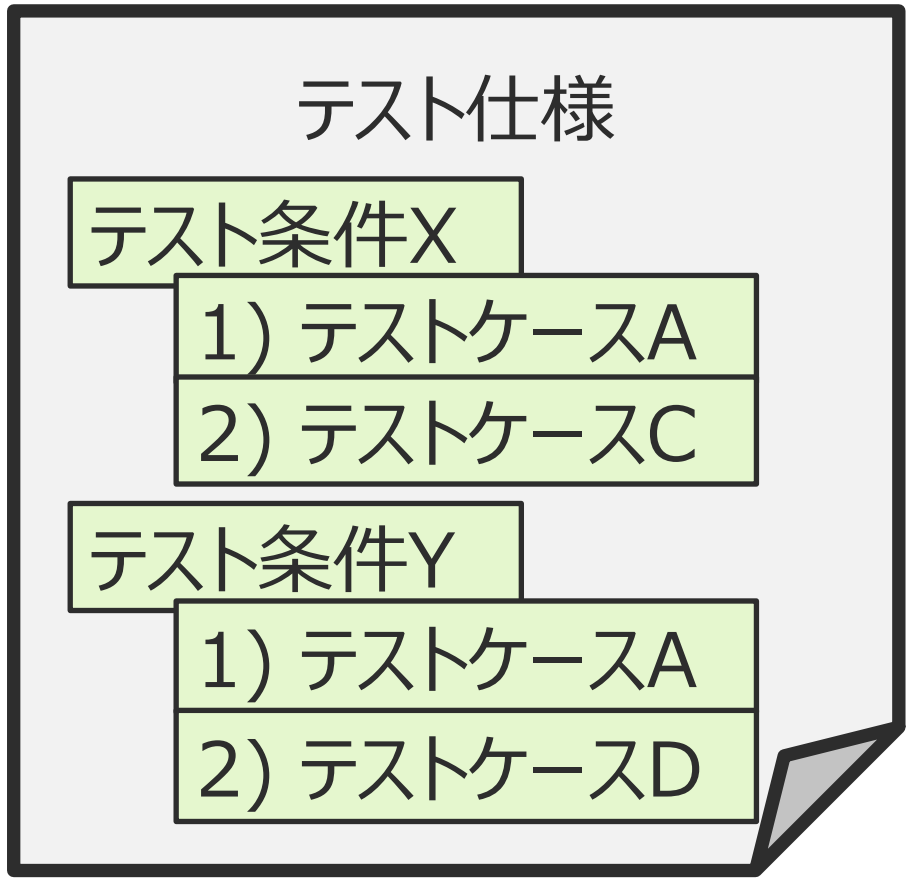


テスト仕様書の構成は…

機能単位でまとめたり



実行単位でまとめたり



実行効率型の問題

個別の機能に対して十分なテストが行えているかの把握が困難である
機能改修時にテストケースの追加・編集が困難である。

テストスイート1
対象機能：A、B、C

テストスイート2
対象機能：A、C

テストスイート3
対象機能：C、D

テストスイート4
対象機能：B、D

テストスイート5

機能Dのテストを
追加したい…

機能Bは
大丈夫か？

課題と取り組み

リライトの前提として

課題①
リライト後に既存のテストケースが
抜け落ちないようにする。

実行効率型の問題として

抽象的

具体的

テストケースの表現

レベル１：処理だけを特定したテストケース

レベル2：入出力の内容・件数まで表現したテストケース

レベル3：同値クラスを用いて組み合わせ論理条件（ペア）まで表現したテストケース

レベル4：実際の値まで具体化したテストケース

実行効率型は
レベル4のテストケースが階層を持たずに
散らばっている。

実行効率型の問題として

「実行効率型はレベル4のテストケースが階層を持たずに散らばっている。」ことで…
A) 個別の機能に対して十分なテストが行えているかの把握が困難。
B) 機能改修時に個別の機能に対するテストケースの追加・編集が困難。

課題②
どこにどのようなテストケースがあるかを
把握しやすくする。

課題

リライト時に
既存のテストケースが
抜け落ちないようにする。

どこにどのような
テストケースがあるかを
把握しやすくする。

抜け落ちないようにするために

1. 既存の実行効率型の各具体的なテストケースに対して、「テスト対象」と「テストケース名」を明記したラベルを貼る。
2. リライト後にラベルに関連付けられたテストケースが、テスト仕様書へ反映できていることを確認する。

