不具合モード分析手法の研究と実践へのアプローチ

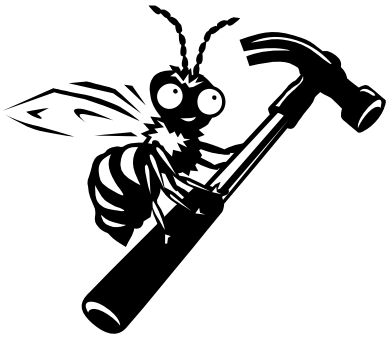
株式会社ベリサーブ
システム検証第二事業部
ITS検証サービス部
江澤　宏和

VERISERVE
verification service
株式会社ベリサーブ

Agenda

- 不具合モードとは
  - 研究開発の背景
  - 不具合モード概要
  - 不具合モード分析プロセス
  - 抽出事例
  - 適用結果
- 実践へのアプローチ
- 今後の展開
- Q&A

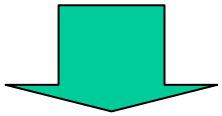
不具合モードとは



研究開発の背景

- **テスト項目が膨大になると、全てを網羅的にテストすることは不可能**

- バグが偏在している部分を狙ってテストし、テスト精度を向上する
  - プロジェクトが異なっても、似たような原因によるバグが多い
  - 経験の豊富なテスト技術者は偏在しているバグを推測してテストしている



- **「攻めの検証」・・・狙いを定めたテスト手法が必要**
  - バグが多いところを狙ってテストするための手法の開発
  - 不具合モードを整理しておき、水平展開してバグを狙いテスト精度を上げる
    - 製品にとってリスクの高い順にバグを見つけることができるようになる
    - 少ない手間で早くバグを見つけることができるようになる

協力体制

- 不具合モード分析研究にご協力頂いている方々
  - 電気通信大学
    - システム工学科　西　康晴講師
    - 西研究室　　河野　哲也氏

※ 他、本研究に賛同し、ご協力頂いている企業様

不具合モード概要

- バグを作りこむメカニズム
  - 難易度（新技術の開発等、実現が難しい開発）
  - 過酷な労働（残業過多）
  - プログラミング言語特有の陥り易い間違い（C言語でのメモリ解放忘れ等）
  - 個人のスキルetc．．．
- これらに共通して言えること
  - どのような理由、条件があろうと、人間が陥り易い「間違いやすさのパターン」があるはず。

間違え易いパターンを整理・分類し、第三者検証に適用する手法が、「不具合モード分析」

例:境界値分析

- なぜこのようなテスト観点があるのか？
  - 仕様:「このフィールドに入力できる最大文字数は英数半角10文字です」
  - テスト項目例
    - 何も入力しない
    - 1文字入力する
    - 10文字入力する
    - 11文字入力する
    - 英数半角以外の文字を入力する

