潜在不具合炙り出しテスト
～テストは計画的に～

中岫　信
東京エレクトロン ソフトウェア・テクノロジーズ(株)

東京エレクトロン

目次

- 背景
- 炙り出しプロジェクト
  - テスト計画
  - テスト設計以降
- まとめ

背景

悩み

提案

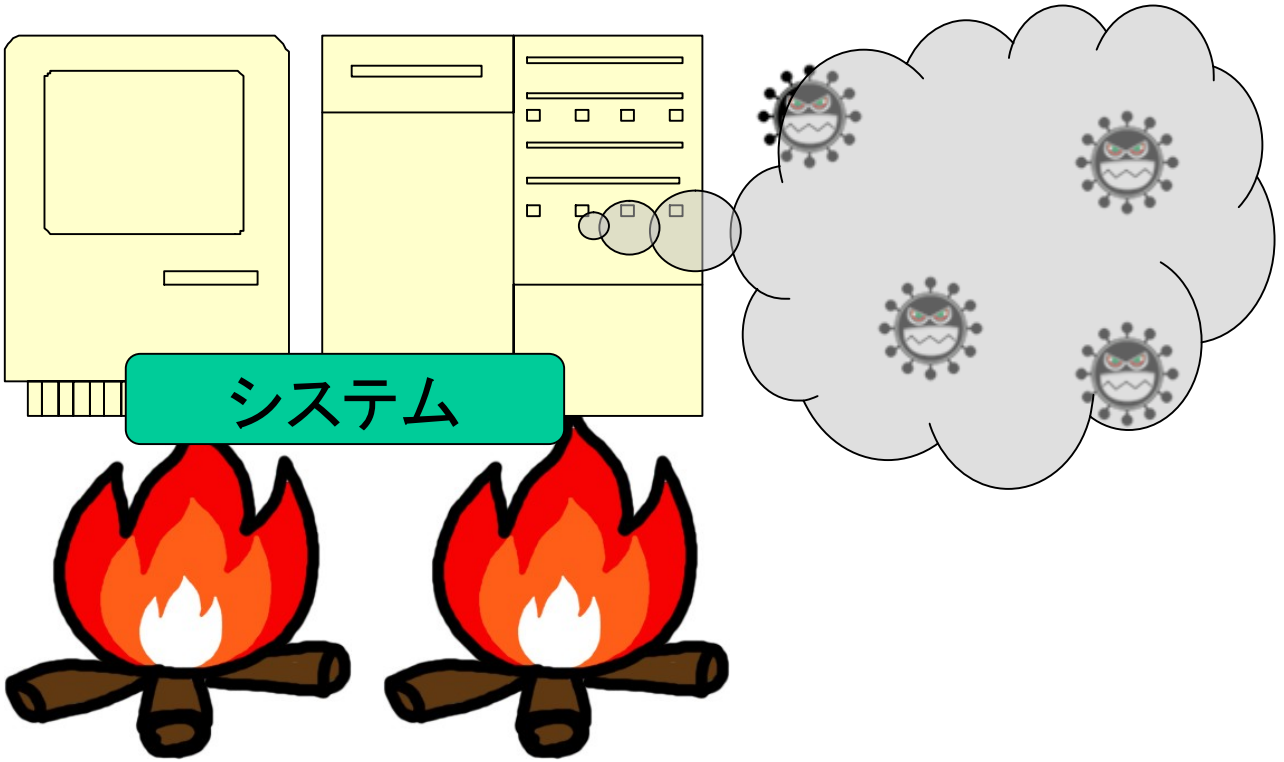
少なくてもシステムが止まるような
重要な不具合は解消したい

今以上に運用面を
考慮したテストが必要だ！！

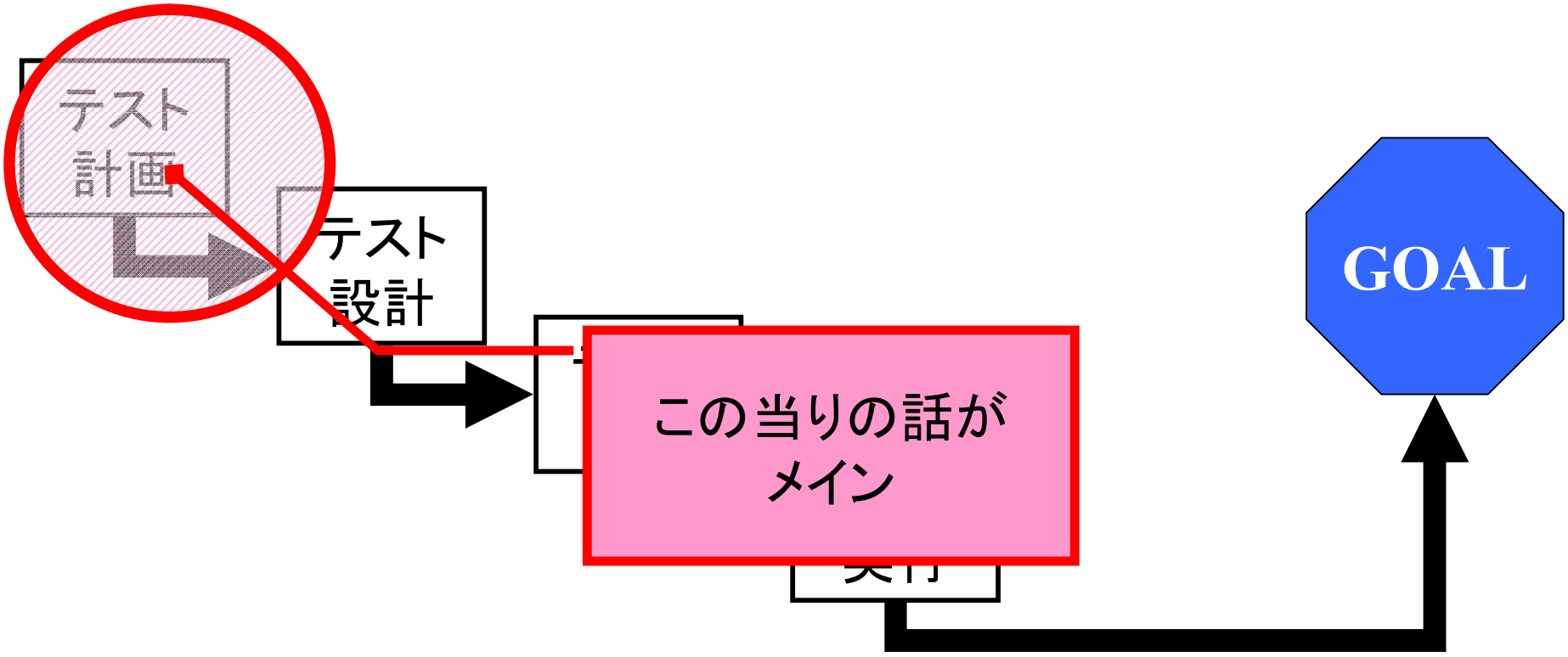
炙り出しプロジェクトの始動

炙り出しプロジェクトとは？

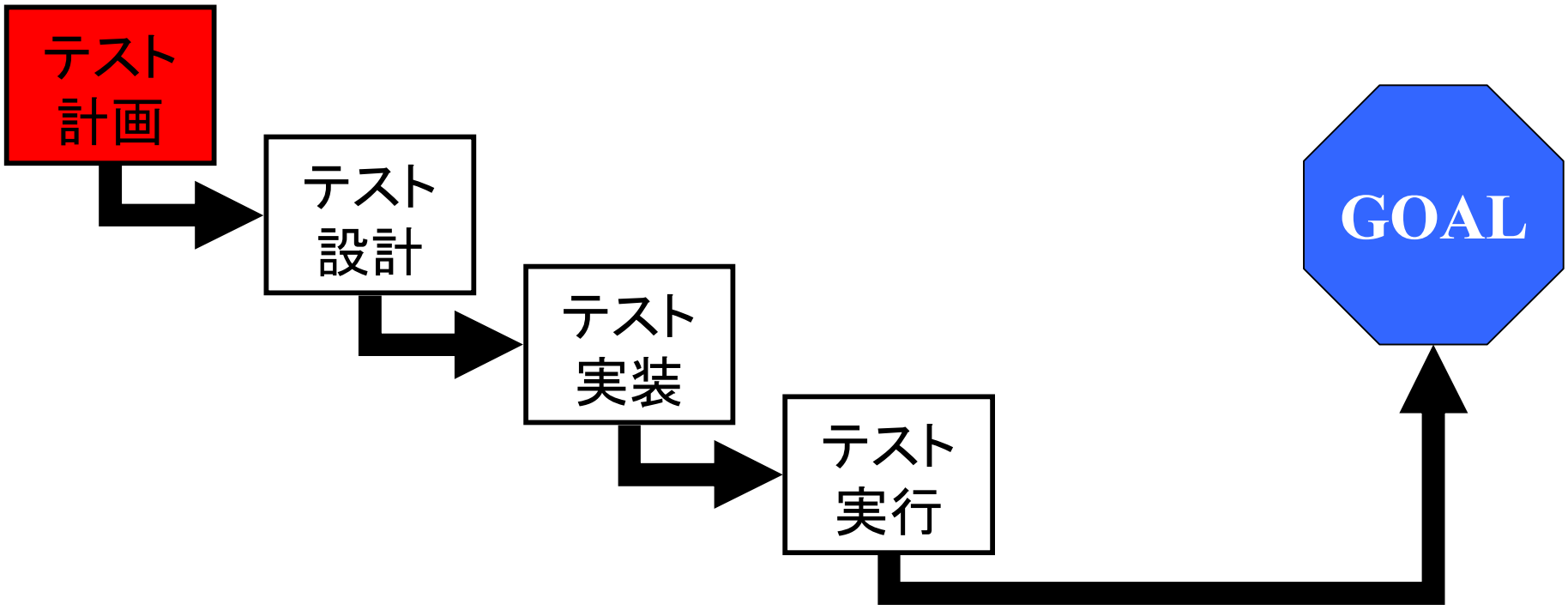
- システムに潜在している運用に関わる不具合を