1. テストケースへのラベル貼り

ラベルの説明

「テスト対象」

機能、ユースケース、など

「テストケース名」

テストの内容が分かるように命名する

記入例

データ表示チェックボックス

データAの表示／非表示の切り替えの確認

テストケース名の表現レベルとしてはレベル3相当になるようにする。

抽象的

レベル1：処理だけを特定したテストケース

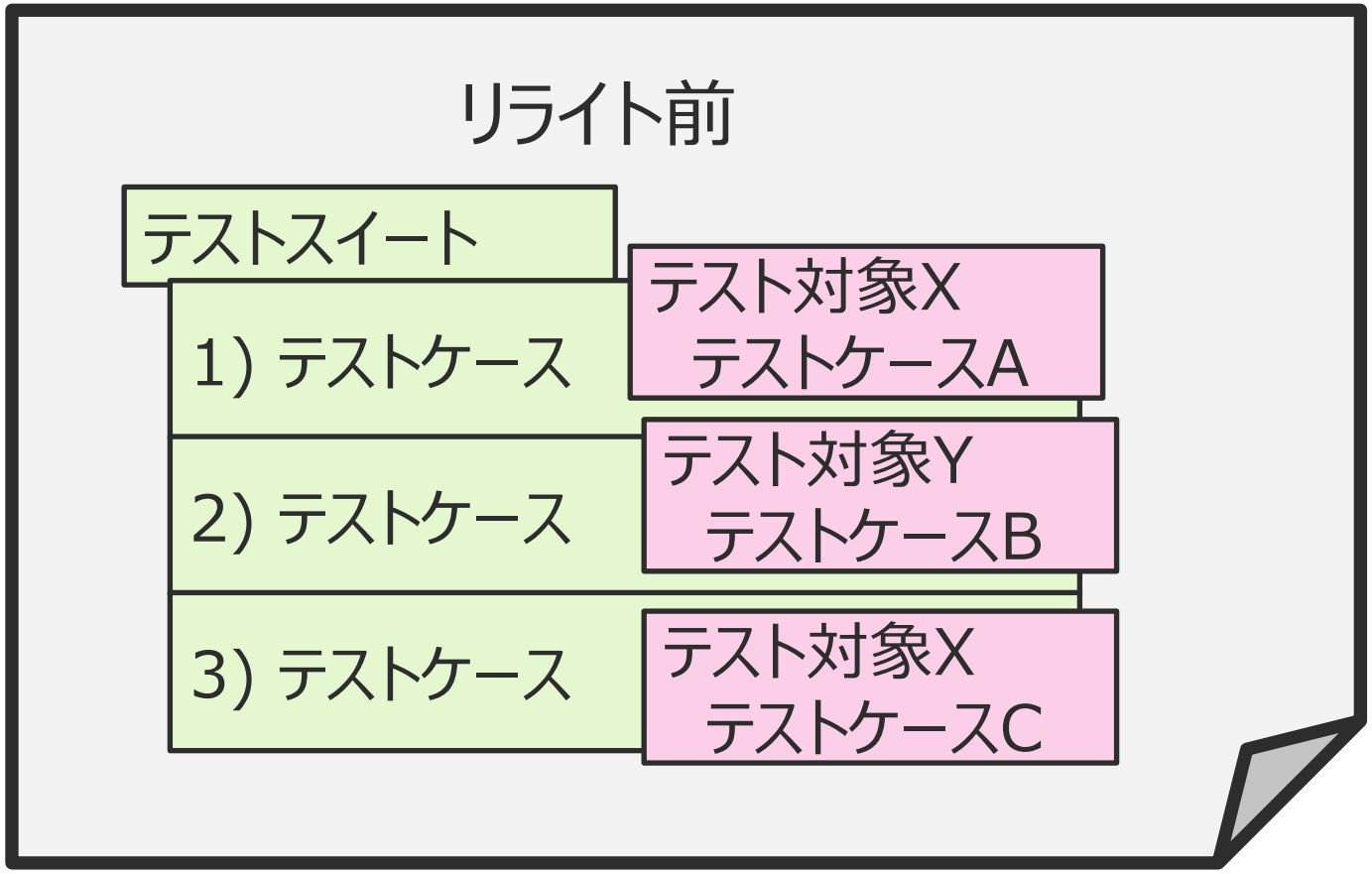
レベル2：入出力のパラメータ項目まで表現したテストケース

レベル3：同値クラスを用いて組み合わせ論理条件（ルール）まで表現したテストケース

レベル4：実際の値まで具体化したテストケース

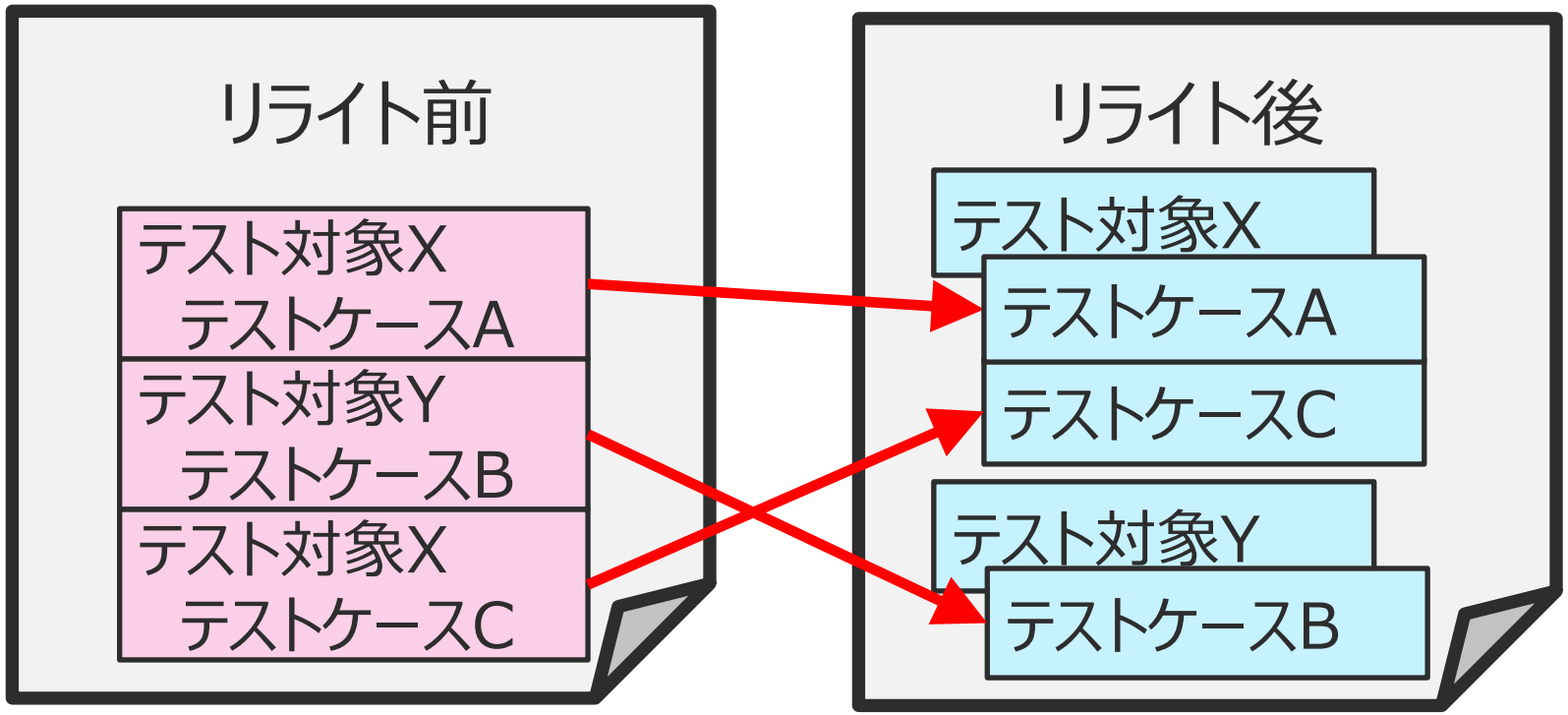
具体的

1. テストケースへのラベル貼り



2. テスト仕様書への反映の確認

リライト後に確認



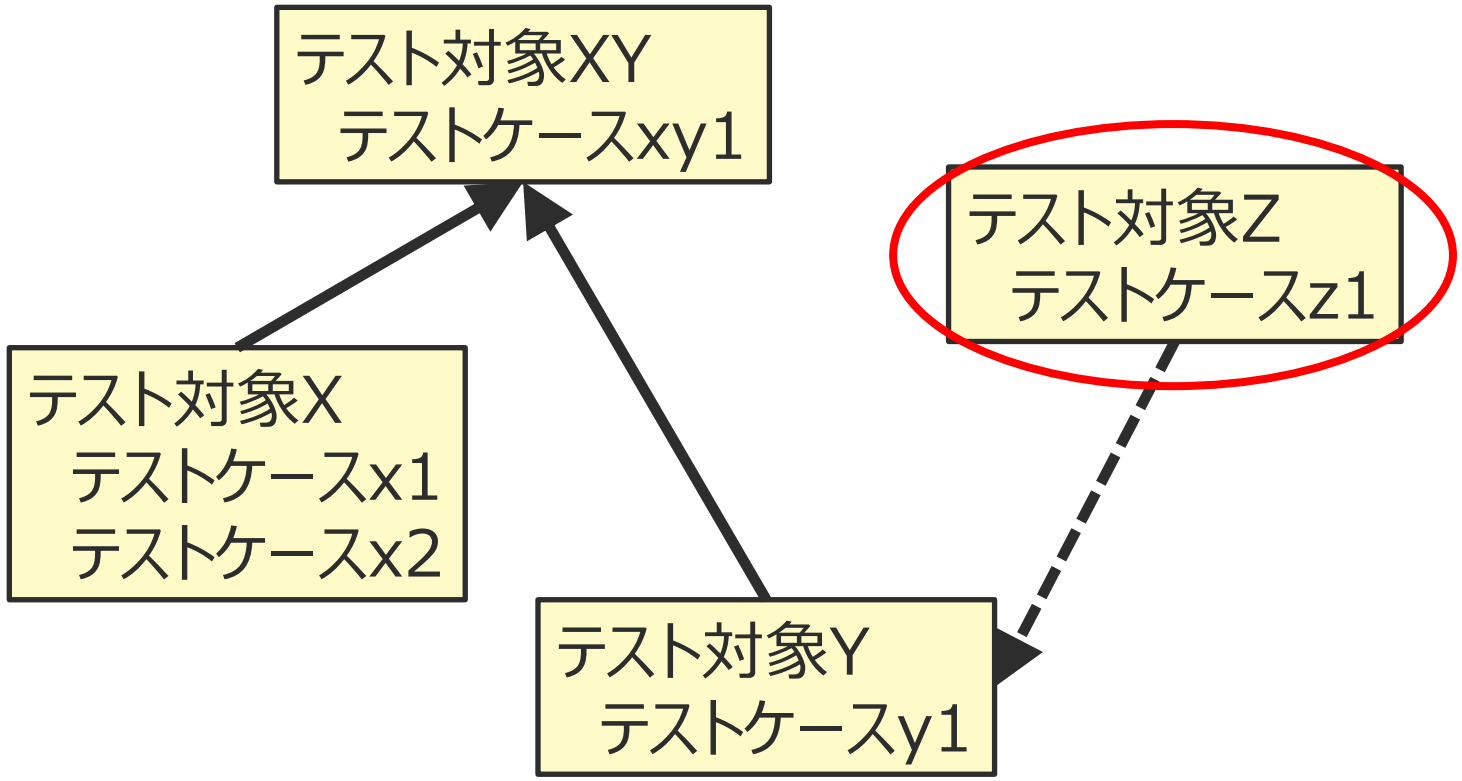
抜けなし！！

課題

リライト時に
既存のテストケースが
抜け落ちないようにする。

どこにどのような
テストケースがあるかを
把握しやすくする。

テストケースをモデル化して構成を可視化する



機能Zのテスト！

流れ

1. 既存の具体的なテストケースへのラベルの貼り付けを行う。
2. ラベルからテストケースの整理と構成の再構築を行う。
3. 再構築後に具体的なテストケースを貼り付ける。