例:境界値分析

- C言語の例

```
char a[10];
int i;

for (i=0;i<=10;i++) {
      a[i] = "a";
};
```

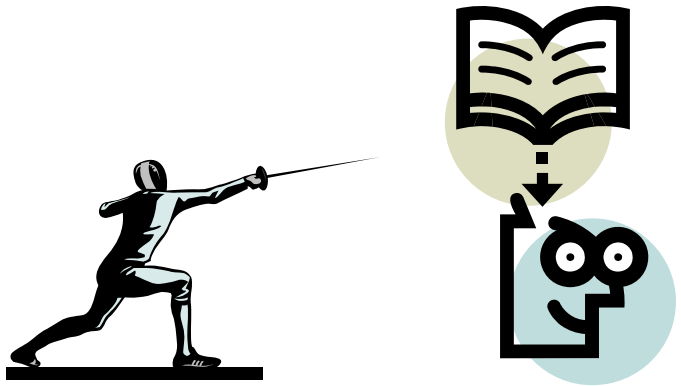
配列は、a[0] ・・・a[9]まで。
しかし、a[10]に値を入れてしまう

こうした事例から、「境界値は間違い易い」を経験値として持っているので、テスト観点として存在する。

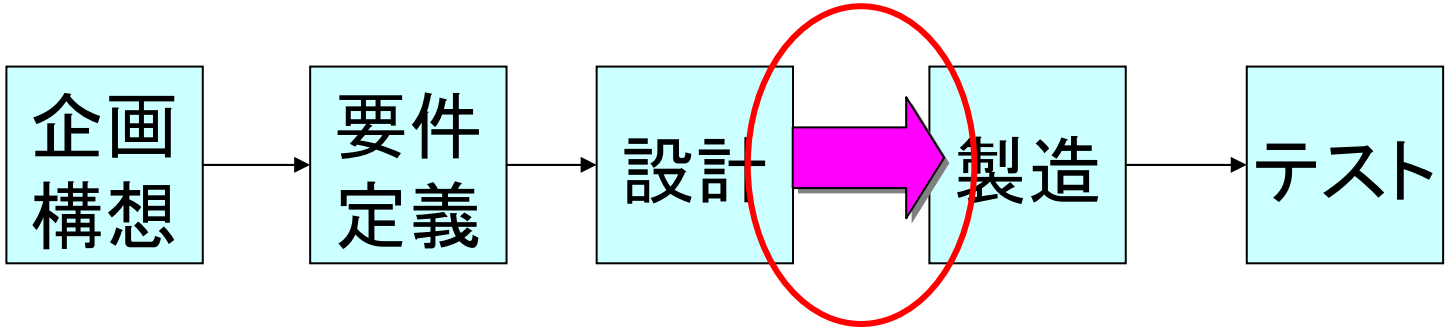
概要総括

- これらの間違えやすさと同様に、「間違え易い仕様」というのがあるのでは？

- 「間違えやすい仕様」に狙いを定め、ピンポイントで不具合を検出できる手法を確立する。

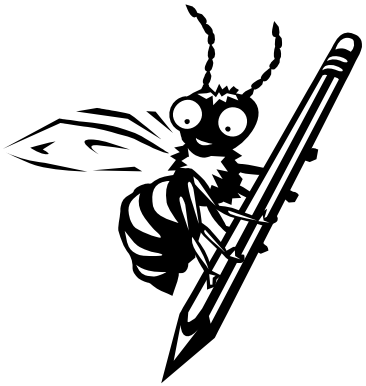


不具合モードの位置づけ

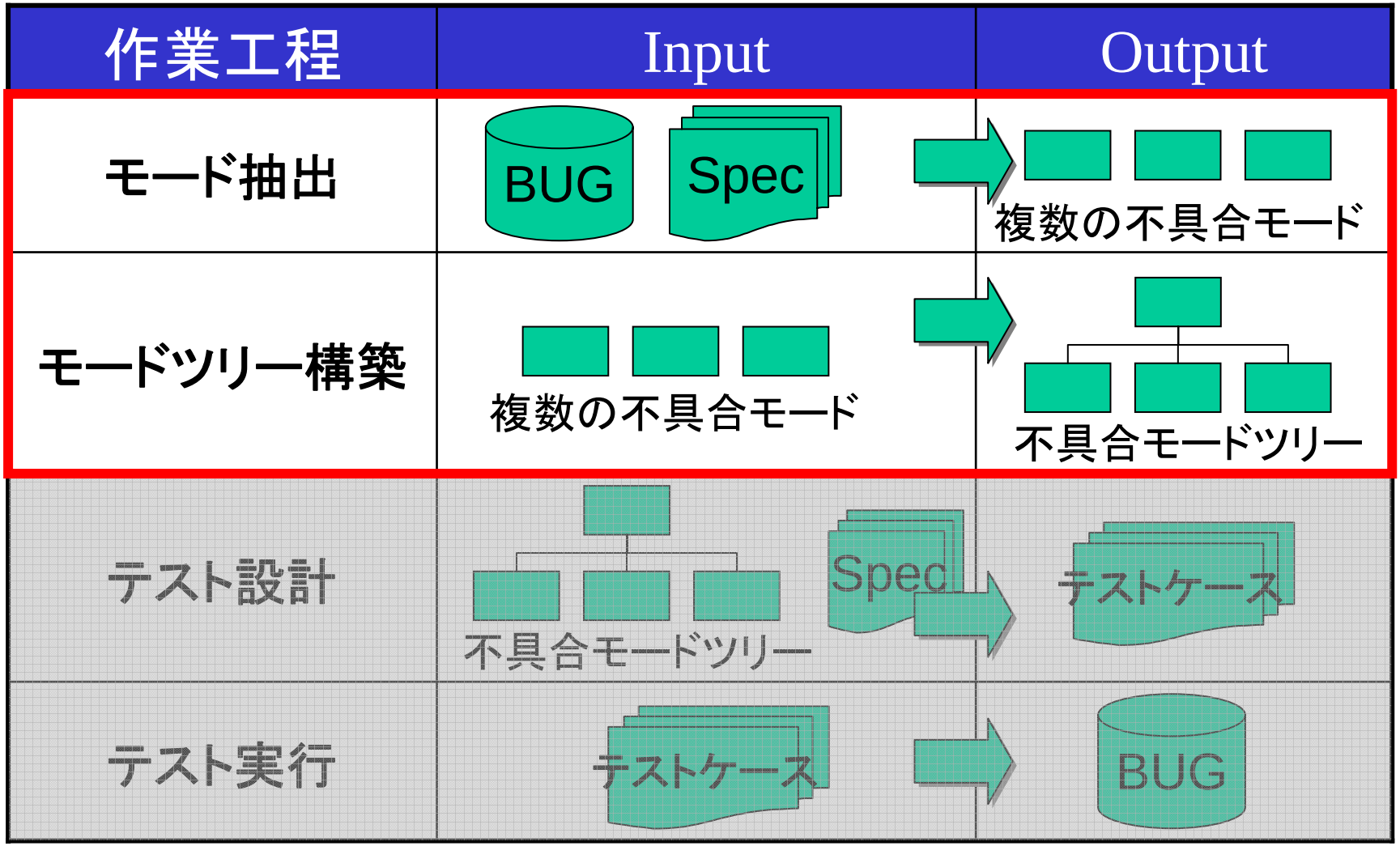


不具合モード分析は、設計レビューによる妥当性確認があったにも関わらず、製造工程でバグを作りこんでしまうところを狙った、分析手法。

不具合モード分析プロセス



不具合モード分析プロセス概要

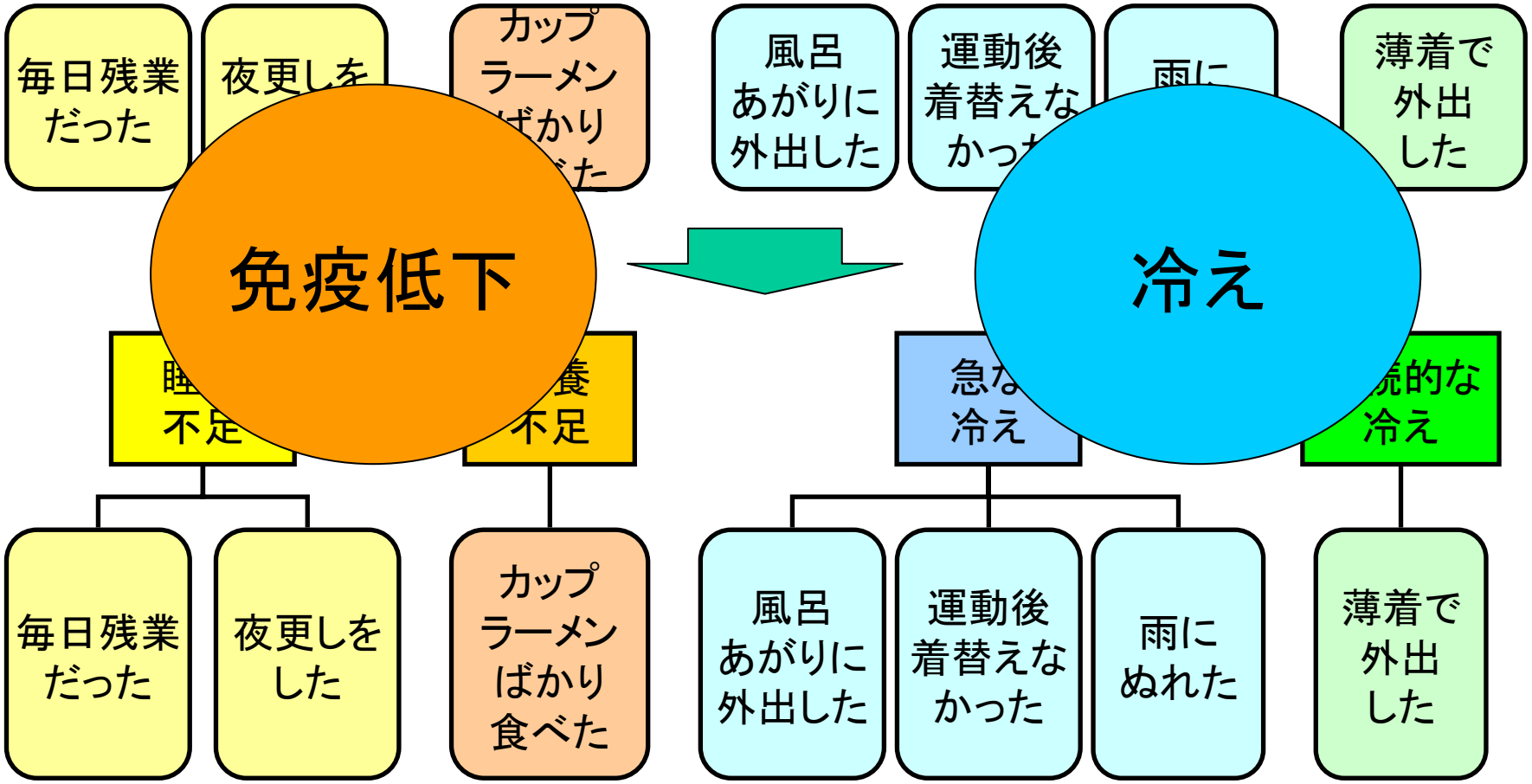


不具合モード抽出プロセス

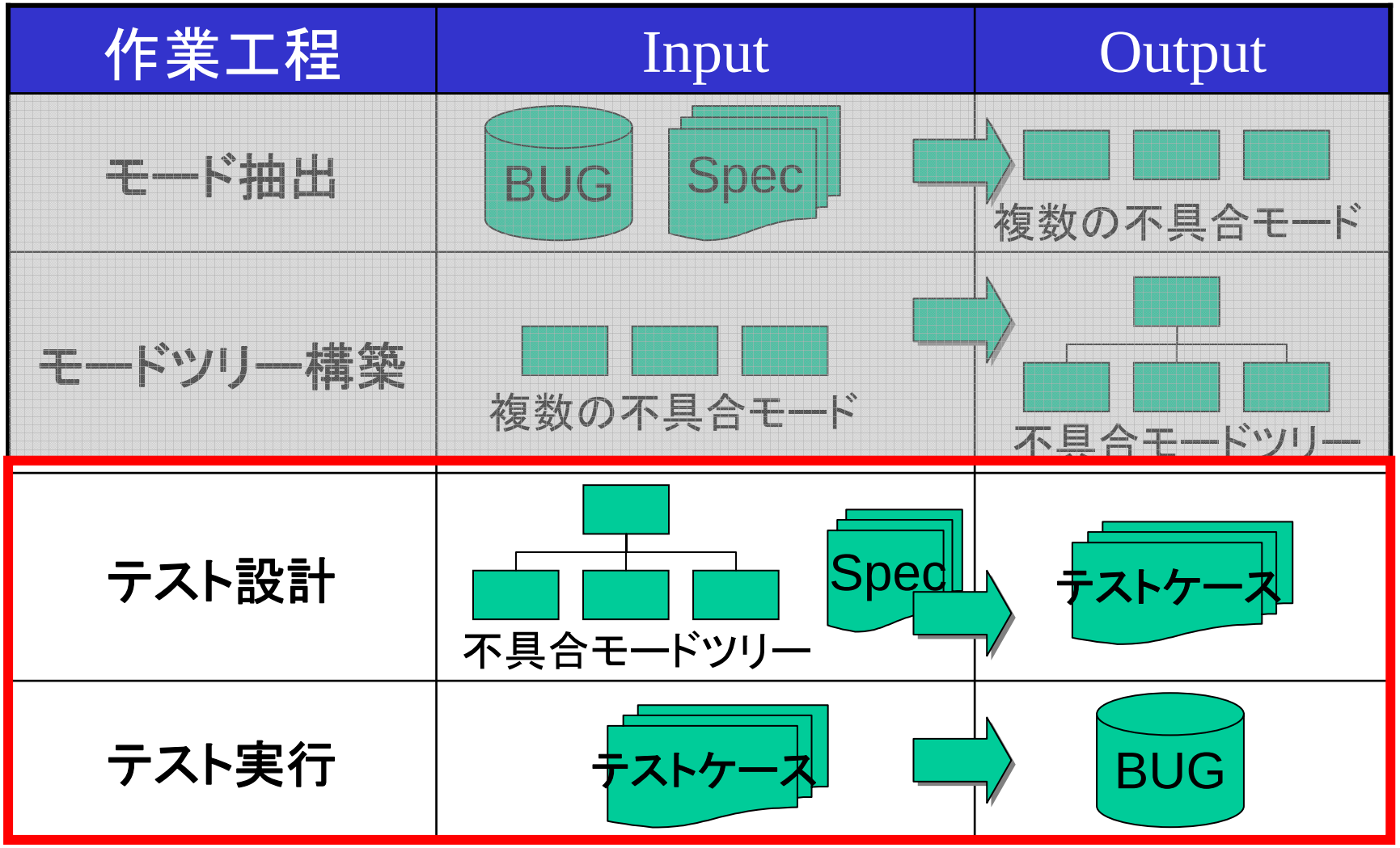
- ①類似の不具合を収集する
  - 不具合の事例をインスタンスとして収集する
    - 風邪をひいた原因の事例の列挙
      - 雨にぬれた
      - 毎日残業だった
      - カップラーメンばかり食べた
      - 風呂あがりに外出した
      - 運動後着替えなかった
      - 夜更しをした
      - 薄着で外出した

