JaSST'18 Kansai

テスコンからの納得できるテスト設計への挑戦

2018年6月15日

大段智広（おおだんともひろ）

こすにゃん

自己紹介

名前：

大段　智広　（おおだん　ともひろ）

お仕事：

開発技術導入支援（CI、静的解析 等）

テスト関連の研修講師

社外活動：

ソフトウェアテスト系のコミュニティ&イベントに出没中

- ✓JaSST Kansai 実行委員
- ✓テスト設計コンテスト`15～18（てすにゃん）
- ✓関西ソフトウェアテスト勉強会(WARAI)
- ✓バグ票ワーストプラクティス検討プロジェクト

本日お伝えしたいこと

自分のテスト設計がいまいち（と言われる）

自分もテスト設計にいまいち納得できない！

そのためには…

本当に解決したい！

〝テストの意図や目的〟

を理解し、押さえる！

テスコンの活動を通じて
これから２つのことを
お話しします

※テスコン：ASTER テスト設計コンテスト

１つめは、
「〃テストの目的や意図〃
　　を理解し、押さえる方法」

２つめは、
「実践した
自分たちのテスト設計」

アジェンダ

1. テスコン（テスト設計コンテスト）とは
2. なぜテスコン？
3. 納得できないテスト設計
4. “テスト設計”とは
5. “納得”とは
6. “納得できるテスト設計にするために”
7. 自分たちのテスト設計
8. まとめ

テスコン(テスト設計コンテスト)とは

指定のテストベースに対するテスト設計を行い、優劣を競う。

コンテストは各地域予選を行い、優秀と認められたチームが決勝戦で争う。

http://aster.or.jp/business/contest.html#system

なぜテスコン？

参加当時のテストの悩み

- ✓会社のテストプロセスに沿ったテストしかしていない
- ✓現状でもテスト設計を行えるがより改善できるのでは？と感じている
- ✓開発が楽になる方法やQAの人ともっと連携できる方法があるはず…

納得できないテスト設計をしている

(より良いテスト設計をしたい)

納得できないテスト設計

テスト設計の悩み具体例

- テストの項目自体がなんか足りない
- あるけどロジカルじゃない
- あるテストとあるテストのやり方に整合性や統一性がない
- 網羅度が十分じゃない
- テスト項目、テストケース導出の根拠が不十分
- 項目の洗い出しやテストの実施が俗人的
- そもそも開発製品が何やりたいかさっぱりわからん

etc,...

"テスト設計"と"納得"を改めて考える

〝テスト設計〟ってなんだ

テスト開発プロセス

ソフトウェア開発プロセスと対応したテスト開発プロセスを推奨している。

"テスト設計"とは

テスト設計（test design）】(JSTQB 引用)
(1) test design specification を参照のこと。
テストアイテムのテスト条件（カバレッジアイテム）、詳細なテストアプローチ、及び、関連する高位レベルテストケースを記述したドキュメント。

(2)
概略的なテスト目的を具体的なテスト条件とテストケースに変換するプロセス。

〝納得〟ってなんだ

"納得"とは

『他人の考え・行為を理解し、もっともだと認めること』

(デジタル大辞泉引用)

他人

考え

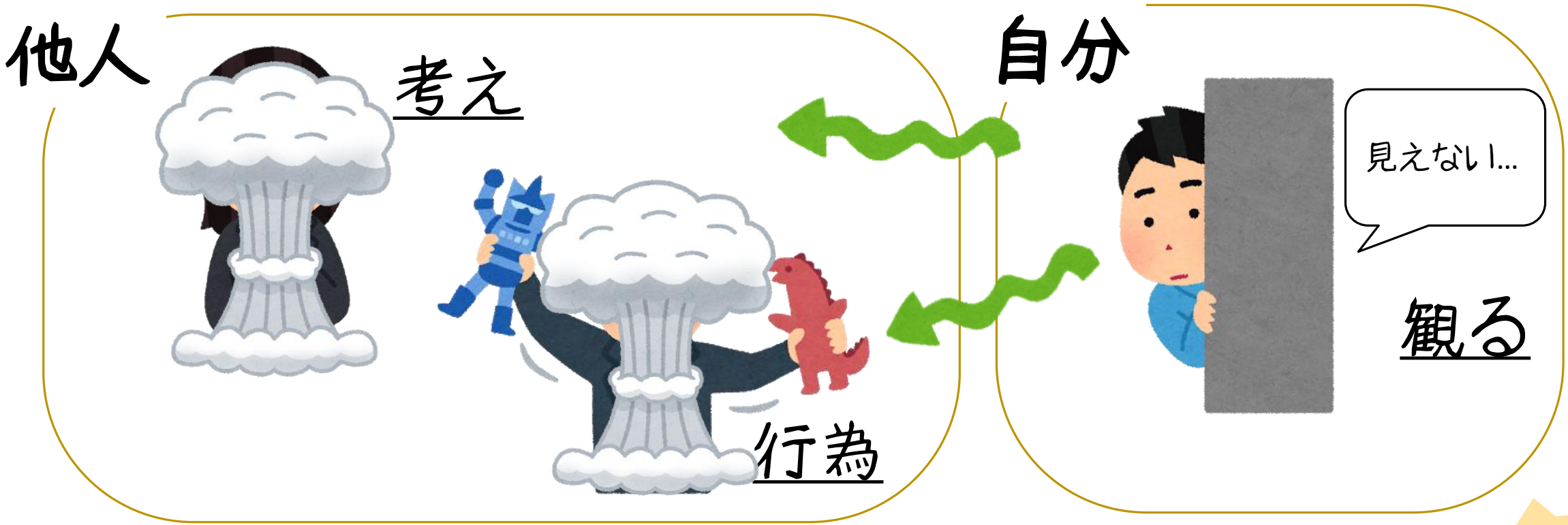
行為

自分

理解
&
認める

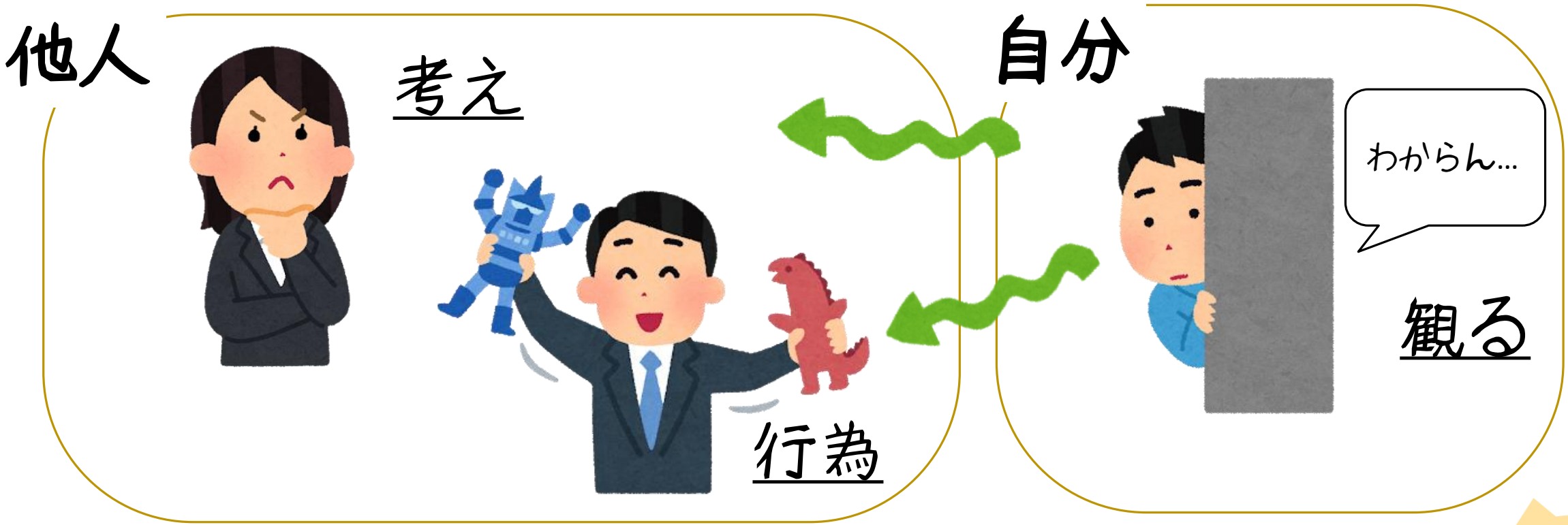
"納得"とは

例えば…他人の考え・行為が見えなかったり…



"納得"とは

見えたとしても、理解のための意図や目的は理解できないと…



"納得"とは

認められず、無意識に拒否したり、理解できず混乱する。

他人

考え

行為

自分

拒否

混乱

"納得"とは

自分自身の拒否がその相手（他人）に伝わると…

"納得"とは

行く先は「争い」…。

他人

自分

拒否

混乱

"納得"するためには…

下記の取り組みが必要になる。

① 他人（自分含む）を決める
② 他人の行為・考えに触れる
③ 理解するための方法を持つ
④ 他人と考えを認め合う術を持つ

「他人①の考え・行為②を理解③し、もっともだと認める④」

"納得できるテスト設計"にするために

「テストに対する他人の考え・行為の目的や意図を理解し、
もっともだと認める」

そのためには

① 他人（自分含む）を決める ⇒ ①文脈、場面状況を決める
② 他人の行為・考えに触れる ⇒ ②他人の行為・考えを話し合う
③ 理解するための方法を持つ ⇒ ③可視化・整理する
④ 他人と考えを認め合う術を持つ ⇒ ④文脈、場面状況を確認する

テスト設計した結果

自分たちのテスト設計

① 文脈、場面状況を決める

上から実践

No.	内容
1	OSSプロジェクトへのアジャイルQAが分かる
2	エンドユーザのニーズに合わせた市場リリースができる
3	アジャイルのテストプロセスが知りたい
4	テストタイプ教えて
5	自動テストってどこをどんだけやればいいの？
6	テストケースの作り方が分かる
7	テストケースがメンテしやすい
8	アジャイルテスト参考文献を探すときに便利
9	これやってみよう、うまく行きそうと思える
10	だれでもアジャイルテストがわかる
11	抽象的にも具体的にもテストの内容を説明できる
12	アジャイルプロジェクトでのテストについてこれ見てと言える