4. リライト後のテスト仕様書に機能改修に対するテストケースを追加する。

1．ラベルの貼り付け

1	システムを起動して、運用時のロギングデータをグラフで可視化して確認できること。			
事前条件	・専用画面を操作する権限があるユーザが登録されている。 ・表示用データがA、B、Cがロギングされていること。 ・システムが待機状態であること。			
ステップ	内容	操作	期待結果	判定
1	システムの起動	システムを起動する。	・システムが起動すること。	
2	ログイン	専用画面を操作できるユーザでログインする。	・ログインすること。	
3	…			
4	データAの表示	データAのチェックボックスをOFFにする。	・画面の にすること	
5		データAのチェックボックスをONにする。	・画面の すること。	
:				

操作制限
操作制限の確認

データ表示チェックボックス
データAの表示／非表示の切り替えの確認

データ表示チェックボックス
データAの表示／非表示の切り替えの確認

内在していた問題への対処

イ） 複数のテストケースを一つのテストケースとして扱っている。
　→混在するテストケース分のラベルを複数貼る。
ロ） 同じテストケースが何度も登場する。
ハ） 対象機能としてはスコープ外のテストケースがある。
　→ラベルの整理する際に、重複や不要なラベルを削除する。
ニ） テストケース自体の粒度（抽象度）が異なっている。
ホ） テストレベルが異なっている。
　→テストケースの再構成の際に、粒度とレベルを調整する。

