アジャイルなテストの見積もりと計画づくり

JaSST 12 Tokai
2012.11.30

presented by きょん(@kyon_mm)

自己紹介

- なまえ : きょん@kyon_mm
- 対象 :

  開発環境改善

  Groovy, SCM, Test, Agile, CD, 関数/証明プログラミング
- Study

  SCMBootCamp, Nagoya.Testing, CDStudy, Cafe.Testing, TDDBootCamp Cafe.Testing

Attention!

- これは私の独断と偏見と体験です。所属する組織、コミュニティの意見ではございません。

What's Agile?

- アジャイルはソフトウェア開発スタイルであってプロセスではない!
- アジャイルの基礎は「アジャイル宣言」と「アジャイルの12の原則」
- Haskellが関数プログラミングスタイルの1つの実装であるように、

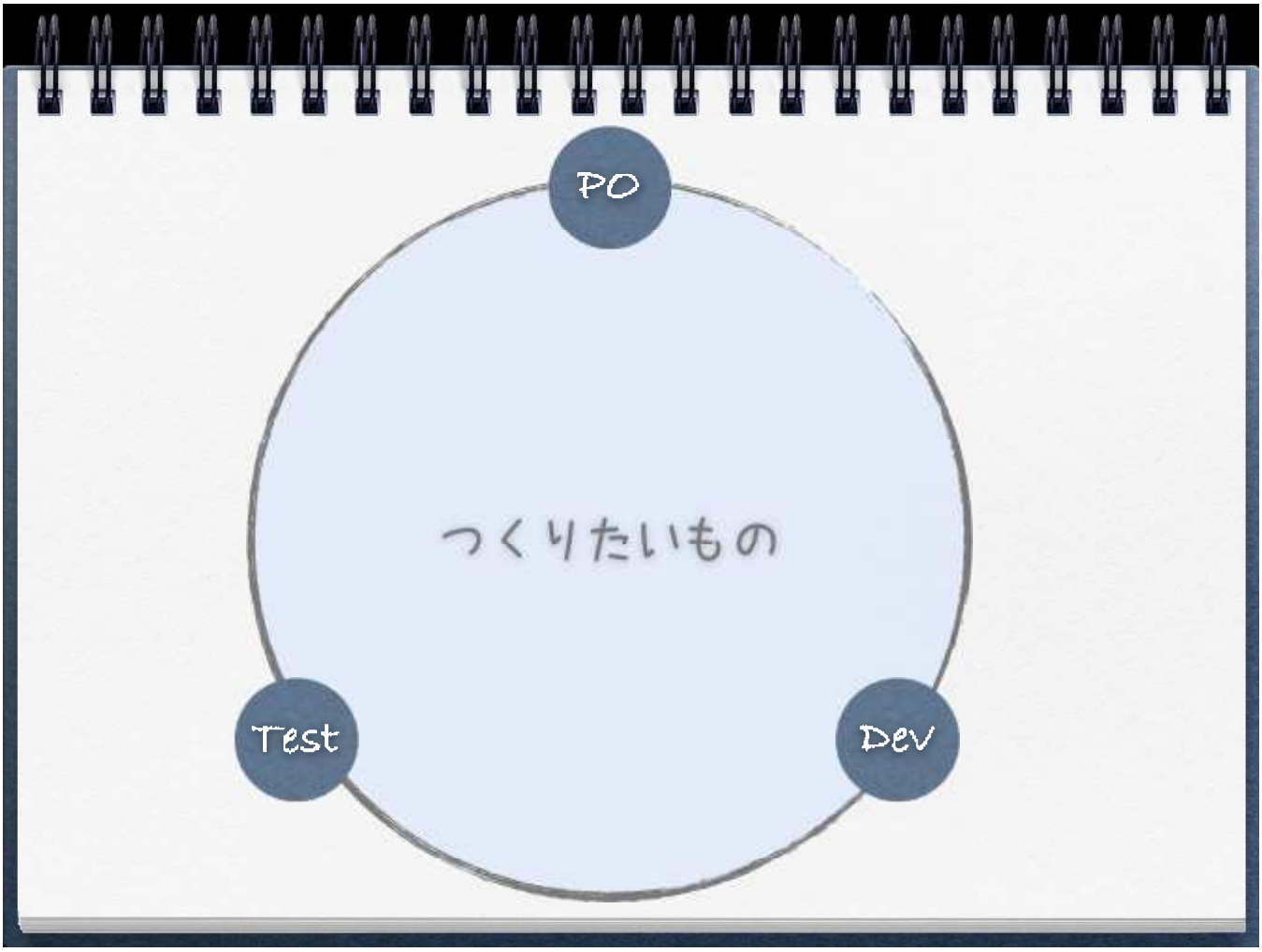
  XP, Scrum, Lean , etc はアジャイルの実装の1つである。

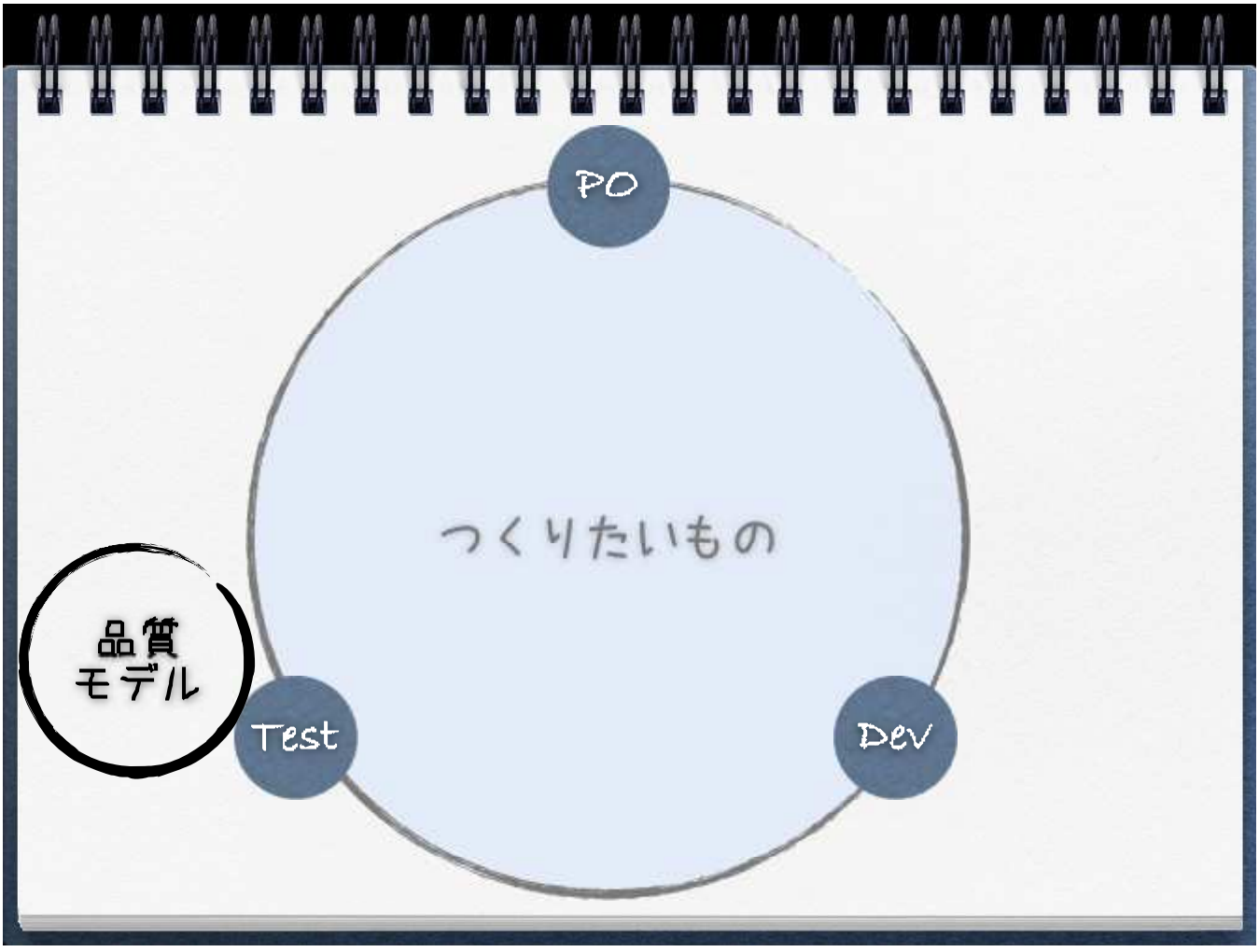
BackGround

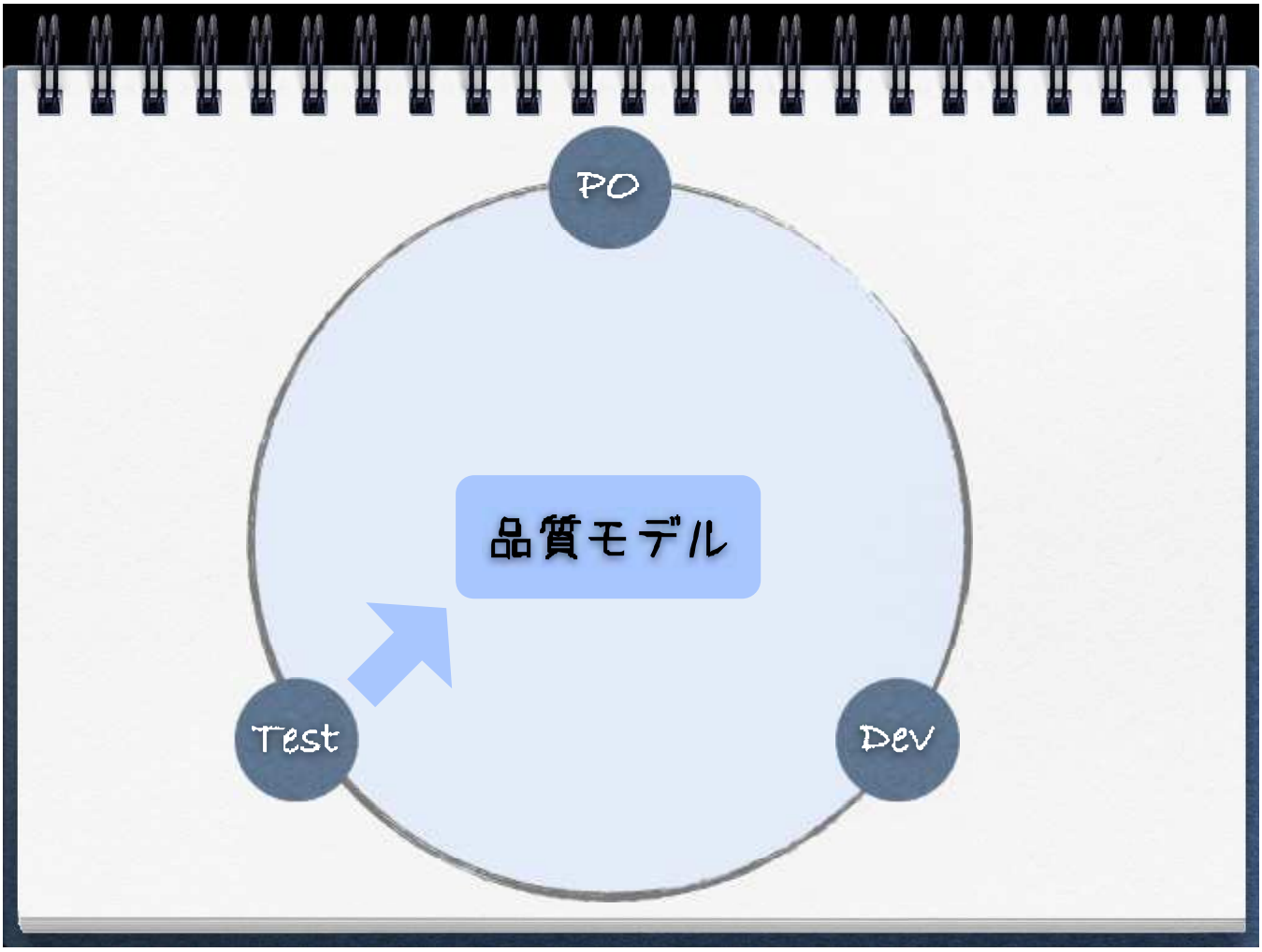
- テストエンジニア一年生！
- GUIのないWebアプリでサーバーサイド
  スマートフォンアプリ
  Web用のフレームワーク
  ライブラリ

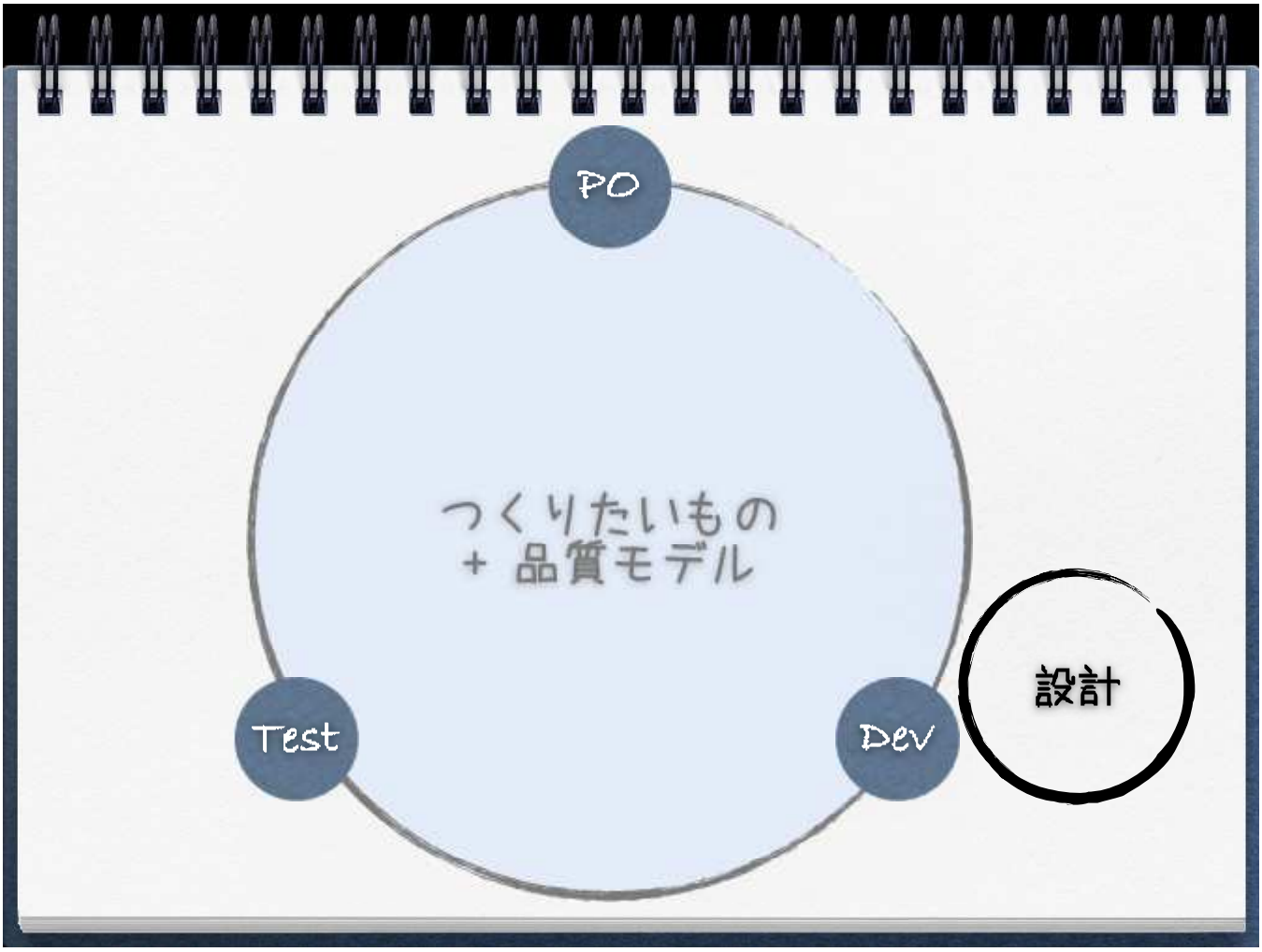
Problem

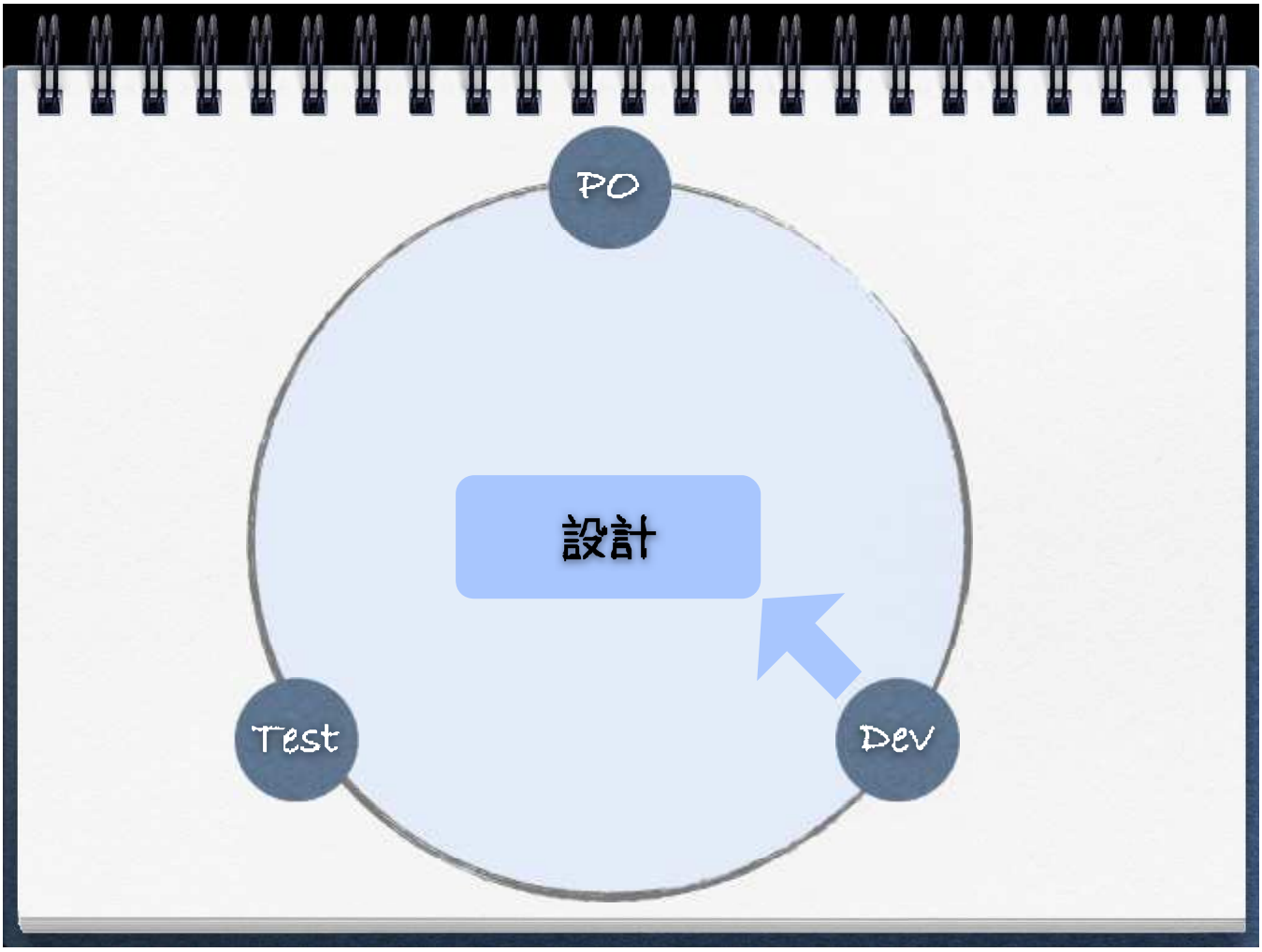
- テスト観点ってなに？
- テストの見積もりが難しい。。。
- 品質ってなに？
- テストはどうやって区切るの？
- どこまでテストすればいいの？