不具合モード抽出プロセス

- ②抽象化して不具合モードを抽出する
  - 似たようなバグを抽象化し不具合モードを抽出する

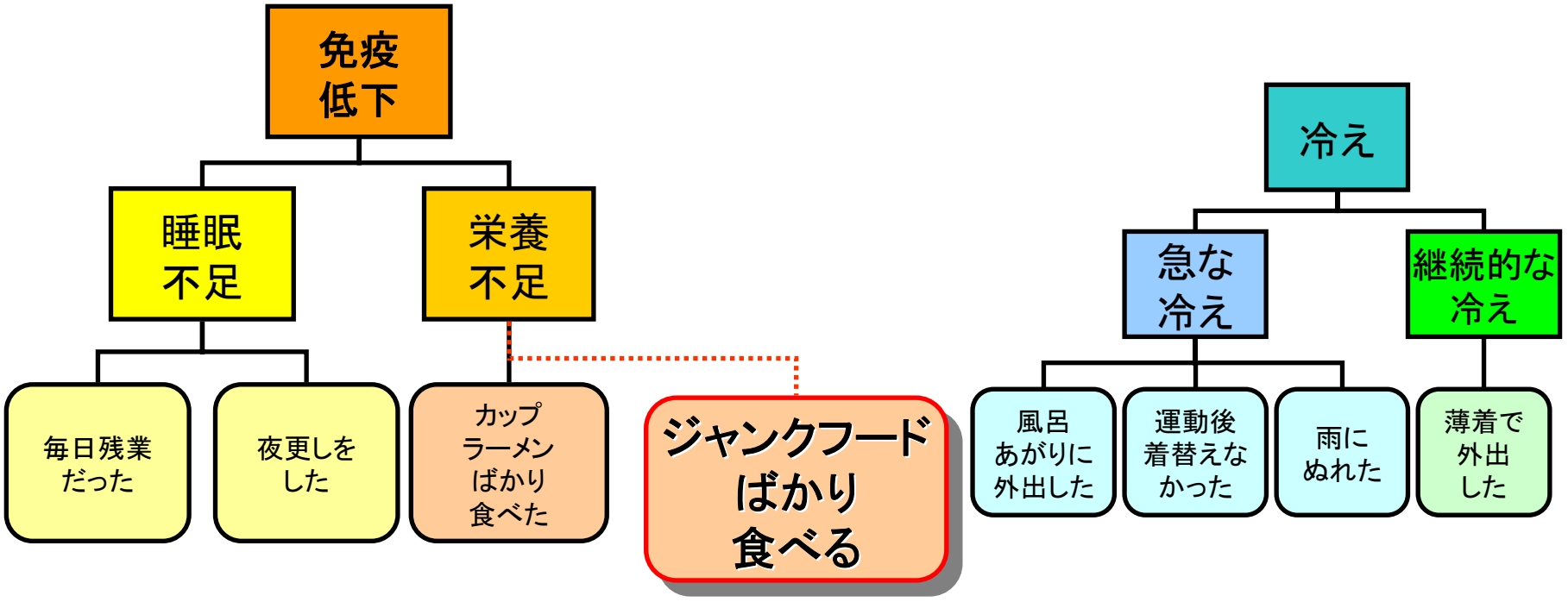


不具合モード分析プロセス概要



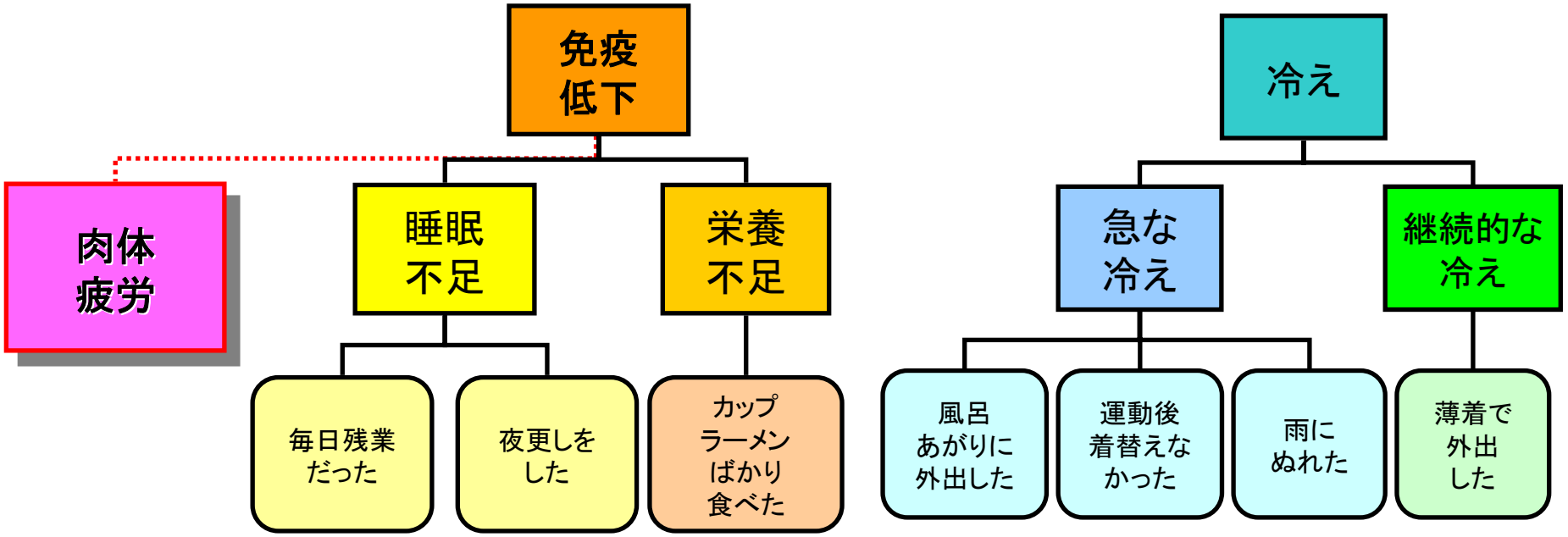
不具合モード使用プロセス

- ③同じ不具合モードに属する新たな不具合のインスタンスを推測する
  - 不具合モードの記述に当てはまるテスト対象の機能やデータ構造を探す
  - 当てはまった不具合モードの示すメカニズムや兆候を持つ不具合を推測する



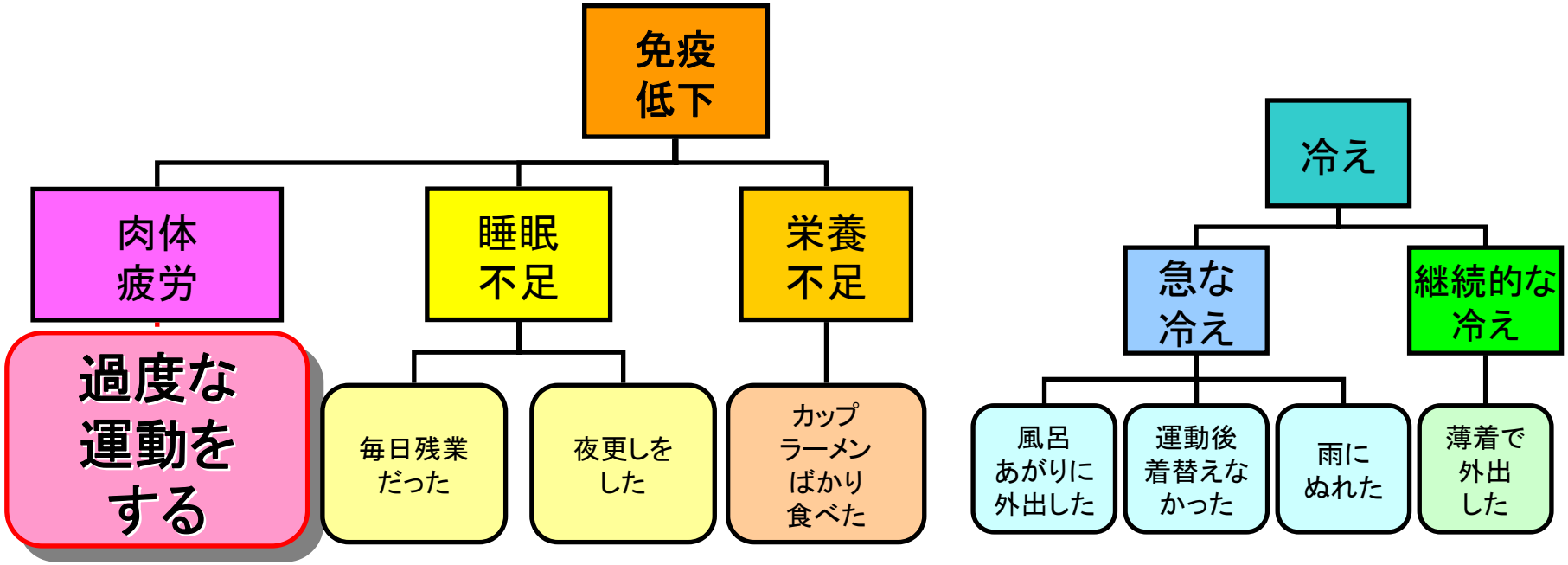
不具合モード使用プロセス

- ④類似の不具合モードを推測する
  - 上位の不具合モードを抽出し、そこから新たに下位の不具合モードを推測する

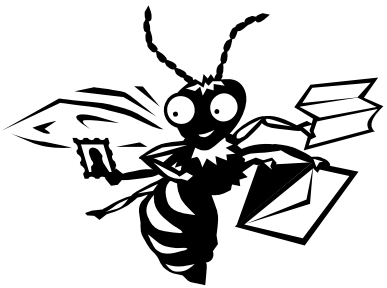


不具合モード使用プロセス

- ⑤推測された不具合モードから新たな不具合のインスタンスを推測する
  - 不具合モードの記述に当てはまるテスト対象の機能やデータ構造を探す
  - 当てはまった不具合モードの示すメカニズムや兆候を持つ不具合を推測する
  - 当てはまった機能やデータ構造などと、具体化した不具合からテスト項目を設計する