  テストによって炙り出すプロジェクトのことである。



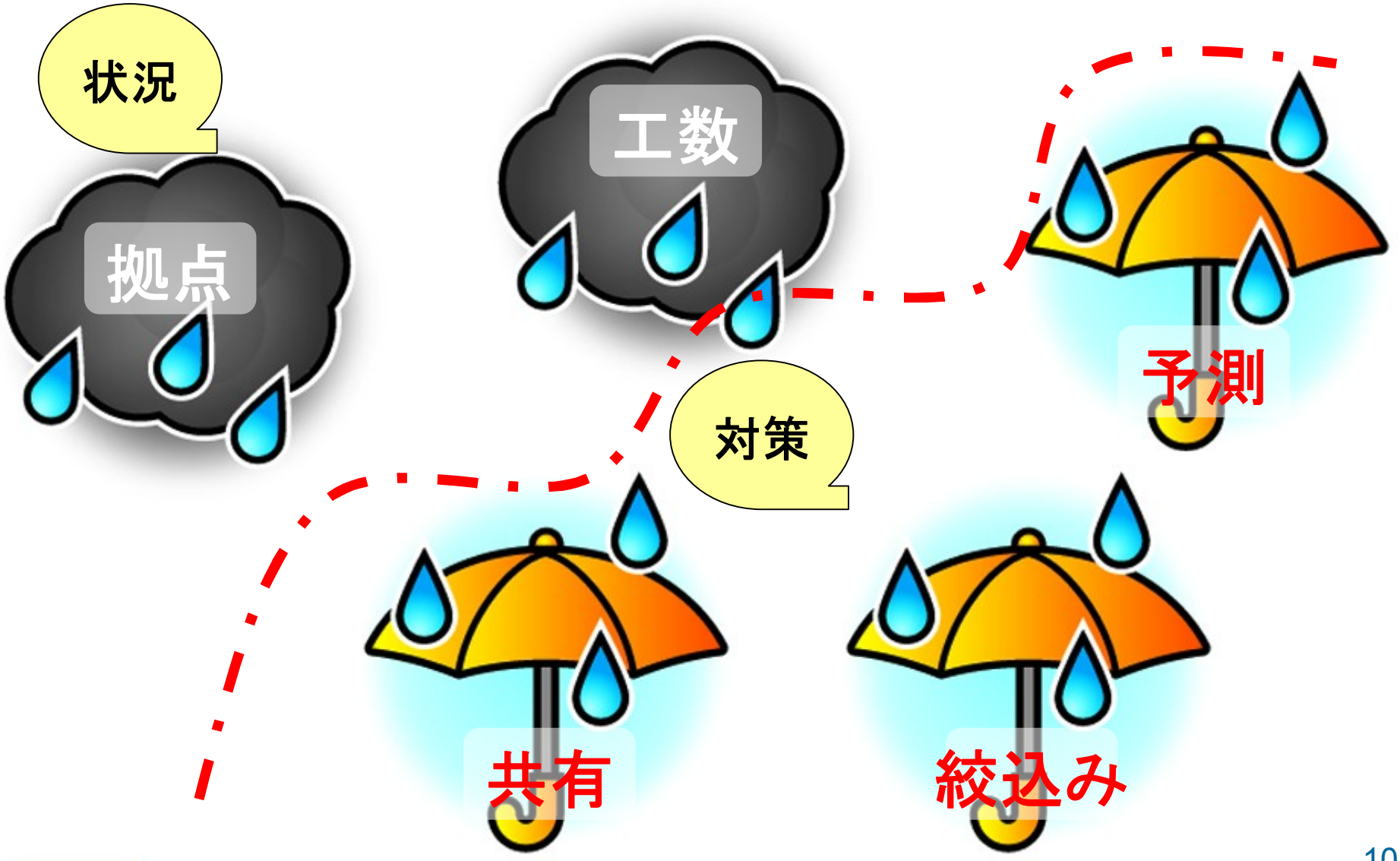
想定する大まかな
作業の流れ



まずは
テストの計画



状況と対策



実施可能な量の予測

- プロジェクトに与えられた期間で使用できる工数を算出
- 与えられた期間で消化できるテストケース数を過去データより算出
  - 単位時間当たりのテストケース作成数
  - 単位時間当たりのテスト実行数

  などなど

⇒ これらをスケジュール化

機能の絞り込み

- お客様が良く使用され、かつ、運用インパクト、製品インパクトの大きい機能をピックアップ
- 過去不具合データより、機能と不具合の発生頻度をマッピング
  ※ハザードマップと呼んでます