PO
つくりたいもの
Test
Dev

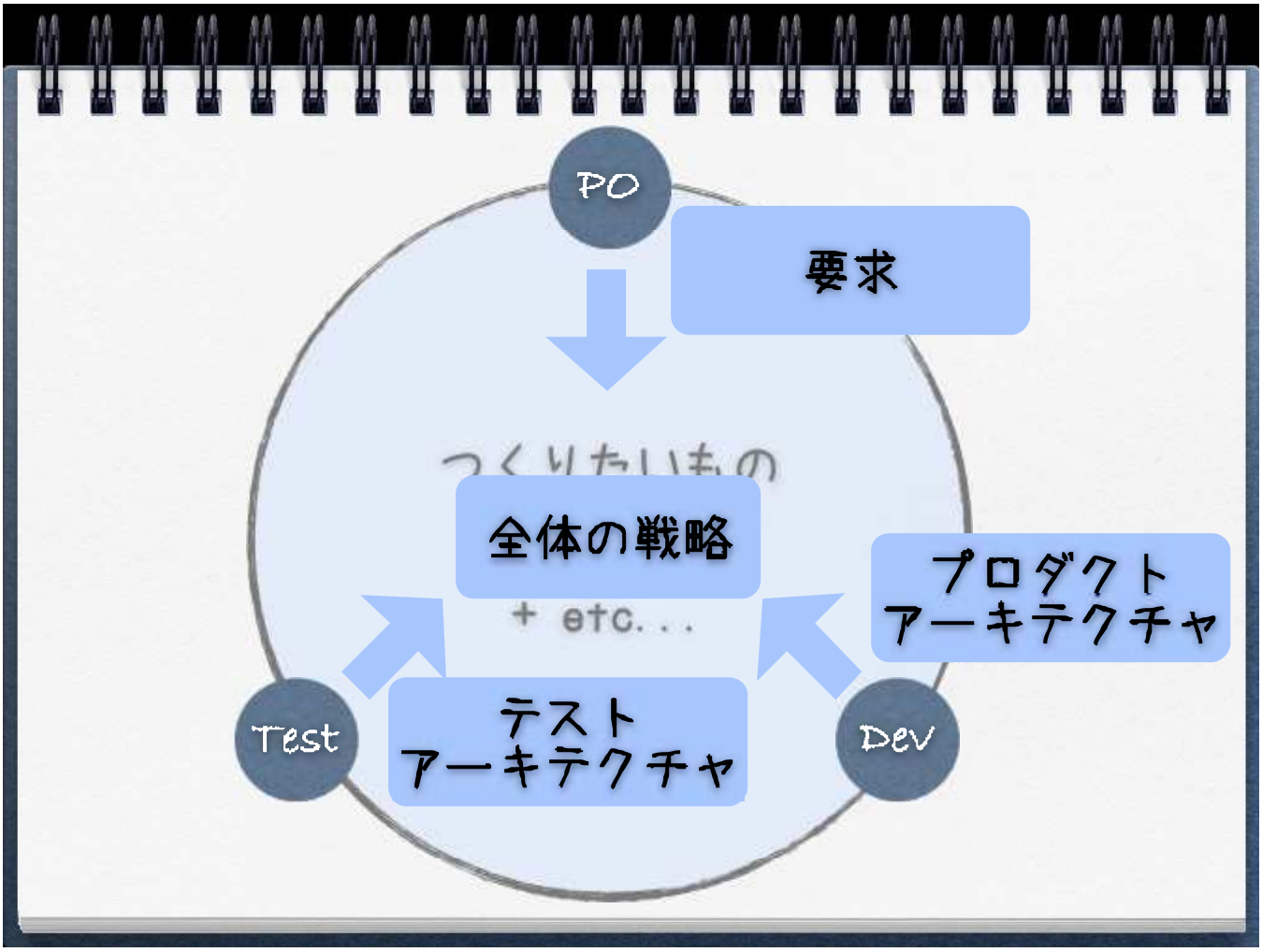

PO
つくりたいもの
品質
モデル
Test
Dev


PO
品質モデル
Test
Dev

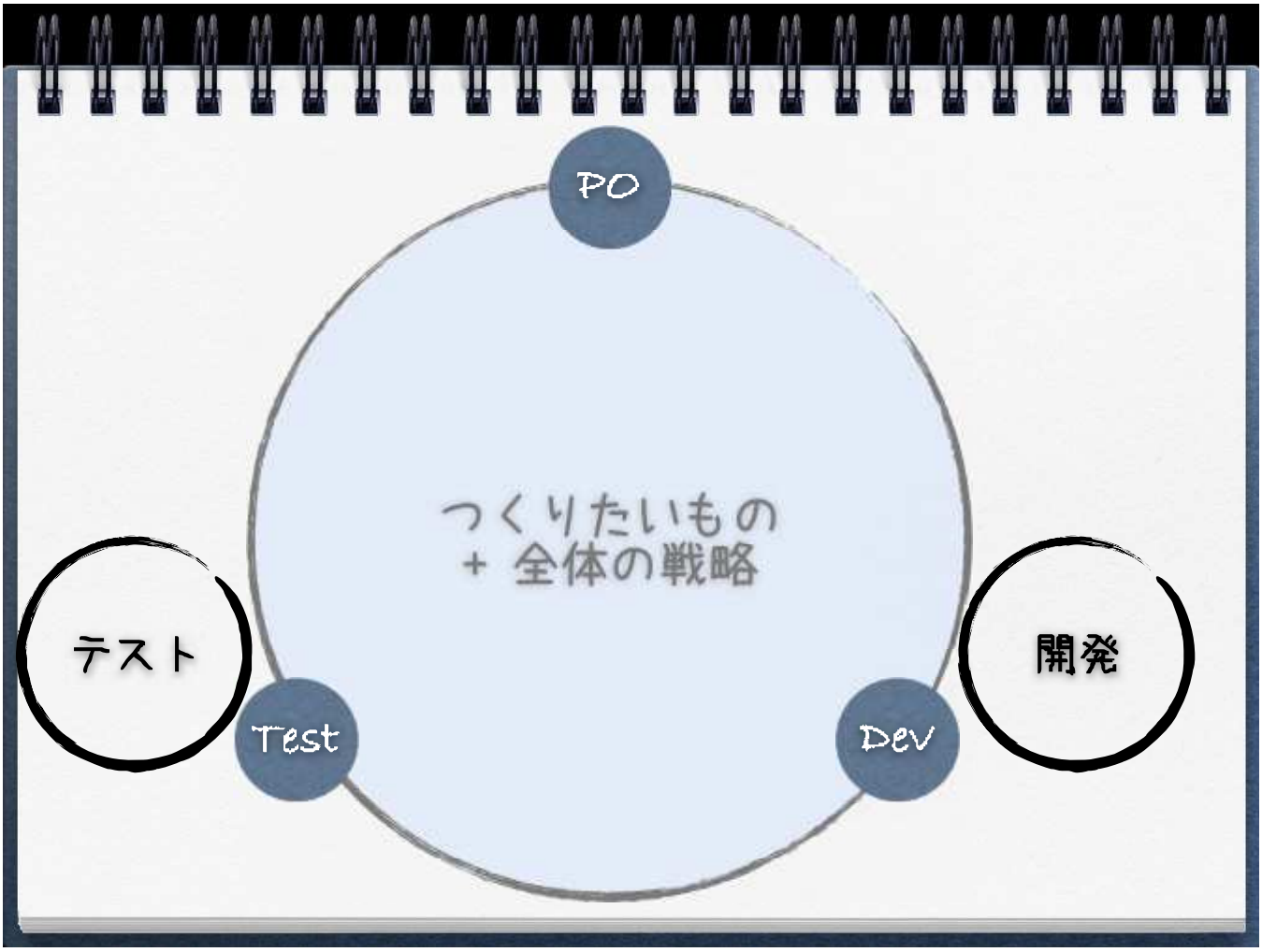

PO
つくりたいもの
+ 品質モデル
設計
Test
Dev

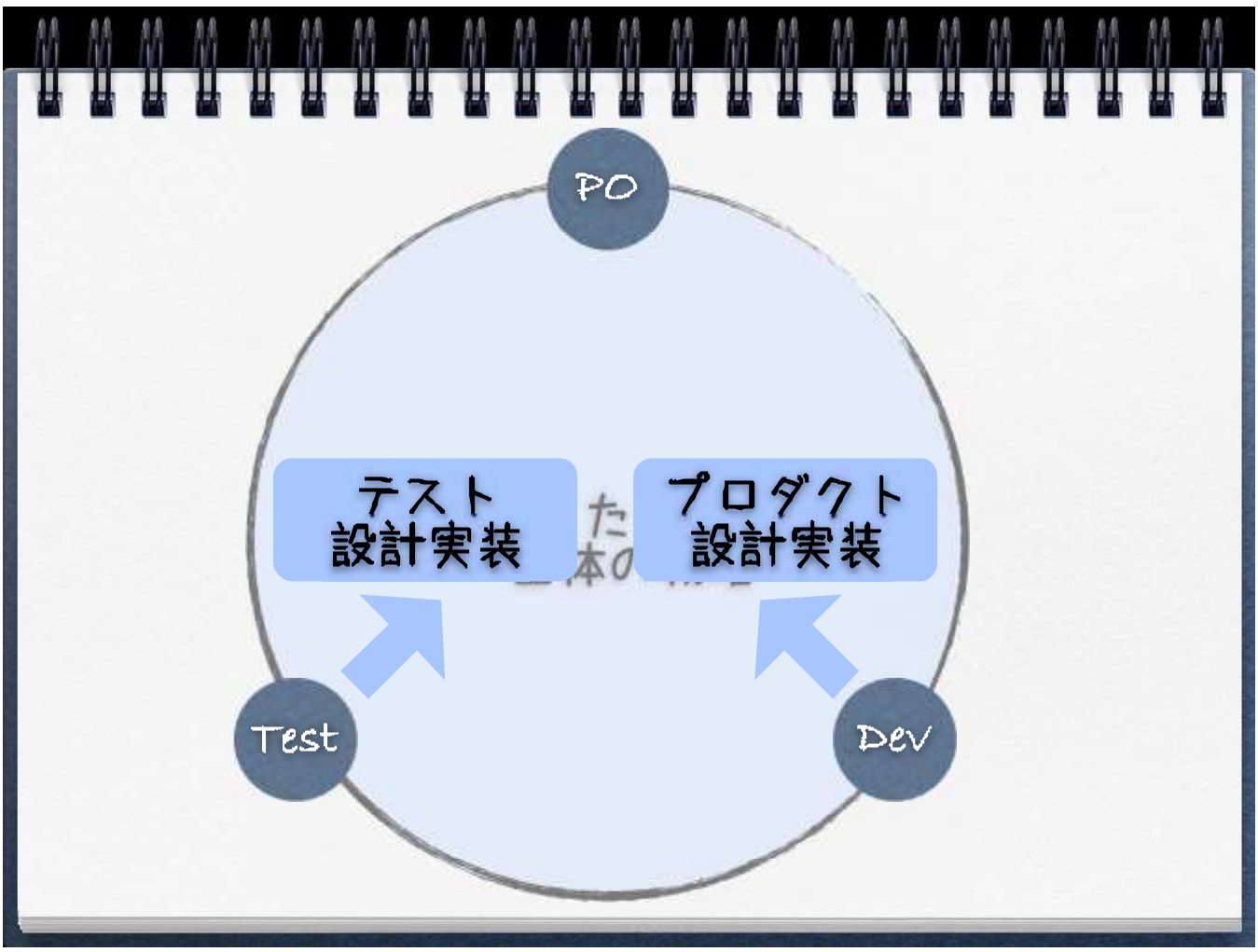

PO
設計
Test
Dev

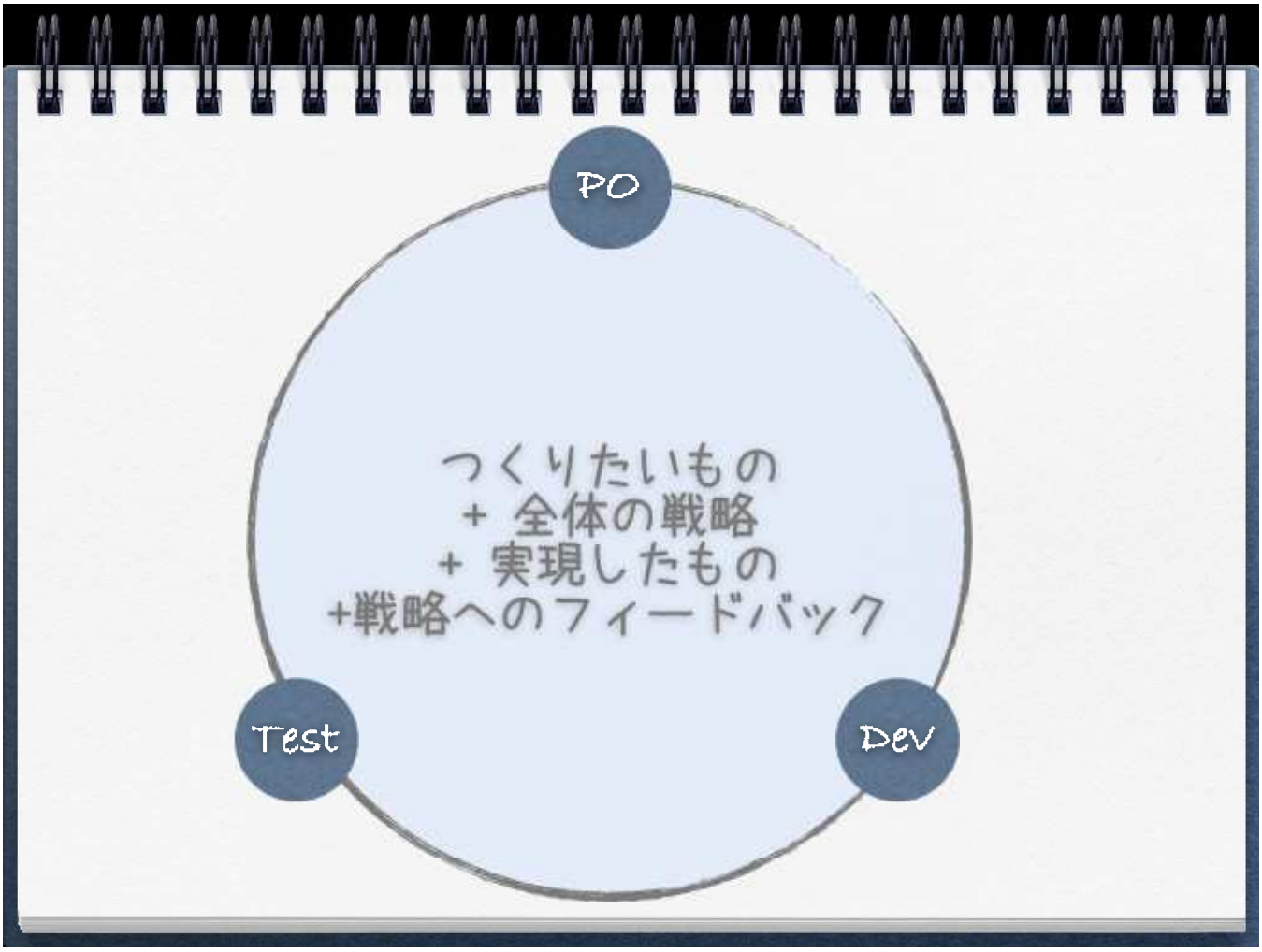

PO
つくりたいもの
+ 品質モデル
+ 設計
+ etc...
Test
Dev

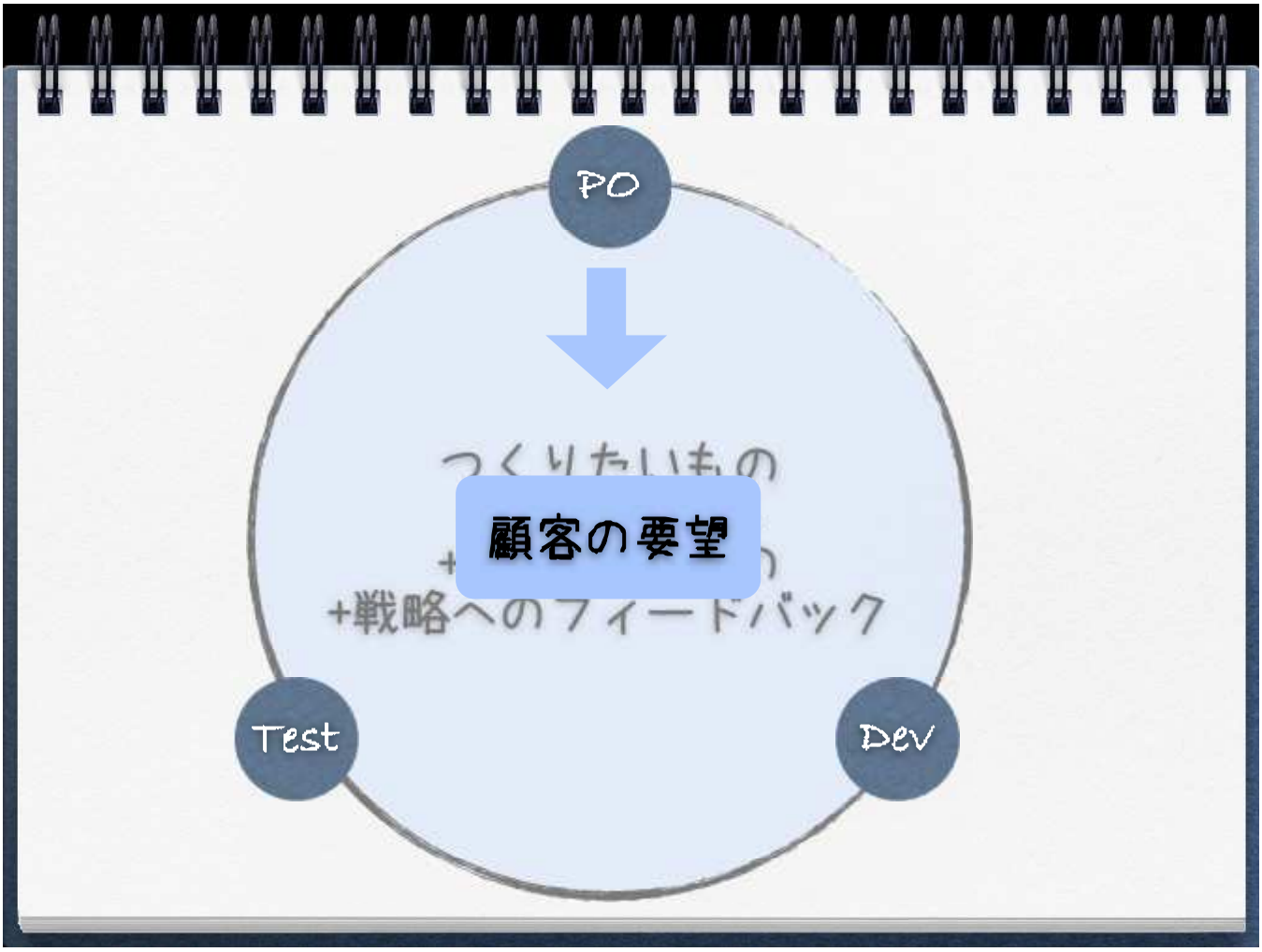

PO
要求
つくりたいもの
全体の戦略
+ etc...
プロダクト
アーキテクチャ
Test
テスト
アーキテクチャ
Dev

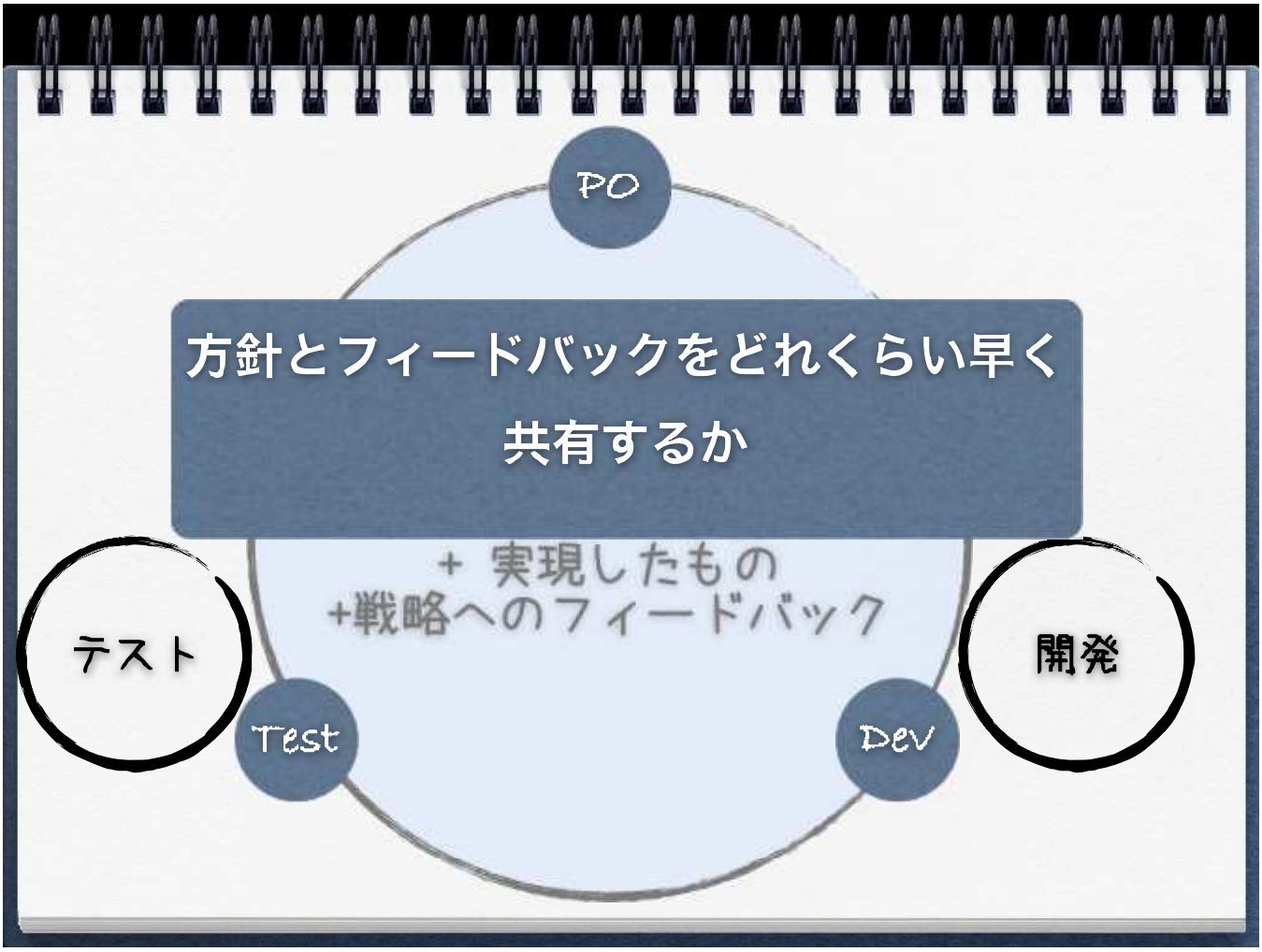

PO
つくりたいもの
+ 全体の戦略
Test
Dev


PO
つくりたいもの
＋ 全体の戦略
テスト
開発
Test
Dev


PO
テスト
設計実装
プロダクト
設計実装
Test
Dev


PO
つくりたいもの
+ 全体の戦略
+ 実現したもの
+戦略へのフィードバック
Test
Dev


PO
顧客の要望
+戦略へのフィードバック
Test
Dev



Agenda

- チームコラボ
- 見積もり
- まとめ

Team Collaboration

Analyze Requirement

- フィーチャー
- 技術的アーキテクチャ
- スケジュール
- チームのリソース
- クライアントのリソース

Use Tools

- 5W2H
- マインドマップ
- フィーチャーボード
- テスト観点モデル

Share

- テストレベル
- 品質モデル
- テストタイプ
- 技術的リスク
- 市場リスク

Test Level

- コミットステージ
- ストーリー受け入れ
- 結合

Test Level

- コミットステージ：ユニット
- ストーリー受け入れ：ハッピーパス
- 結合：テストエンジニアによるテスト

Test Level

自動化範囲はプロジェクト毎に違う

- コミットステージ：ユニット
- ストーリー受け入れ：ハッピーパス
- 結合：テストエンジニアによるテスト

Quality Model

- ISO9126(品質特性) + 経験
- FCM(Walters & McCallのモデル)