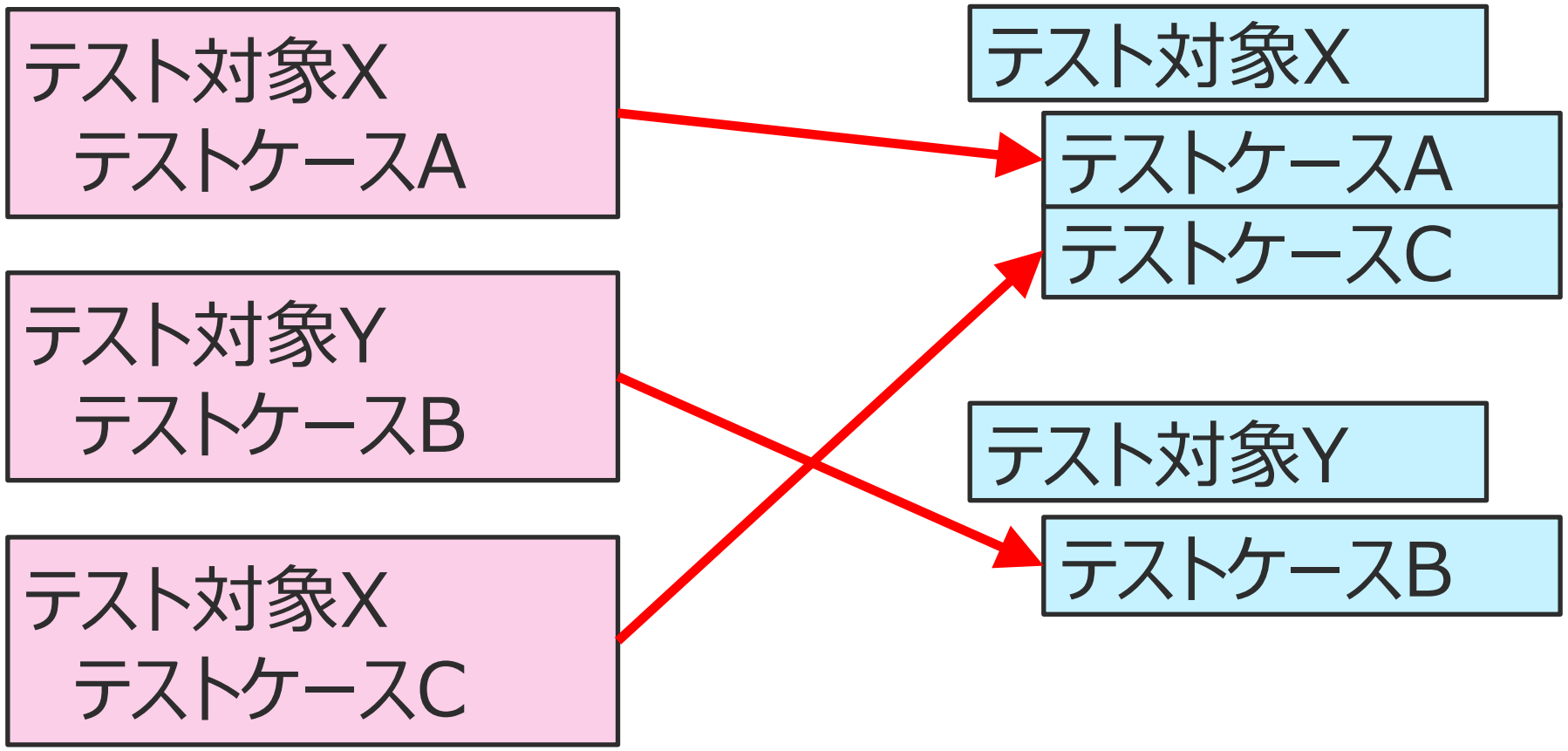
流れ

1. 既存の具体的なテストケースへのラベルの貼り付けを行う。
2. ラベルからテストケースの整理と構成の再構築を行う。
3. 再構築後に具体的なテストケースを貼り付ける。
4. リライト後のテスト仕様書に機能改修に対するテストケースを追加する。