⇒これらの情報からプライオリティ付け

テストプロセス、手法の共有

- 実行するテストプロセスを定義し、アウトプットのレベルを統一

  ⇒PFDの作成

- 対象の機能に対するテストの内容を合わせるためテスト方法を統一
  - 運用を意識したテスト
  - 炙り出すために対象機能をくまなくテスト

とは言ったものの・・・

いままでだって

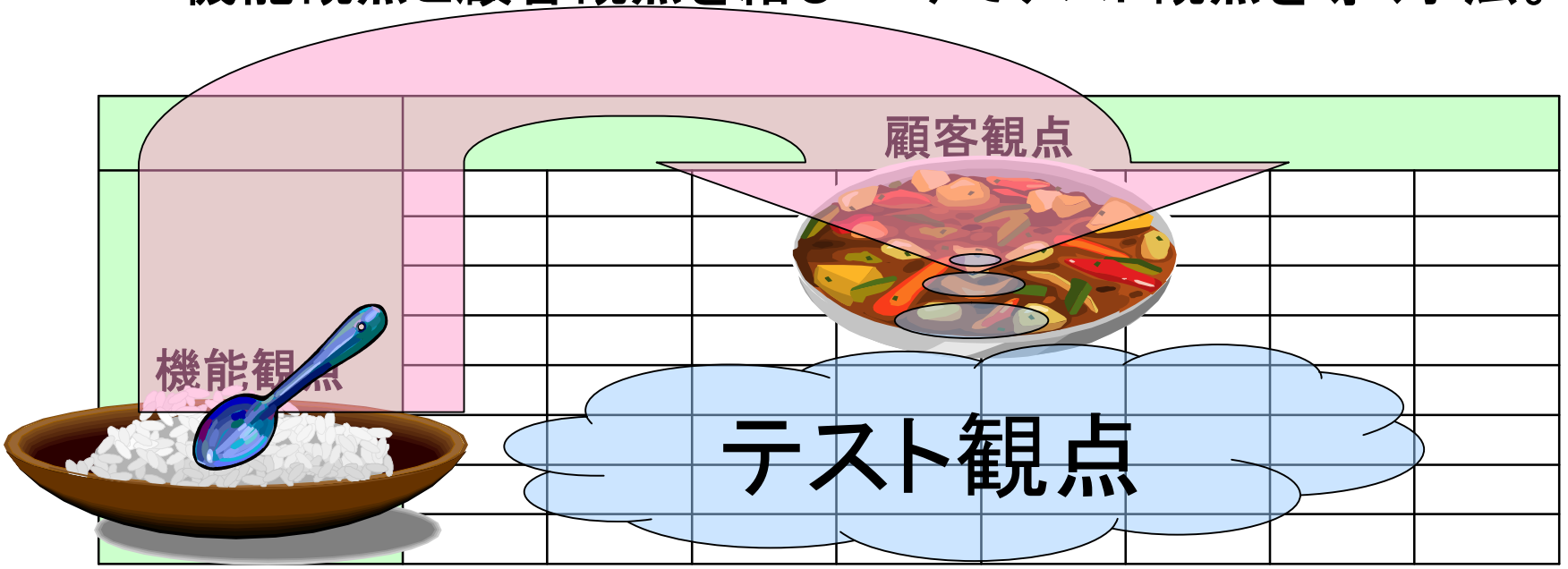

QAでもテストは行っている
いままでと同じテストだと
意味がない
運用面を考慮していないわけではない
網羅的に今までと異なる観点の
テストを導く良い方法はないだろうか？

閃き

※JaSST'09 Hokkaidoで
紹介されたテスト手法

スープカレー表とは？

- 機能観点と顧客観点のマトリクスで、
- 機能観点をご飯、顧客観点をスープに見立て、
- ご飯をスープにくぐらせて食べるスープカレーのように機能観点を顧客観点にくぐらすことで、
- 機能観点と顧客観点を結びつけてテスト観点を導く手法。



初めて取り入れる手法なので

検証

特定の機能に絞って、手法を試用

使える手法なのか？

- 今までとは異なる観点が出てきた
- 使えそうな手法だ

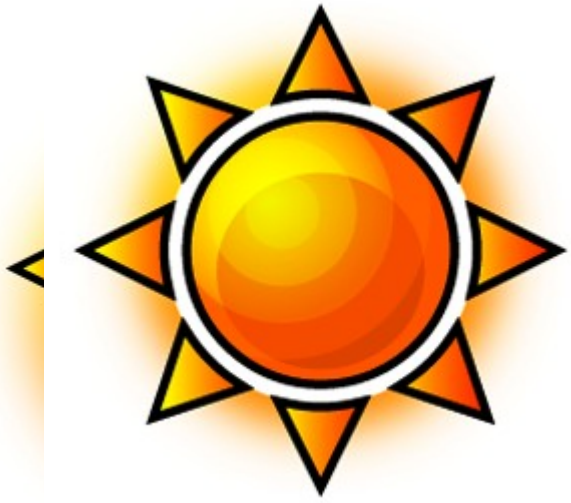
使い方に問題はないか？

- 使い方はわかった
- 手順が人によって異なる
- 観点の幅が人によって異なる

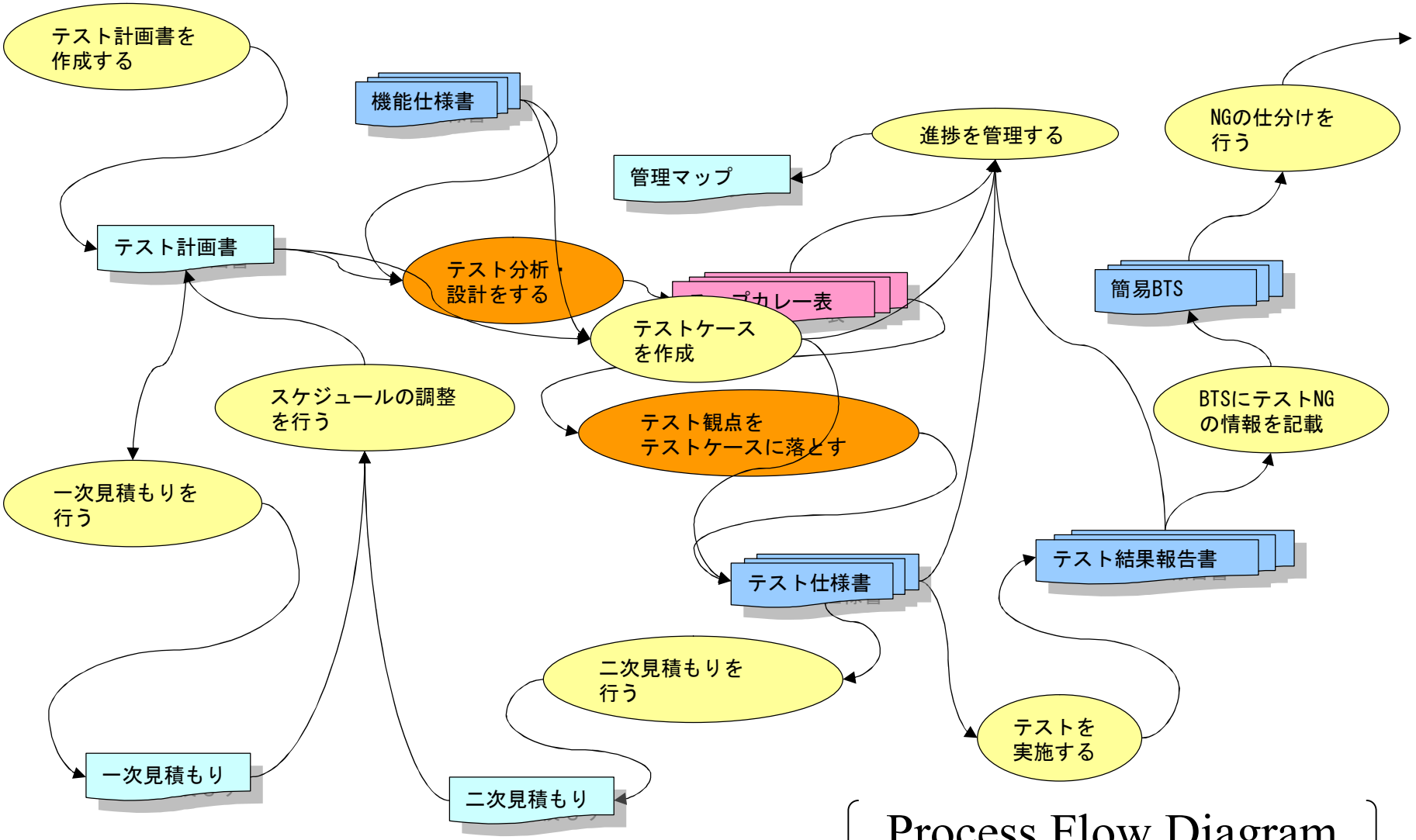
課題の解決へ

- **導入手法の手順を整備とレクチャー**
  - 手順書作成
  - 全員一度経験するまで、特定の担当者が観点出しに参加する
- **観点の幅をあわせるため**
  - 最低限考えるテスト観点を設定
    ⇒スープカレー表をテンプレート化