ISO9126

- わかりやすそう
- 型から入る(Typeじゃなくて Formだよ！

ISO9126

- ISO9126ベースにどんな品質が必要か考えてみた。
- 結果、それを見ただけではどんなサービスか想像つかないものができあがりやすかった。
- 自分には使いこなせない系。

そこで経験をだな

.NET? Android?
経験ありませんでした。

ということで、チームに聞いてみた

ISO9126 + Team

- 開発者が思う次のような不安点を分類しなおした。

  仕様が曖昧だけど作らなければいけない部分の漏れ

  テストが困難故に単体でしかテストしていない範囲

ISO9126 + Team

- PO(プロダクトオーナー)が思う次のような不安点を分類しなおした。

  運用時に言われそうな課題について

  確認出来ていない受け入れ基準

ISO9126 + Team

- 「何ができるのか」と「どう使われるか」が少しずつ鮮明になっていった。
- これは他の品質モデルを使っても一緒だった。

Risk

- 設計
- ビューティフル・コード
- パフォーマンス
- バグ修正
- 顧客のビジネス影響
- 連携サービス

Effective by share

- テストの優先順位の意識付け
- どの品質を対象にしているかの意識付け
- まずは、マインドマップ + ISO9126で議論をスタートするチームが増えた。

Agenda

- チームコラボ
- 見積もり
- まとめ

Estimation

Use Tools

- 5W2H
- マインドマップ
- フィーチャーボード
- テスト観点モデル

テスト観点 =
何かを実証するアプローチのこと
と定義します。

テスト観点毎にテストするとやりやすいかもしれない。。。
という直感。

そこで、テスト観点を列挙！

あるプロジェクトで
テスト観点を列挙すると150個以上になった。。。

あるプロジェクトで
テスト観点を列挙すると150個以上になった。。。

僕には見積もれないよ。。。

そこで

相対見積もりですよ

Relative Estimation

- テスト観点を相対的に見積もる
- 例)

  プロパティファイルの変更 : 5 points

  クライアントツールのインストール : 8 points

テストの相対見積もりに挑戦！

やってみてすぐに悩んでしまう。。。

やってみてすぐに悩んでしまう。。。

何を見積もればいいのだろう？

やってみてすぐに悩んでしまう。。。

何を見積もればいいのだろう？

- 規模？
- 複雑さ？
- 時間？
- 他のもの？

3つに絞った

- テストに関わりそうなパラメータ数
- テスト実施の難易度
- どれくらい組み合わせるか

Definition of Test Point

- テストポイント =

  テストに関わりそうなパラメータ数

  ×テスト実施の難易度

  ×どれくらい組み合わせるか

Examples of Test Point

	パラメータ	実施難易度	組合せ	TP
言語を跨ぐコード	5	1	3	15
DB基本操作	3	3	2	18
インストール	3	10	2	60

これを全てのテスト観点に適用する！

of Test Point

	パラメータ	実施難易度	組合せ	TP
言語を跨ぐコード	5	1	3	15
DB基本操作	3	3	2	18
インストール	3	10	2	60

テスト観点を列挙すると150個以上になった。。。

6000テストポイント
と変換できた。

理想日での見積もりは？

実施したテストをもとに
理想日を見積もる

Flowの一例

- テストアーキテクチャ構築
- テストポイント見積もり
- テスト戦略策定
- 1日だけうすくひろくテストを実施する
- テストポイントの再見積もり
  テスト戦略再策定
- テスト実施

Flowの一例

- テストアーキテクチャ構築
- テストポイント見積もり

テスト観点の集合をつくる。
次のような事をすることが多かった。
・品質モデルの共有などを通してテスト目的をつくる
・テスト対象をつくる
・テスト観点をつくる
・テスト観点のリファクタリングをする
・必要な情報が抜けていないか考える

Flowの一例

- テストアーキテクチャ構築
- テストポイント見積もり
- テスト戦略策定

先のテストアーキテクチャは不完全で使えないので、プロジェクトの息を吹き込む。（なにをいっｔ
リソースや日々の変化、日程などを加味した全体での作戦になるもの。

Flowの一例

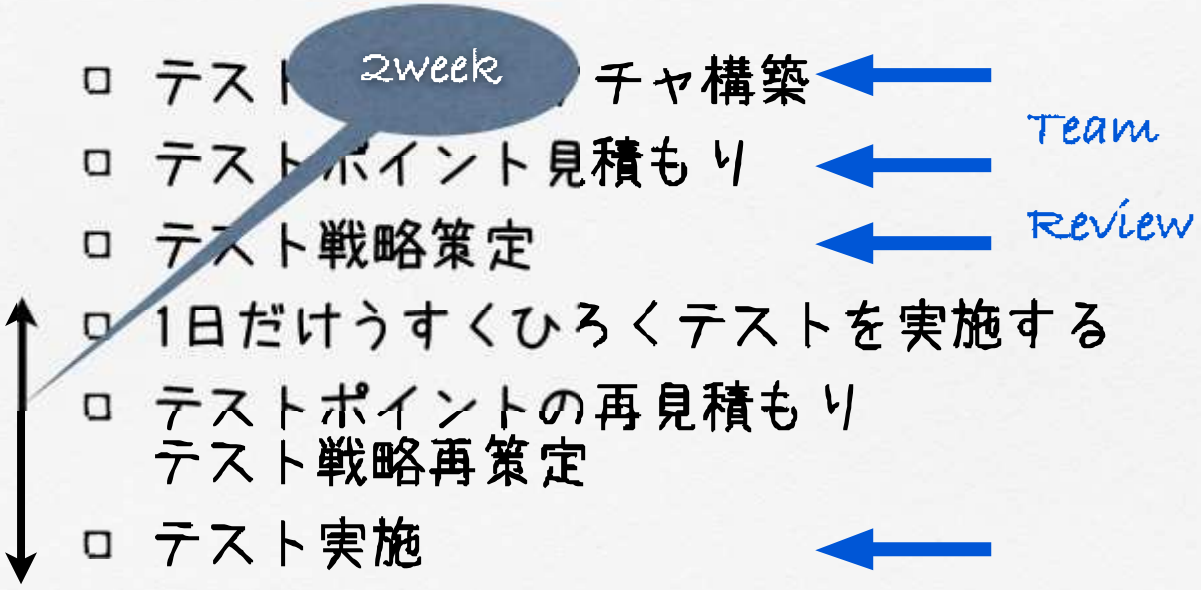
- テストアーキテクチャ構築
- テストポイント見積もり
- テスト戦略策定
- 1日だけうすくひろくテストを実施する
- テストポイントの再見積もり
  テスト戦略再策定
- テスト実施

Flowの一例

- テストアーキテクチャ構築 ←
- テストポイント見積もり ←
- テスト戦略策定 ←
- 1日だけうすくひろくテストを実施する
- テストポイントの再見積もり
  テスト戦略再策定
- テスト実施 ←

Team Review

Flowの一例



Test Strategy

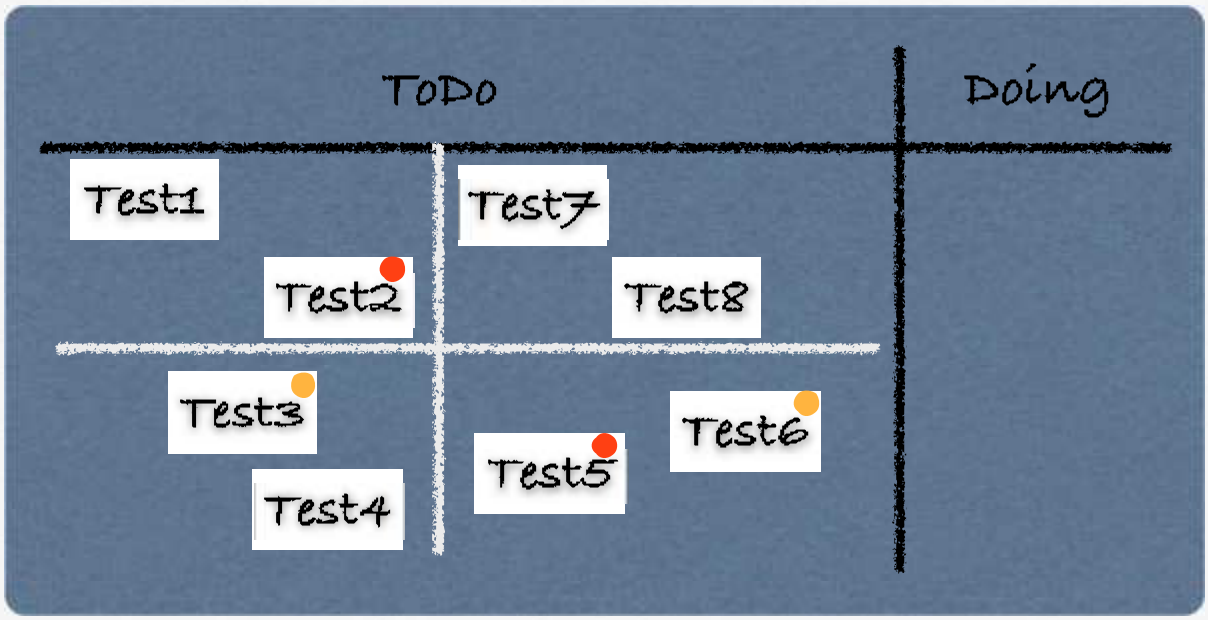
- テスト観点のマトリクス
- POと相談してどのテスト観点のテストを実施するか決定

Test Strategy

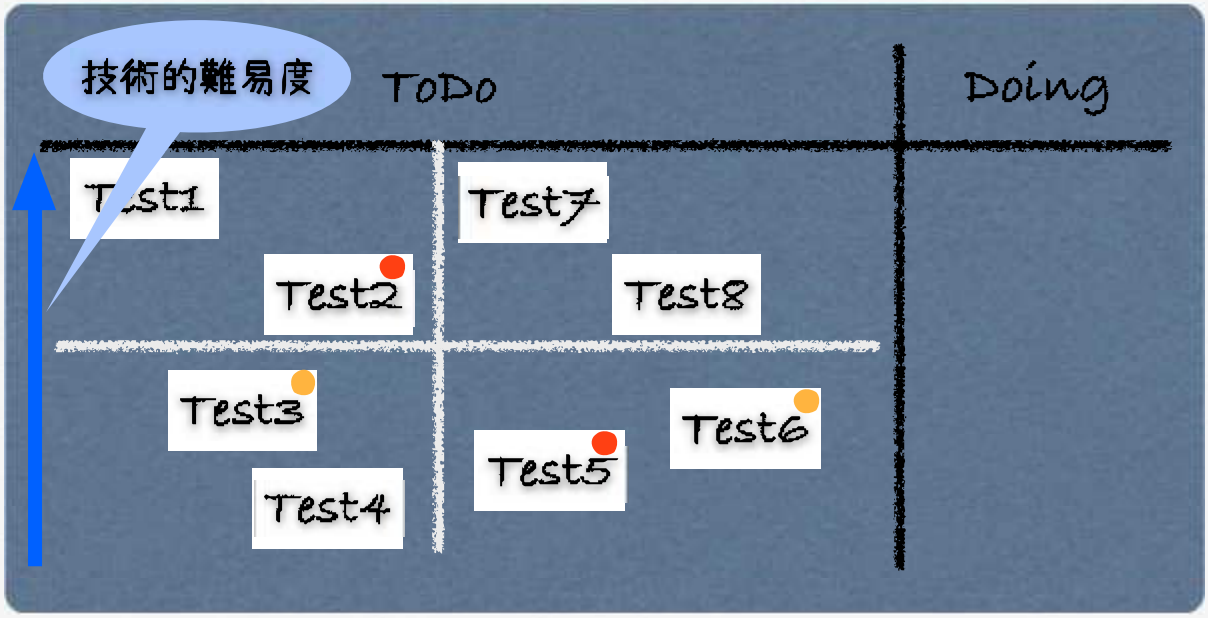
- テスト観点のマトリクス
- POと相談してどのテスト観点のテストを実施するか決定

テスト用のKanbanを用意してみた

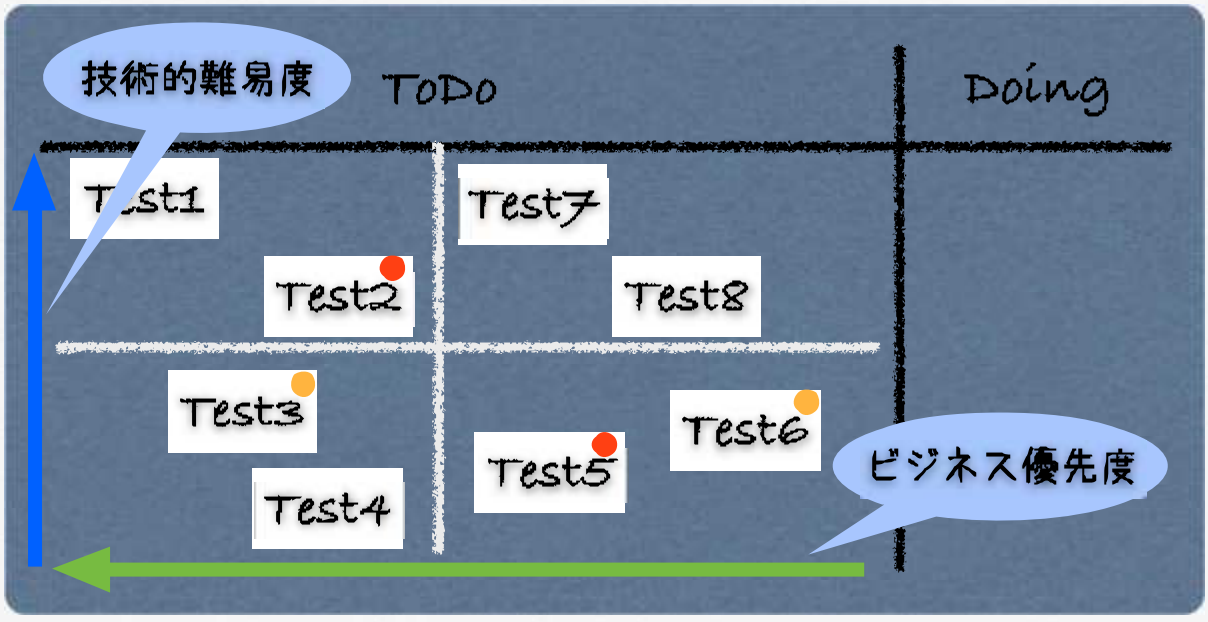
Test Board

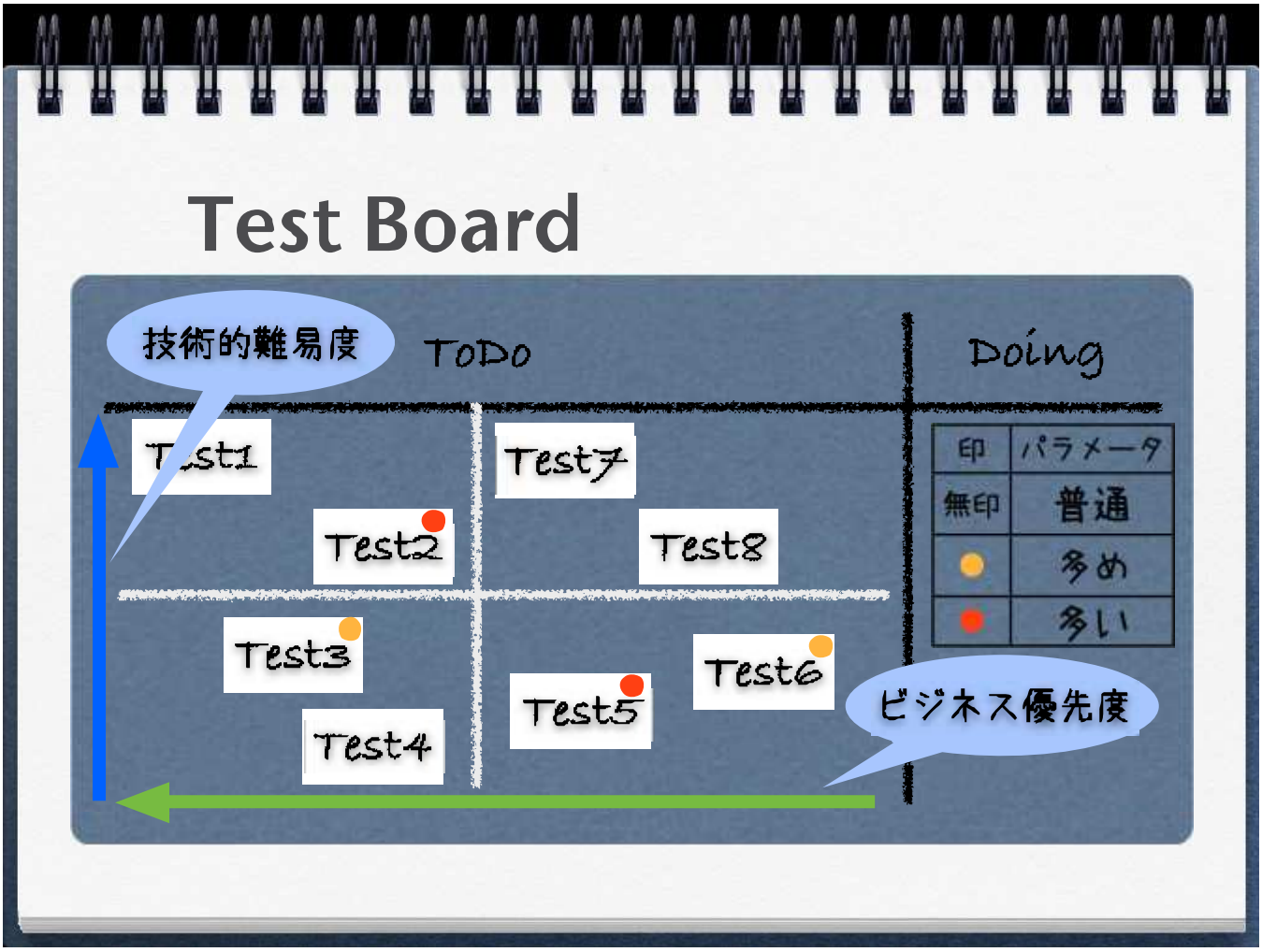


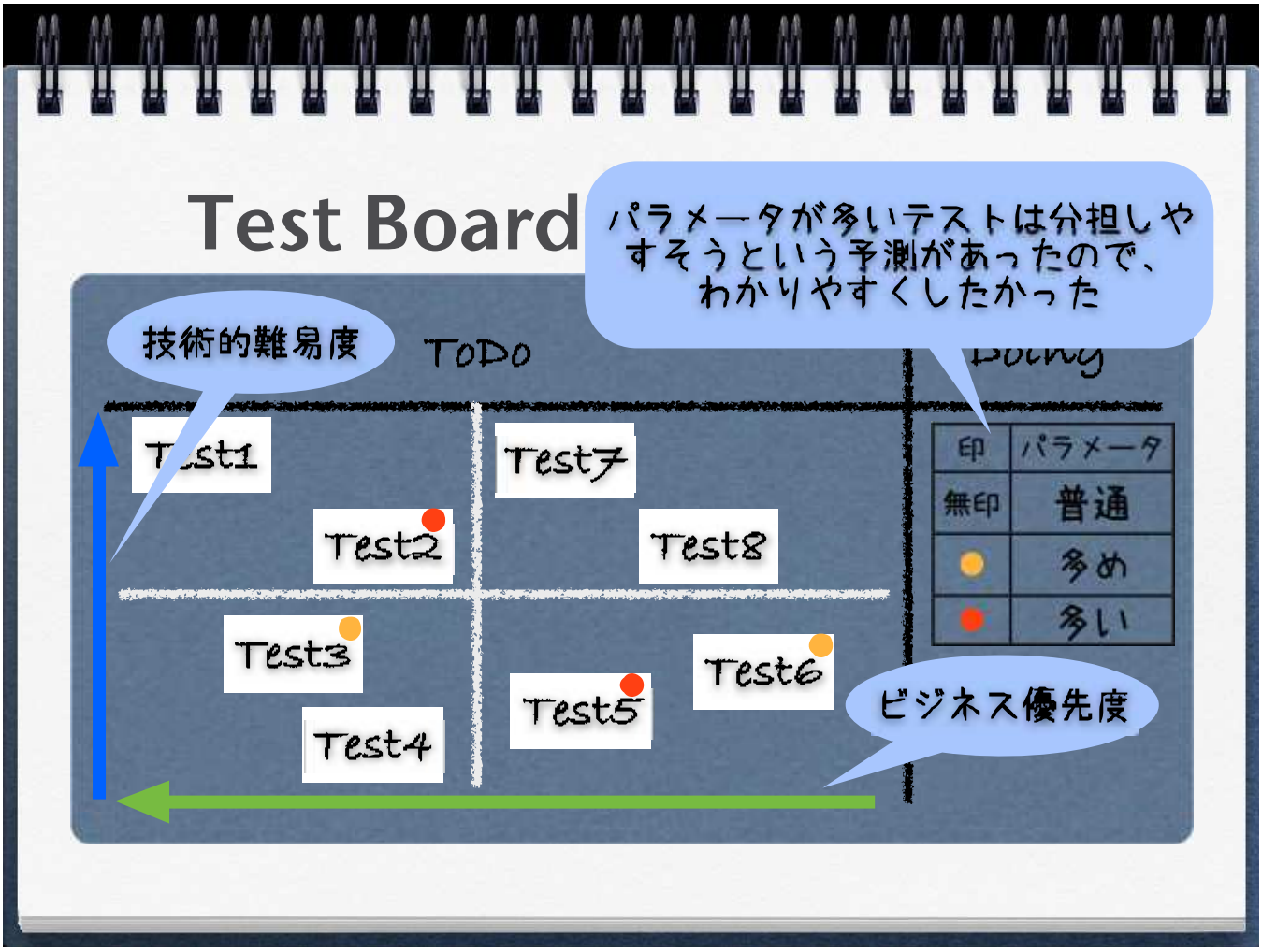
Test Board



Test Board

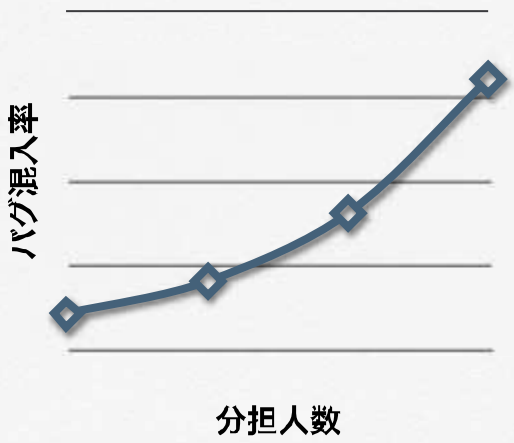



Test Board
技術的難易度
ToDo
Doing
Test1
Test2
Test3
Test4
Test5
Test6
Test7
Test8
印
パラメータ
無印
普通
多め
多い
ビジネス優先度