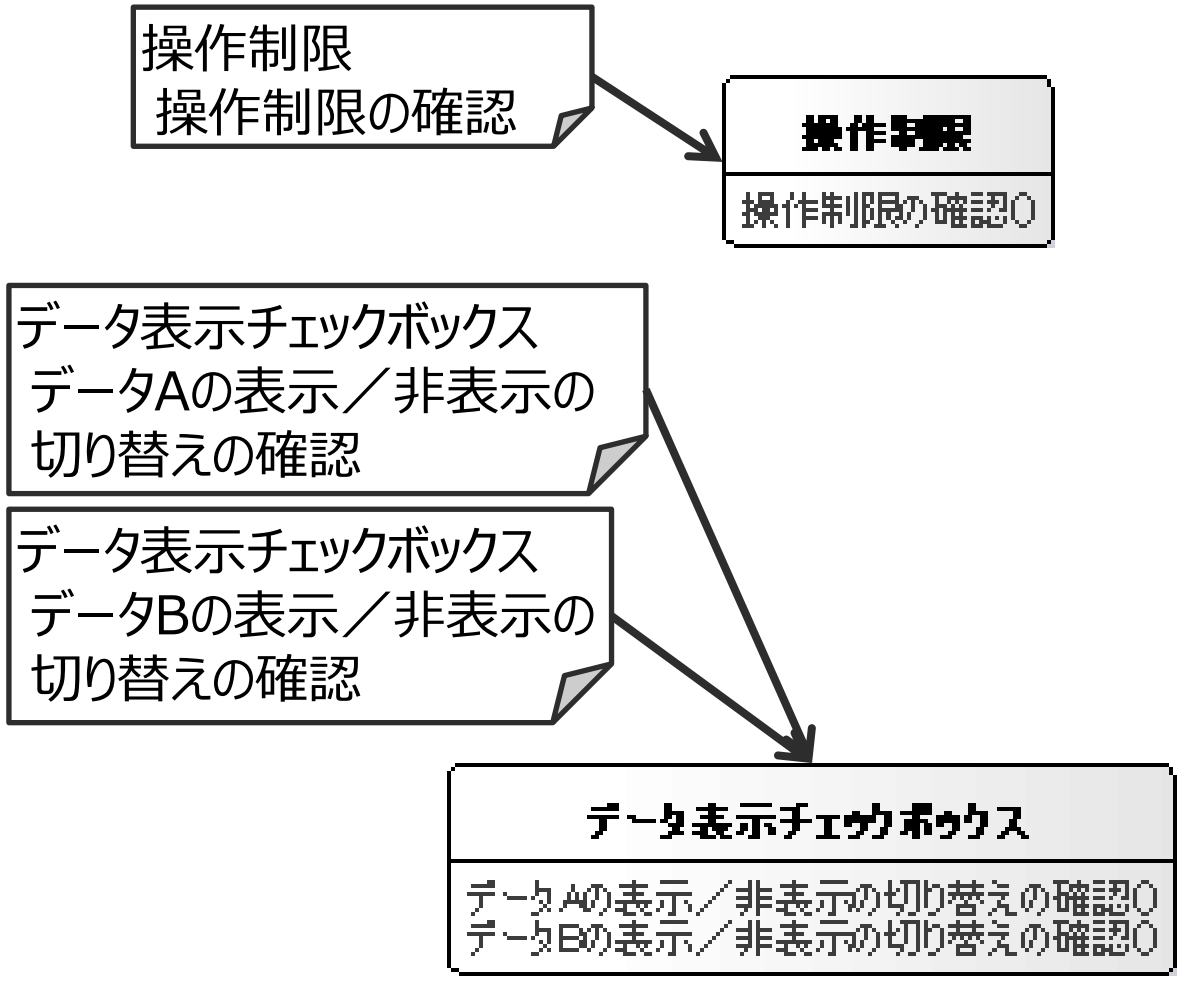
2. モデル化＆再構築

テストケースの整理と構成の再構築の流れ
1. ラベルをテスト対象ごとグルーピングする。
2. グルーピング後は、テストカタマリーの記法にあてはめる。

2-1. ラベルをテスト対象ごとグルーピングする

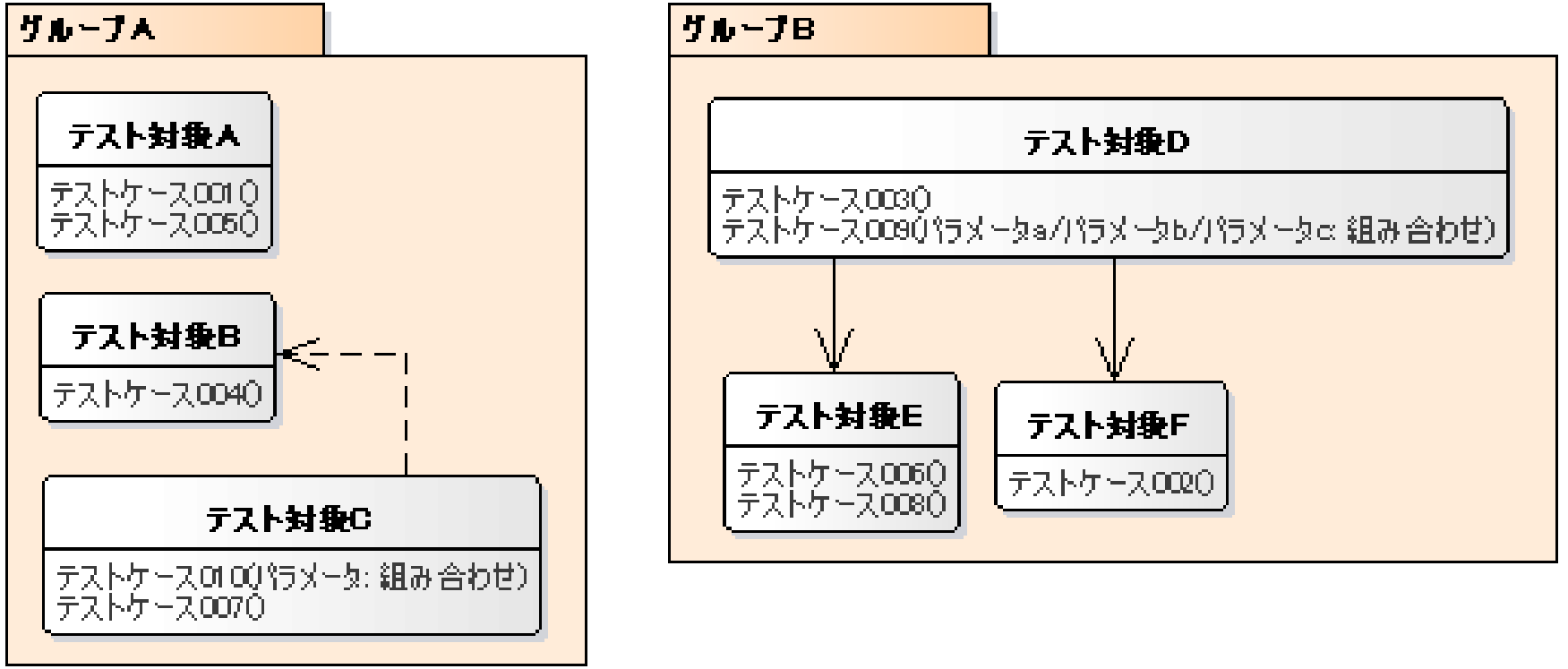


2-2. テストカタマリーの記法にあてはめる



- モデル図を俯瞰してみることで、テストケース名の表現や粒度が異なるものがないかを確認し、異なっている箇所は表現や粒度の修正を行う。
- テスト対象の仕様との比較やテスト条件の組み合わせに対して不足しているテストケースがあった場合、テストケースを補充する。
- 関連のあるテスト対象との関係を関係線、依存線を利用して図示する。

2-2. テストカタマリーの記法にあてはめる

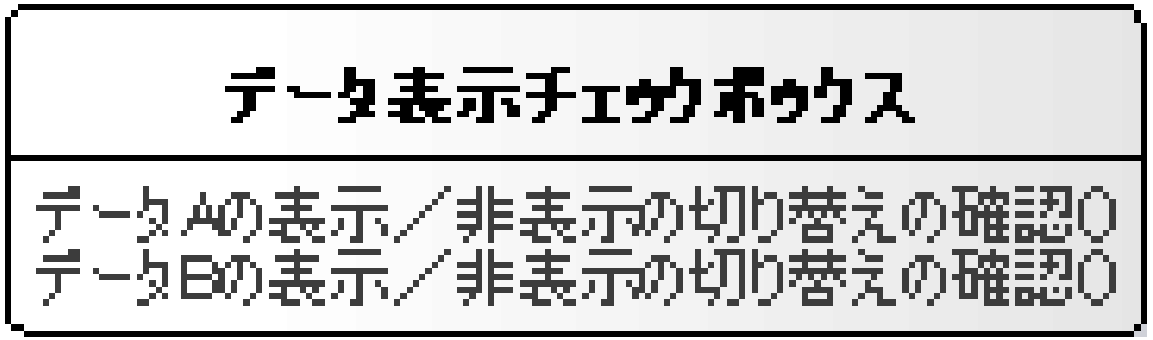


テスト対象A：独立したテストケース
テスト対処B・C：テスト対象Cはテスト対象Bの結果の影響を受ける
テスト対象D・E・F：テスト対象Dはテスト対象Eとテスト対象Fの統合テスト

流れ

1. 既存の具体的なテストケースへのラベルの貼り付けを行う。
2. ラベルからテストケースの整理と構成の再構築を行う。
3. 再構築後に具体的なテストケースを貼り付ける。
4. リライト後のテスト仕様書に機能改修に対するテストケースを追加する。

3. 具体的テストケースの貼り付け



データ表示チェックボックスの確認

ケース名	ステップ	操作	期待結果	判定
データAの表示	1	専用画面を起動する。	データAを表示すること。	Pass/Fail
	2	データAチェックボックスをOFFにする。	データAを非表示にすること。	Pass/Fail
	:			
データBの表示	1	専用画面を起動する。	データBを表示すること。	Pass/Fail
	2	データBチェックボックスをOFFにする。	データBを非表示にすること。	Pass/Fail
	:			