自分たちのテスト設計

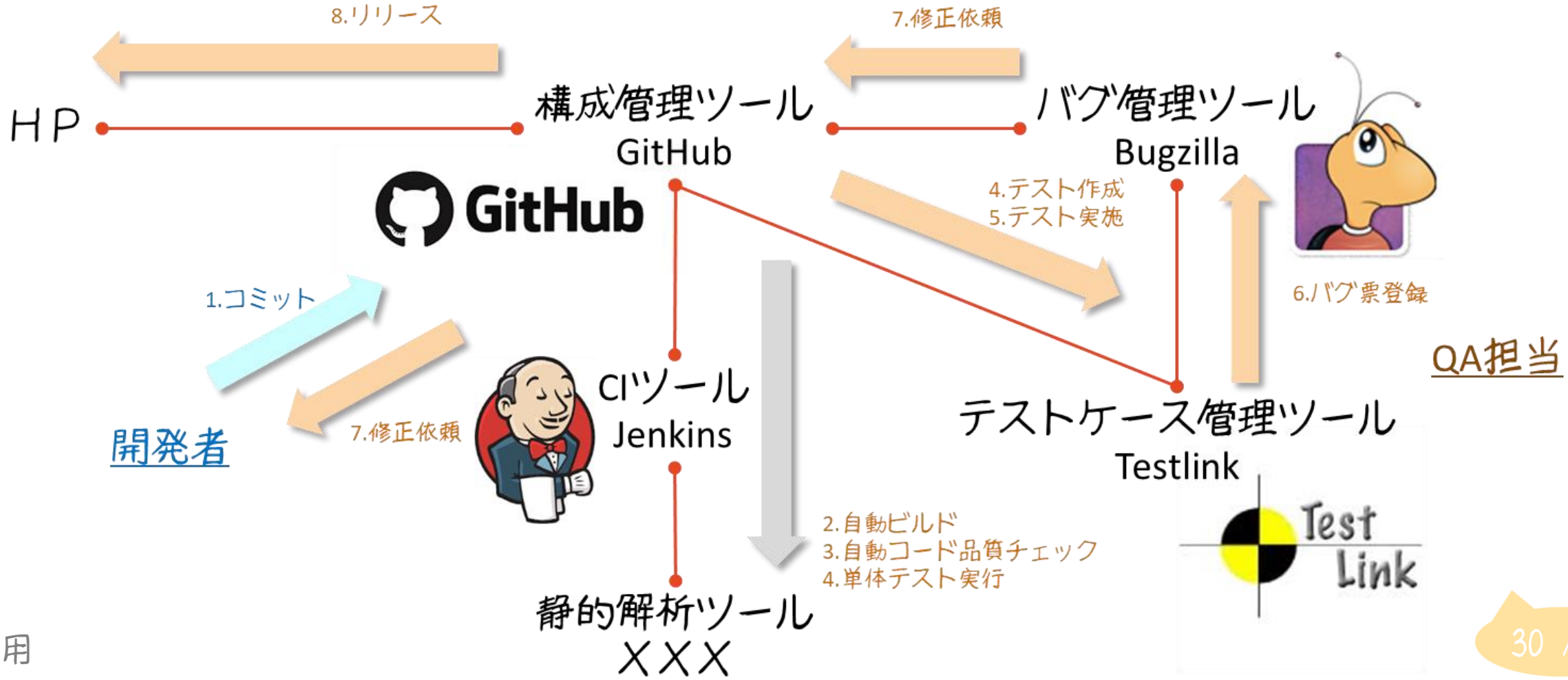
① 文脈、場面状況を決める

- 開発製品　　:オープンソースソフトウェアの通信カラオケシステム
- 立場　　:コミュニティメンバ QA担当
- リソース　　:2人
- リリース周期　　:2週間(最新版)、6か月(安定板)

製品のフィーチャーはコミュニティ開発者の意思で、適宜実装される。

自分たちのテスト設計

① 文脈、場面状況を決める



自分たちのテスト設計

② 他人の行為・考えを話し合う

「製品のリリースを止めるバグ
を許容範囲まで減らしたい！」

なんで？

自分たちのテスト設計

③ 可視化・整理する
理由を理解できる要素まで分解する。

バグがなかなか治らない！

ソフトウェア構造が
分からない！

フィードバックが多い！
(バグ票が大量に出る)