Test Board
パラメータが多いテストは分担しやすそうという予測があったので、わかりやすくしたかった
技術的難易度
ToDo
Doing
Test1
Test2
Test3
Test4
Test5
Test6
Test7
Test8
印
パラメータ
無印
普通
多め
多い
ビジネス優先度

Ease of divide test

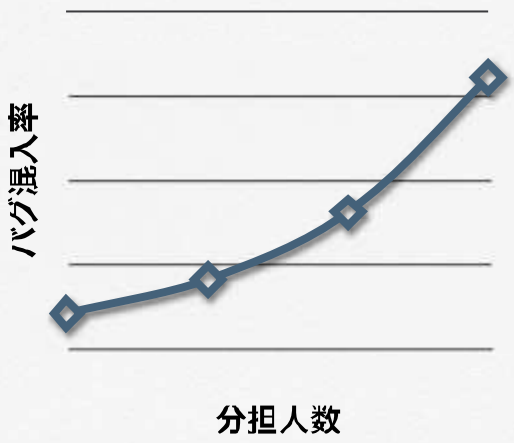
ある規模当たりの作業品質

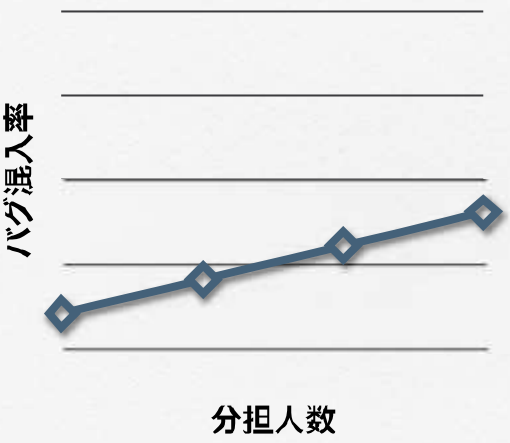


Ease of divide test

ある規模当たりの作業品質

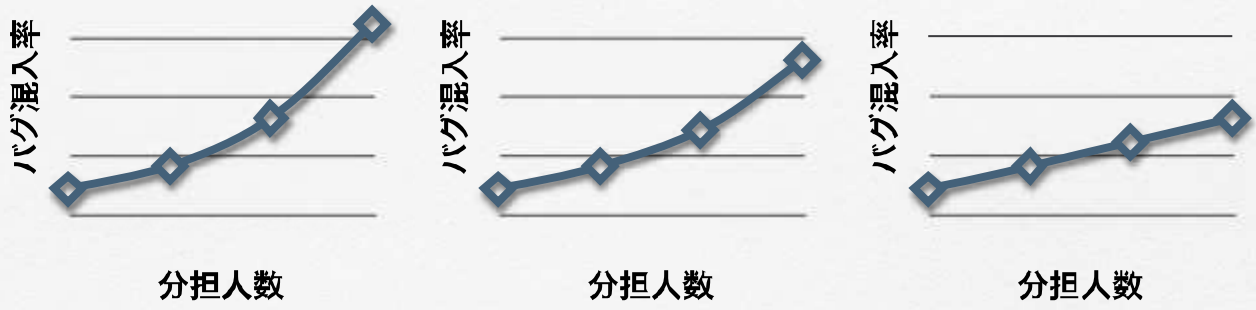
できるだけ右のようなグラフにしたい





Ease of divide test

ある規模当たりの作業品質



分担人数　分担人数　分担人数

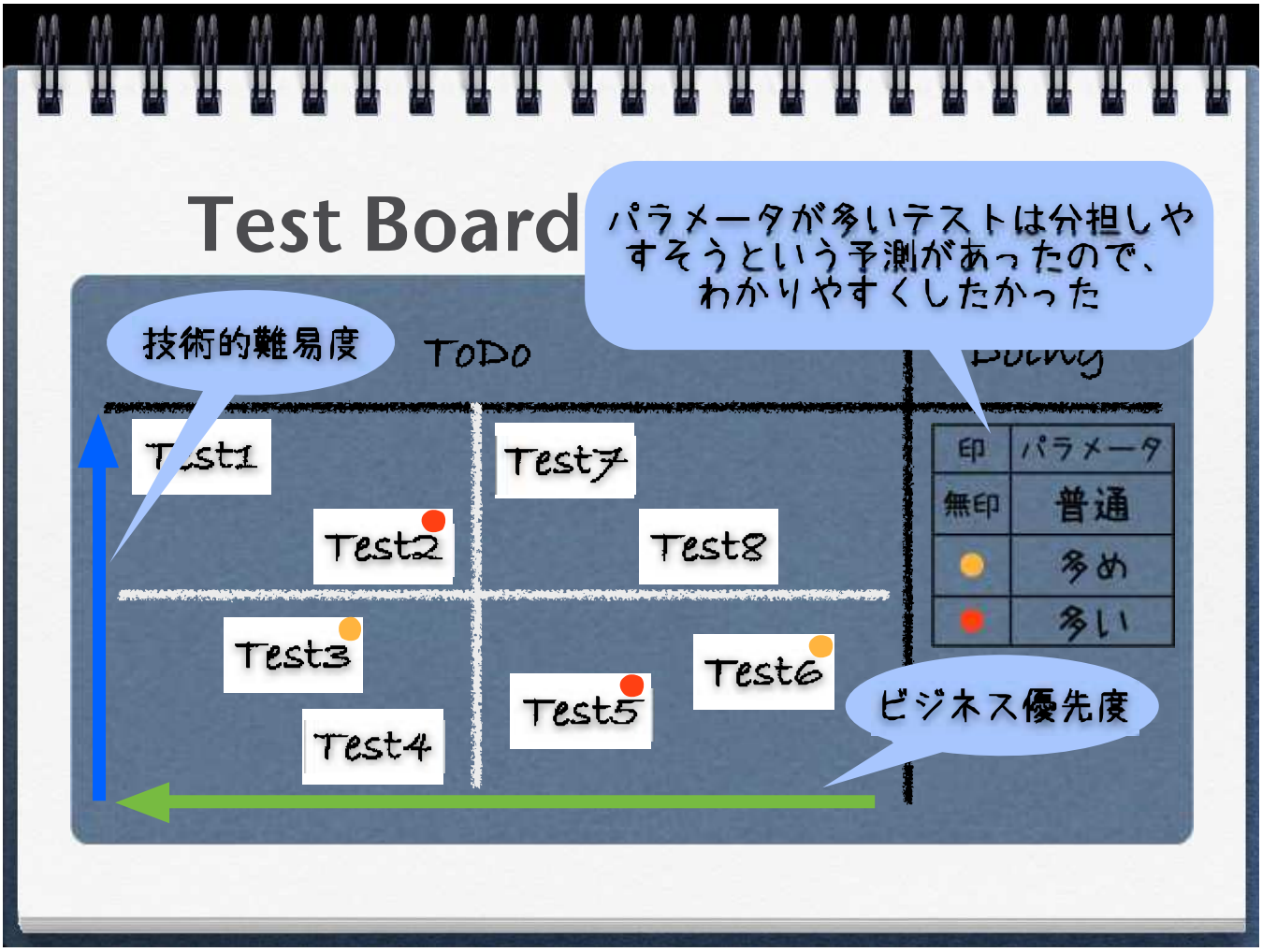
テストの独立性

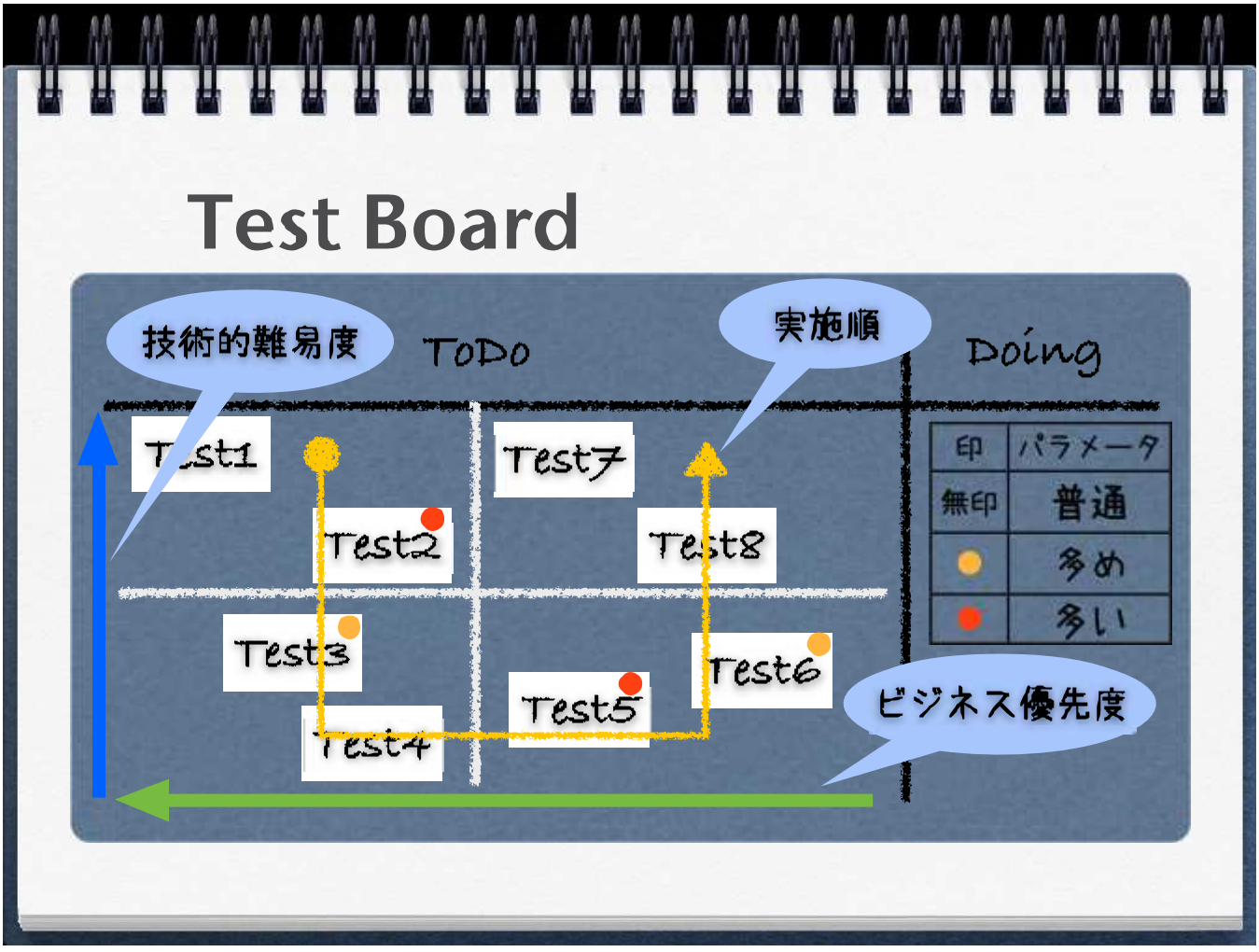
Ease of divide test

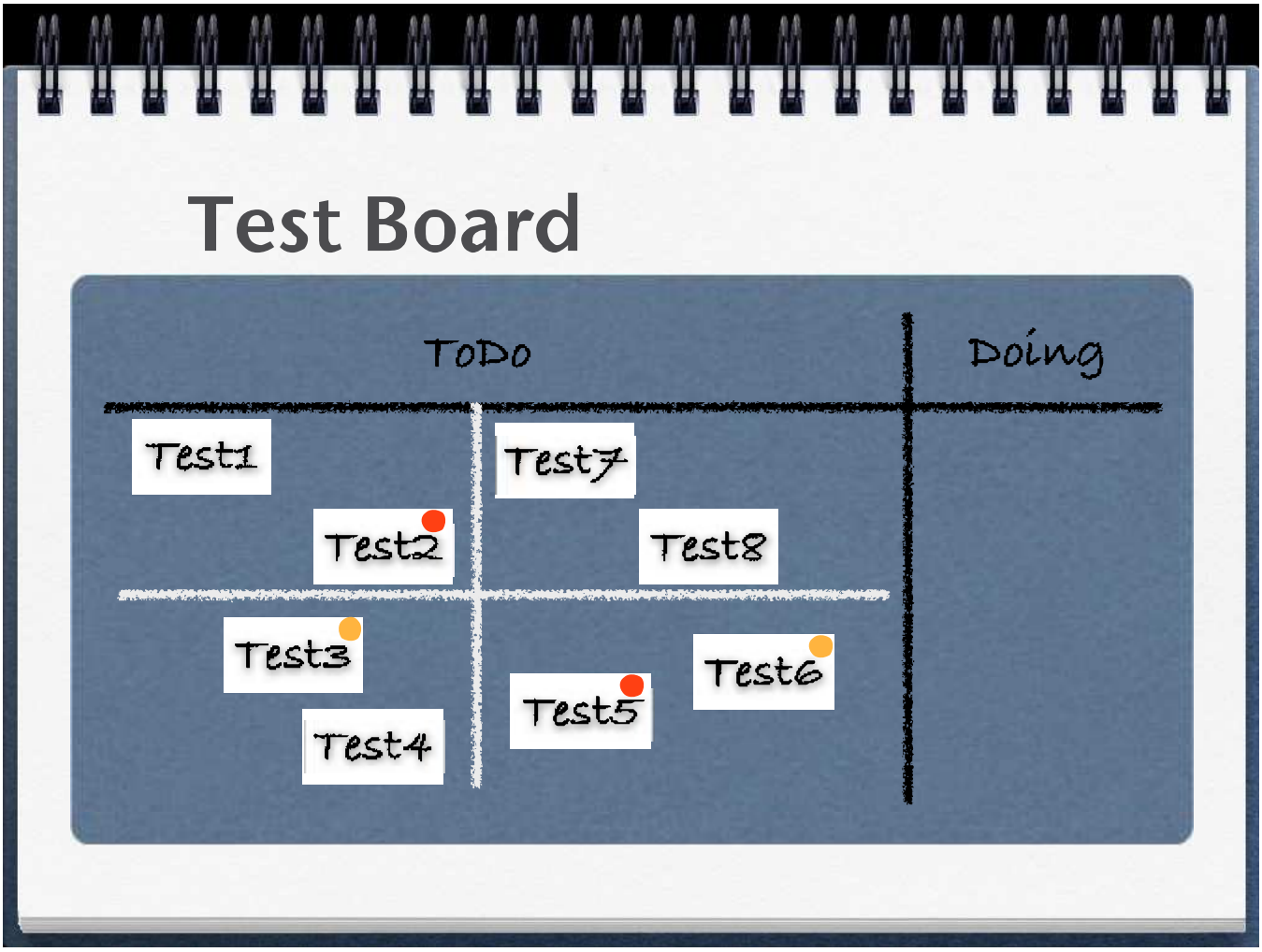
様々な要因があるが、
パラメータが多い事によって規模が大きくなるテストはテストを分割しやすい(独立性を保ち易い)のではないだろうか。という経験則。