手法の採用、運用へ



手法を取り入れたプロセス

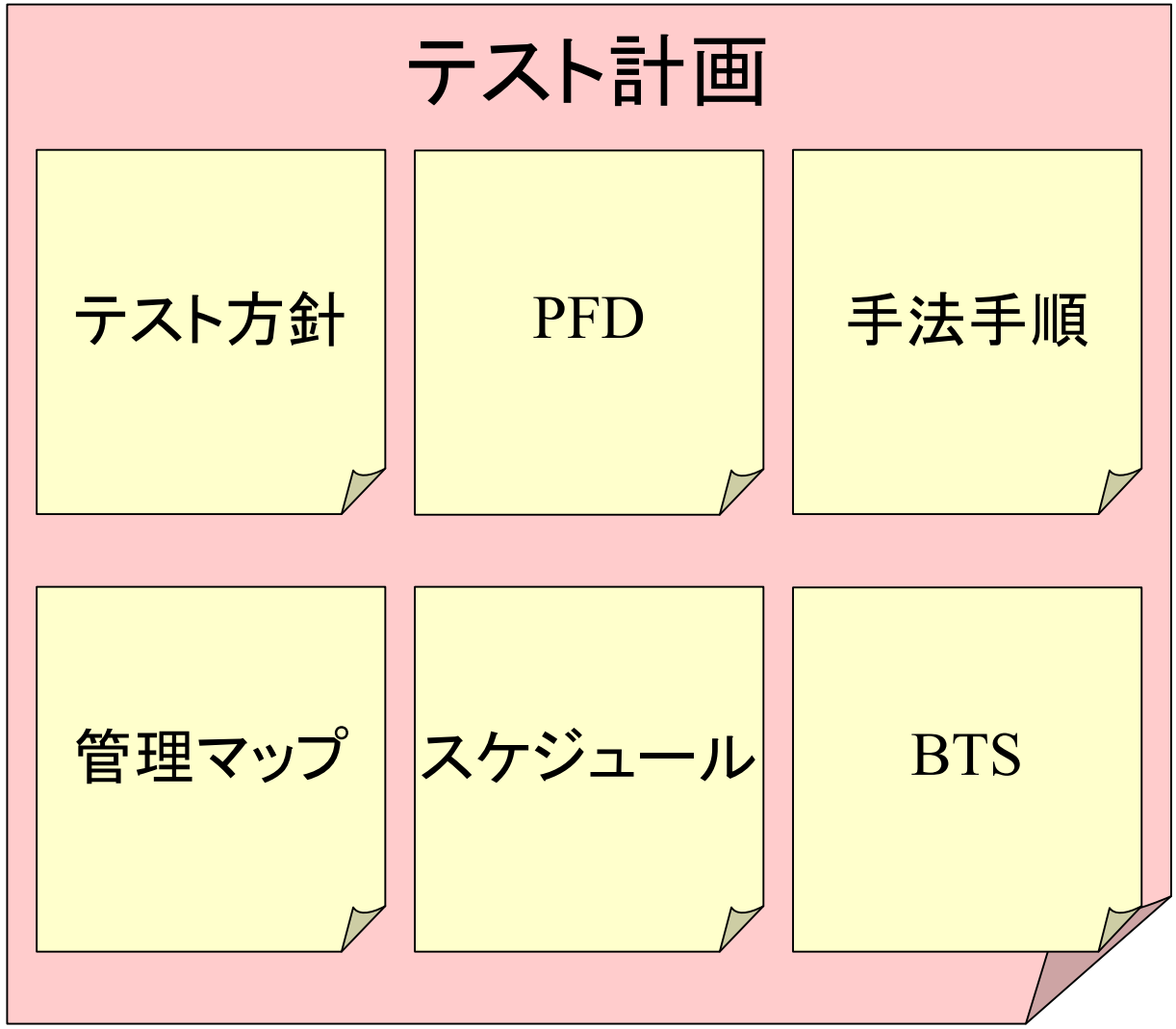


Process Flow Diagram

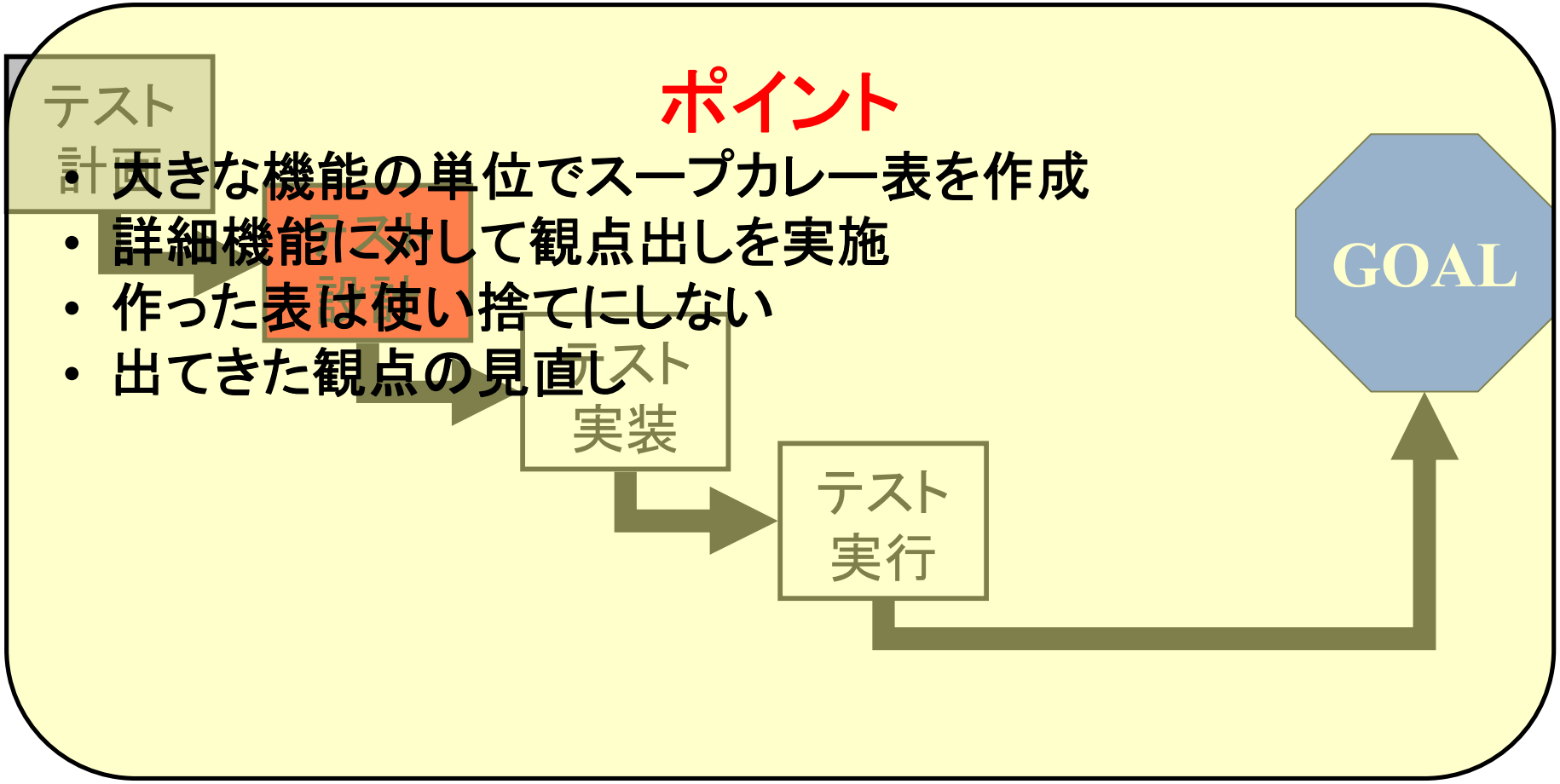
目的、手法をテスト方針として整理

テストレベル	システムテスト
テストタイプ	機能テスト、および、非機能テスト、エラー推測テスト
テスト目的	製品不良を引き起こさず、顧客の製品処理が続けられる。 装置を停止せずに顧客の製品処理が続けられる。 そのためのテストを実施し、問題なく装置が使える事を確認する。
テストベース	機能仕様書
テストウェア	テスト観点・概要(スープカレー表) テスト仕様書 管理マップ テスト結果報告書(顧客提出用含む) 簡易BTS
テスト手法	ブラックボックス スープカレー表
テストリスク	NGが発生した場合は、不具合かどうかの切り分けフェーズにて切り分けを行う。 不具合の内容により顧客インパクトある場合は、修正スケジュールを検討し、顧客報告する。 ただし、重大不具合が発生した場合は、都度緊急取り込みの判断をする。