提起課題の全体像
が分からない！

課題の全体を整理
する人がいない！

修正箇所の影響範囲が
分からない！

バグが見過ごされる

コードをいじる
と不具合が出る

ソフトウェア構造の全体像を簡
単に把握できる手段がない！

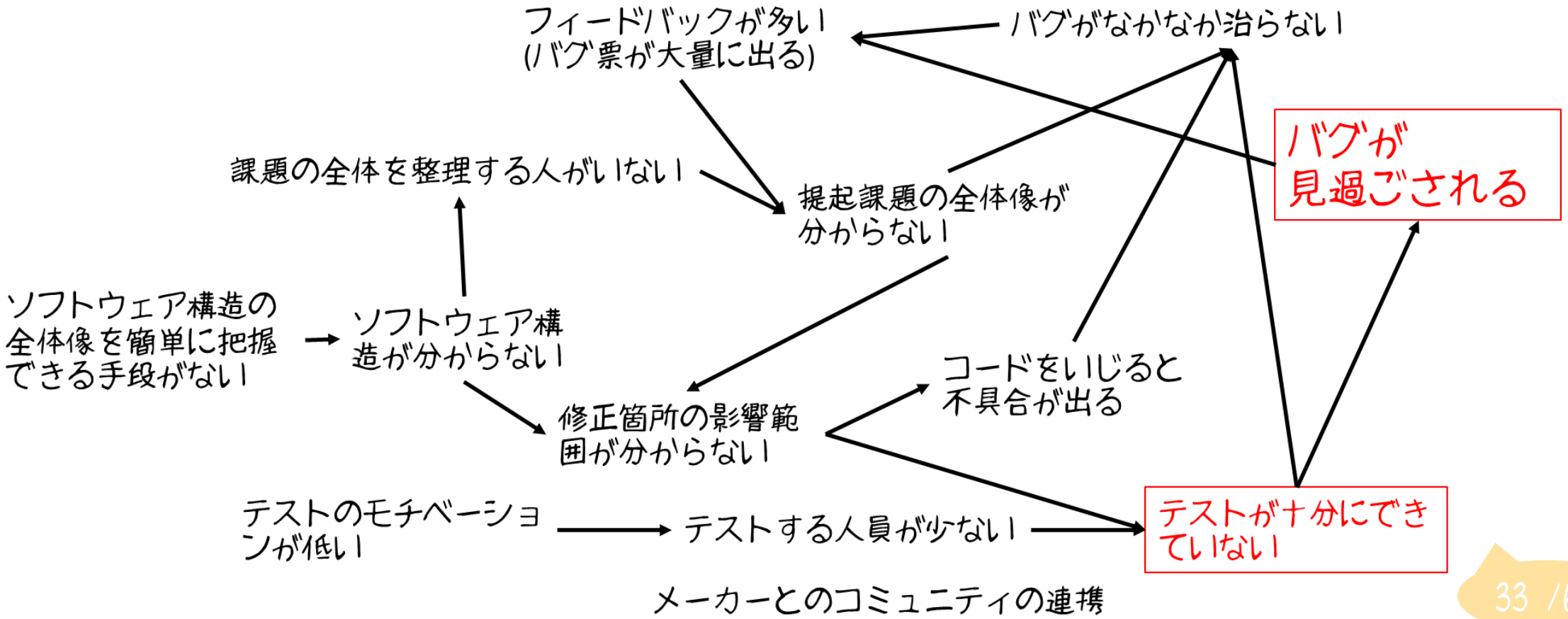
テストのモチベー
ションが低い！

テストが十分にでき
ていない！

テストする人員が少ない！

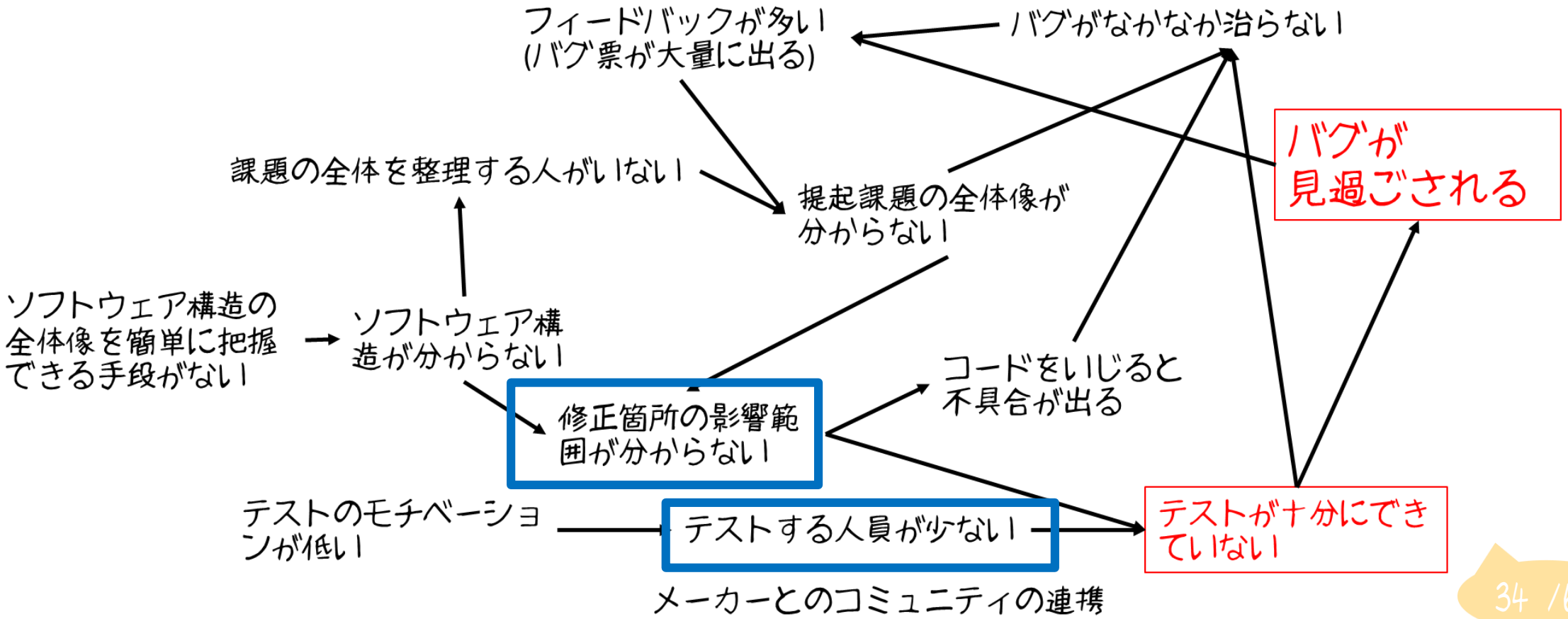
自分たちのテスト設計

③ 可視化・整理する
問題の因果関係を整理する。



自分たちのテスト設計

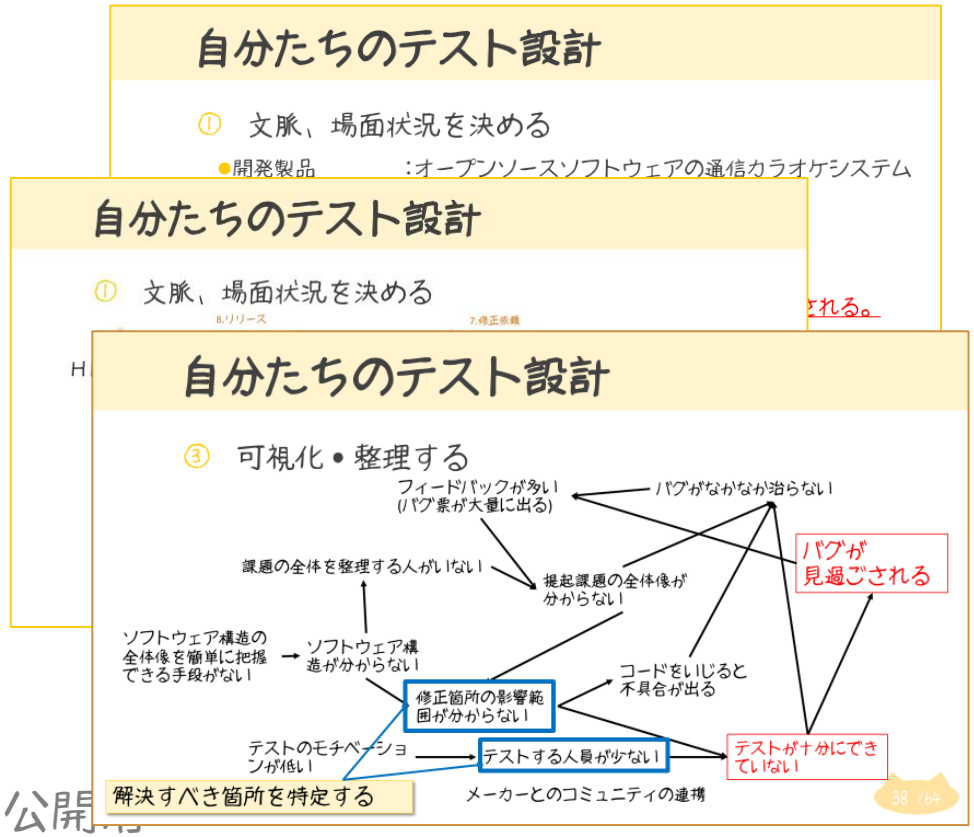
③ 可視化・整理する
解決すべき箇所を特定する。



自分たちのテスト設計

④ 文脈、場面状況を確認する

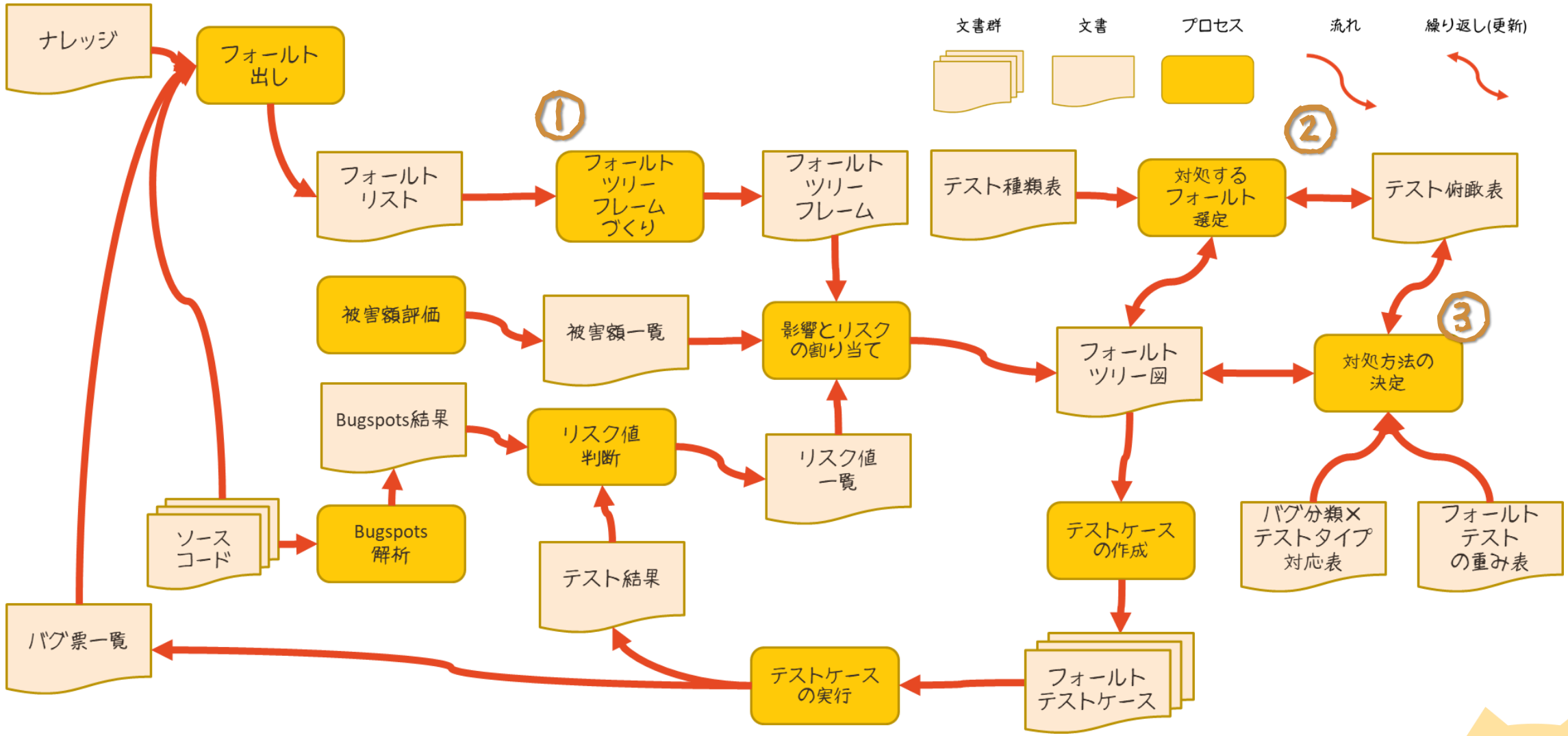
確認しながら開発しているテスト設計を行う。



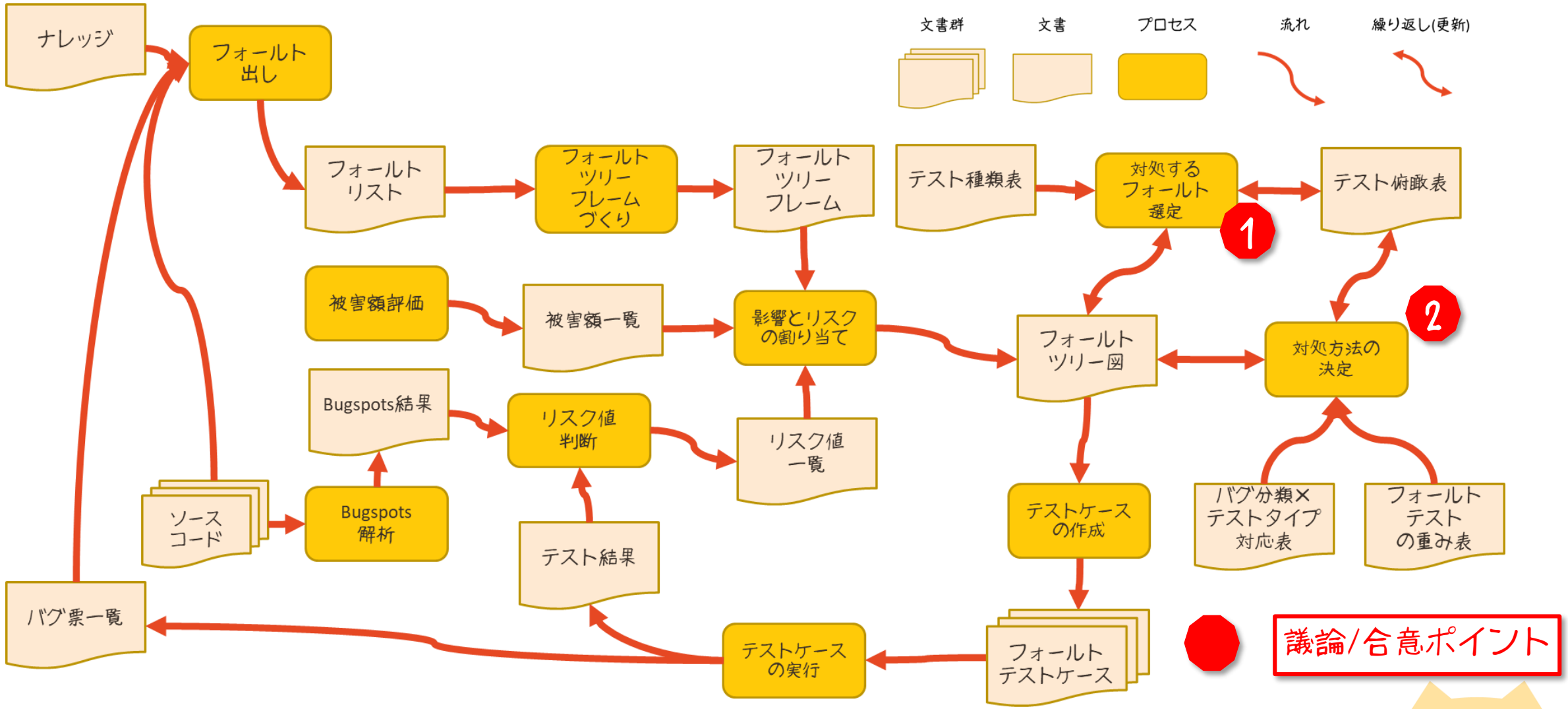
リリースについての課題

① 製品のリリースを止める事象およびバグをどう特定するか？
② テストで対処する/しない事象はどのように決めるか？
③ 対処するためにどんなテストをどれくらい行うか？