流れ

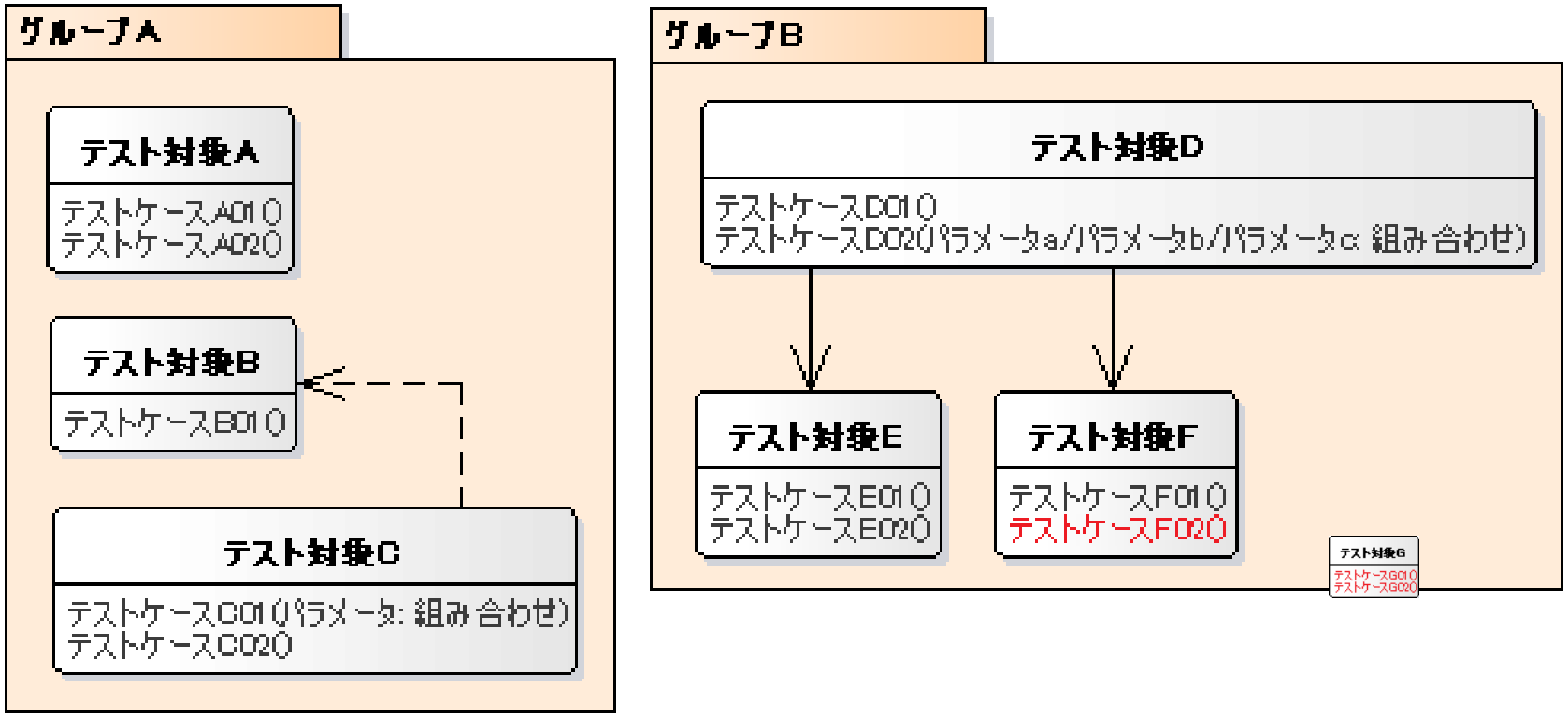
1. 既存の具体的なテストケースへのラベルの貼り付けを行う。
2. ラベルからテストケースの整理と構成の再構築を行う。
3. 再構築後に具体的なテストケースを貼り付ける。
4. リライト後のテスト仕様書に機能改修に対するテストケースを追加する。

4. テストケースの追加対応

機能改修に対するテストケースの追加・編集

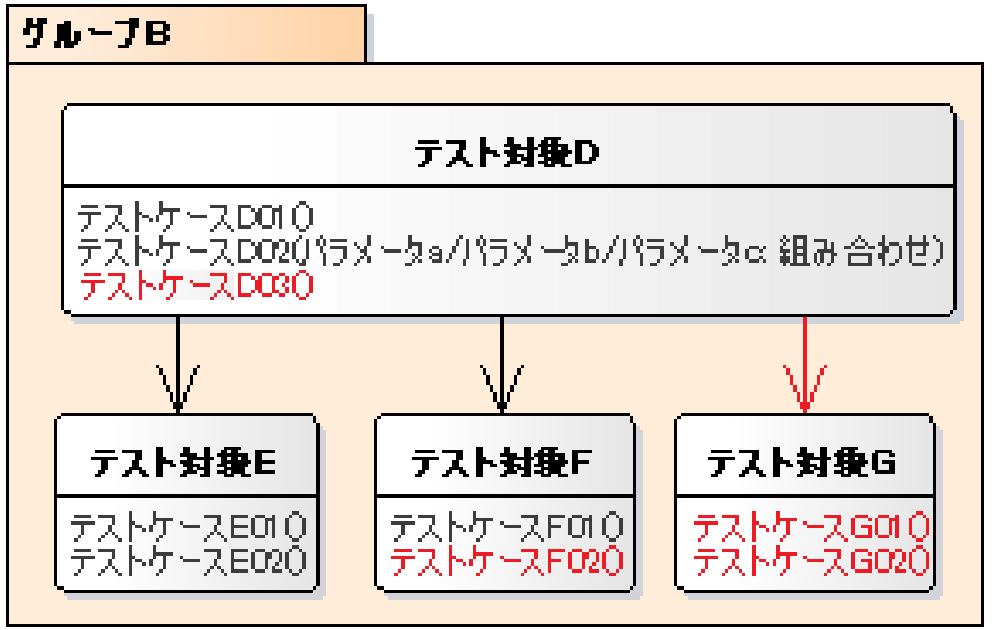
1. 機能改修によるテストケースの追加を行う。テスト対象が追加になっている場合、新たなテストカタマリーを追加する。
2. 追加したテストカタマリーが関係するテスト対象との関連を図示し、関連先のテストカタマリーに抽象的なテストケースを追加する。
3. 追加した抽象的なテストケースを具体的なテストケースに落とし込む。

4-1. テストケース&テスト対象の追加



テスト対象F：テストケースF02を追加
テスト対処G：新規テスト対象。テストケースG01、G02を持つ。

4-2. テストカタマリーに抽象的テストケースを追加



テスト対象D：テスト対象Gは、テスト対象E、Fと関連があるため、
テスト対象Dで統合テストのケースを追加

4-3. 具体的なテストケースに落とし込む

データ表示チェックボックス
データAの表示／非表示の切り替えの確認 データBの表示／非表示の切り替えの確認

データ表示チェックボックスの確認

ケース名	ステップ	操作	期待結果	判定
データAの表示	1	専用画面を起動する。	データAを表示すること。	Pass/Fail
	2	データAチェックボックスをOFFにする。	データAを非表示にすること。	Pass/Fail
	:			