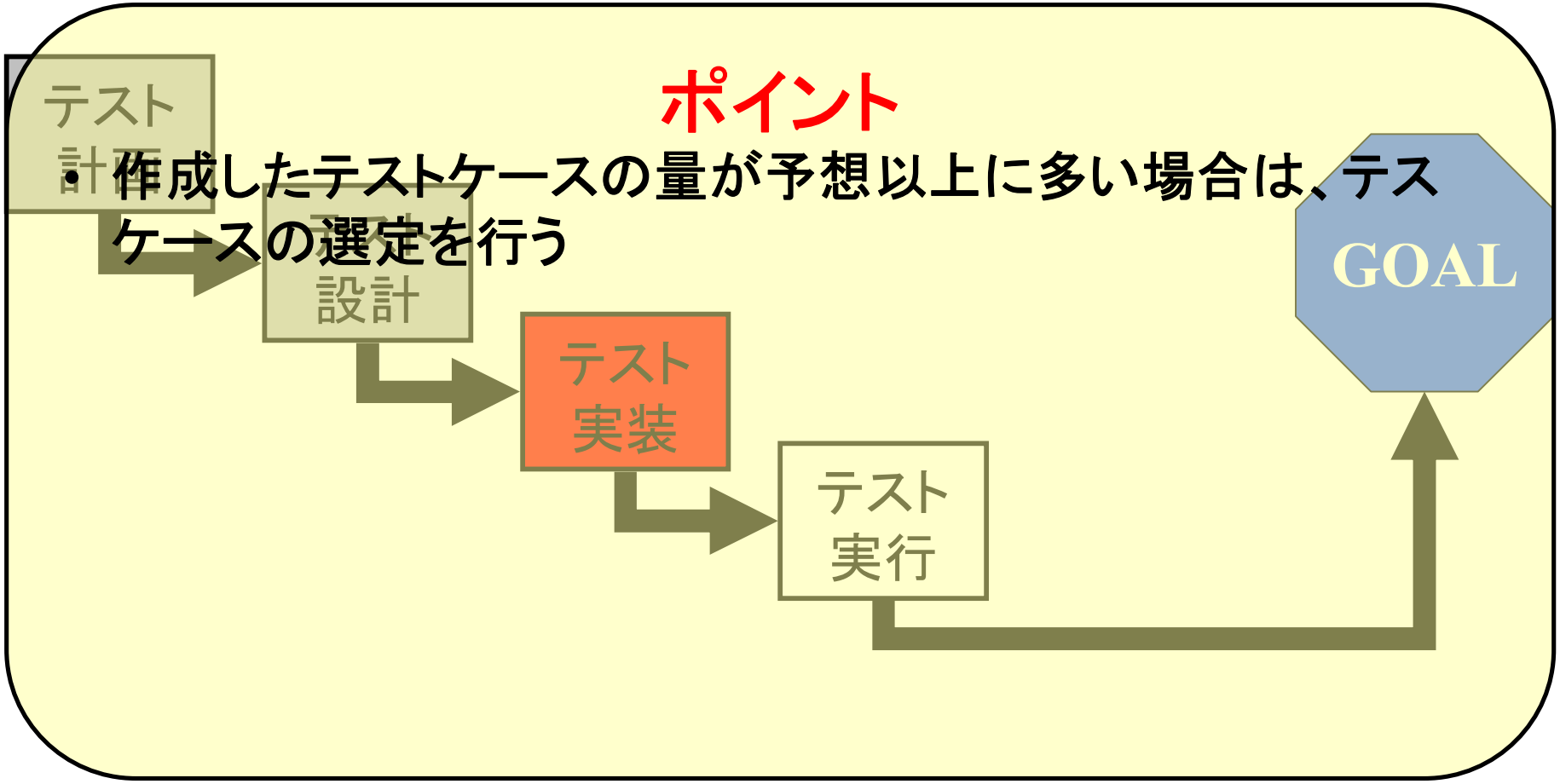
これまでの一式を整理



計画の次は
テストの設計



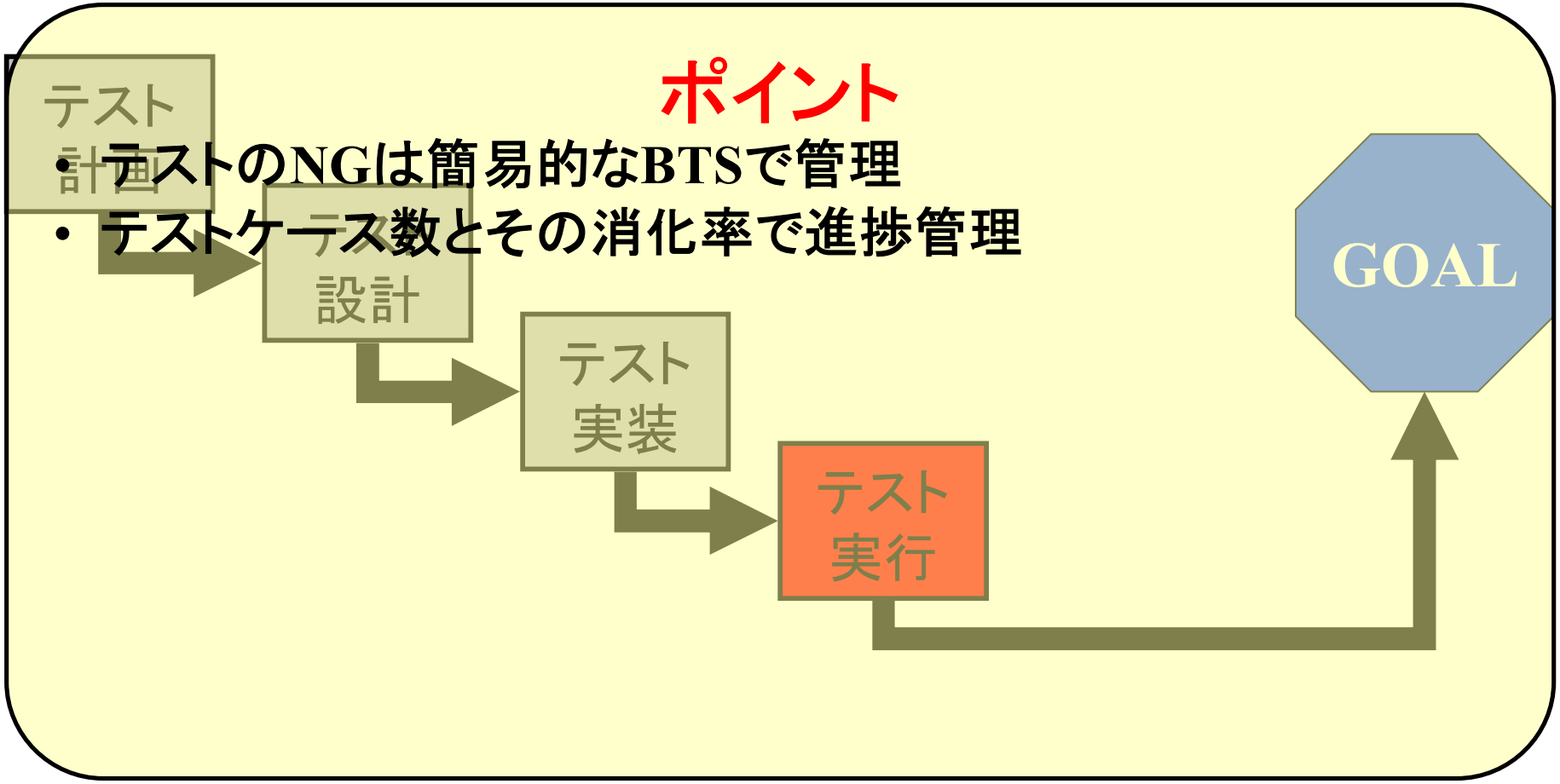
設計したら
テストを実装



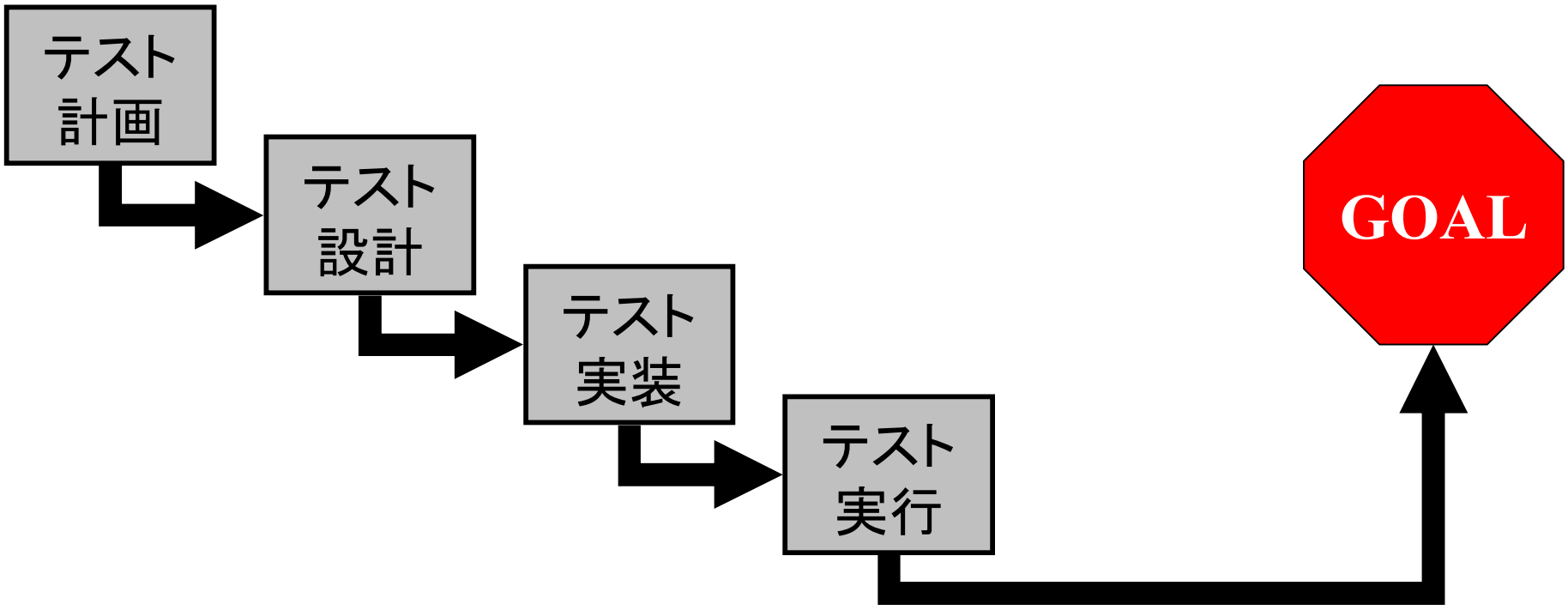
ポイント

- 作成したテストケースの量が予想以上に多い場合は、テスケースの選定を行う

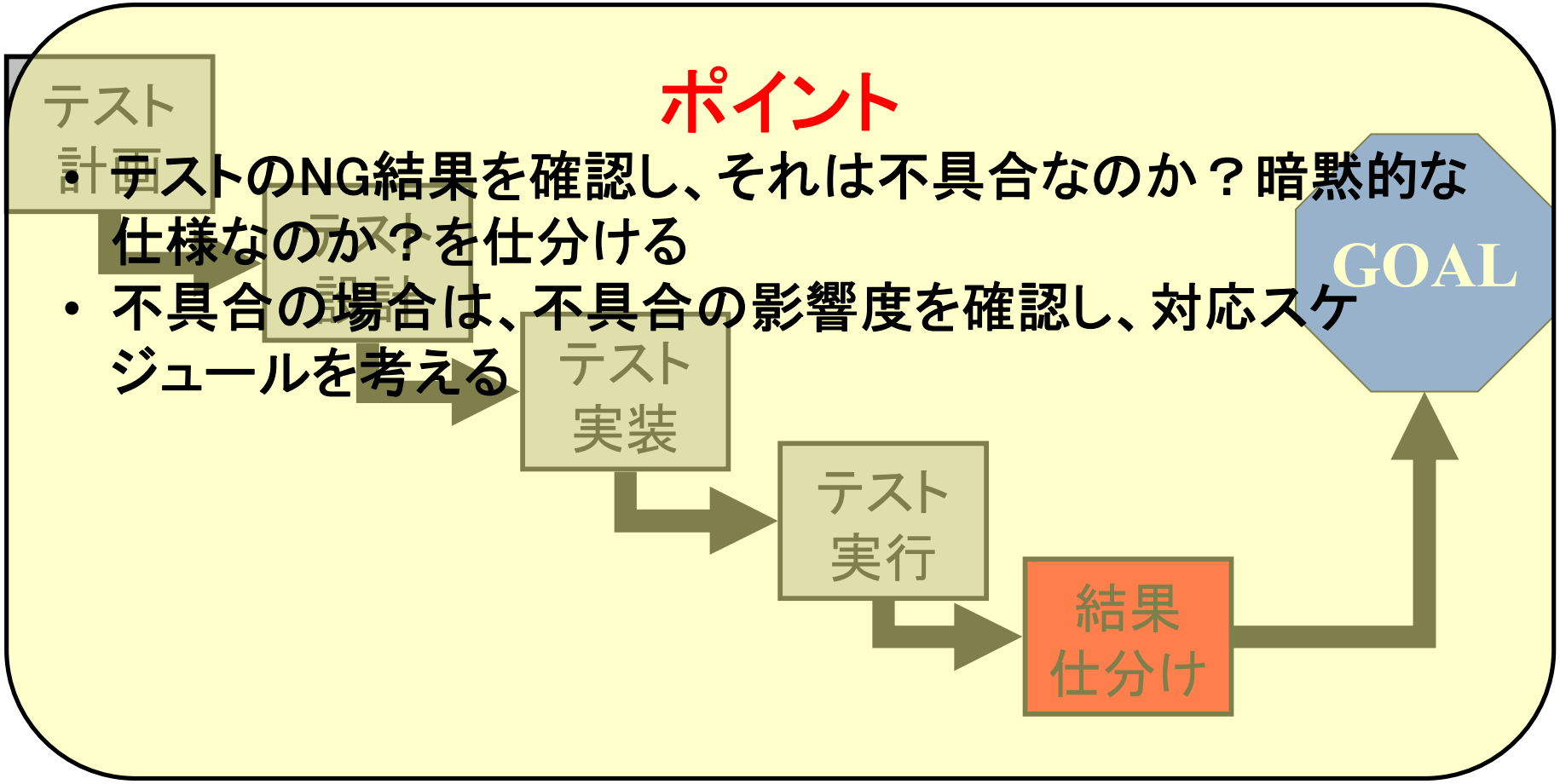
実装できたら
テストを実行