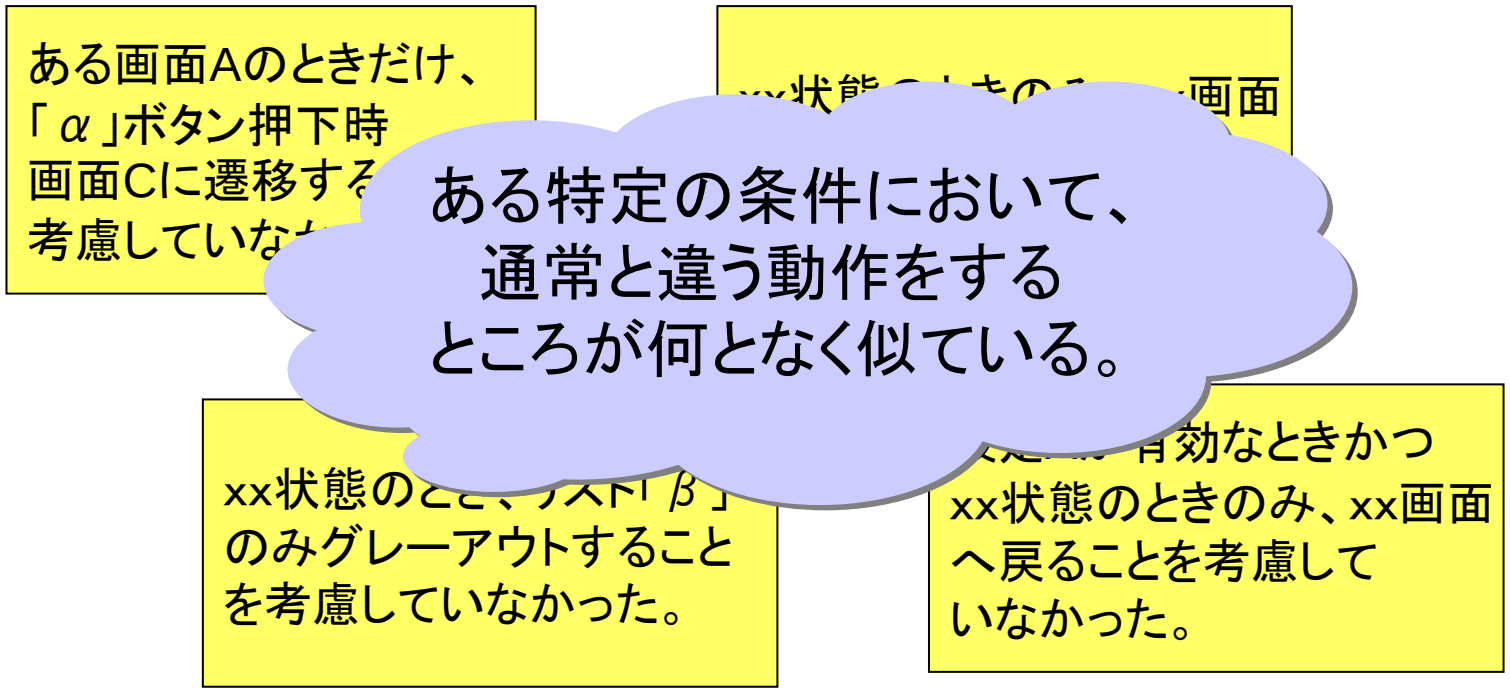


不具合モード事例:「例外」



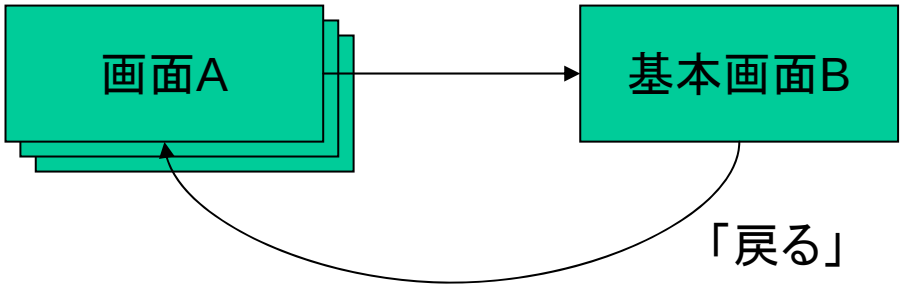
不具合モード「例外」

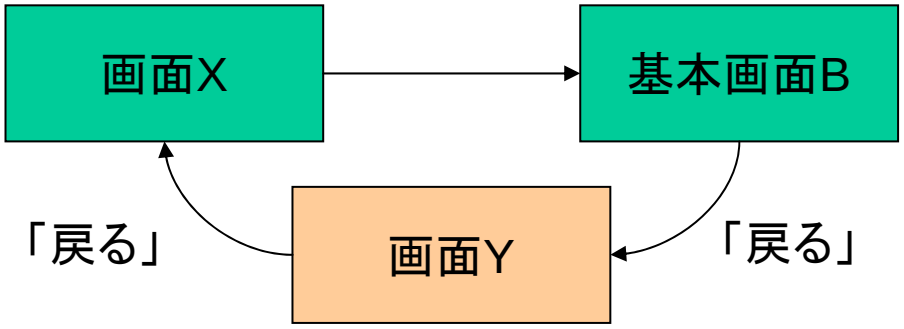
類似の原因の不具合を収集する



不具合モード：例外

- 「戻る」ボタンの基本仕様
  - 一つ前の画面に「戻る」というのが一般的

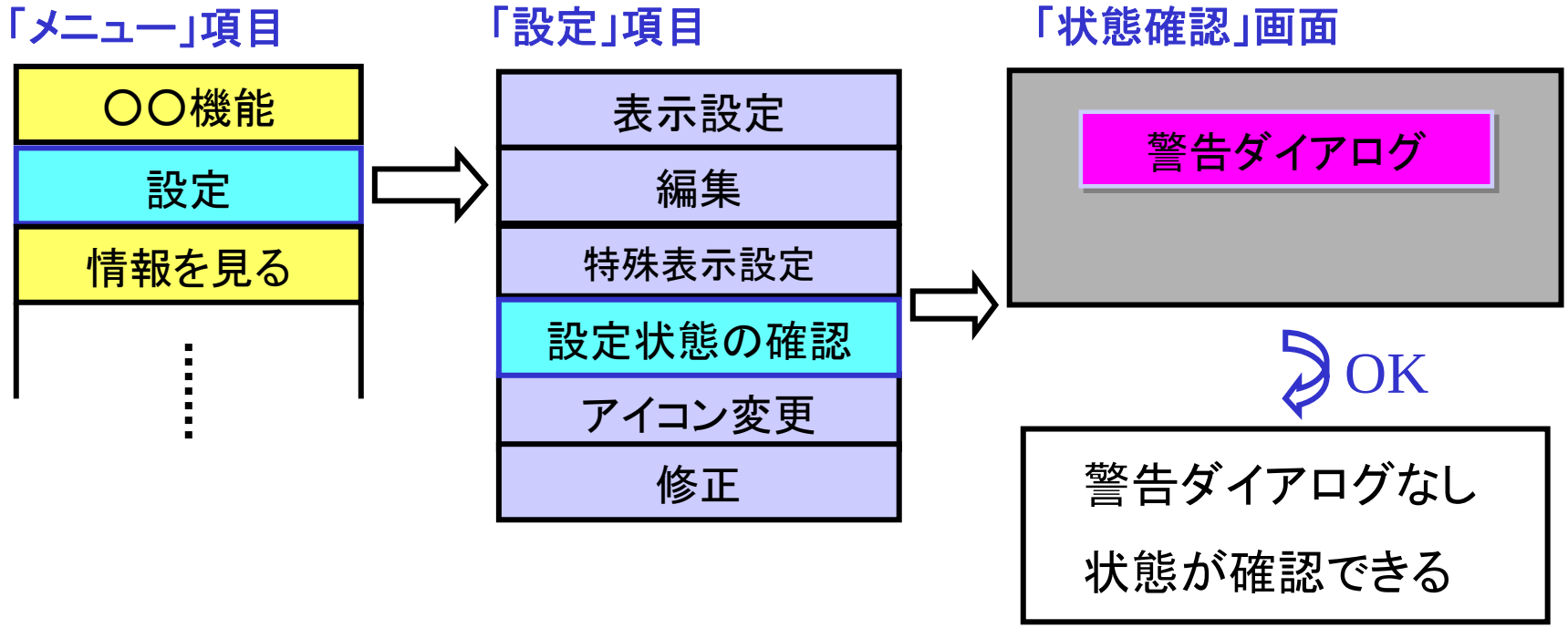




画面Xから画面Bに遷移したときのみ、「戻る」ボタンは画面Yに遷移する。

不具合モード事例

- 基本不具合モード「例外」の典型例
  - ある状態の時、「状態確認」画面だけ操作を制限しない仕様だが、「状態確認」画面も制限してしまう不具合



不具合モード事例

- 原因の推測
  - 対象の一項目を除いて利用できないようにする仕様であるため、「**全部利用できないようにするもの**」と間違えてしまった。
  - ある動作中、設定を「変更する」や「修正する」は、その動作自体に影響を与えるので利用できなくしているが、設定状態確認については影響を与えないのでボタンが有効である仕様である。