結果と考察

取り組み評価

a. 既存のテストケースが漏れないようにする。
→リライト後のテストケースの既存のテストケースに対する網羅率で評価する。

b. どこにどのようなテストケースがあるかを把握しやすくする。
→関係者による定性的な評価を行う。

既存テストケースの網羅率

- 既存のラベルに関連付けられた具体的テストケースがリライト後のテスト仕様書に実装されている割合。
- 不要テストケースを除く具体的テストケースを用いる。

既存テストケースの網羅率

a.　既存のテストケースが漏れないようにする。
→リライト後のテストケースの既存のテストケースに対する網羅率で評価する。

網羅率は100%
リライトによるテストケースの漏れはない

関係者による定性的な評価

関係者は次の各1名

全員ドメイン知見者

- テスト設計者
- レビューア
- テスト実行者

関係者による定性的な評価

評価時の質問

- テストケース保守性はどのようになったか？
- 実行しやすさに変化はあったか？

テストケース保守性はどのようになったか？

- どこにどのようなテストケースを追加するかがわかりやすい。
- 実施するテストケースの取捨選択が容易になっている。
- 変更内容のレビューが容易で、ケース漏れに気づきやすい。

実行しやすさに変化はあったか？

- テストケースの粒度が揃っているため、作業工数が見積もりやすい。
- 同じ操作が、何度も登場してテストの実行効率は下がったと感じる。

結果は

b. どこにどのようなテストケースがあるかを把握しやすくする。
→関係者による定性的な評価を行う。

実行効率が下がったという評価はあるが、
テストケースの所在が把握しやすくなった

リライト版テスト仕様書に対する考察

重複、不要、不足のテストケースの数から、
リライト後のテスト仕様書を評価する。

タイプ	説明	リライト後の影響
混在テストケース	一つのテストケースで複数の確認が行われているテストケース。	1ケース1意にするため、テストケース数は増加になる。
重複テストケース	目的も確認内容も同一のテストケース。	余分なテストケースのため、テストケース数は減少となる。
不要テストケース	対象のスコープ外のテストケース。	余分なテストケースのため、テストケース数は減少となる。
不足テストケース	実装されていない必要なテストケース。	追加実装となるため、テストケース数は増加になる。

リライト版テスト仕様書に対する考察

リライト前後の1機能あたりの具体的なテストケース数の増減を比較

	Before	After
単機能テスト		
機能グループA	1.80	2.20
機能グループB	2.56	2.94
機能グループC	3.42	3.17
機能統合テスト	0.75	1.00

リライト版テスト仕様書に対する考察

	Before	After
単機能テスト		
機能グループA	1.80	2.20
機能グループB	2.56	2.94
機能グループC	3.42	3.17
機能統合テスト	0.75	1.00

機能グループCのリライト後のテストケース数が減少

リライト版テスト仕様書に対する考察

リライト前のテスト仕様書

- 混在、重複等のテストケースが存在する理由
  →度重なる機能改修などで適切にテストケースの追加・編集が出来なかった

リライト後のテスト仕様書

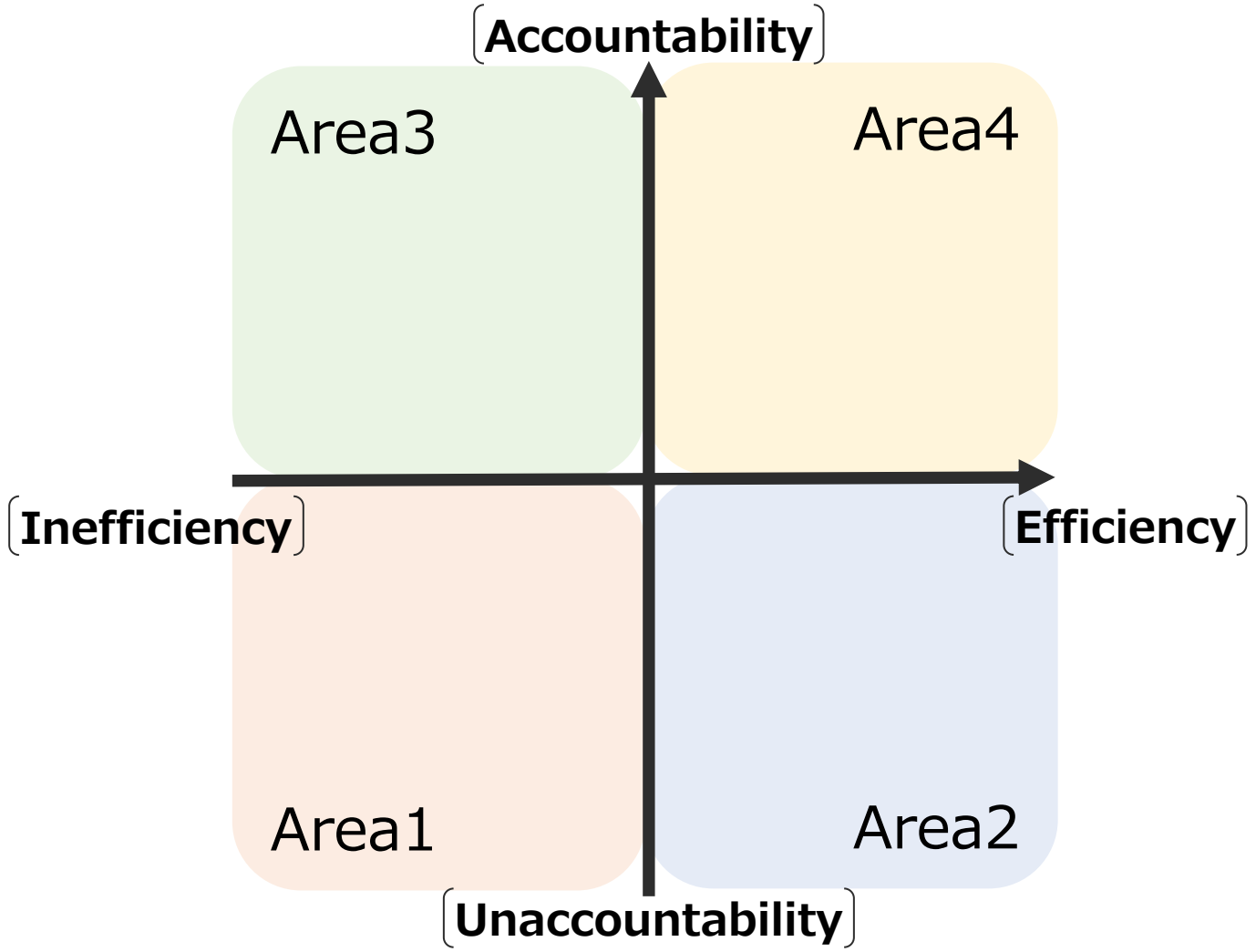
- テストケースでは無駄なテストケースが排除でき
  →純粋に仕様に対するテストケースだけを実装