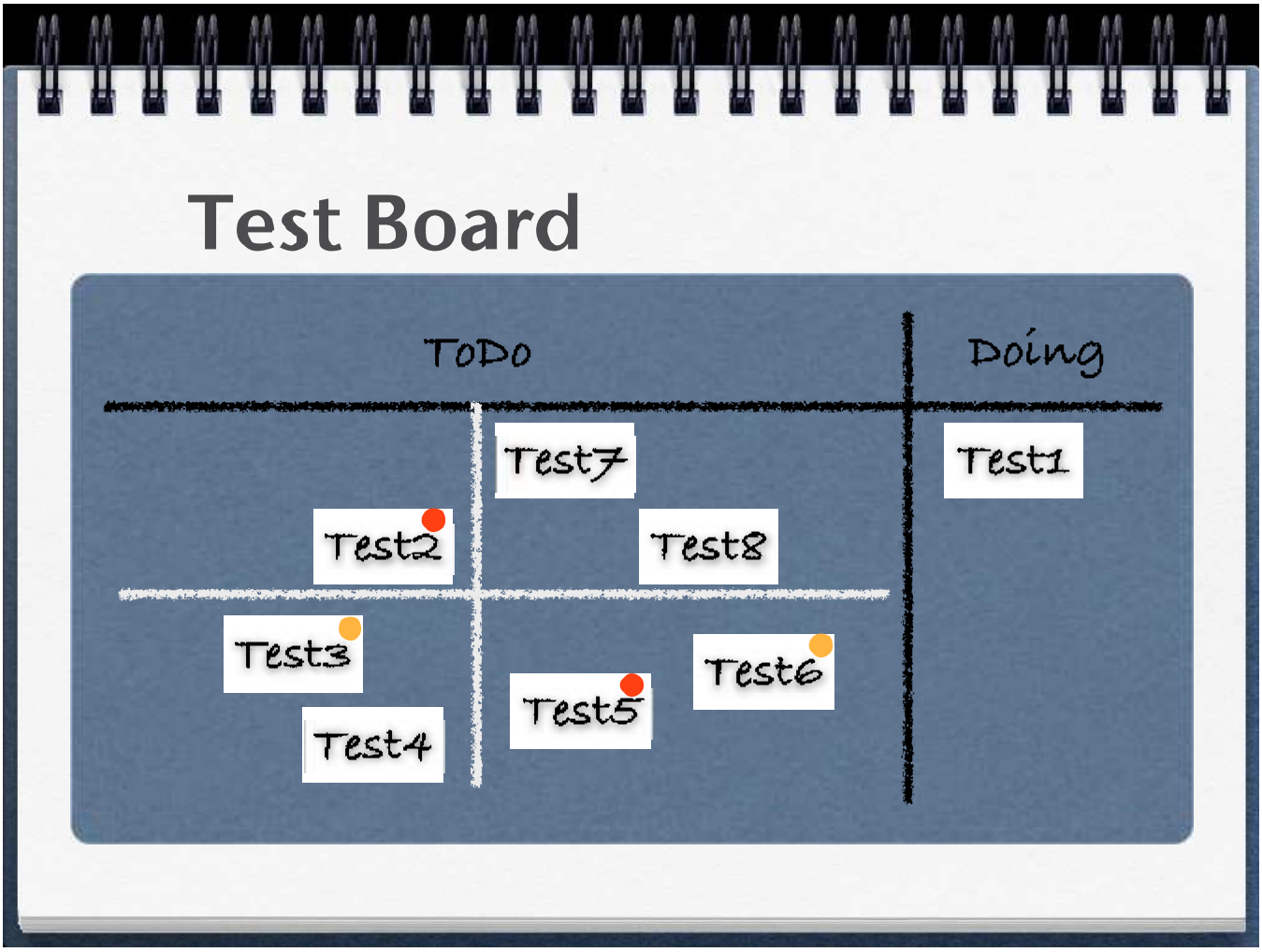
分担人数　分担人数　分担人数

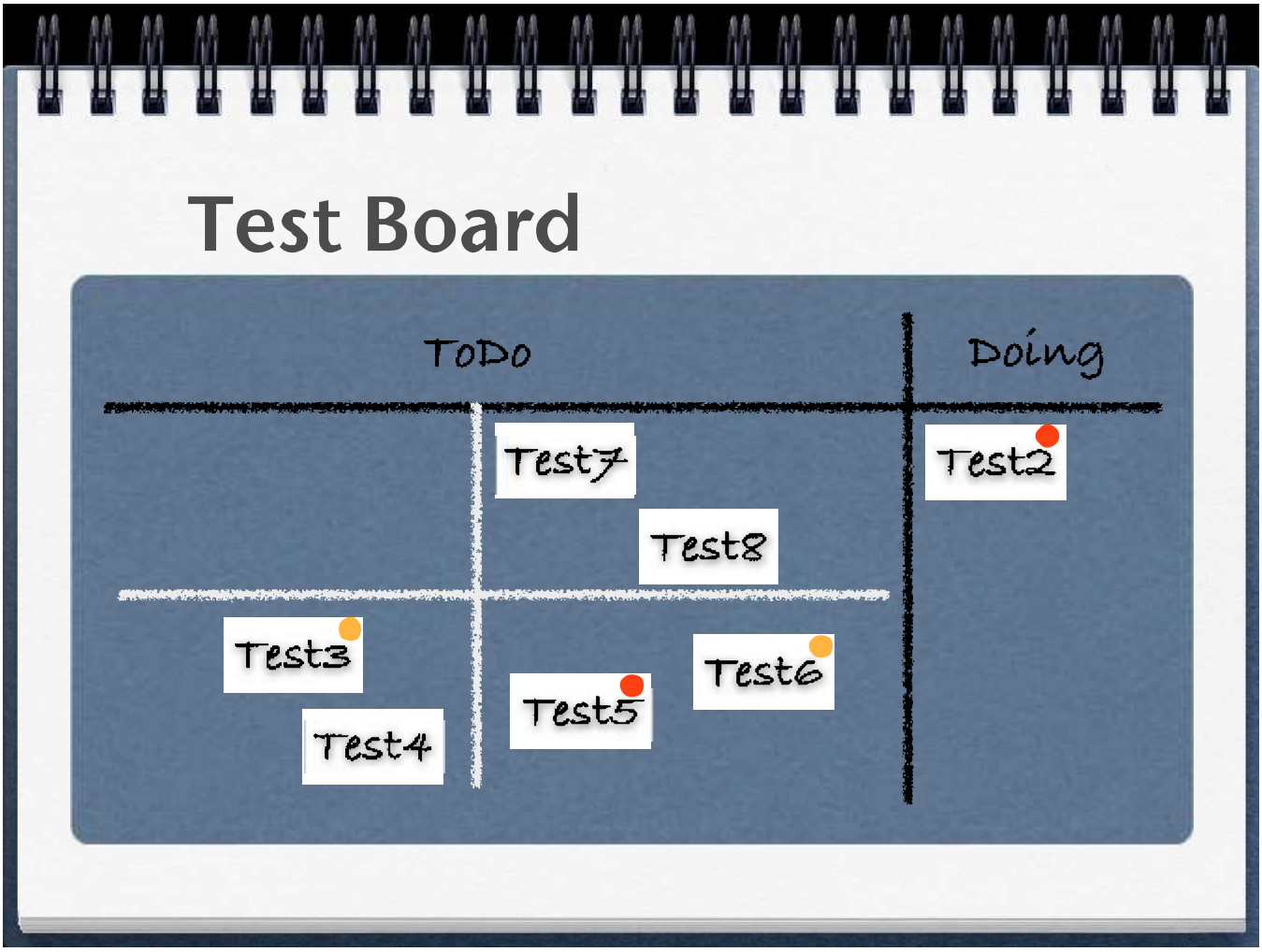
テストの独立性

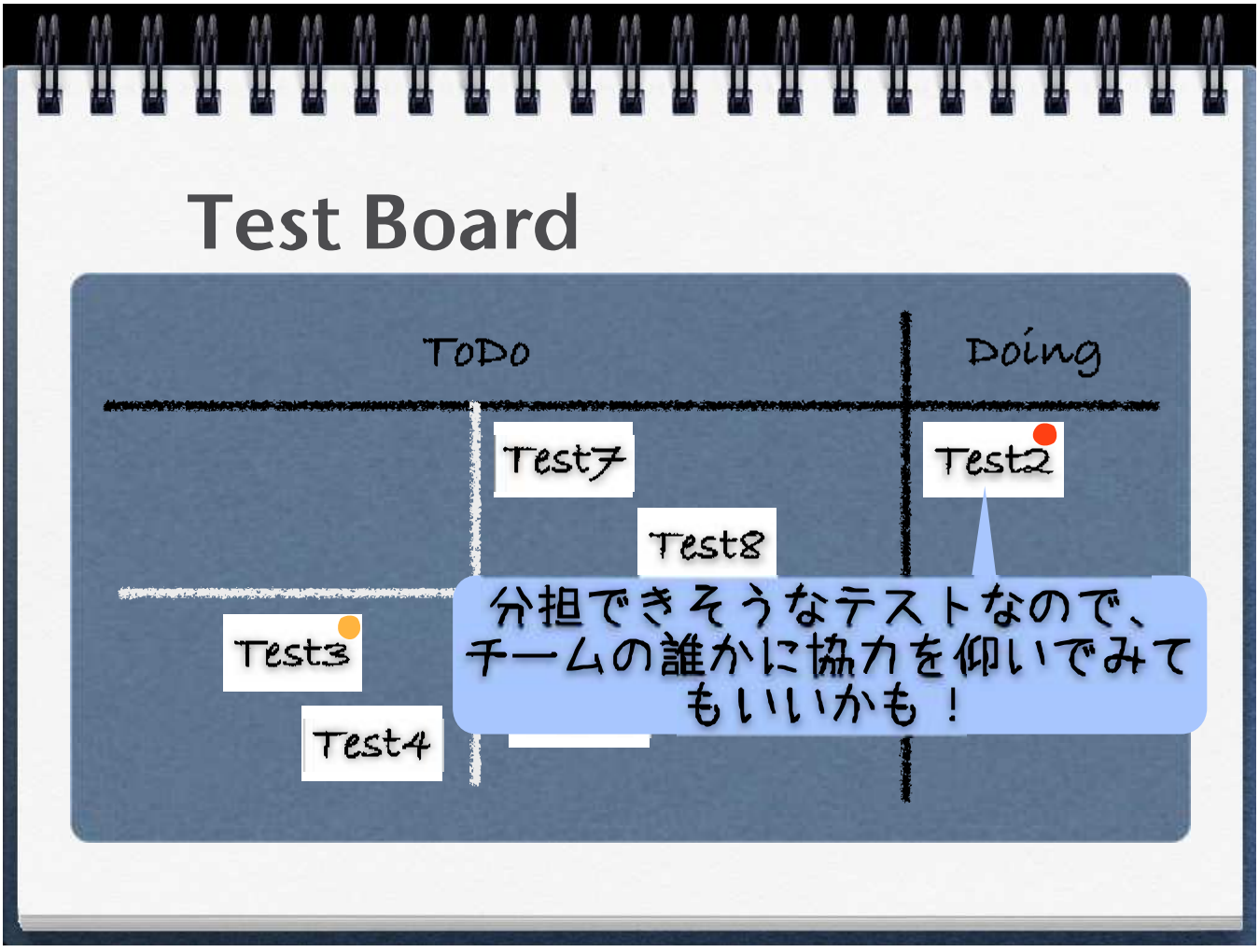

Test Board
パラメータが多いテストは分担しやすそうという予測があったので、わかりやすくしたかった
技術的難易度
ToDo
Doing
Test1
Test2
Test3
Test4
Test5
Test6
Test7
Test8
印
パラメータ
無印
普通
多め
多い
ビジネス優先度


Test Board
技術的難易度
ToDo
実施順
Doing
Test1
Test2
Test3
Test4
Test5
Test6
Test7
Test8
印
パラメータ
無印
普通
多め
多い
ビジネス優先度

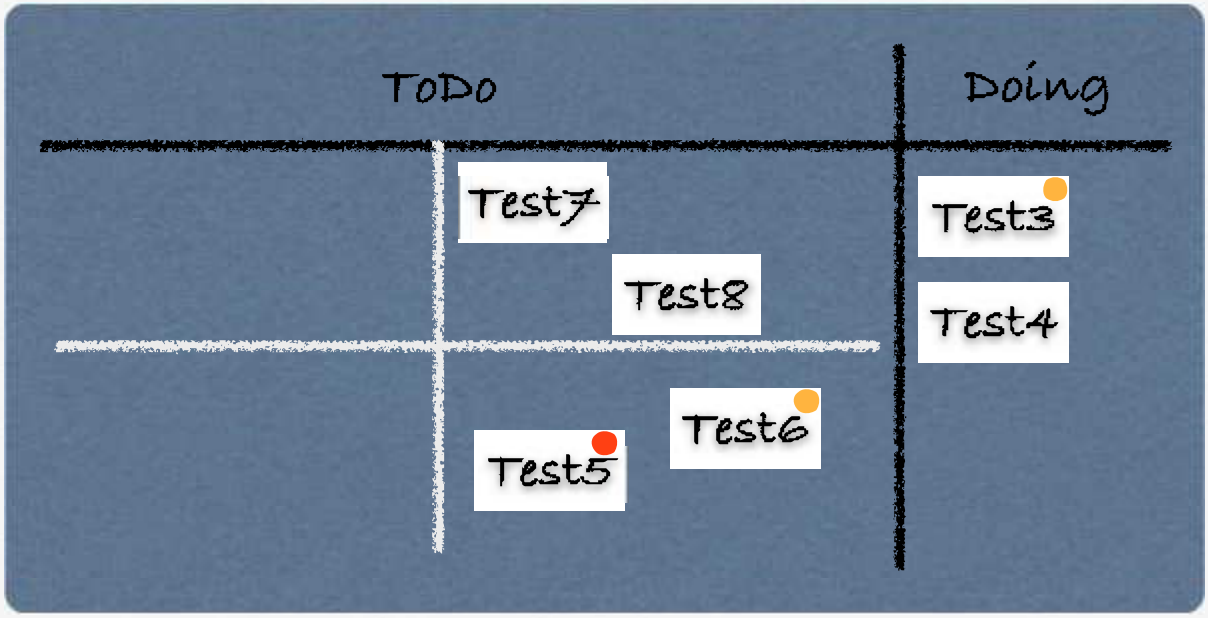

Test Board
ToDo
Doing
Test1
Test7
Test2
Test8
Test3
Test6
Test5
Test4


Test Board
ToDo
Doing
Test7
Test1
Test2
Test8
Test3
Test6
Test5
Test4


Test Board
ToDo
Doing
Test7
Test2
Test8
Test3
Test6
Test5
Test4


Test Board
ToDo
Doing
Test7
Test2
Test8
Test3
Test4
分担できそうなテストなので、
チームの誰かに協力を仰いでみて
もいいかも！

Test Board



Test Strategy

- テスト観点のマトリクス
- POと相談してどのテスト観点をテストを実施するか決定

Flowの一例

- テストアーキテクチャ構築
- テストポイント見積もり
- テスト戦略策定
- 1日だけうすくひろくテストを実施する
- テストポイントの再見積もり
  テスト戦略再策定
- テスト実施

調査的テスト（仮）

- 見積もりのために実施する1dayのテストを調査的テストと仮に名前つけた。
- できるだけ、満遍なく多くのテスト観点をテストする。
- 目的は「テストの見積もり」「プロダクトの現状調査」「必要そうなスキルの確認」

調査的テスト（仮）

探針とは違う？

- 見積もりのために実施する1dayのテストを調査的テストと仮に名前つけた。
- できるだけ、満遍なく多くのテスト観点をテストする。
- 目的は「テストの見積もり」「プロダクトの現状調査」「必要そうなスキルの確認」

調査的テスト（仮）

- さらっと言えば、Test Boardを1日で1周することで、何かを掴む。

Velocity

- 1st Sprint -> 1500p / 1week
- 2nd Sprint -> 2000p / 1week
- 3rd Sprint -> 1800p / 1week

見積り可能なテスト

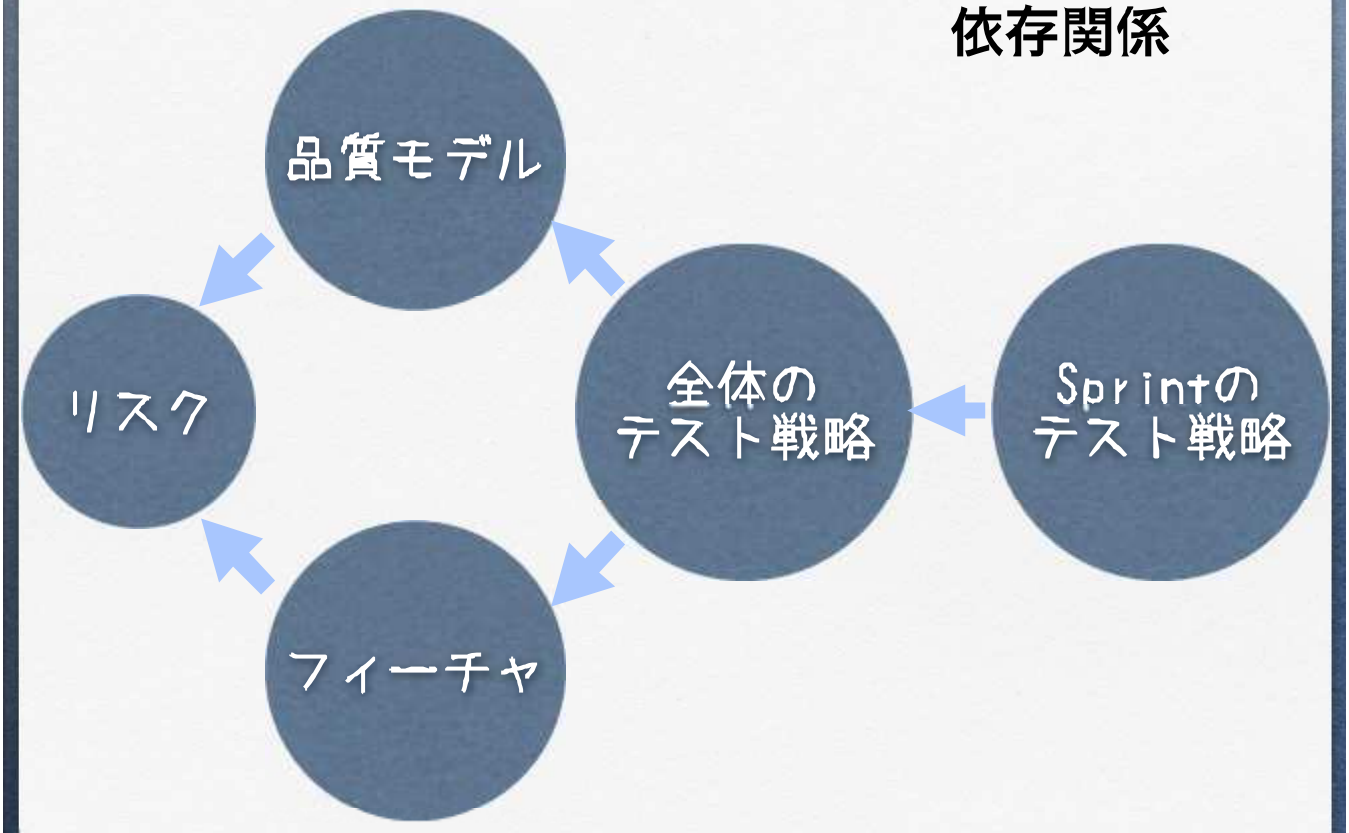
- 見積もり可能なテストはテスティングを導いてくれる。
- 見積もれないテストはテスティングが行き当たりばったりになる。

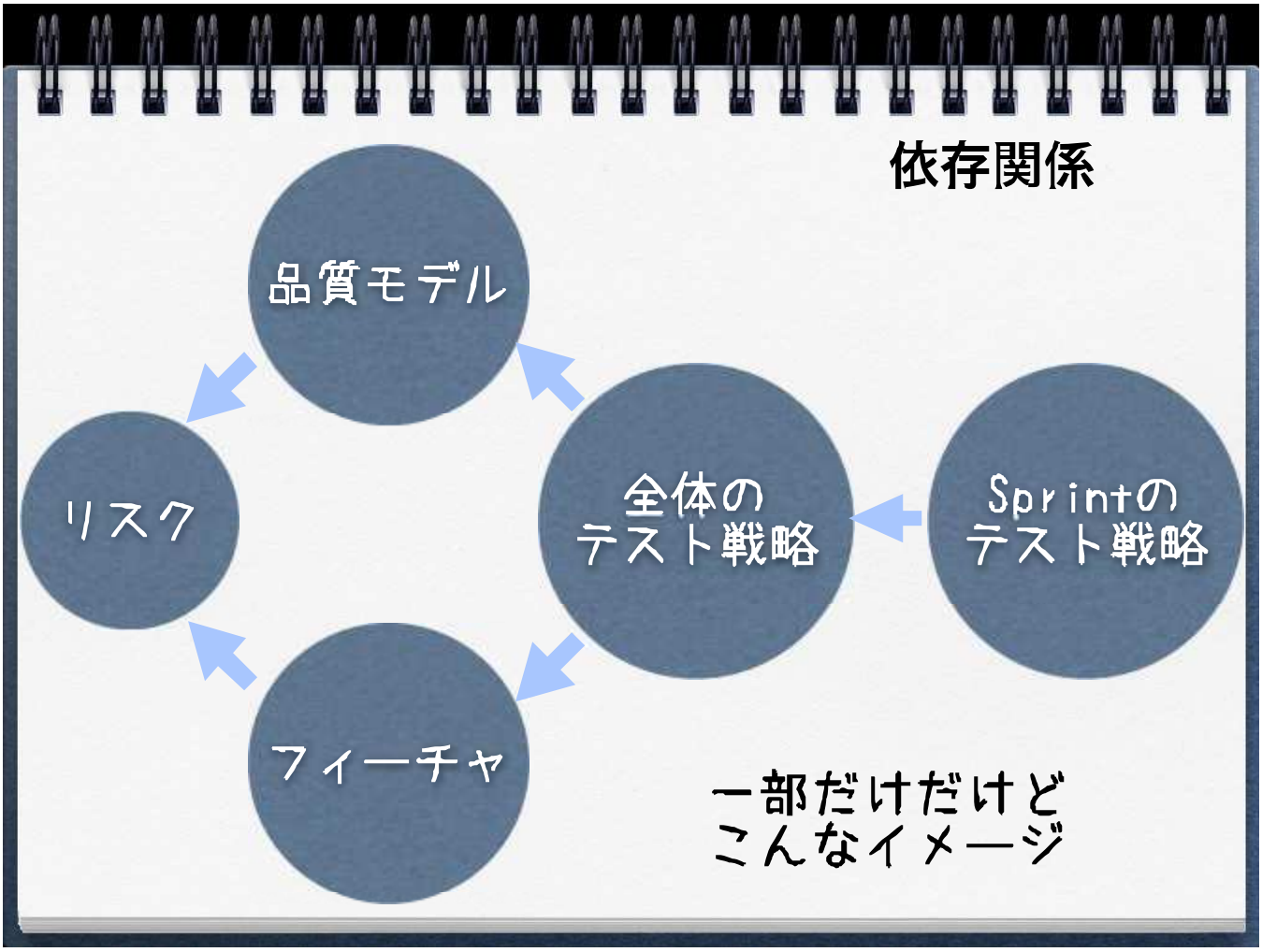
見積り可能なテスト

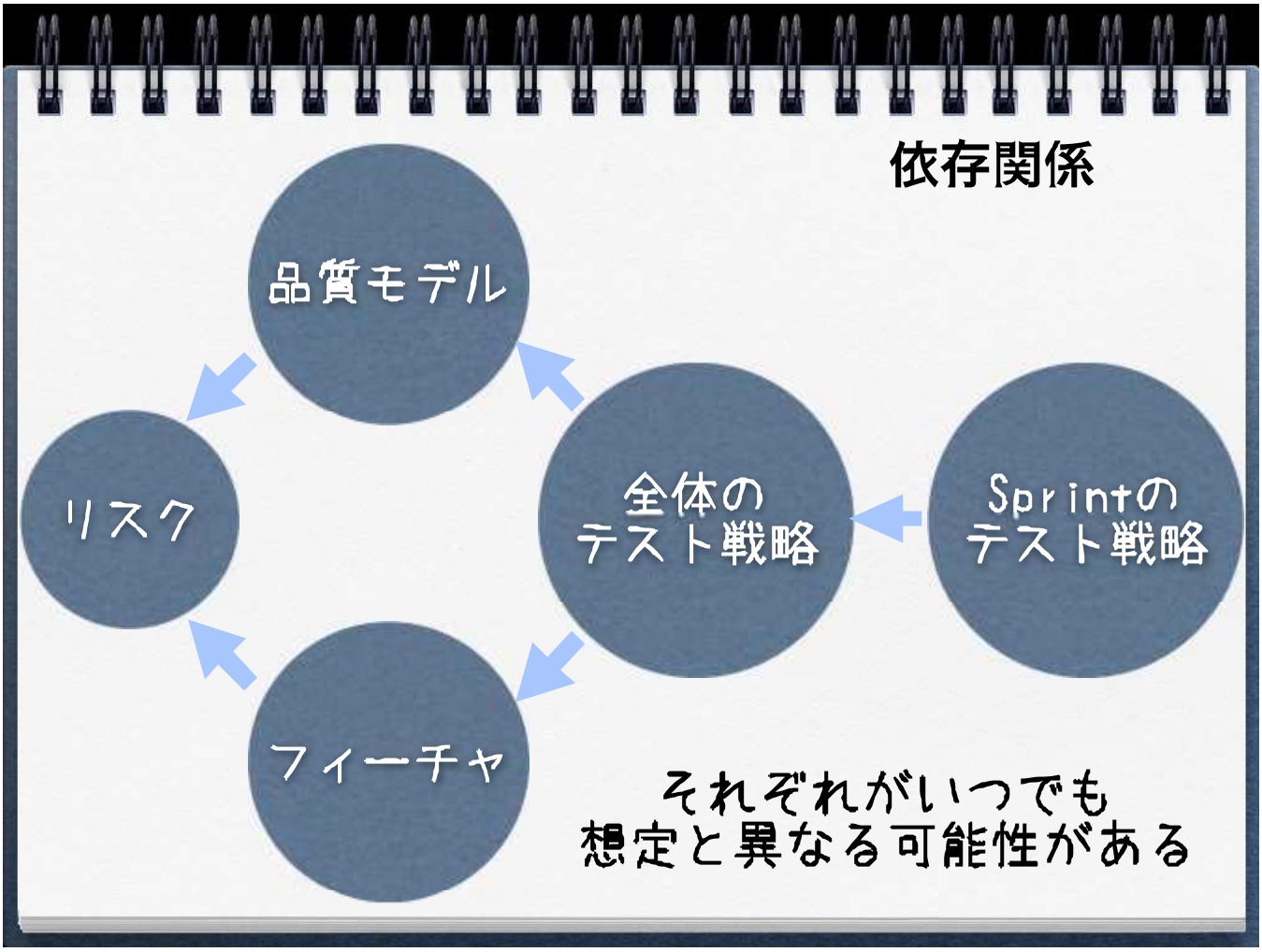
- 見積もり可能なテストはテスティングを導いてくれる。
- 見積もれないテストはテスティングが行き当たりばったりになる。