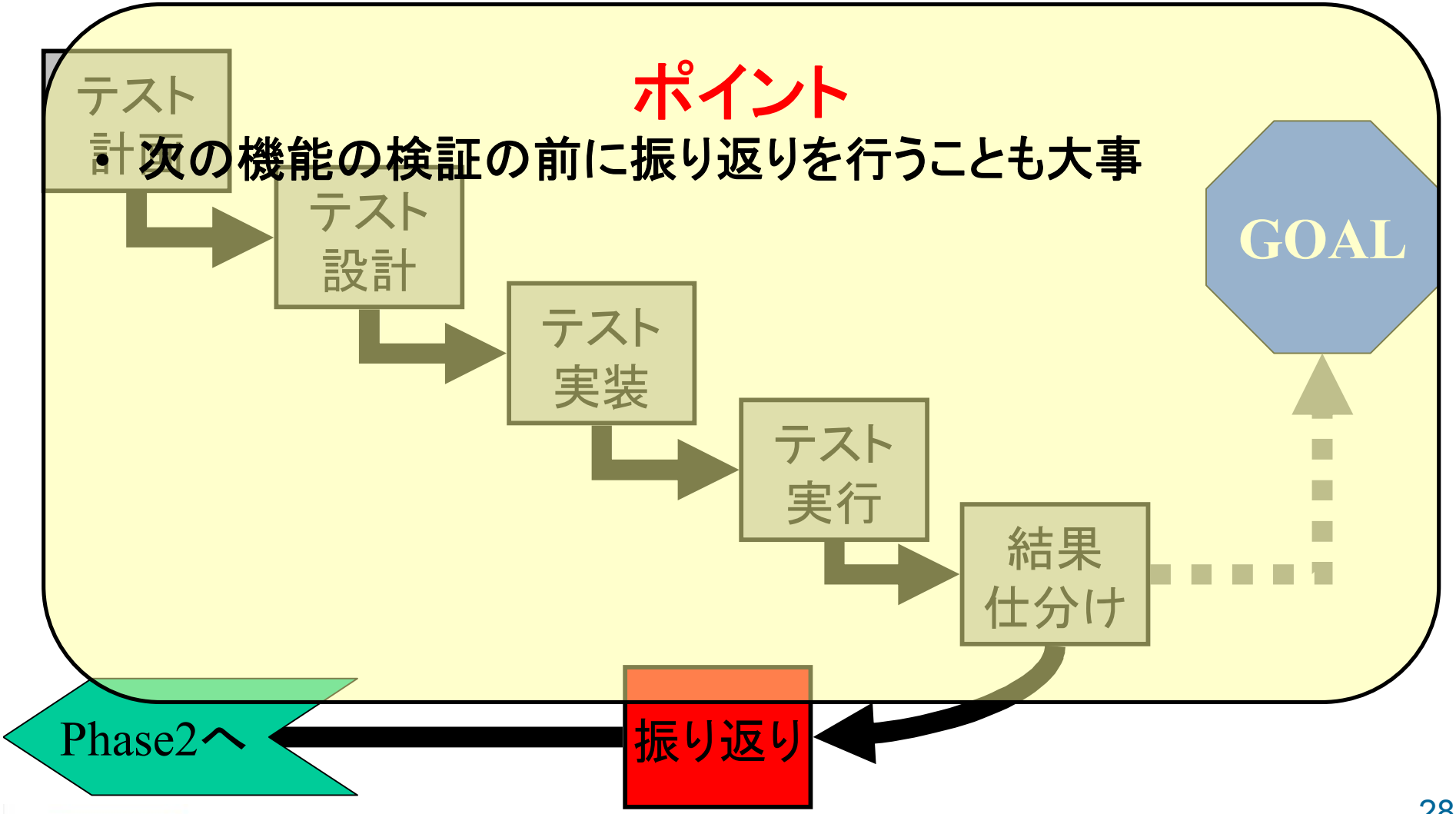
全テストケースを消化
GOAL!!



ではない


ポイント
• テストのNG結果を確認し、それは不具合なのか？暗黙的な仕様なのか？を仕分ける
• 不具合の場合は、不具合の影響度を確認し、対応スケジュールを考える
テスト計画
テスト設計
テスト実装
テスト実行
結果仕分け
GOAL

そして振り返りを実施


ポイント
• 次の機能の検証の前に振り返りを行うことも大事
テスト
計画
テスト
設計
テスト
実装
テスト
実行
結果
仕分け
GOAL
振り返り
Phase2へ

まとめ

やったこと

- テスト計画の実施