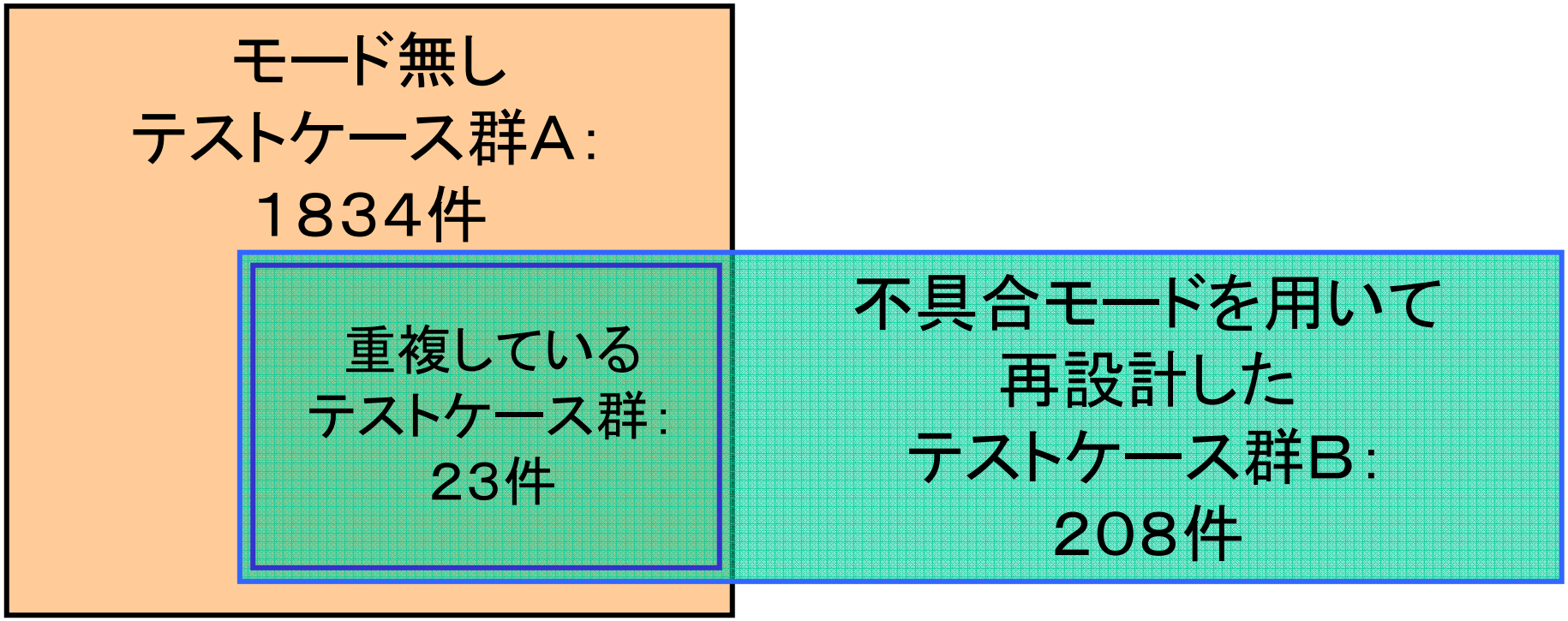
あるグループの中に一つだけ属性や要素が違うものが存在すると、共通に変化するイベントは「例外」という不具合を作りこみやすいのでは？

プロダクト・アフォーダンス

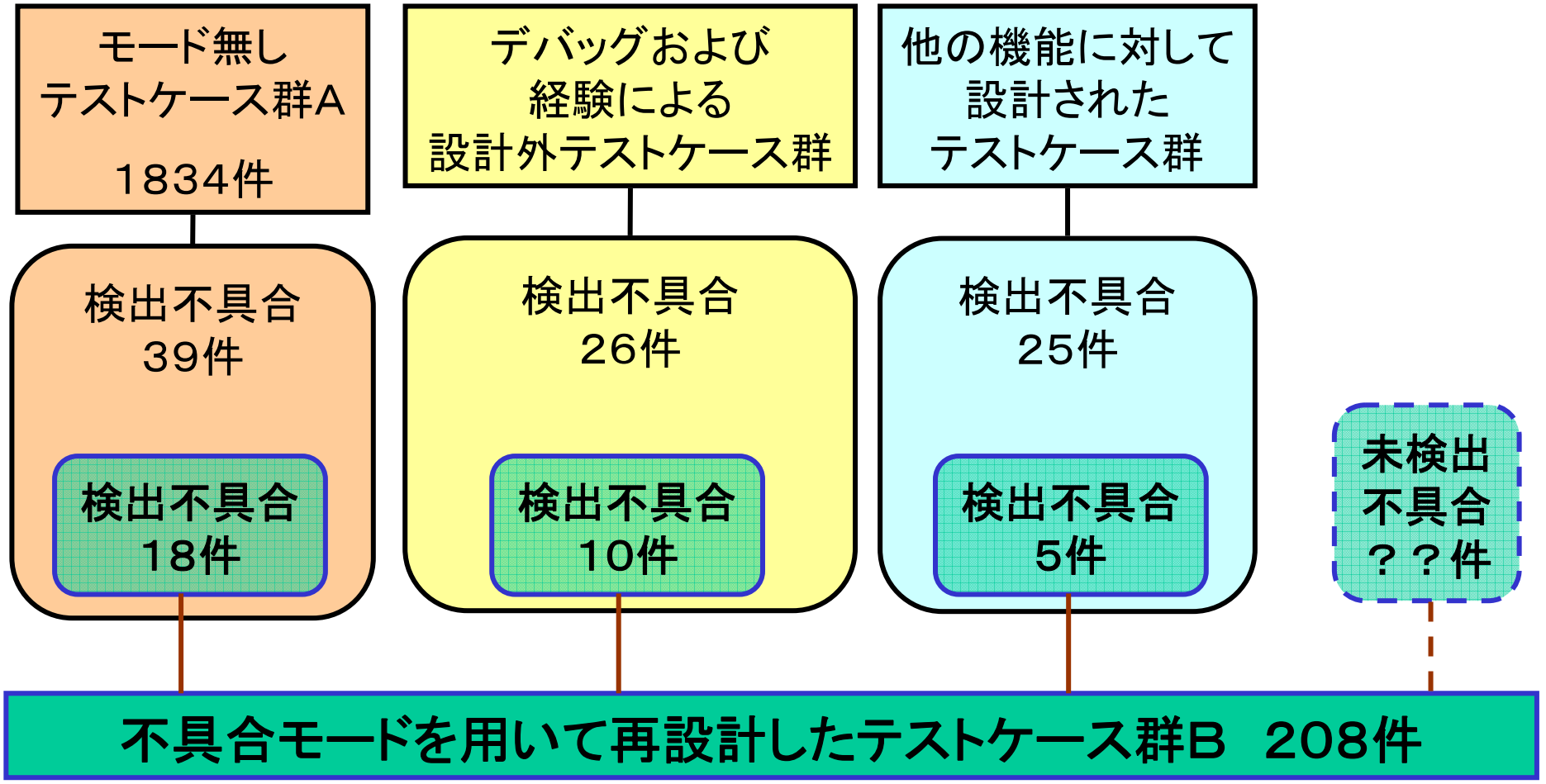
- 「例外」は、開発対象物の持つ
  ヒューマンエラーの起きやすさを示す属性である
  - 「多くのうち少数に例外的な開発事項が必要になる設計である」と抽象化できる
  - これを「**プロダクト・アフォーダンス**」と呼ぶことにする
    - プロダクト・アフォーダンスとは、**ヒューマンエラーの起きやすい開発対象物の属性**である
    - 設計と記法の両方にプロダクト・アフォーダンスがあるだろう
  - 仕様にも構造にも実装にも適用できる