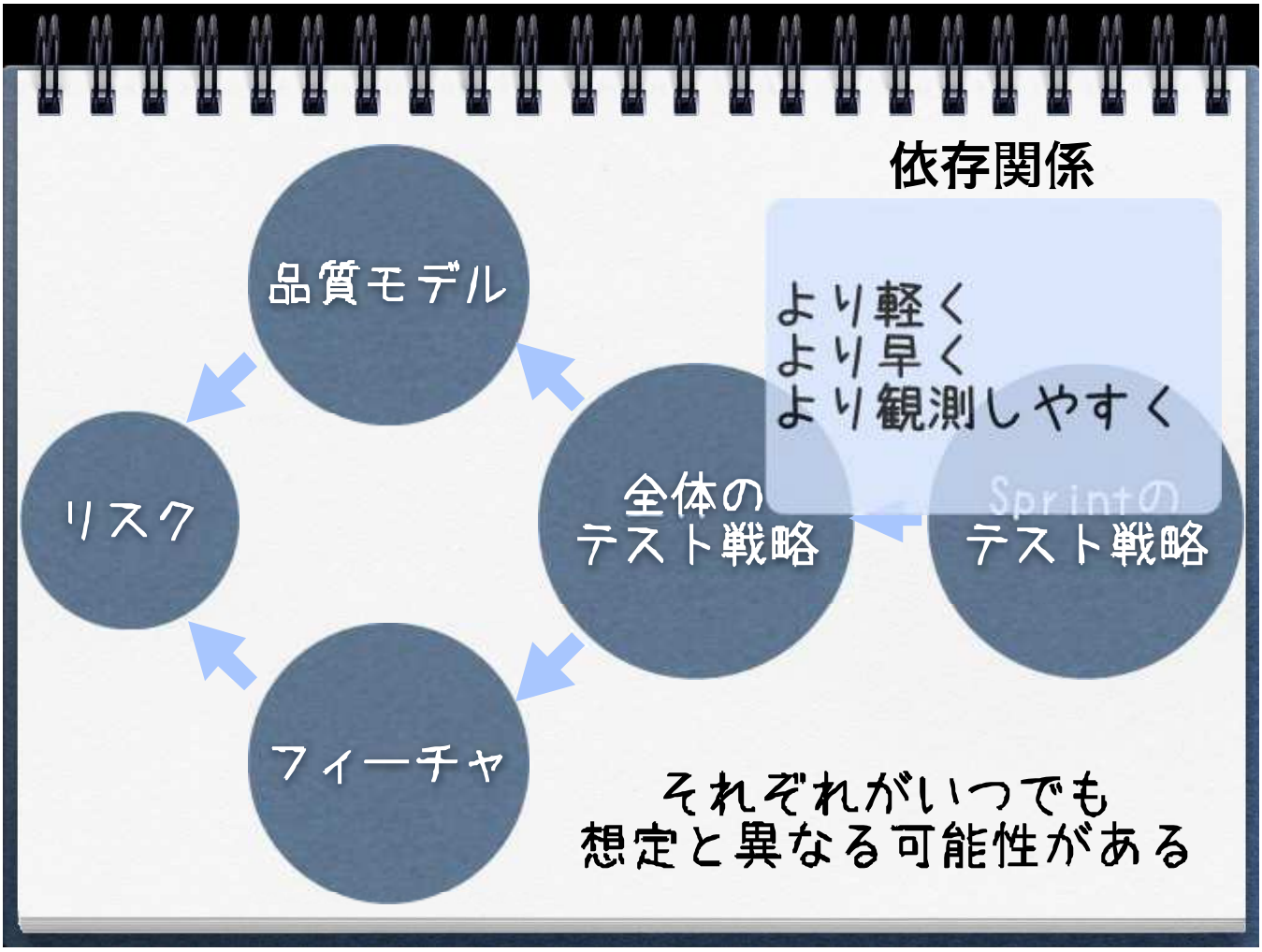
行き当たりばったりになったら、
見積もりが不正確な可能性が高かった

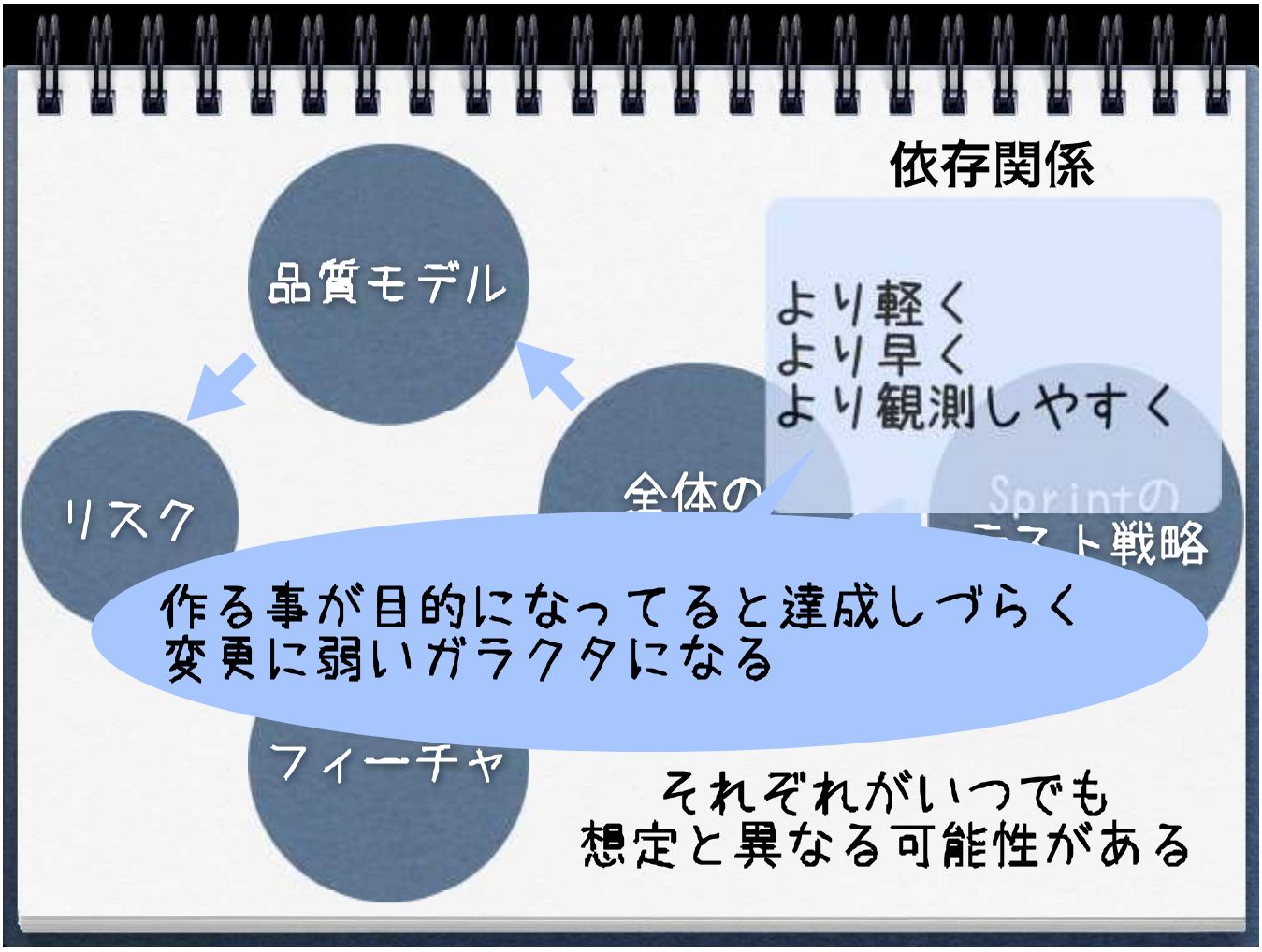
依存関係

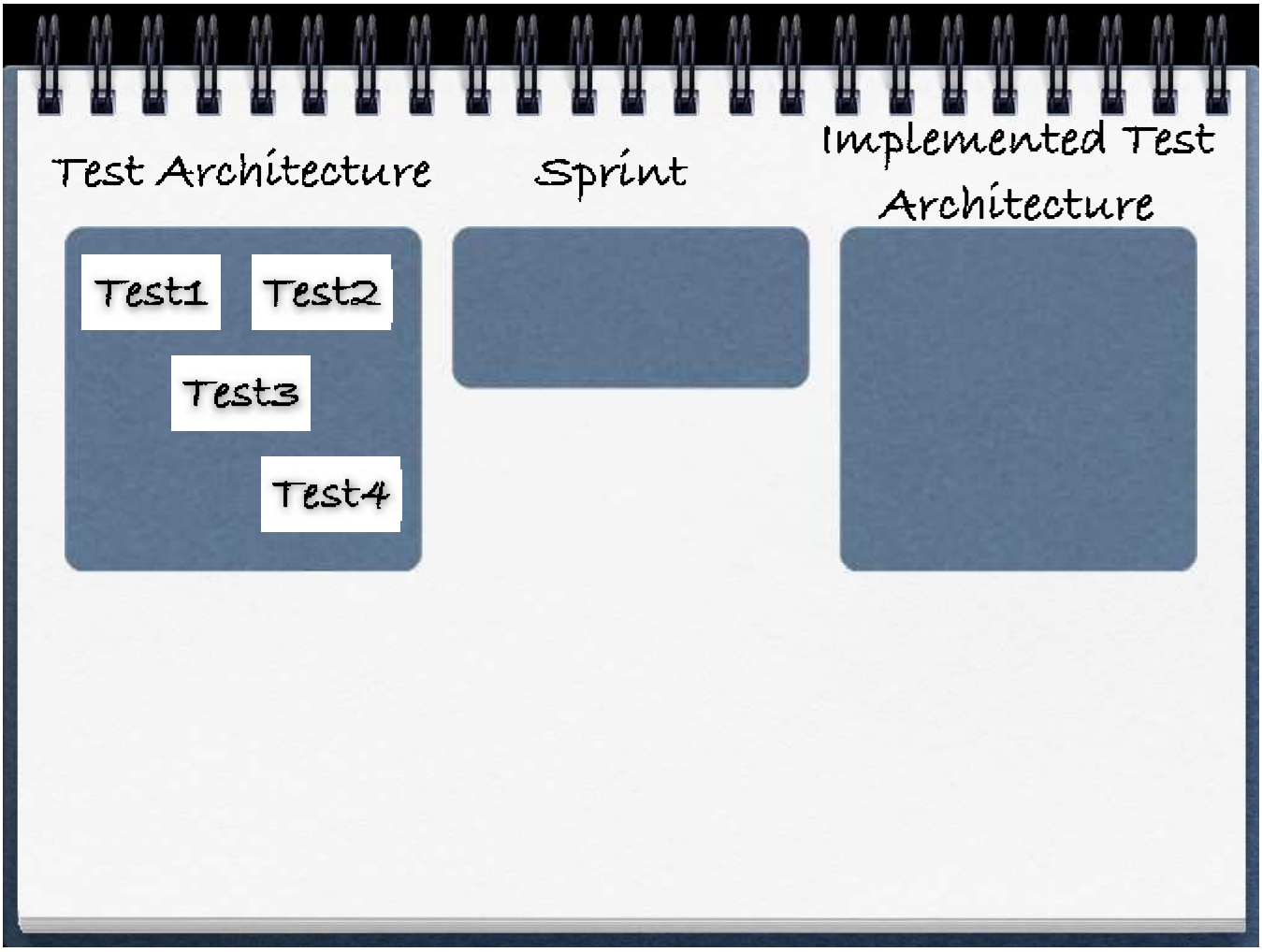


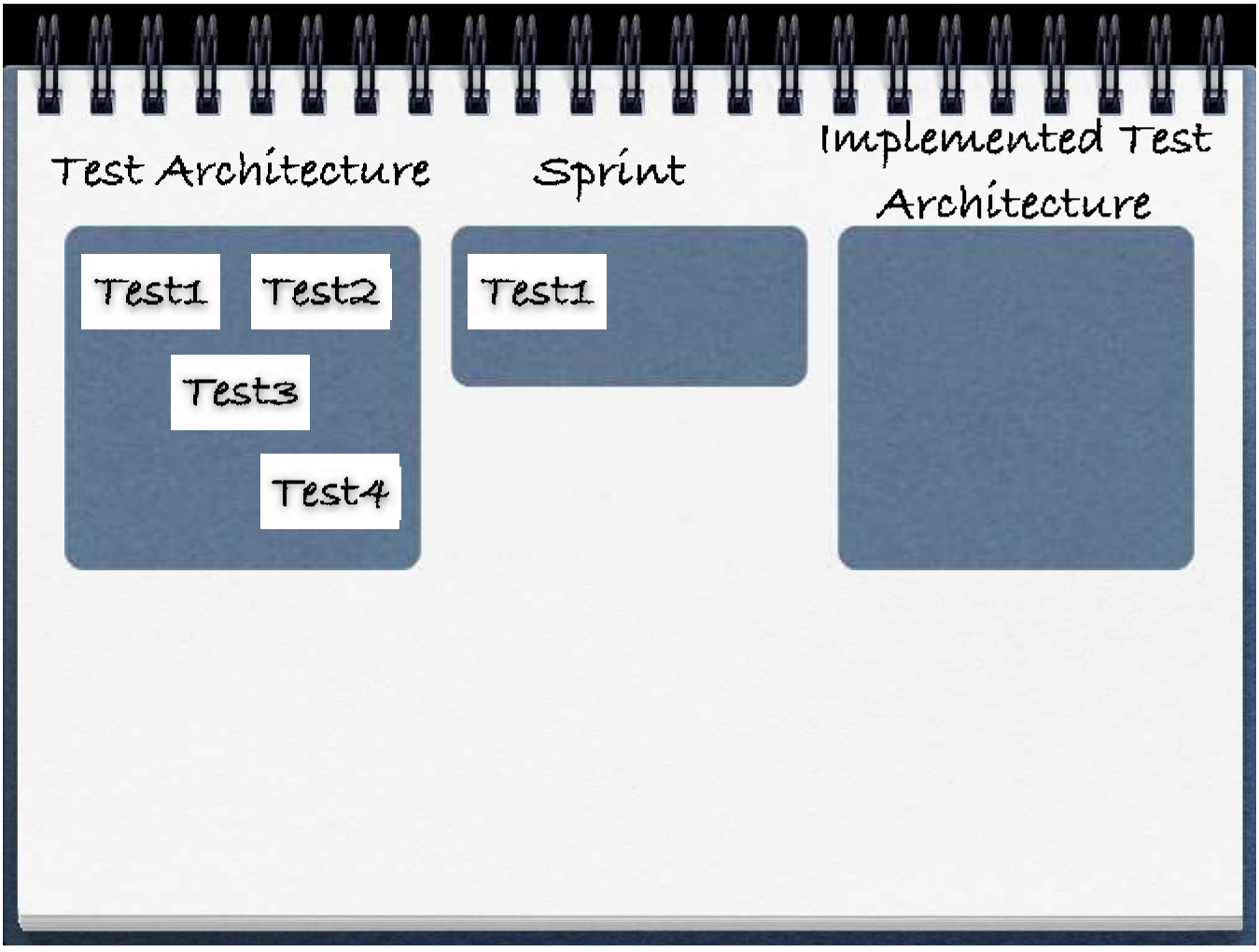

依存関係
品質モデル
リスク
全体の
テスト戦略
Sprintの
テスト戦略
フィーチャ
一部だけだけど
こんなイメージ

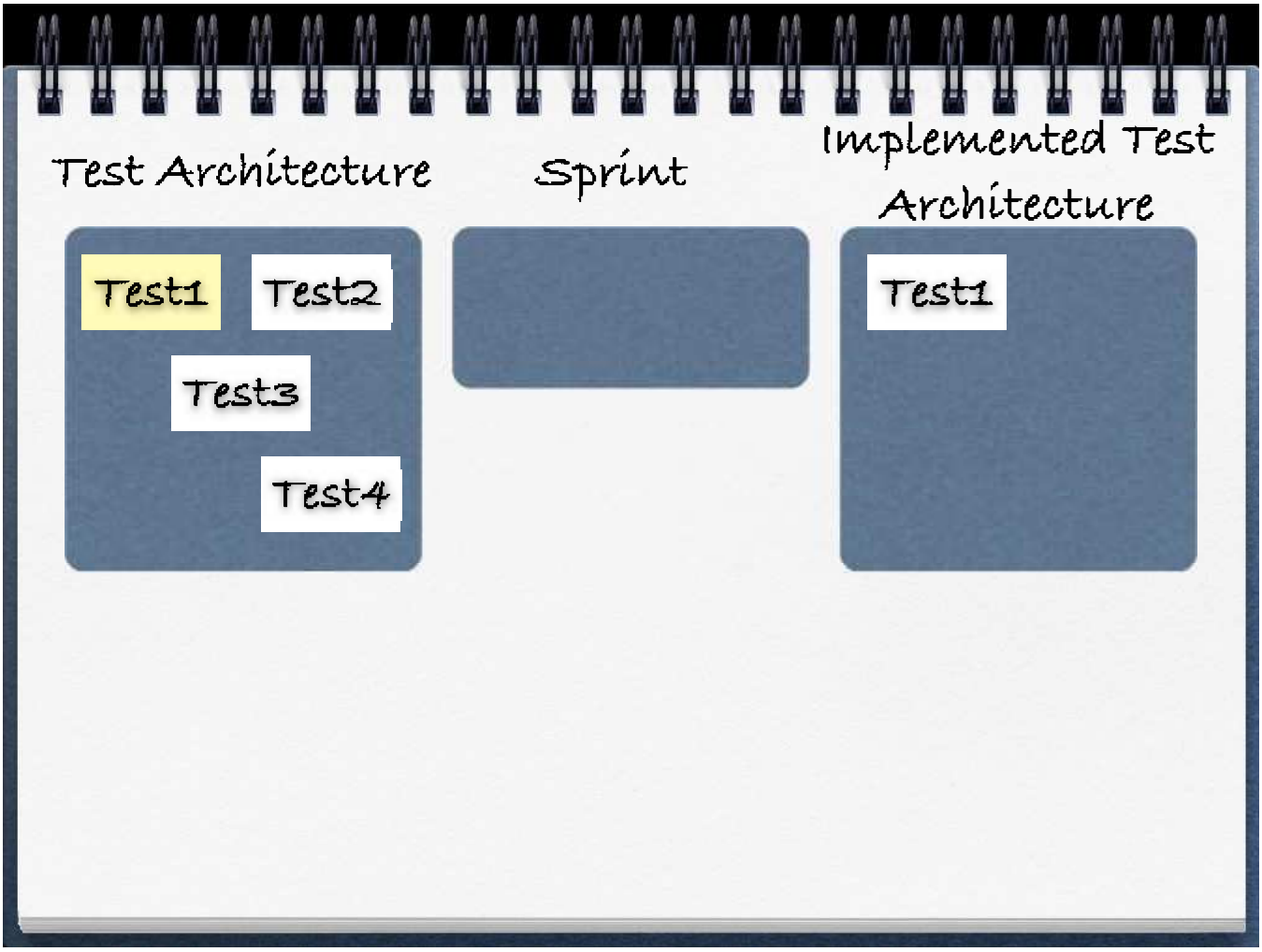

依存関係
品質モデル
リスク
全体の
テスト戦略
Sprintの
テスト戦略
フィーチャ
それぞれがいつでも
想定と異なる可能性がある

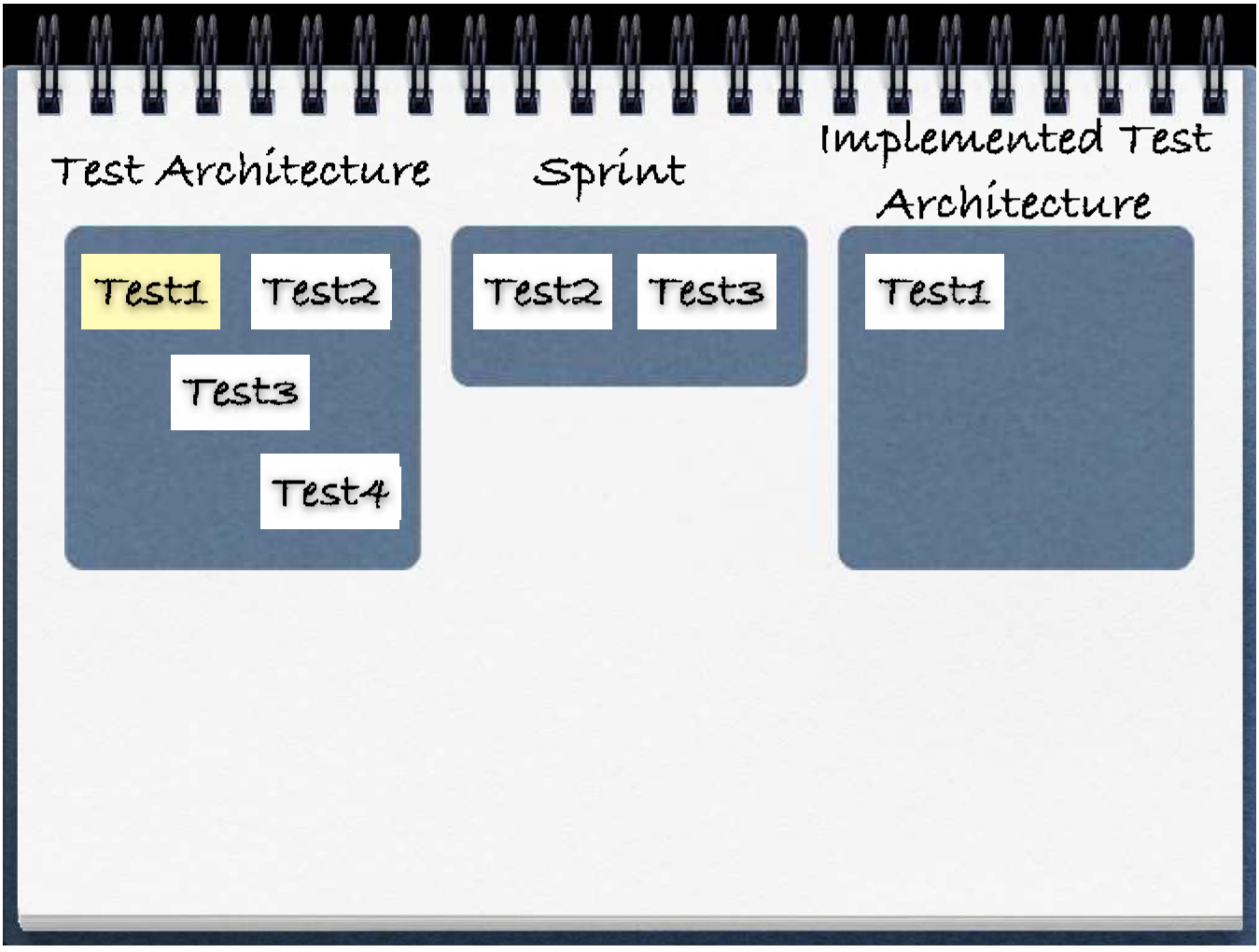

依存関係
品質モデル
より軽く
より早く
より観測しやすく
リスク
全体の
テスト戦略
Sprintの
テスト戦略
フィーチャ
それぞれがいつでも
想定と異なる可能性がある