- 新たな手法の導入

わかったこと

- テスト計画で狙っていた不具合が発見できたため、機能や観点の絞込みの手法が正しかったと言える
- 本格的に運用する前に手法を検証することで、効果と問題点の整理ができ、導入がスムーズになること
- スープカレー表は網羅的にテスト観点を導くのに効果的な手法であること

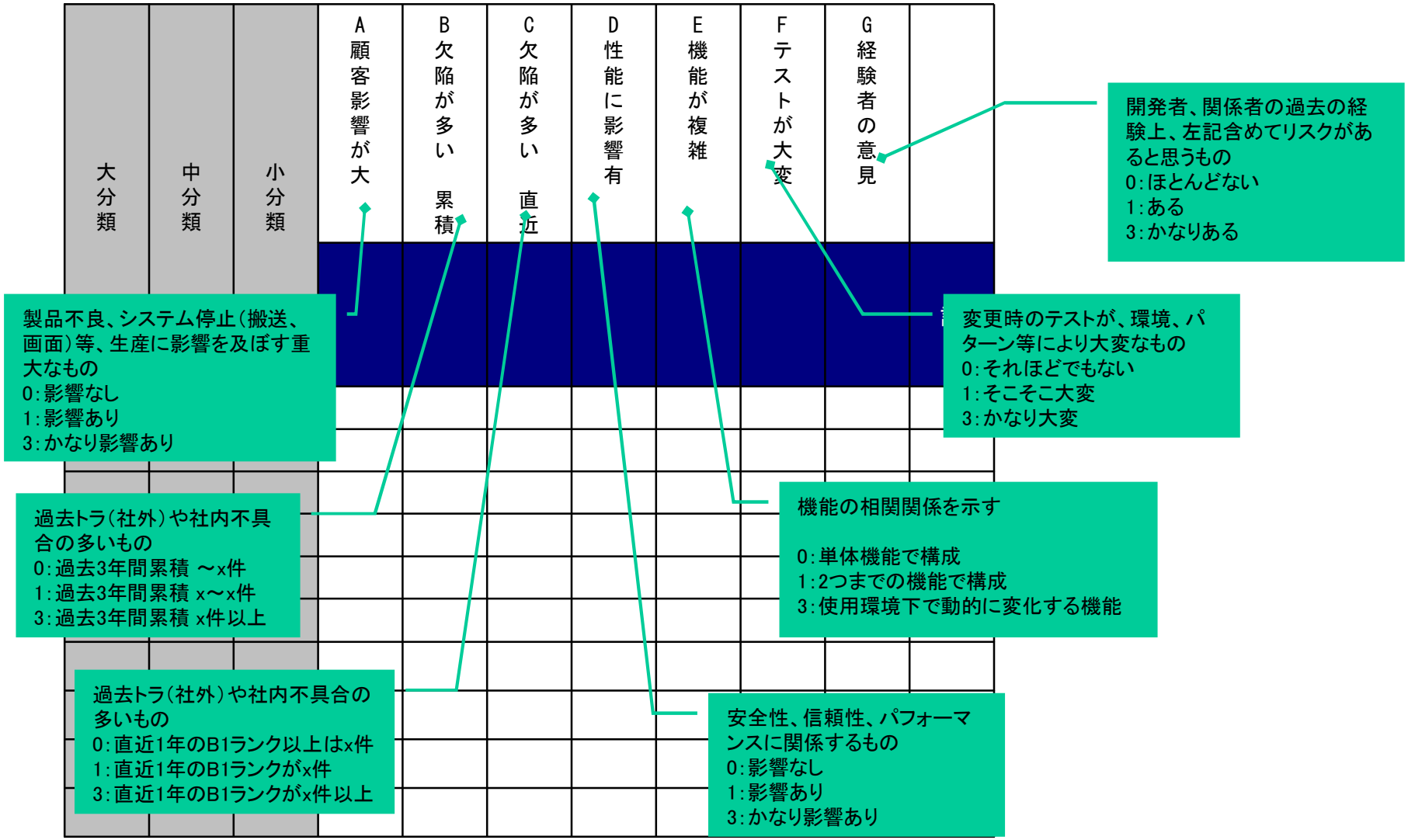
つぎにやること

- システムテストを考える上で、誰が、いつ、どのように、その機能を使うのかをもっと具体的に考えること
- 今回の炙り出しで得たことを開発プロセスの上流工程に取り入れること