テストの全体像


文書群
文書
プロセス
流れ
繰り返し(更新)
ナレッジ
フォールト出し
フォールトリスト
①
フォールトツリーフレームづくり
フォールトツリーフレーム
テスト種類表
②
対処するフォールト選定
テスト網羅表
被害額評価
被害額一覧
影響とリスクの割り当て
フォールトツリー図
③
対処方法の決定
Bugspots結果
リスク値判断
リスク値一覧
ソースコード
Bugspots解析
テスト結果
テストケースの作成
バグ分類×テストタイプ対応表
フォールトテストの重み表
バグ票一覧
テストケースの実行
フォールトテストケース

テストの全体像


文書群
文書
プロセス
流れ
繰り返し(更新)
ナレッジ
フォールト出し
フォールトリスト
フォールトツリーフレームづくり
フォールトツリーフレーム
テスト種類表
対処するフォールト選定
テスト網羅表
1
被害額評価
被害額一覧
影響とリスクの割り当て
フォールトツリー図
対処方法の決定
2
Bugspots結果
リスク値判断
リスク値一覧
ソースコード
Bugspots解析
テスト結果
テストケースの作成
バグ分類×テストタイプ対応表
フォールトテストの重み表
バグ票一覧
テストケースの実行
フォールトテストケース
議論/合意ポイント

テストアーキテクチャ(フォールト)

1

事象の影響度(被害金額)と発生確率(リスク値)から
対処すべき事象(フォールト)を特定し、対応者について合意する。

被害金額

コード変更リスク

フォールトの
やばさ

テストアーキテクチャ(フォールト)

フォールトリスト

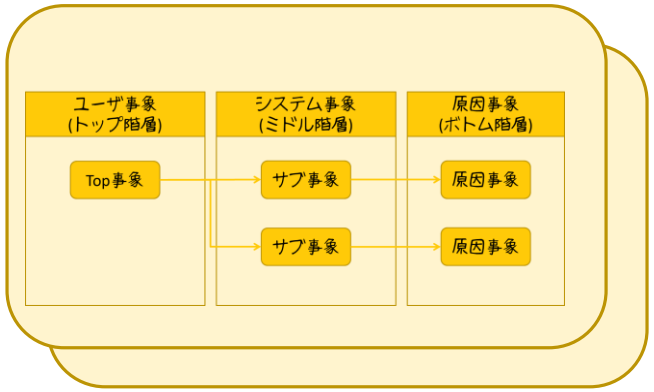
身体的リスク	不快な音波
	大音量
	気分が悪くなる映像が流れる
	チカチカ(光の点滅)
金銭的リスク	課金ができない！
	課金の計算が正しくない！
	課金データが改ざんされる
社会的要請	クラッキングされる
	踏み台にされる
	操作データが外部にもれる
	楽曲データがとられる
物理的破壊リスク	ハードウェアが壊れる
	バックアップデータが壊れる
	サーバーのデータを破壊する
	大量にデータをDL