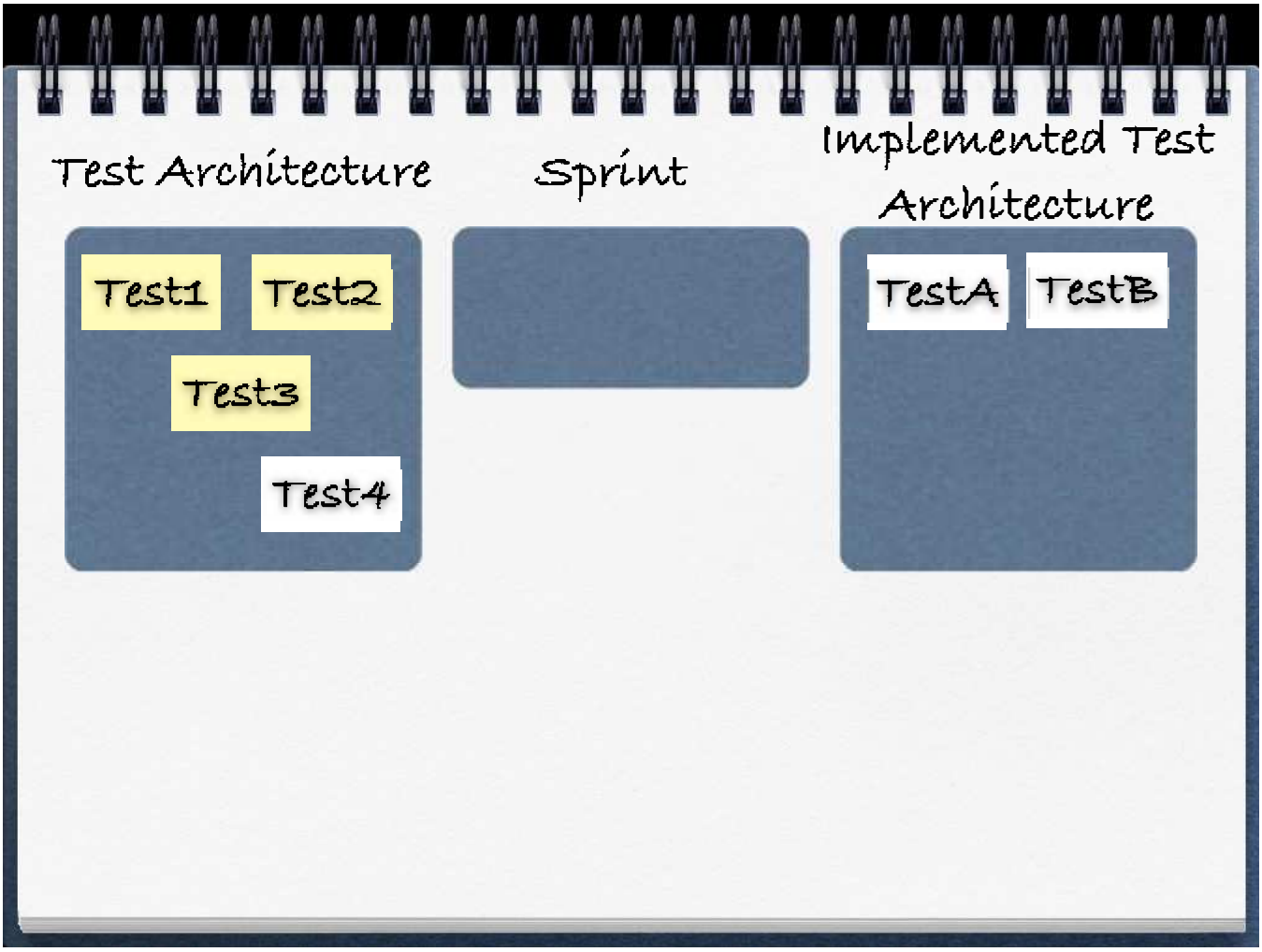

依存関係
品質モデル
より軽く
より早く
より観測しやすく
リスク
全体の
Sprintの
テスト戦略
作る事が目的になってると達成しづらく
変更に弱いガラクタになる
フィーチャ
それぞれがいつでも
想定と異なる可能性がある