ご清聴ありがとうございました

Appendix

ハザードマップ


大分類
中分類
小分類
A 顧客影響が大
B 欠陥が多い 累積
C 欠陥が多い 直近
D 性能に影響有
E 機能が複雑
F テストが大変
G 経験者の意見
開発者、関係者の過去の経験上、左記含めてリスクがあると思うもの
0:ほとんどない
1:ある
3:かなりある
製品不良、システム停止(搬送、画面)等、生産に影響を及ぼす重大なもの
0:影響なし
1:影響あり
3:かなり影響あり
変更時のテストが、環境、パターン等により大変なもの
0:それほどでもない
1:そこそこ大変
3:かなり大変
過去トラ(社外)や社内不具合の多いもの
0:過去3年間累積 ～x件
1:過去3年間累積 x～x件
3:過去3年間累積 x件以上
機能の相関関係を示す
0:単体機能で構成
1:2つまでの機能で構成
3:使用環境下で動的に変化する機能
過去トラ(社外)や社内不具合の多いもの
0:直近1年のB1ランク以上はx件
1:直近1年のB1ランクがx件
3:直近1年のB1ランクがx件以上
安全性、信頼性、パフォーマンスに関係するもの
0:影響なし
1:影響あり
3:かなり影響あり

テスト対象とした品質特性

品質特性	副特性
機能性	合目的性
	正確性
	相互運用性
	セキュリティ
	機能性標準適合性
信頼性	成熟性
	障害許容性
	回復性
	信頼性標準適合性
使用性	理解性
	習得性
	運用性
	魅力性
	使用性標準適合性

品質特性	副特性
効率性	時間効率性
	資源効率性
	効率性標準適合性
保守性	解析性
	変更性
	安定性
	試験性
	保守性標準適合性
移植性	環境適応性
	設置性
	共存性
	置換性
	移植性標準適合性

スープカレー表
概要

- 縦軸に「機能観点」、横軸に「顧客観点」
- 機能観点
  どんな機能を使うのか(機能要求)
- 顧客観点
  - その機能はどのような目的で存在しているか？(目的機能)
  - その機能が特定の利用状況で顧客満足を満たせるか？(想定ユーザ)
  - その機能がどのように動作するか？(非機能要件)

スープカレー表
イメージ

機能		目的	想定ユーザ		非機能観点						

今回のスープカレー表
テンプレート

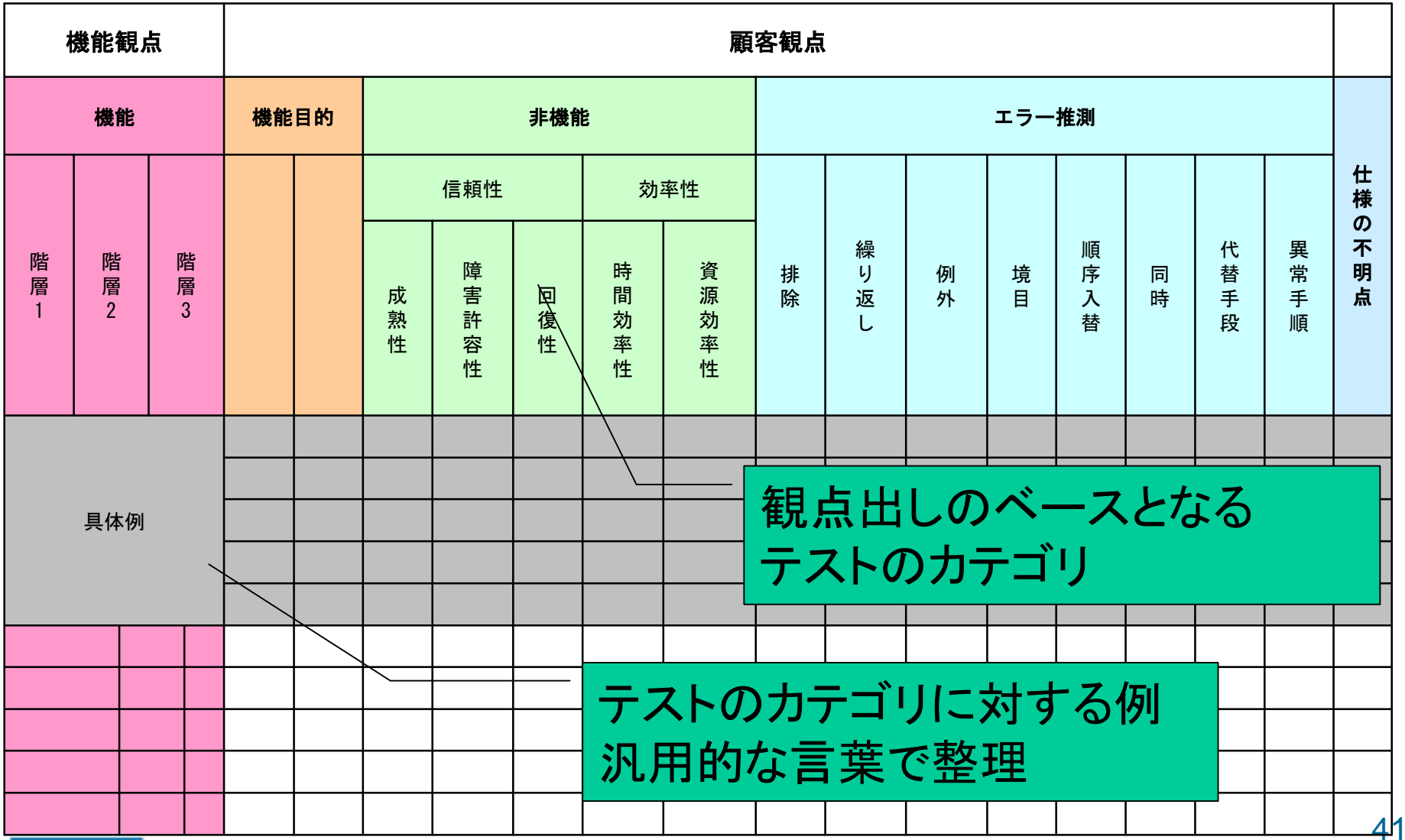
縦軸

- 機能

横軸

- 機能の目的
- 想定ユーザ
  装置を使用するユーザはある程度決まっているため、ここは省略
- 非機能観点
  ベースの観点として品質特性とエラー推測を利用
- 仕様の不明点
  仕様に対する不明点、疑問点、などのメモ

採用した
スープカレー表テンプレート



テストケースの選定基準

- **以下の視点で5段階でポイントを割り振り、ポイントの低いものを間引く**
  a. 手順がお客様の運用上、実施可能かどうかを考える
  b. テストケースで期待する結果が出なかった場合に発生する事象のシステムへどれぐらい影響を与える度合
  c. 機能毎の過去に発生した不具合のランク割合をもとに、ハインリッヒの法則当てはめて、潜在するかどうかを考える
  d. テストケースがどのテストレベルを確認
- **ポイントをつけた後は機械的に間引くが、間引いたテストケースの中に必要なテストケースまで間引いていないか再度確認する**