まとめ

テスト仕様書の構成の変化

ポジショニングマップ

- テストの実行しやすさ（Execution Efficiency）
- テストの目的に対してどのようなテストを実行しているか説明しやすさ（Accountability）

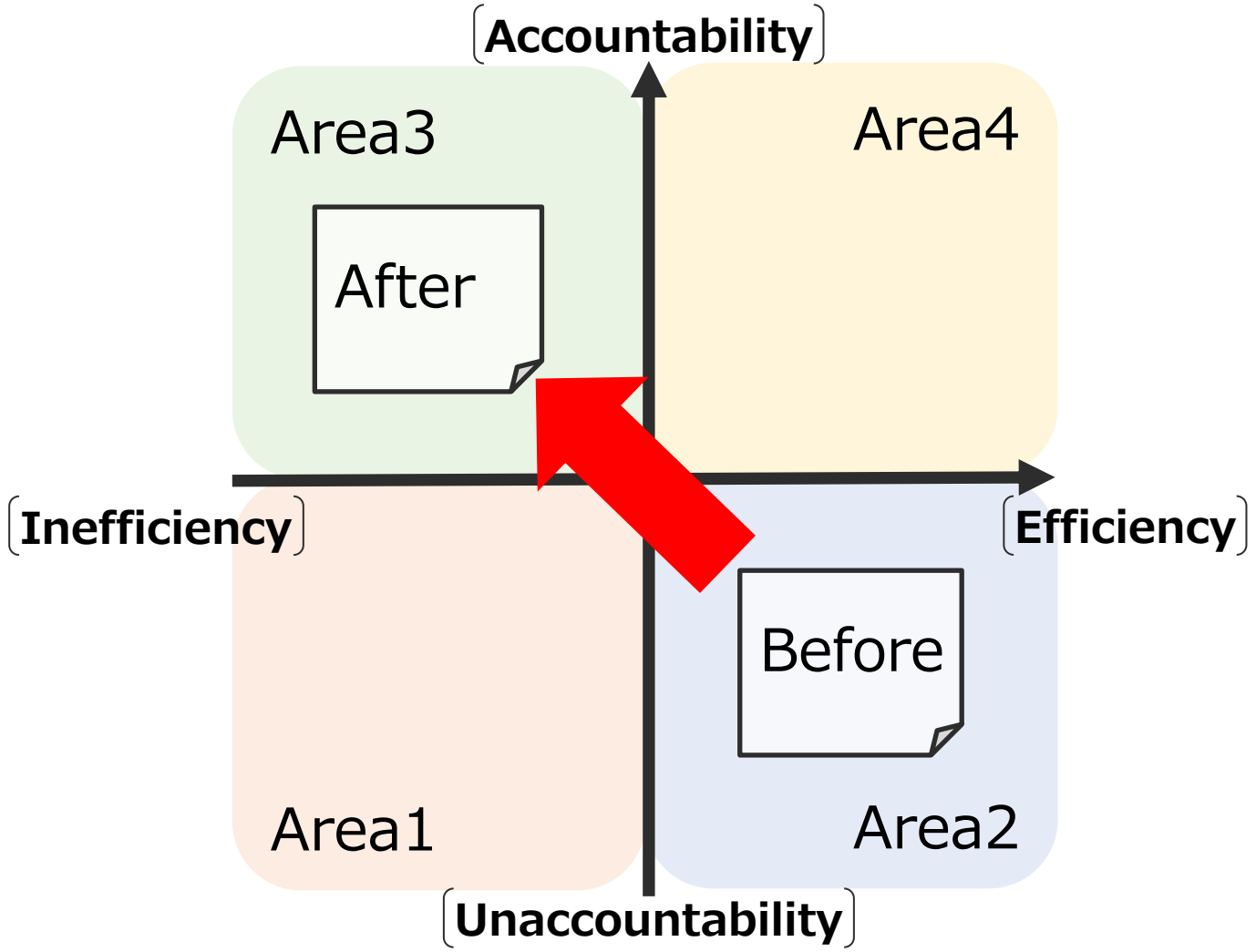


テスト仕様書の構成の変化

【リライト前】
実行しやすさに着目　（実行効率型）

【リライト後】
テスト対象とテストケースの関連が明確

Accountabilityは向上したが、
Efficiencyは下がったと言える。



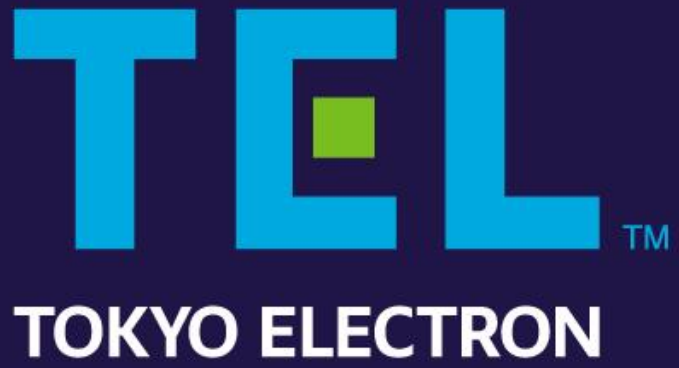
テストケースの構成を可視化することで

- テストの目的に対応したテストケースの所在が把握しやすくなる。
- テストケースの追加・編集が容易になる。
- 機能関連により必要なテストケースの取捨選択が容易になる。
- テストケースの重複、漏れに気付きやすく、レビューがしやすくなる。
- テストケースの粒度が整うことにより、実行時の工数が見積もりやすくなる。

今後の課題、展望

今後の課題、展望

- Accountability重視の対応となったが、Efficiencyも考慮する必要があると考えている。今後は、AccountabilityとEfficiencyを両立できるアプローチも考えたい。
- 他にも今回のような実行効率型のテスト仕様書が多数存在する。機能の改修が入りやすく影響が大きいものから、今回と同様のアプローチでテスト仕様書のリライトを行いたい。
- テストコードに実装されているテストケースとの重複があることがわかっているため、テストコードも同様に整理することで重複を排除できると考えている。
- テスト設計できるメンバーが不足しているため、関係者が今回のような整理ができるよう技術の横展開を行う必要がある。


TEL
TM
TOKYO ELECTRON