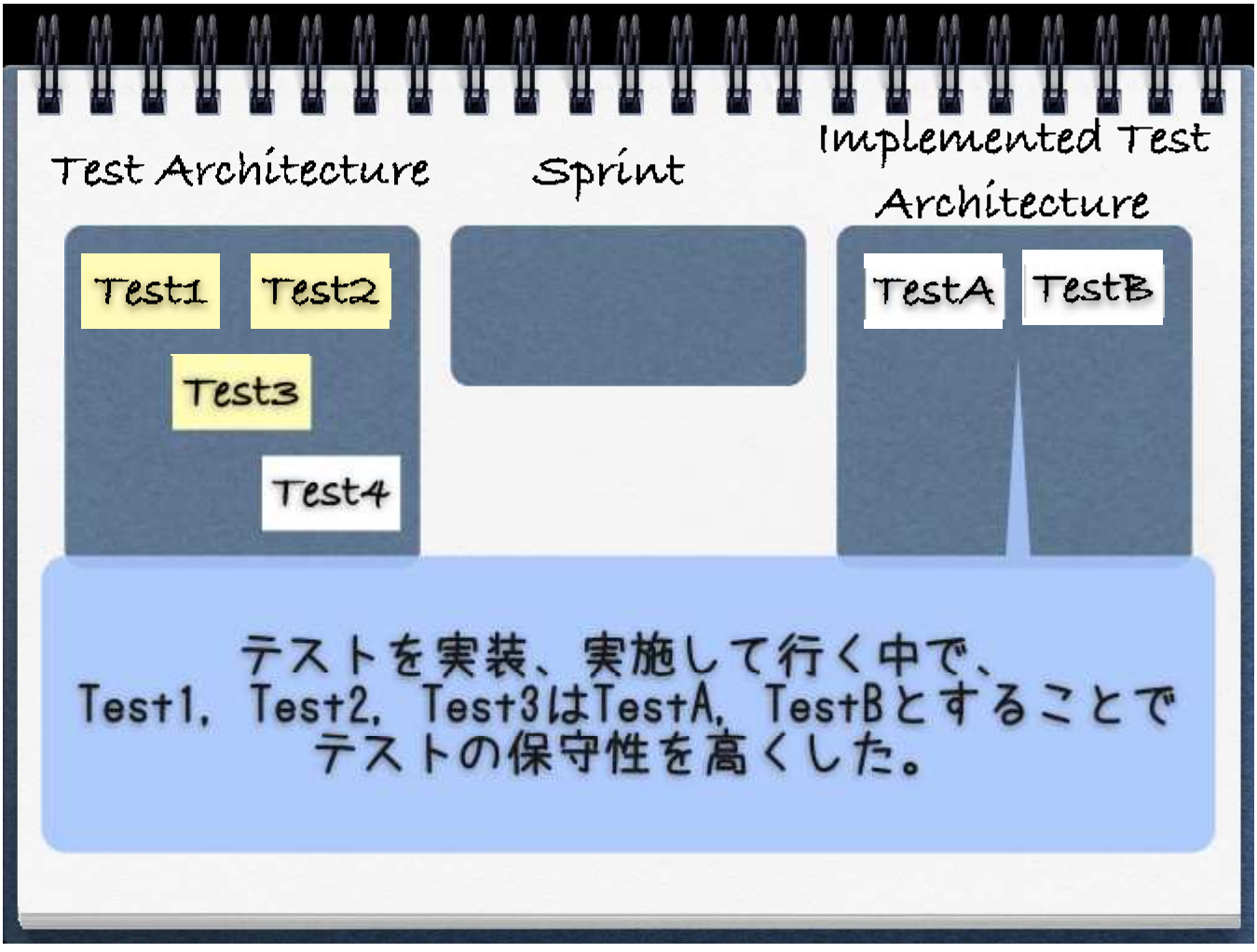

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4

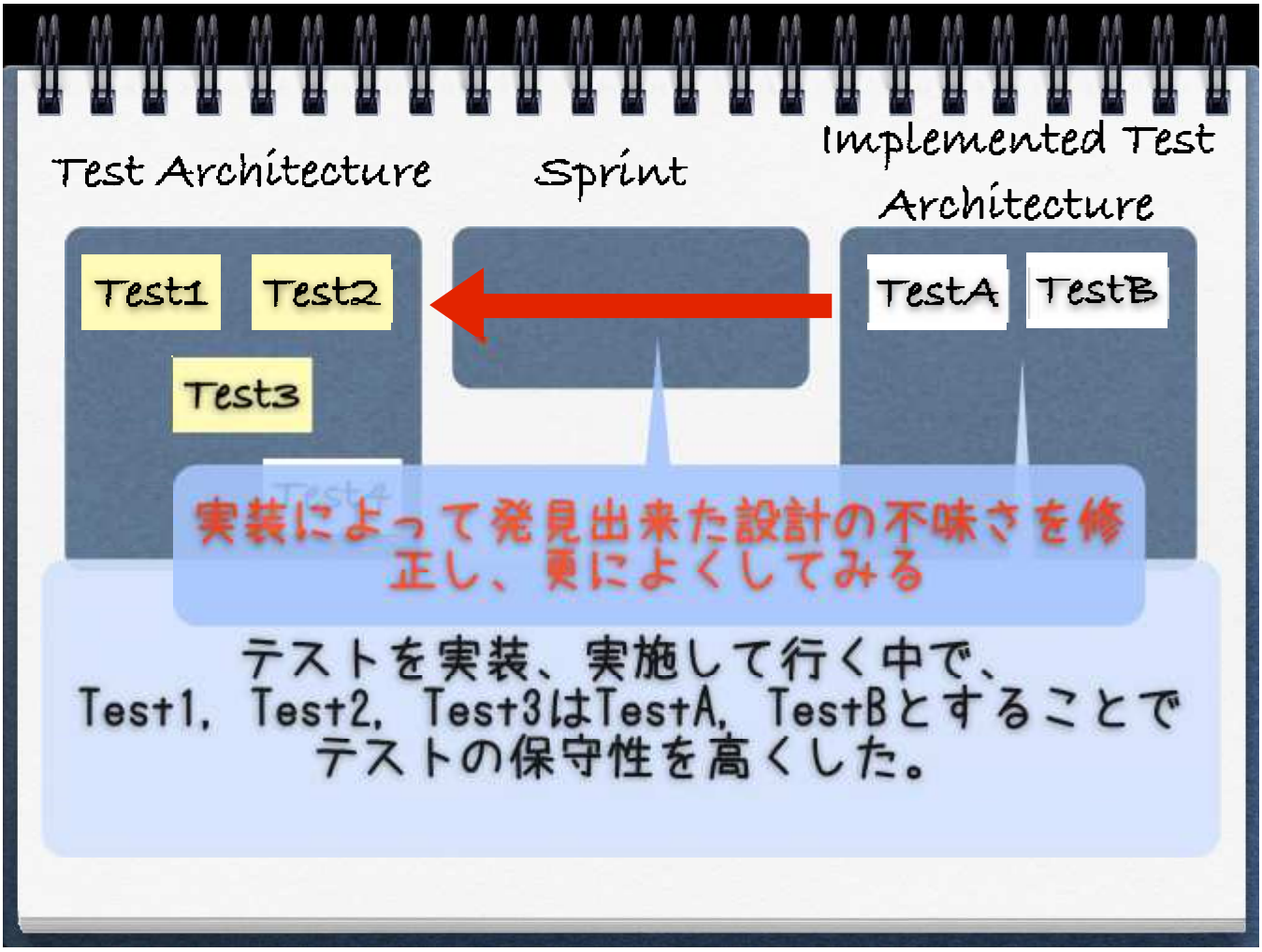

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
Test1

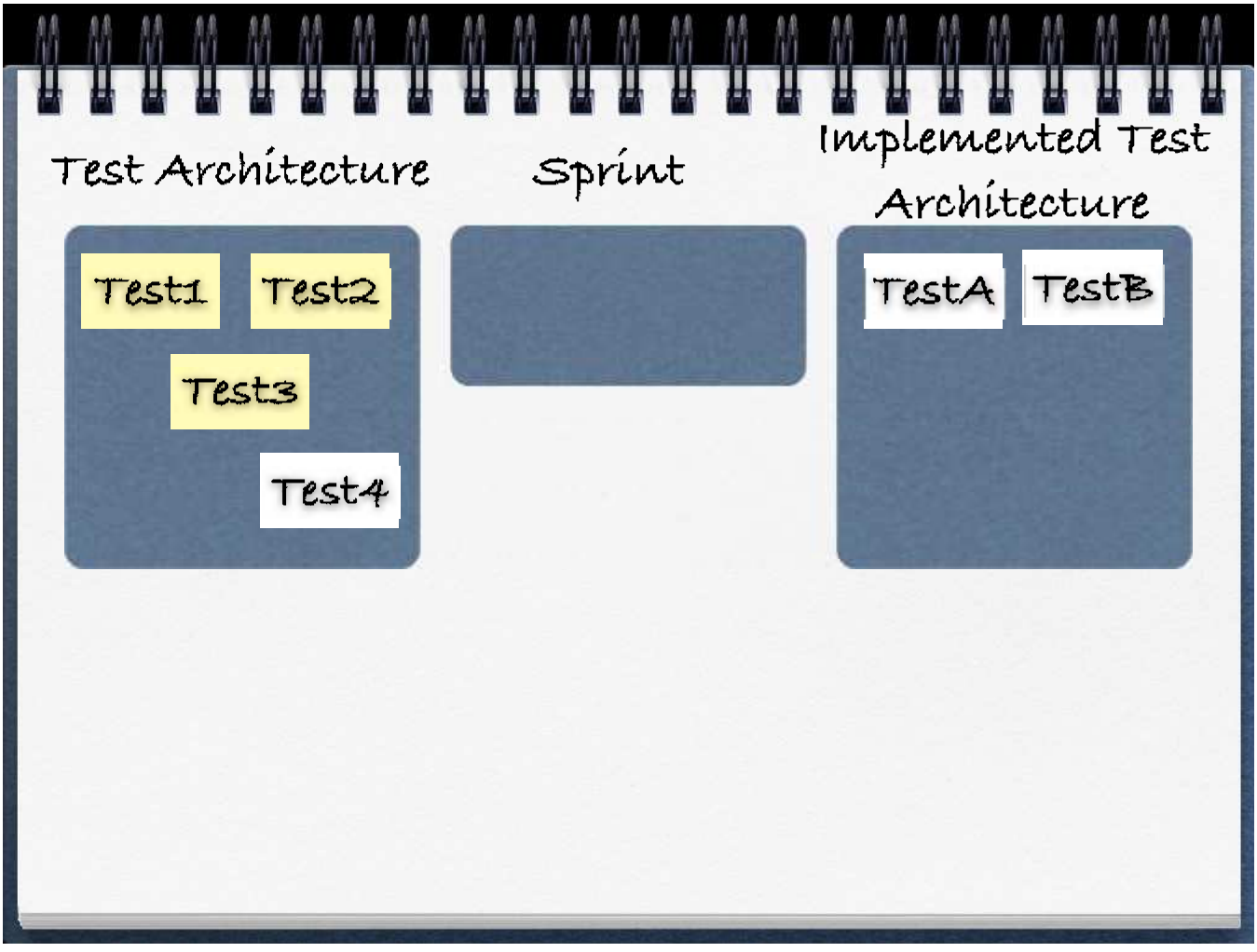

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
Test1

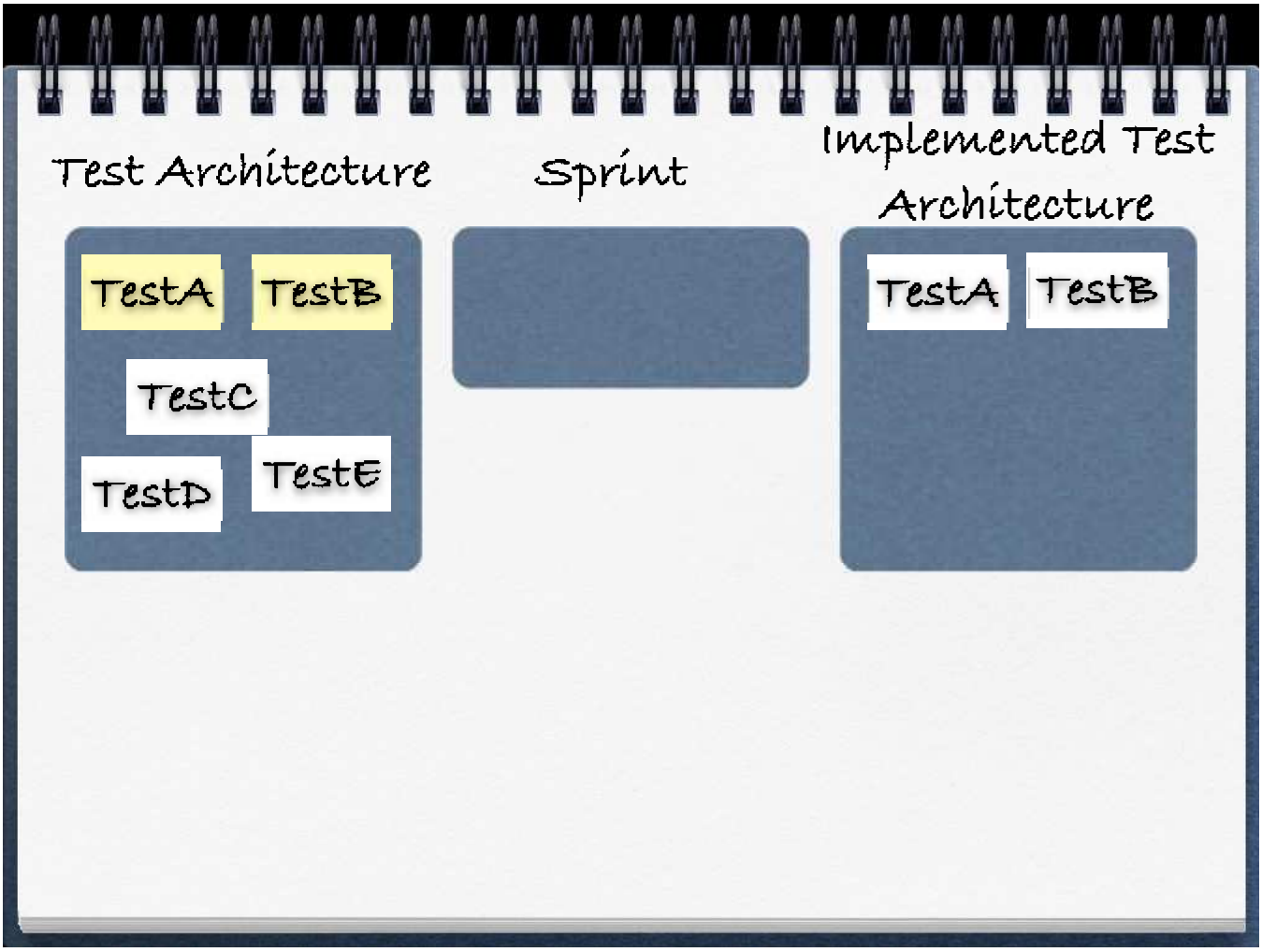

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
Test2
Test3
Test1

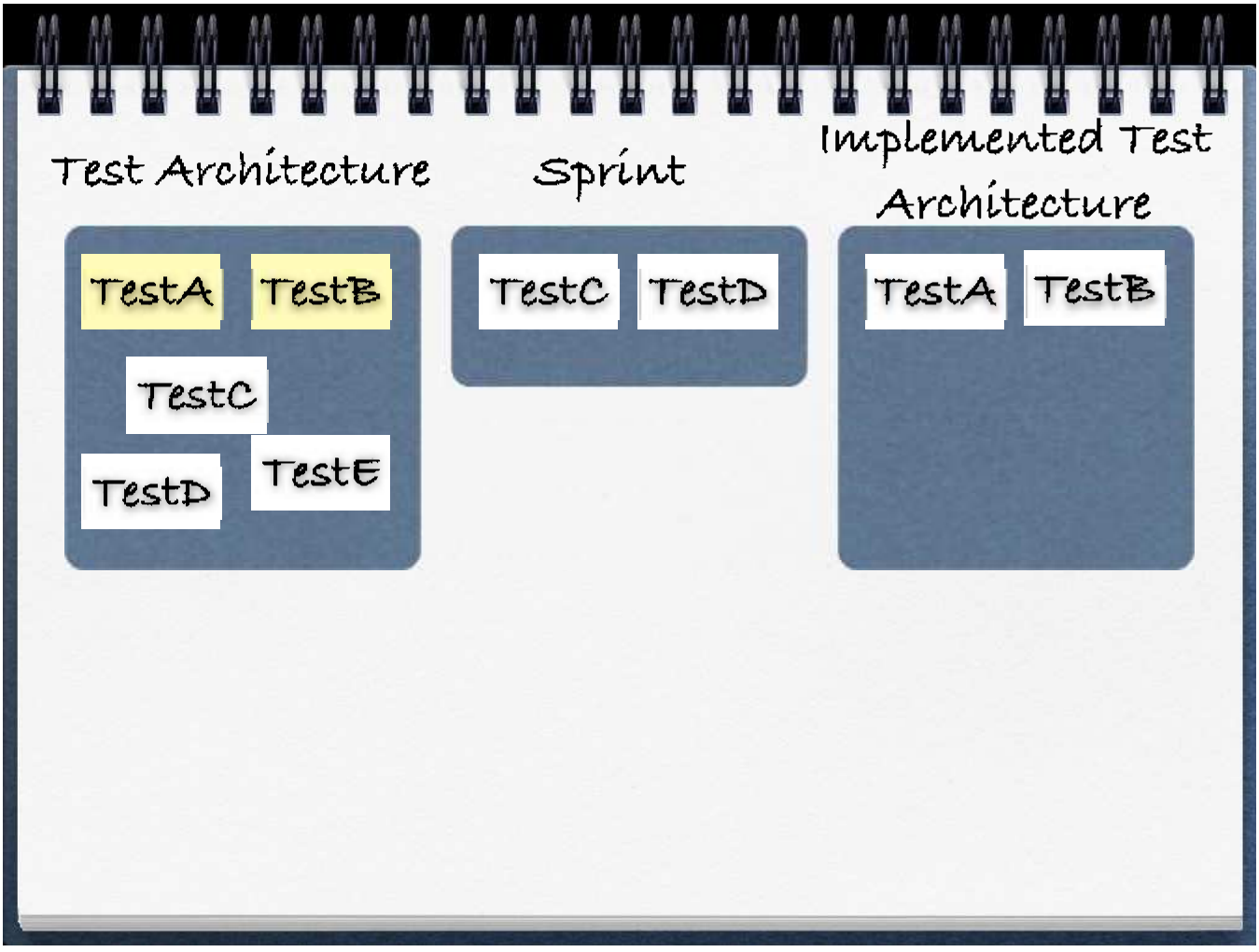

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
TestA
TestB

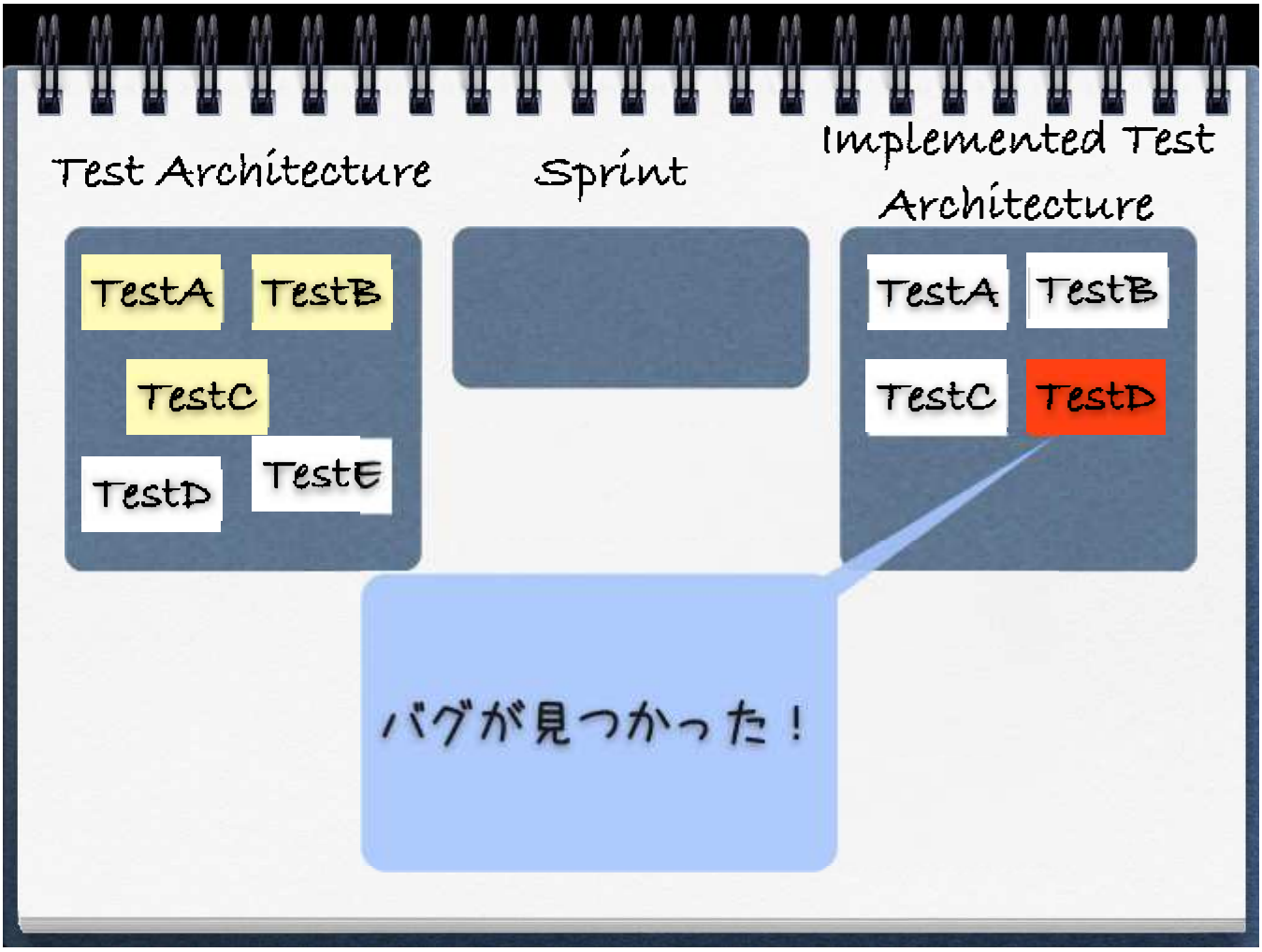

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
TestA
TestB
テストを実装、実施して行く中で、
Test1, Test2, Test3はTestA, TestBとすることで
テストの保守性を高くした。

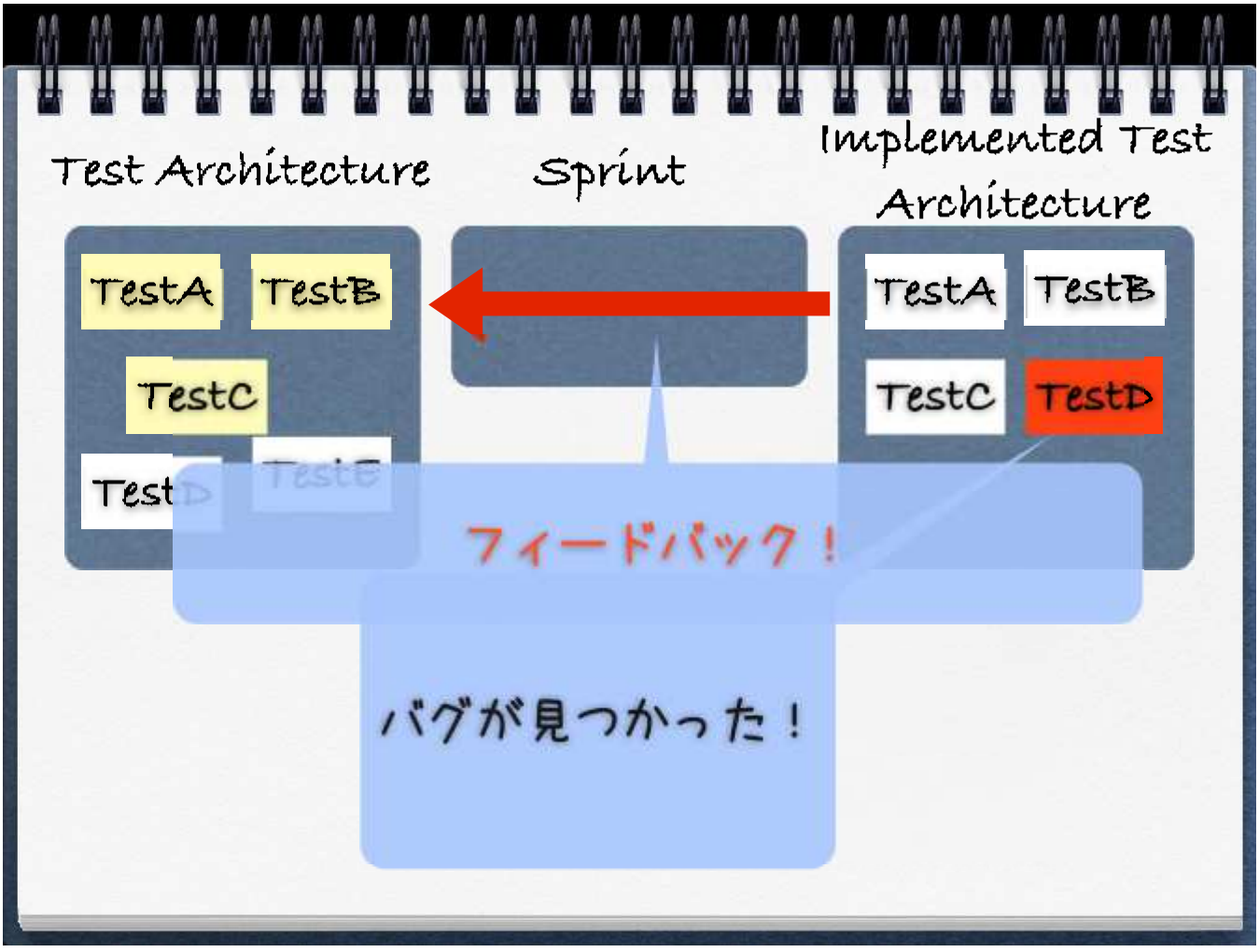

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
TestA
TestB
実装によって発見出来た設計の不味さを修正し、更によくしてみる
テストを実装、実施して行く中で、
Test1, Test2, Test3はTestA, TestBとすることで
テストの保守性を高くした。

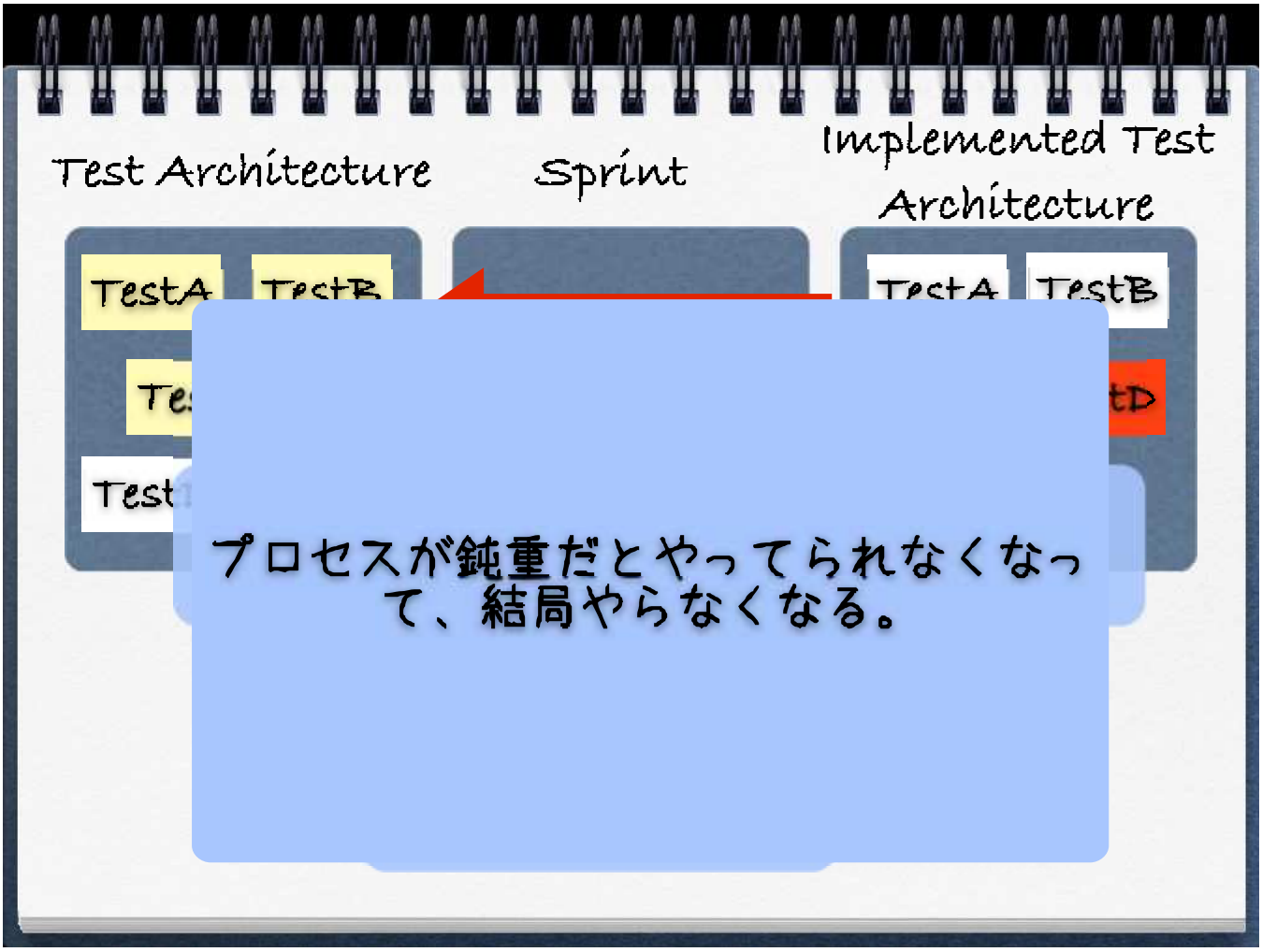

Test Architecture
Sprint
Implemented Test Architecture
Test1
Test2
Test3
Test4
TestA
TestB

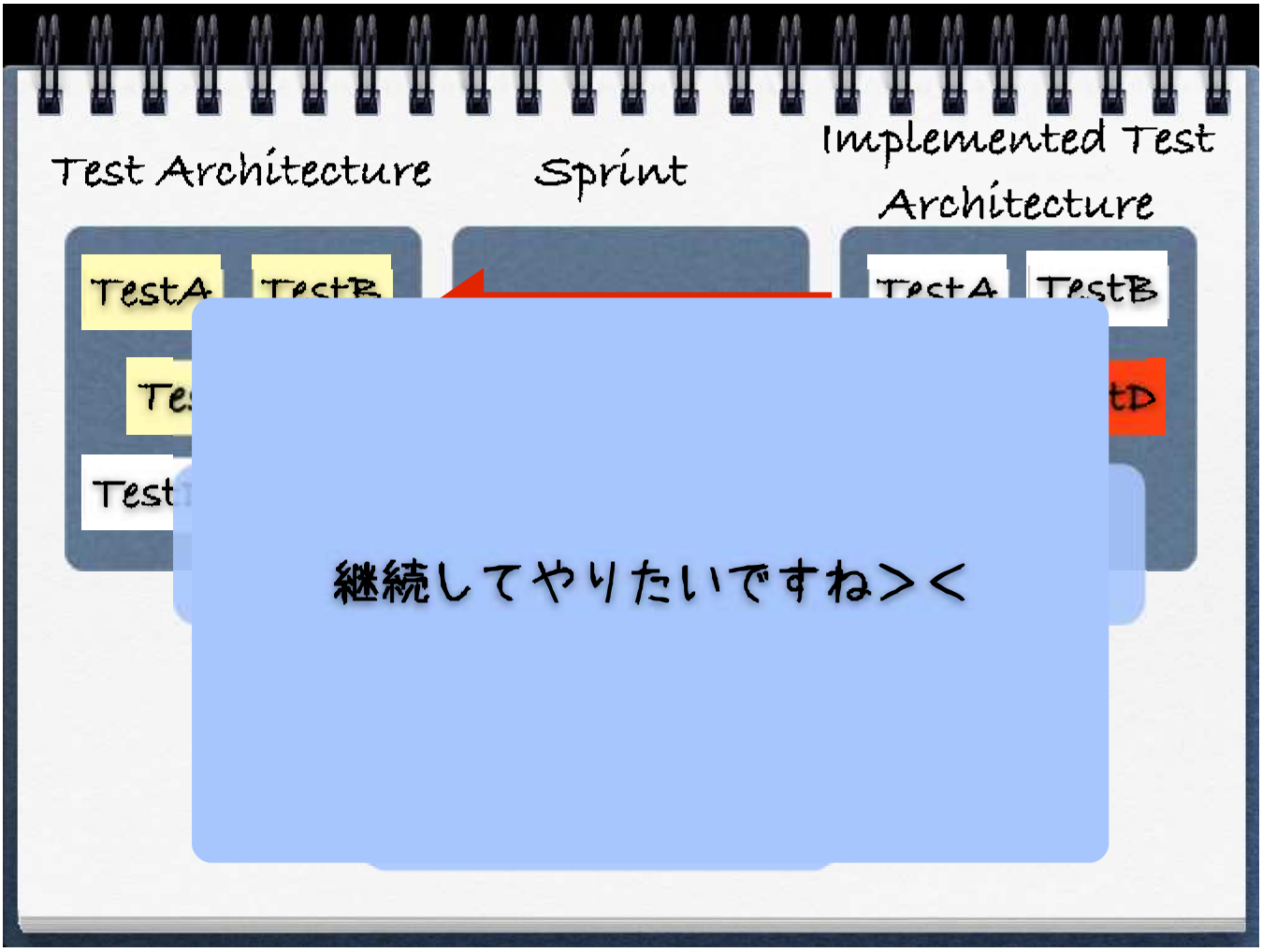

Test Architecture
Sprint
Implemented Test Architecture
TestA
TestB
TestC
TestD
TestE
TestA
TestB


Test Architecture
Sprint
Implemented Test Architecture
TestA
TestB
TestC
TestD
TestE
TestC
TestD
TestA
TestB


Test Architecture
Sprint
Implemented Test Architecture
TestA
TestB
TestC
TestD
TestE
TestA
TestB
TestC
TestD
バグが見つかった！


Test Architecture
Sprint
Implemented Test Architecture
TestA
TestB
TestC
TestA
TestB
TestC
TestD
フィードバック！
バグが見つかった！


Test Architecture
Sprint
Implemented Test Architecture
TestA
TestB
プロセスが鈍重だとやってられなくなって、結局やらなくなる。


Test Architecture
Sprint
Implemented Test Architecture
TestA
TestB
継続してやりたいですね><

Agenda

- チームコラボ
- 見積もり
- まとめ

まとめ

Problem

- テスト観点ってなに？
- テストの見積もりが難しい。。。
- 品質ってなに？
- テストはどうやって区切るの？
- どこまでテストすればいいの？

Problem

- テスト観点ってなに？
- テストの見積もりが難し
- 品質ってなに？
- テストはどうやって区切
- どこまでテストすればい

認識、管理しやすい単位の定義を与える事で、使い易い言葉にして、テストに対する意識を増やした

Problem

- テスト観点ってなに？
- テストの見積もりが難しい。。。

「相対見積もり」によって
ざっくりとしか出来ない範囲
「調査的テスト + Velocity」によって
徐々に正確にできる範囲にわけた。

Problem

QualityModelとしてISO9126 + マインドマップを使ってチームでミーティングする機会を必ずつくるようにすることで、正しさより気軽に考えられるようにして意識を向けた

- 品質ってなに？
- テストはどうやって区切るの？
- どこまでテストすればいいの？

Problem

テスト観点、フィーチャ単位などを実施した。
実際にはなにを知りたいか次第。
品質に直結するなら品質モデルベースで区切る。
開発と同じ単位で知りたいなら開発対象単位。
制約ややりやすさで変わる。

- テストはどうやって区切るの？
- どこまでテストすればいいの？

Problem

「リリースのDone」はテストするときではなく、最初に共有しておき徐々に修正していく。
範囲や組合せを考えるが、基本的には常に優先順位をつける。

- どこまでテストすればいいの？

続けたいこと

- 品質モデルの共有
- チームのレビュー
- 相対見積もりの活用

課題

- 複数のテストエンジニアが関わったときにうまく共有できない。（Test Boardなどはまだチームで常に共有できるほど完成度が高くない

出来そうなこと

- テスト観点のネスト構造について実践してみる。（テストバスケット？
- テスト観点のリファクタリングや設計パターンの構築

気付いた事

- Flowの最初につくるテストアーキテクチャは自分が思った以上に作り込む必要がない
- 今回の例だとテスト観点モデルは徐々に成長させることになる。
  独立性と合成性が高いテスト観点を考える事が重要。

ご清聴ありがとうぴょん◆