被害金額

被害額一覧

身体的リスク	1000万円
金銭的リスク	500万
社会的要請	5億円
物理的破壊リスク	10万円

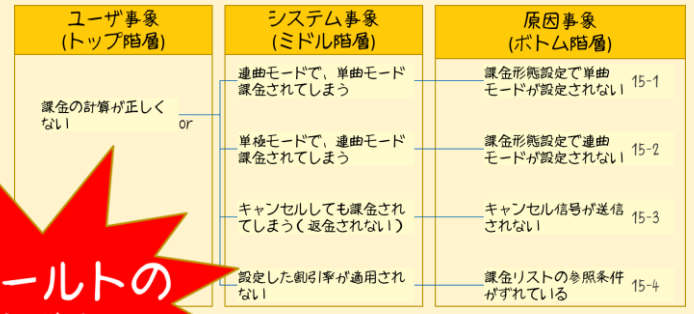
フォールトツリーフレーム



合意ポイント

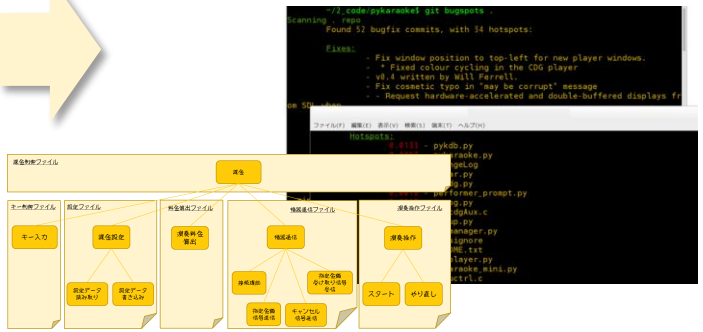
フォールトツリー図 1

被害額×リスク値の総和=やばさ：一位



フォールトのやばさ

リスク値一覧



コード変更リスク

公開用

フォールトツリー作成方法

まず、三層に分けてフォルトツリーを考える。

フォールトツリー作成方法

ステークホルダーの要求からTop事象を出す。

より導出するため、要求に対して
ガイドワードを適用し、Top事象を洗い出す。

要求からのTop事象(フォールト)だし

要求＋HAZOPガイドワードにより、Top事象を効率よく導出する。

オーナ

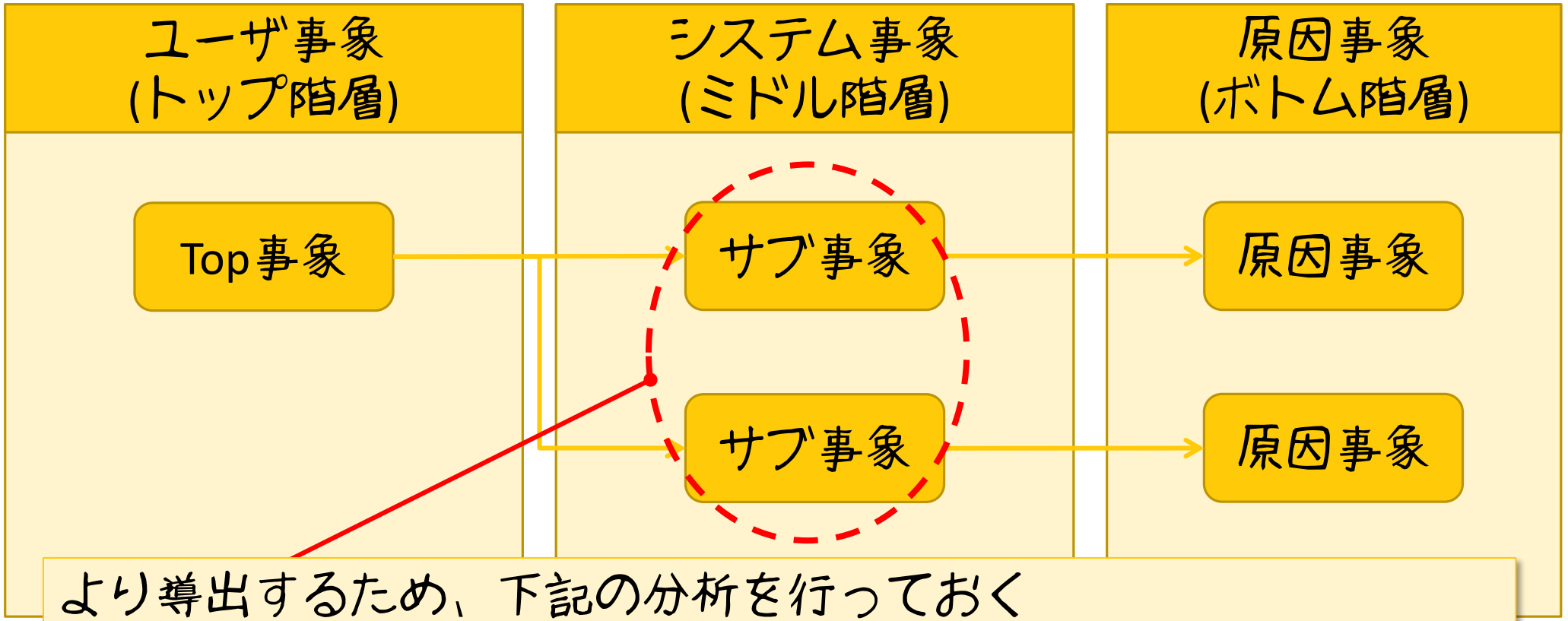
お客さんが利用してくれ、儲かる（課金ができる）

種類：違う

課金額を誤る

フォールトツリー作成方法

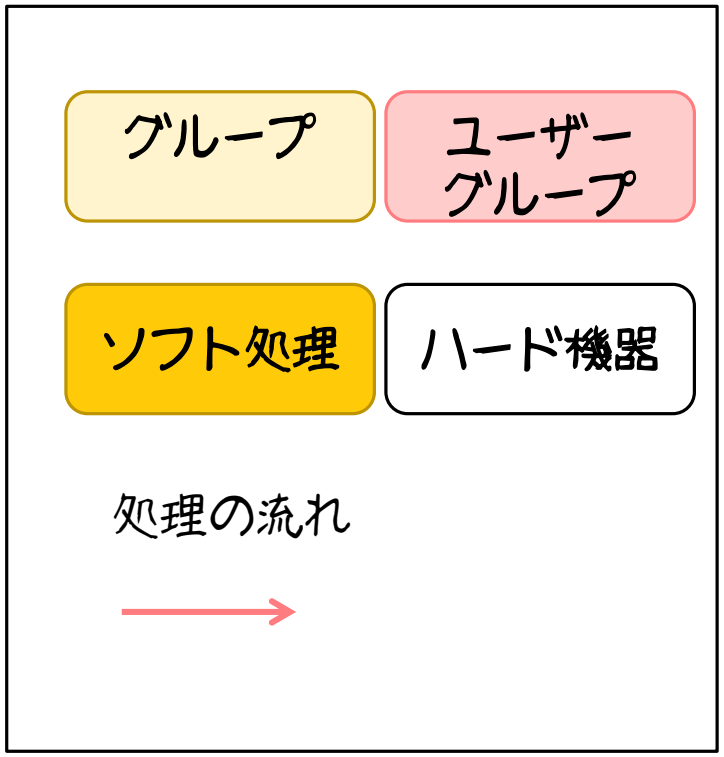
また他にハードとソフトの分析を行ってから作成する。



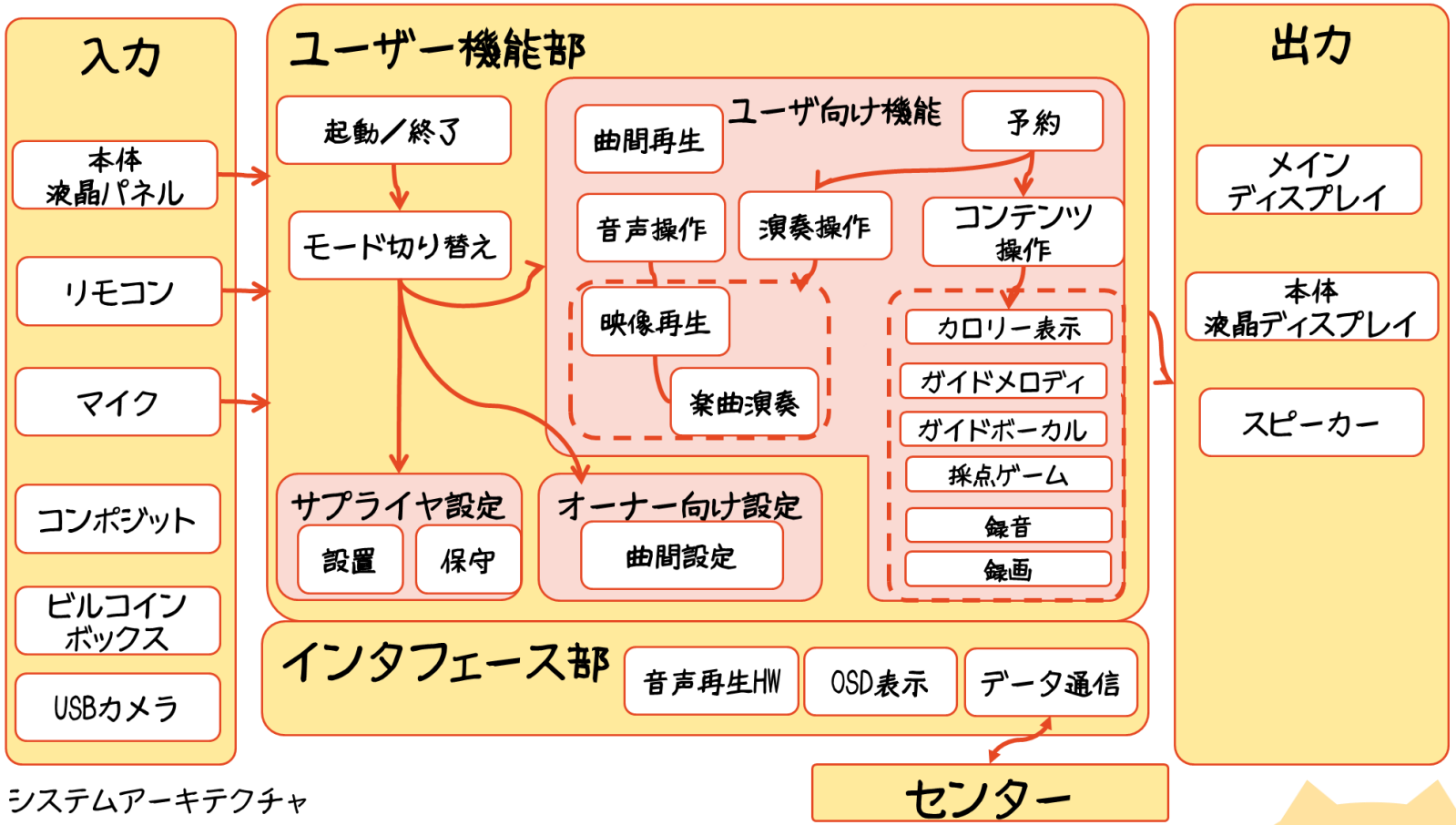
より導出するため、下記の分析を行っておく
1．システムのハード構成:機器の種類、機器の構成
2．システムのソフト構成:処理の種類、処理実装のファイル構成
※想定でもよいので出す