適用結果

- 不具合モード「例外」の適用結果
  - 某コンシューマ向け組込製品での事例
    - 第1世代製品→第1．5世代製品への適用



適用結果



適用結果

- **テスト精度の評価**
  - Aの（モード無し）テストケース群　　：1834件
  - Bの（モードあり）テストケース群　　：208件
  - 重複しているテストケース群　：23件
  - 重複していないテストケース群　　：185件
  - Aで検出された不具合数　　：39件（39/1834 = **2.1%**）
  - Aで検出された不具合のうち
Bで検出されるはずの不具合数　　：18件（18/208 = **8.7%**）
  - Bで検出されるはずの確認不具合数：33件（33 / 208 = 15.9%）
＋α

4倍以上の精度向上となっている

適用結果考察

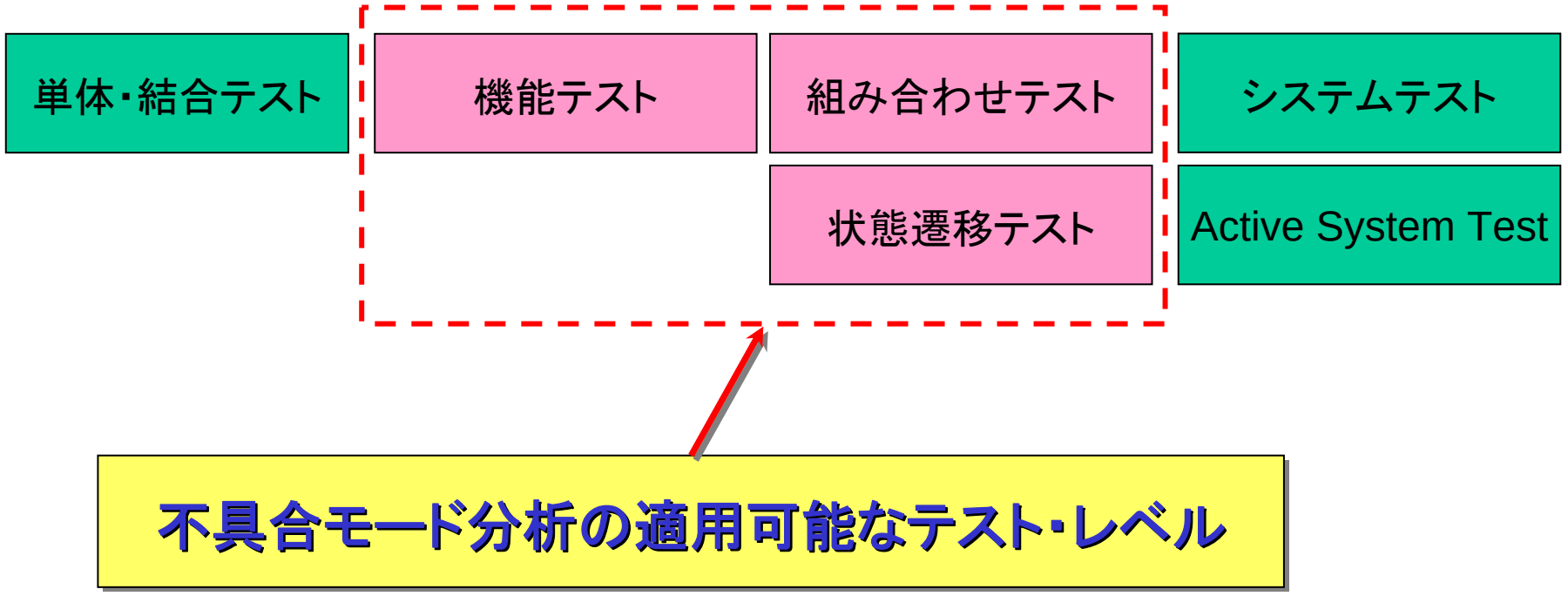
- 不具合モードを用いてテスト設計を行うことにより テスト精度が向上するという適用結果が得られた
  - ただしAは**「保証型」のテスト**であり、Bは**「検出型」のテスト**であるため、単純に比較するにはやや問題がある
    - 一見ムダに見えるAのテスト項目にも十分意味がある
  - 保証型のテストの優先順位設定として用いると強力なツールとなる

- 不具合モード蓄積プロセスには時間がかかる
  - 不具合報告書が曖昧だと調査して書き直す必要が生じる
    - ヒアリングが必要な場合もある
  - 最上位の不具合モードを抽出するには、かなりのスキルが必要

実践へのアプローチ

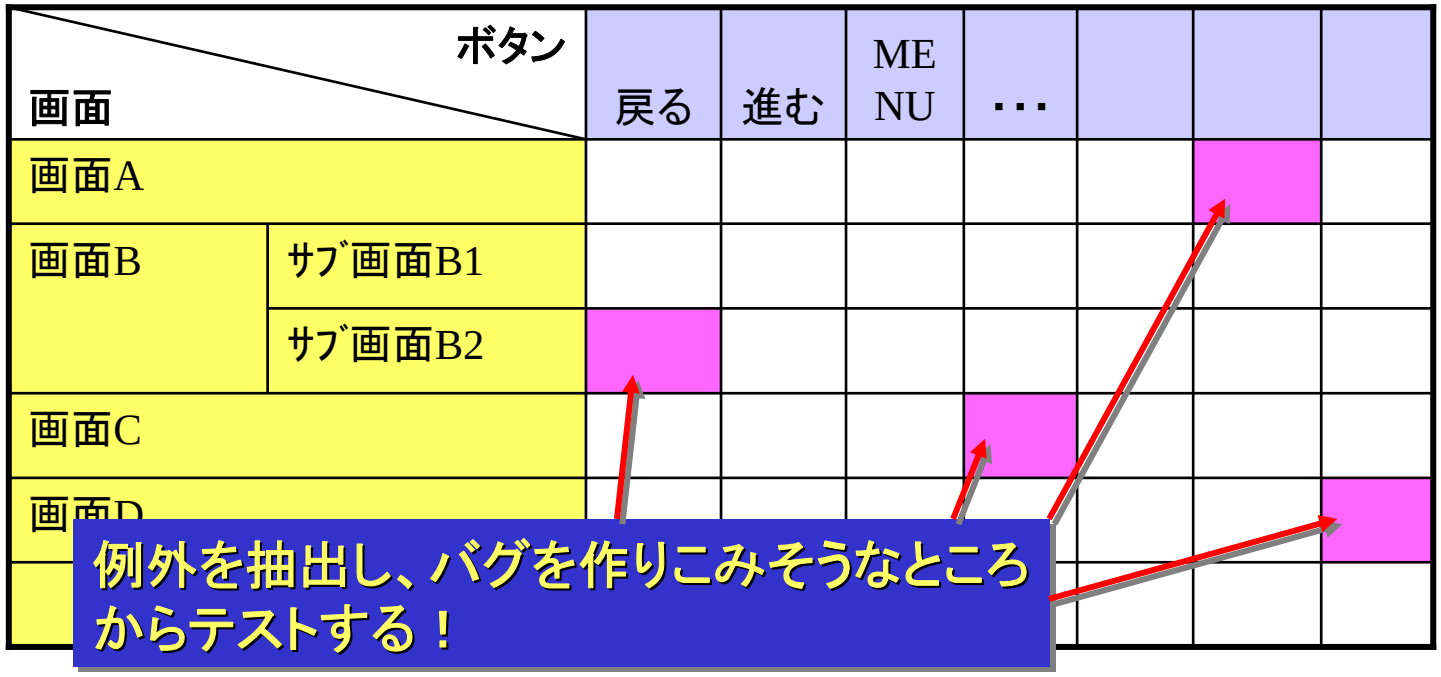
テスト工程への適用

- テスト実行工程



機能テスト・状態遷移テストへの適用

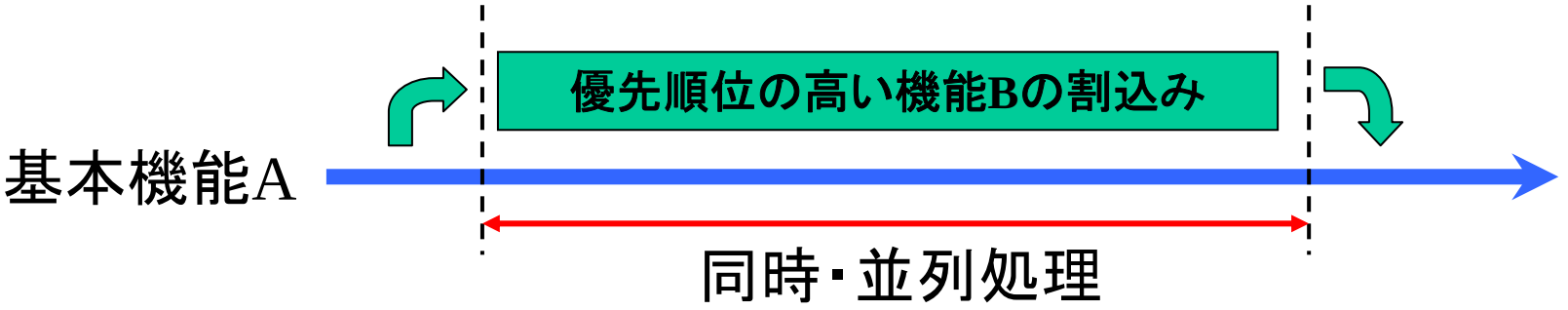
- 網羅的に抽出される項目から、例外に該当するケースを先にテストすることで、バグを早期発見する。
  - 例：「戻る」ボタン
    - 全ての表示画面に対して、「戻る」ボタン押下を網羅的に抽出した場合、項目数が膨大になる。