ハードの分析:機器の構成

ハードウェア機器の構成を考える。



例



システムアーキテクチャ

ソフトの分析:処理実装のファイル構成

ソフトウェア処理実装のファイル構成を考える。

ファイル

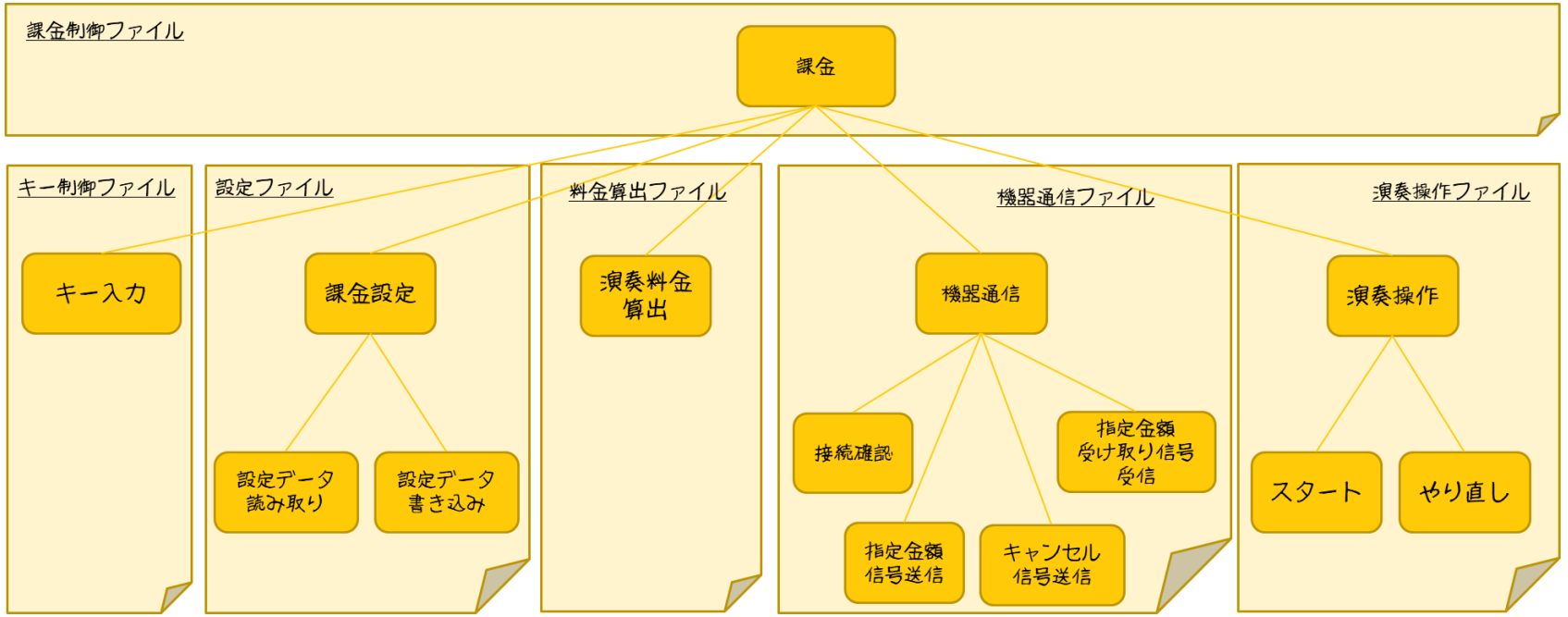
課金制御ファイル

処理

ソフト処理

関連

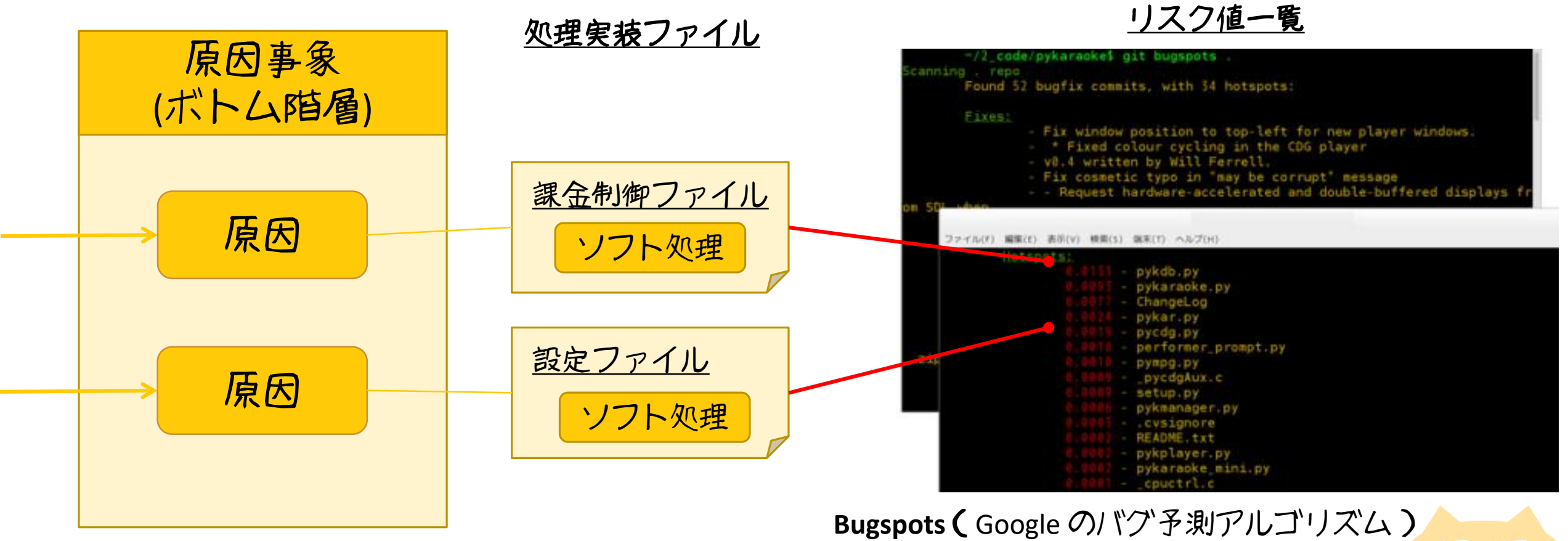
例



※可能であれば、処理の流れも書く。

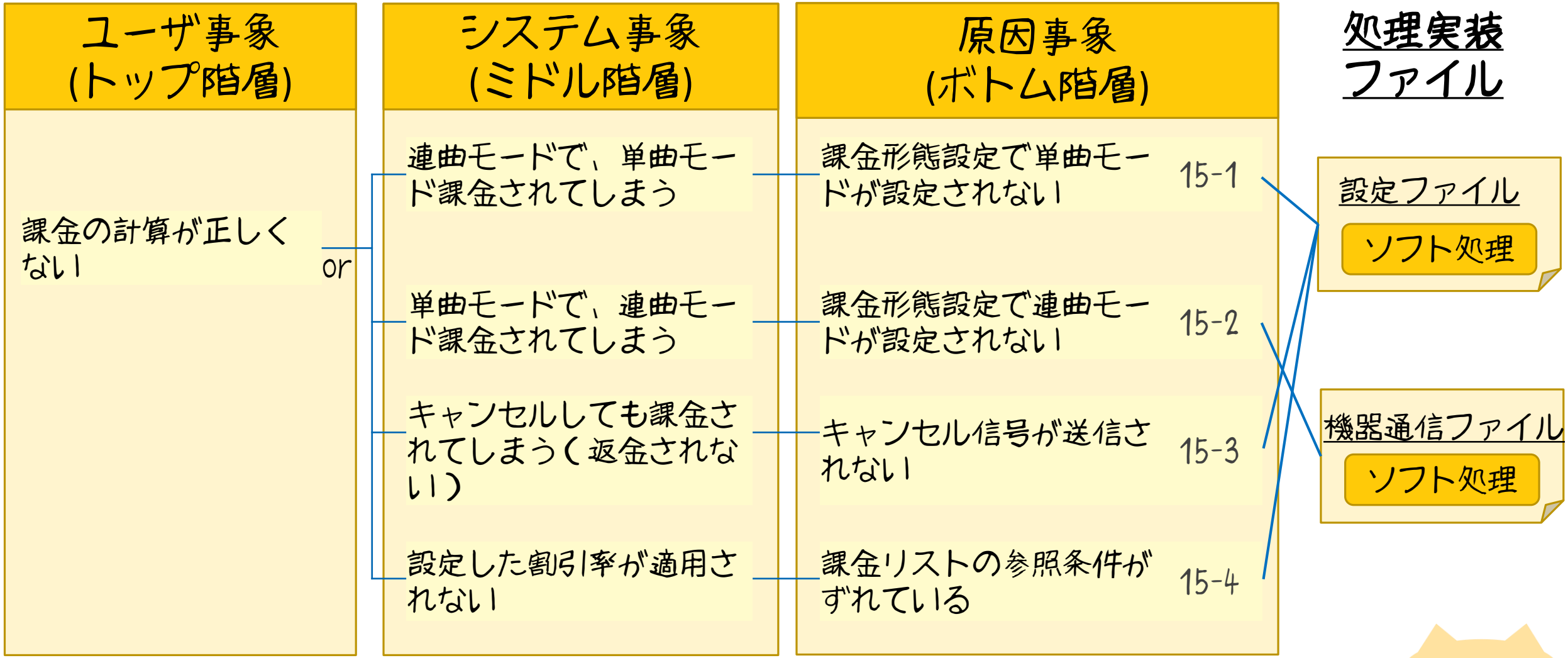
リスク値の算出と割り当て

原因に処理実装ファイルを紐づけ、リスク値を算出・割り当てる。

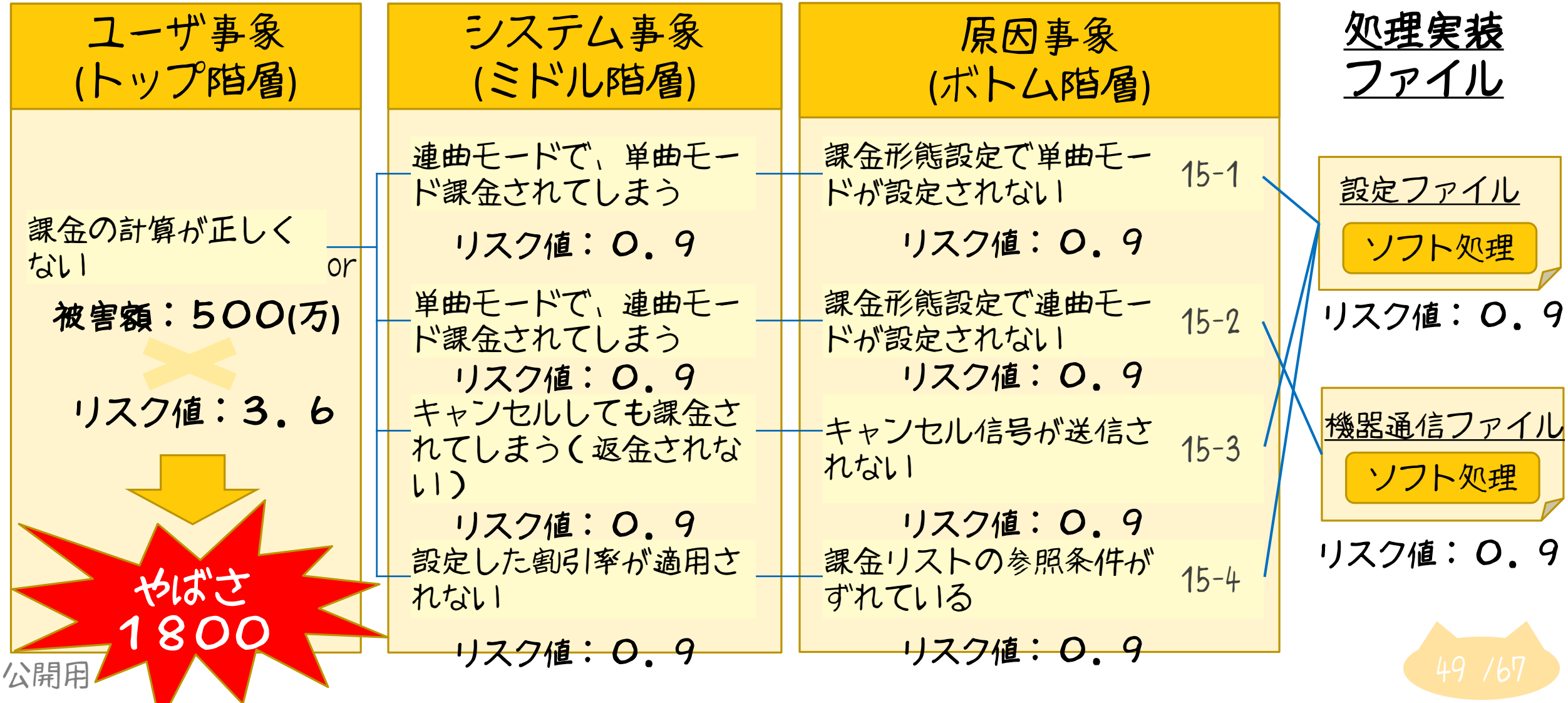


Bugspots（Google のバグ予測アルゴリズム）

フォールトツリーフレームサンプル



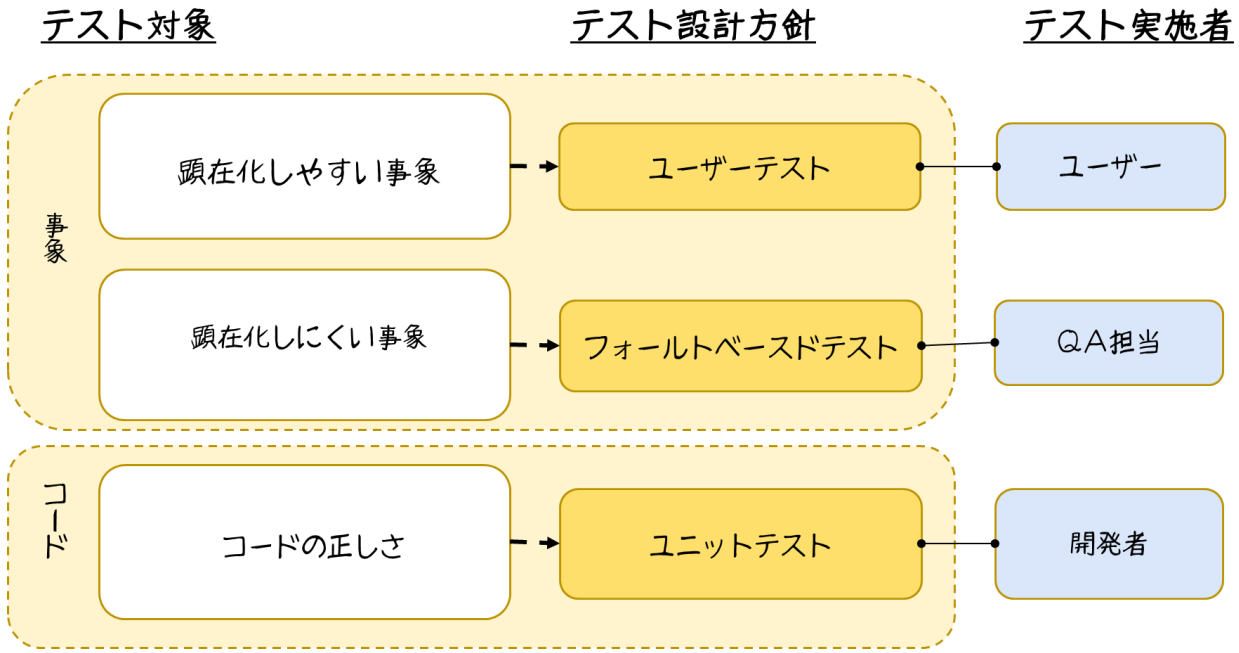
フォールトツリーフレームサンプル



テストアーキテクチャ(テスト設計方針)

2

テストで対処する起こってほしくない事象のために、テスト設計方針を合意する。



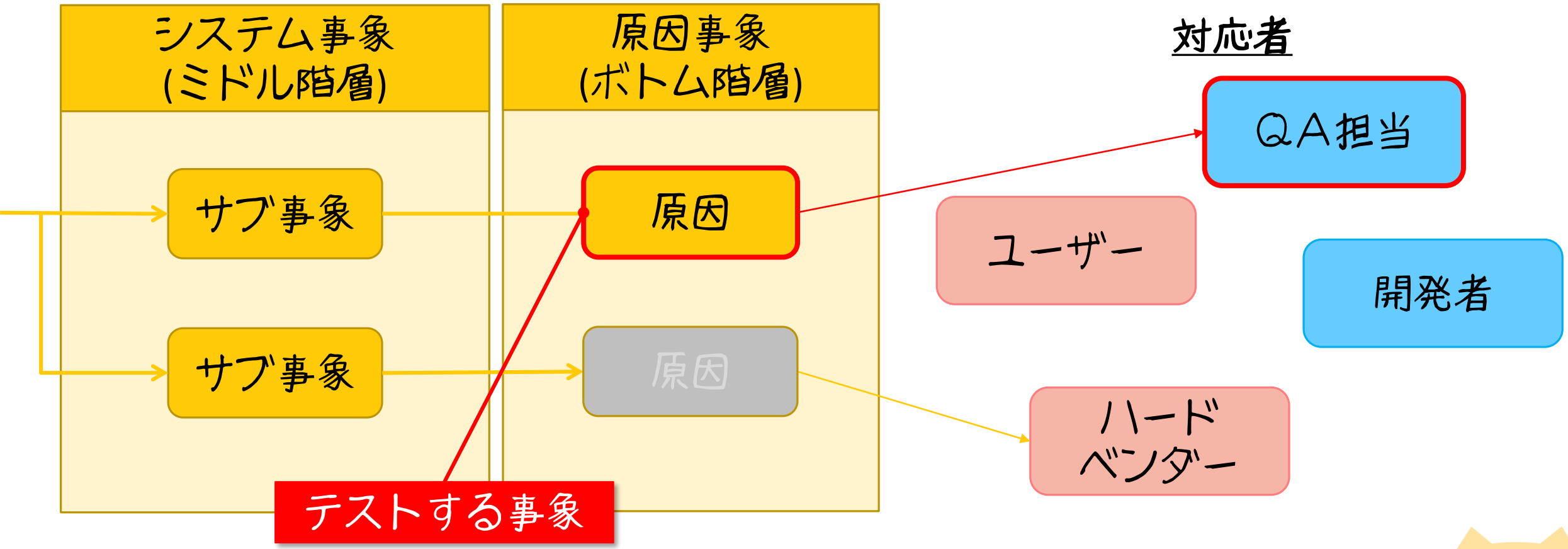
フォールトテストの重み表

		テストタイプ※1				
		データ	条件	シナリオ	イベント/タイミング	構成
テストの重みづけ	軽 最低限のテスト	有効/無効	重複結果なし	C0	0-switch	1-wise
	中	＋同値クラス	無効なし	C1	セル網羅	2-wise
	重	＋境界値	完全	C2	1-switch	3-wise

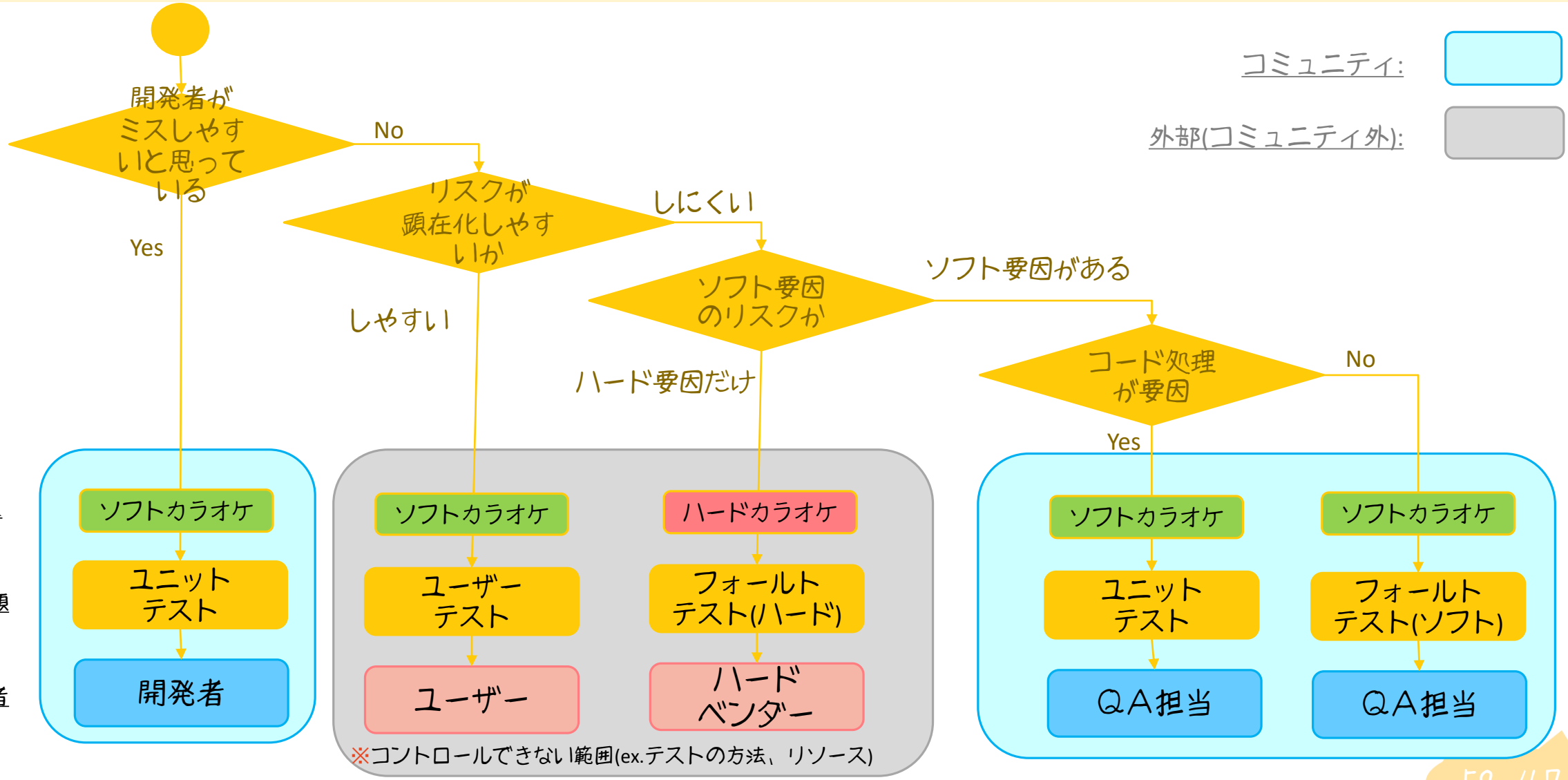
		テストタイプ				
		データ	条件	シナリオ	イベント/タイミング	構成
リスク値低減度	軽 最低限のテスト	31.00%	25.00%	56.76%	61.67%	23.57%
	中	71.00%	45.00%	77.35%	83.89%	80.71%
	重	95.00%	95.00%	95.00%	95.00%	95.00%

対処する事象(フォールト)の選定と対処方法割り当て

QAが対処すべきかを合意する。

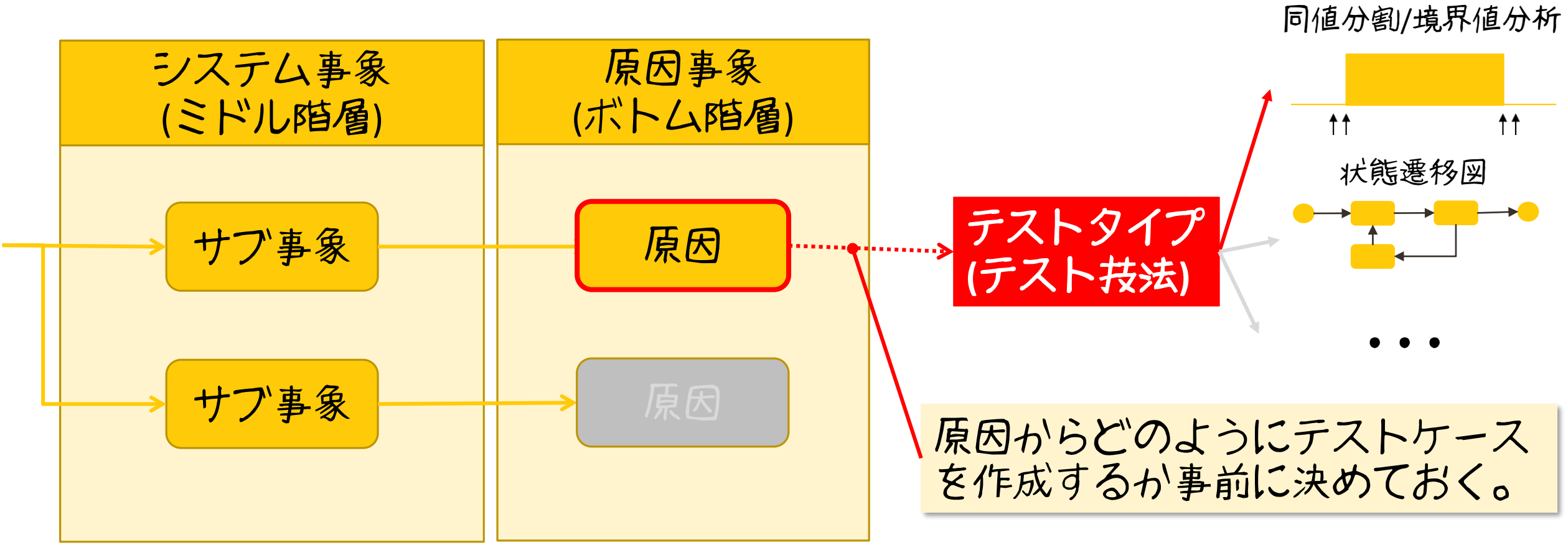


事象ごとのテスト担当決めフロー



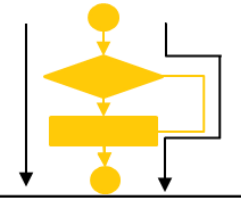
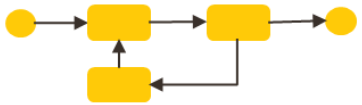
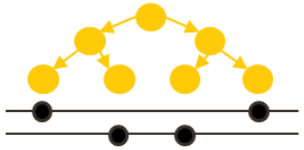
対処する事象(フォールト)の選定と対処方法割り当て