組み合わせテストへの適用

- 組み合わせテストでも、新たなモードが存在する

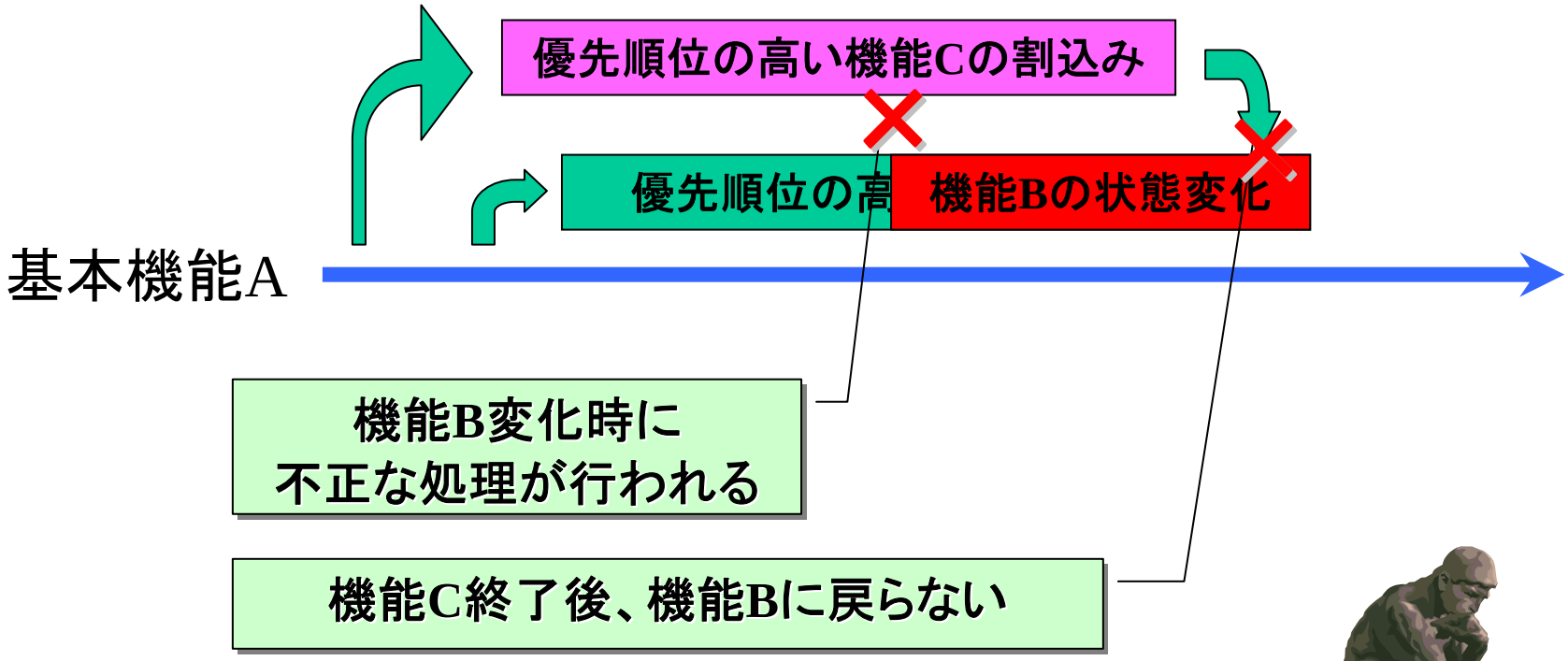
通常のテスト・パターン



確認項目
- 機能A実行中に機能Bが開始・表示されること
- 機能A・Bともに処理が実行されること
- 機能B終了後、表示が機能Aに戻ること

組み合わせテストへの適用

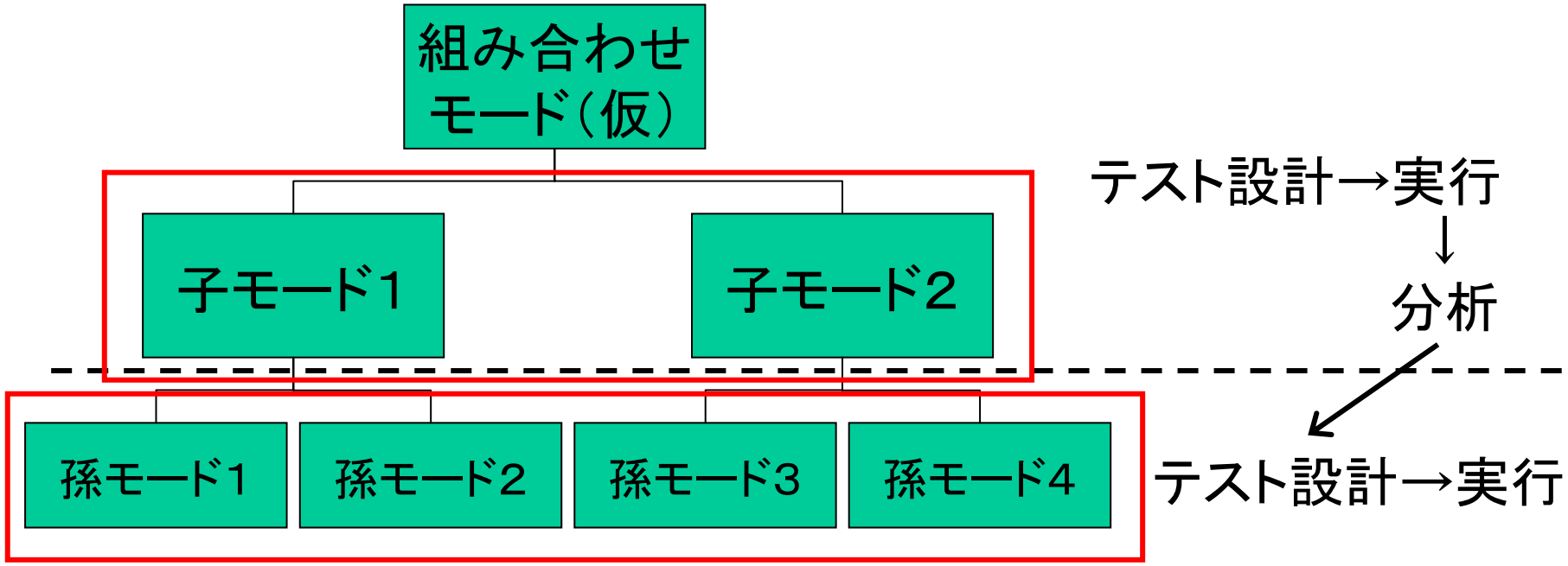
モード・テスト・パターン



こんなパターンを考えはじめたらキリがない！
しかし、バグは実際に検出される。どうしよう？？

組み合わせテストへの適用

- 組み合わせによるモード適用



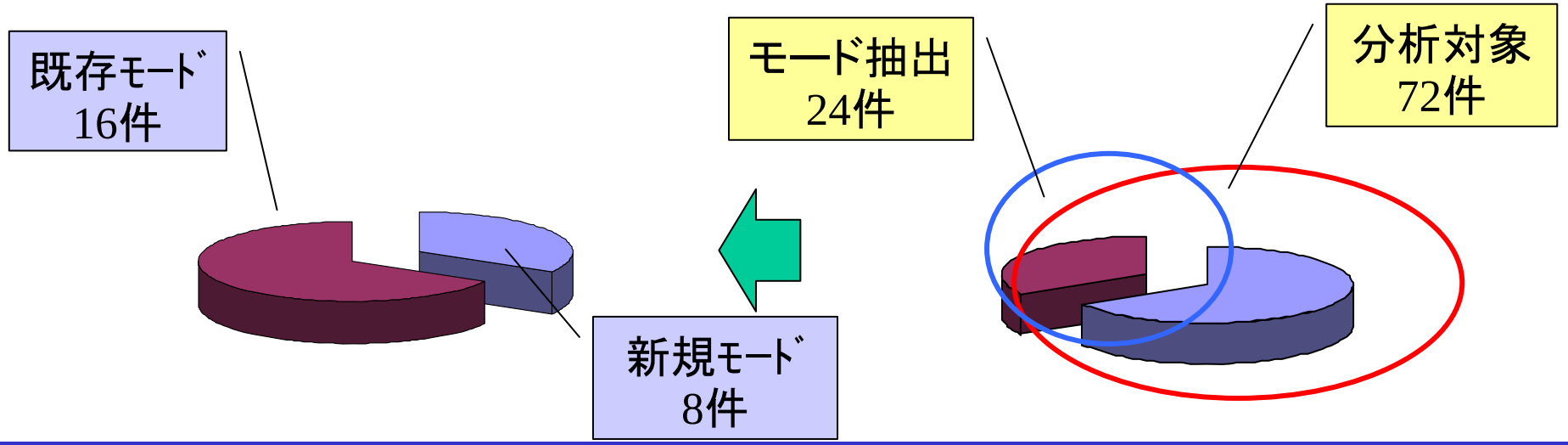
想定効果

- **短期間に多くのバグを検出することができる**
  - ピンポイントでテストができる
  - 早いタイミングで多くのバグを見つけることができる
- **ノウハウに依存せずに検出率の高いテストができる**
  - ノウハウを表出化させることにより誰もがバグを狙ってテストできる
  - テストケースの間引きの観点しても活用できる
- **バグを予測することができる**
  - 開発者が作り込み易いバグを予測しテストケースの設計ができる
  - テスト前に期待結果と想定されるバグを知ることができる

網羅型のテストと組み合わせることで、効果がある！

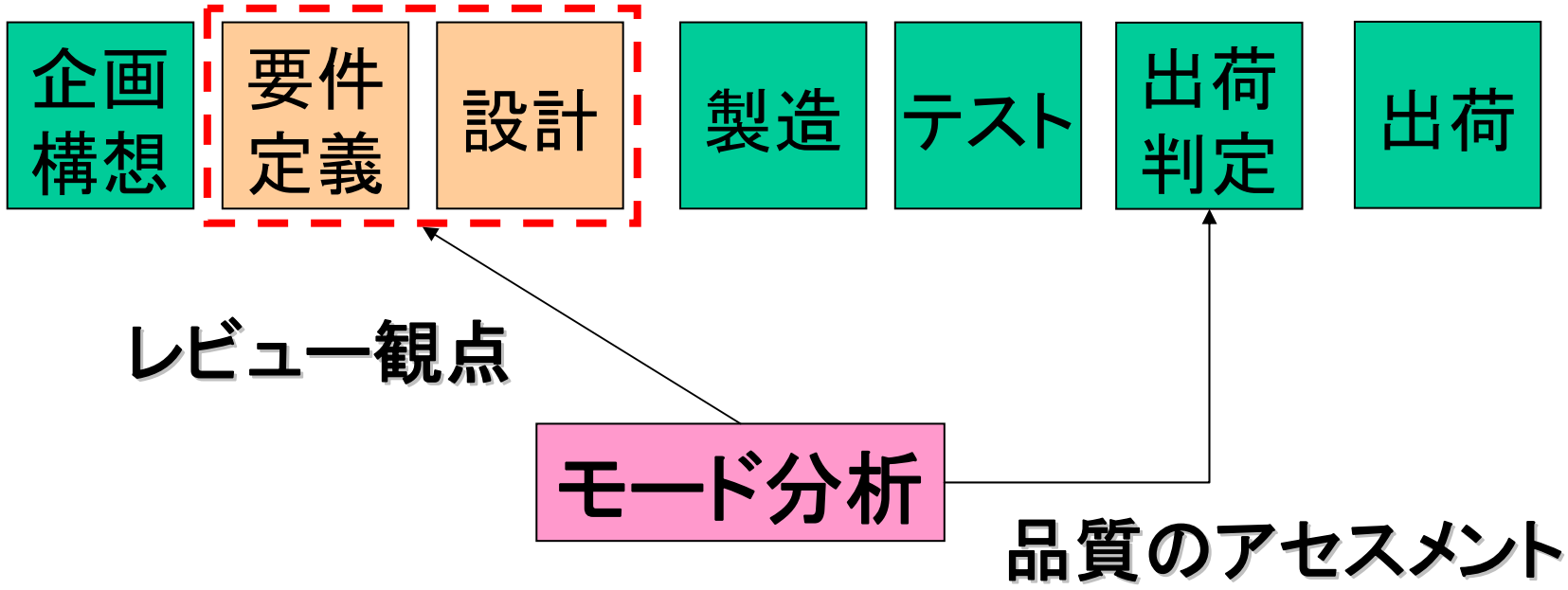
まとめ

- **現在、ツリー全体17のモードについて抽出済**
  - 今後も分析を継続することで、新しいモードの検出、ツリーの精査を行ってゆく。
- **他の組織で不具合モードが適用できることがわかった**
  - ある別の組織で抽出したモードを、ある製品のHMIエリヤで発生している不具合の分析を行った
    - HMIエリヤでの不具合件数:126件
    - 分析対象とした不具合件数:72件
    - 不具合モードで検出されるであろう不具合件数:24件
      - 既存不具合モードに該当する不具合:16件
      - 新規不具合モードに該当する不具合: 8件



今後の展開

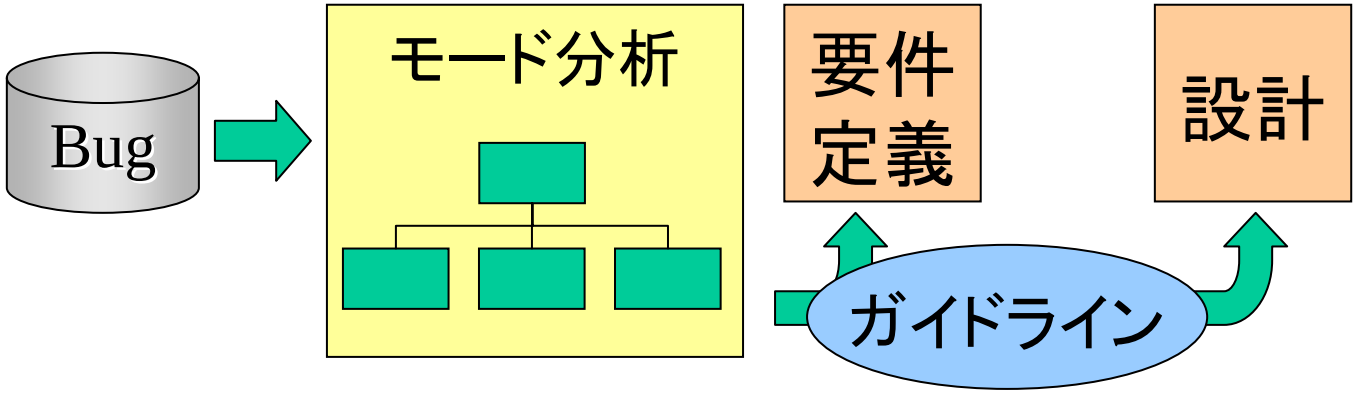
- 不具合モード分析のゴール？
  - バグを作りこまないための仕様、レビュー体系の構築



開発ガイドラインに不具合モードを盛り込むことで、バグを作りこまないための手法となる

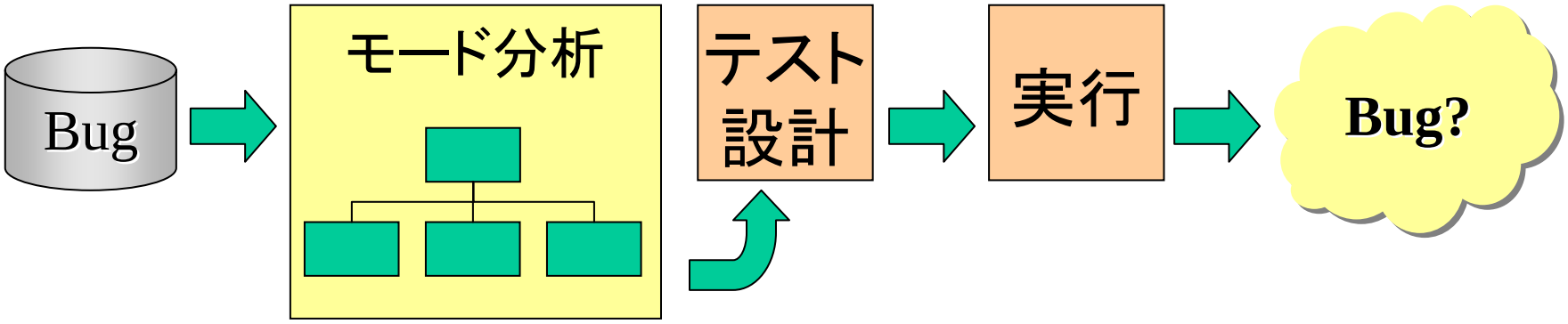
上流工程への取り組み

- **上流工程への改善の取り組みが考えられる**
  - **レビューで指摘すべき不具合がテストで検出されている**ことで修正コストがかかっているのではないか
    - 不具合モードを仕様レビューや設計レビューの観点とすることで多くの不具合を指摘することができるようになるだろう
  - **作り込みの未然防止や再発防止ができる不具合が検出されている**ことで修正コストがかかっているのではないか
    - 不具合モードを仕様作成ガイドラインや設計ガイドラインに組込むことで不具合の作り込みを減らすことができるようになるだろう



品質のアセスメント

- **モード・テストを活用しなかった製品に対して、最終出荷前にモード・ツリーから項目を抽出し、不具合の検出漏れがないかどうか、最終的な品質チェックを行う**
  - 製品の品質向上につながるだろう
  - 不具合が検出された場合、「テストとして何が足りなかったのか？」等原因分析を行うことで、テスト品質の向上につながるだろう



ご清聴、ありがとうございました。

- Q&A