QAが対処すべき事象に対処方法（テストタイプ）を割り当てる。



テストタイプ⇔テスト技法対応表

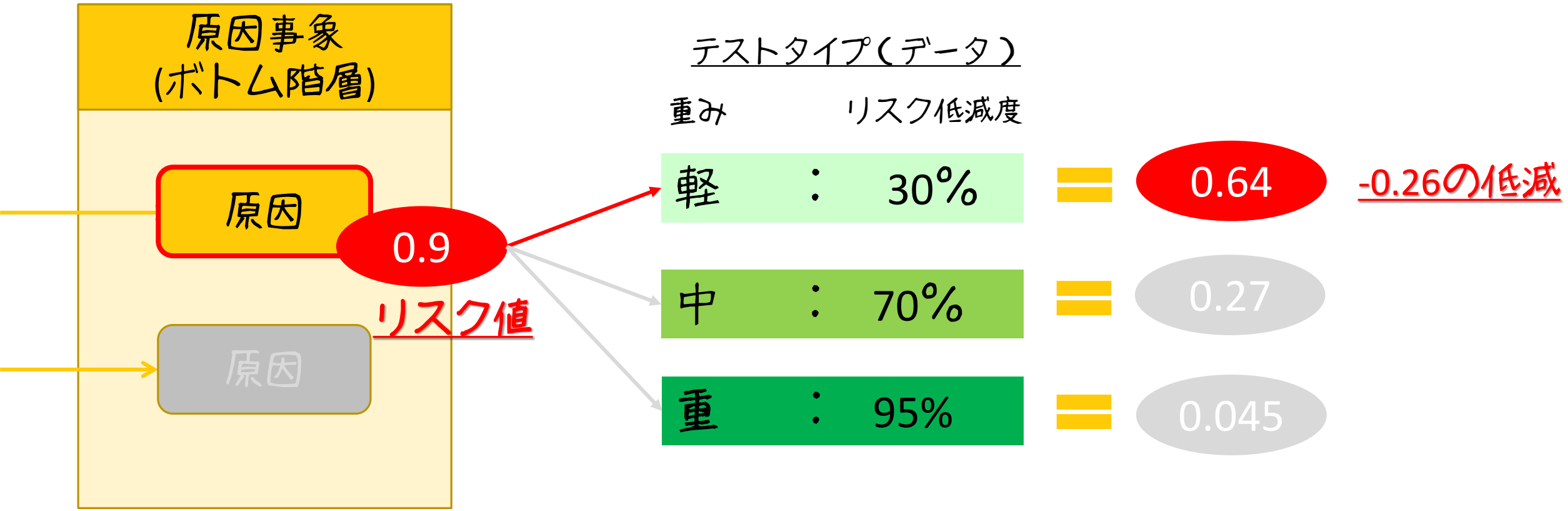
原因からどのようにテストケースを作成するか決めておく。

テストタイプ	(キーワード)	テスト技法	(テストケース導出モデル)
• データ (領域、境界、サイズ) • 性能	テストタイプ+ 抜け/漏れ ズレ/跳び 過剰/不足 矛盾/衝突 未反映/初期値/なし	同値分割 境界値分析	例 
• 流れ (シナリオ、処理)	同上	フローチャート	例 
• イベント/タイミング (状態、モード、遷移)	同上	状態遷移図/表	例 
• 構成 (環境、機器、モジュール)	同上	分類ツリー法	例 
• 条件 (入出力、制約)	同上	デシジョンテーブル	例

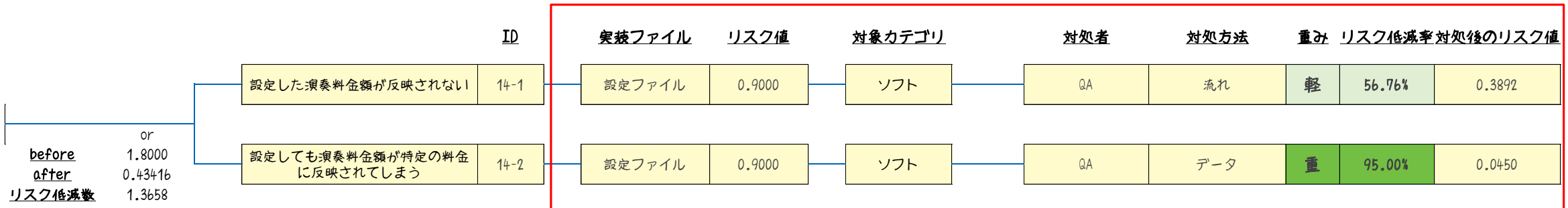
	1	2	3	4
A	F	F	T	T
B	F	T	F	T
Result		○	○	○

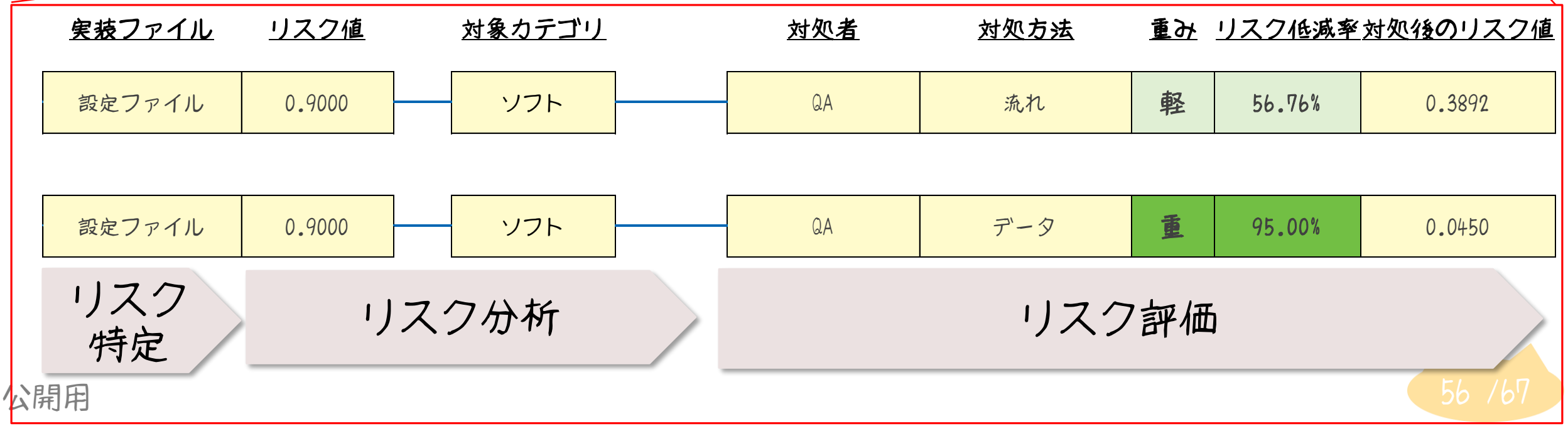
対処する事象(フォールト)の選定と対処方法割り当て

テストタイプの重みを割り当て、事象全体の「やばさ」を下げる。



フォールトツリー図サンプル





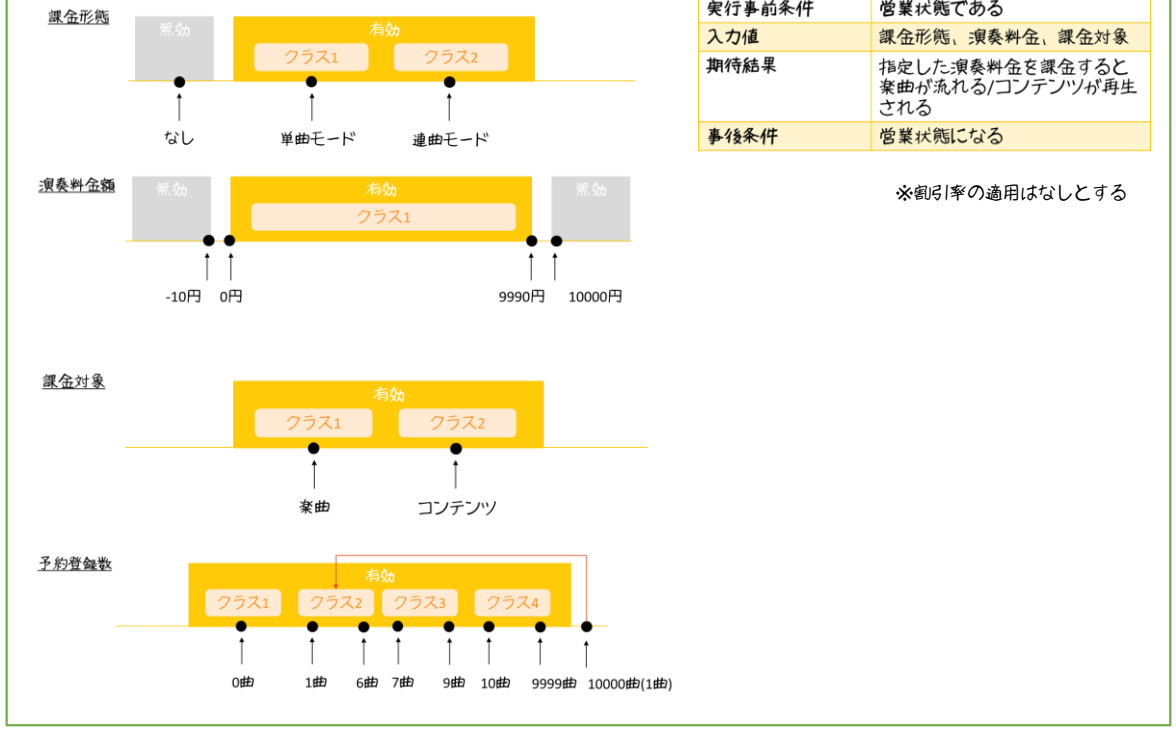
テストケースサンプル

テストケース導出モデルからテストケースを作成する。

テストケース導出モデル

テストケース導出モデル

テストタイプ	データ
テスト技法	同値分割/境界値分析



実行事前条件	営業状態である
入力値	課金形態、演奏料金、課金対象
期待結果	指定した演奏料金を課金すると楽曲が流れる/コンテンツが再生される
事後条件	営業状態になる

※割引率の適用はなしとする

テストケース

テストケース

重み	重
カバレッジ	有効/無効＋同値クラス＋境界値
実施テストケース数	168

テストケースNo.	入力条件（事前状態）					期待結果	事後条件	事象ＩＤ
	課金形態	演奏料金額	課金対象	予約登録数				
1	連曲モード	-10円	コンテンツ	1曲		設定できない	営業状態になる	14-1
2	単曲モード	-10円	コンテンツ	1曲		設定できない	同上	16-1
3	連曲モード	-10円	コンテンツ	10000曲		設定できない	同上	
4	なし	10000円	楽曲	9999曲		設定できない	同上	
5	連曲モード	10000円	コンテンツ	10000曲		設定できない	同上	
6	なし	0円	コンテンツ	6曲		課金されない	同上	
7	単曲モード	10000円	楽曲	9曲		設定できない	同上	
8	単曲モード	0円	コンテンツ	7曲		指定金額で課金される	同上	
9	単曲モード	0円	楽曲	6曲		指定金額で課金される	同上	
10	連曲モード	0円	コンテンツ	7曲		指定金額で課金される	同上	
11	単曲モード	9990円	楽曲	0曲		指定金額で課金される	同上	
12	連曲モード	9990円	コンテンツ	10000曲		指定金額で課金される	同上	
13	なし	9990円	楽曲	6曲		課金されない	同上	
14	単曲モード	0円	コンテンツ	9999曲		指定金額で課金される	同上	
15	なし	-10円	コンテンツ	6曲		設定できない	同上	
16	単曲モード	10000円	楽曲	0曲		設定できない	同上	
17	単曲モード	10000円	コンテンツ	9999曲		設定できない	同上	
18	単曲モード	-10円	コンテンツ	0曲		設定できない	同上	
19	単曲モード	9990円	コンテンツ	10000曲		指定金額で課金される	同上	
20	なし	0円	コンテンツ	7曲		課金されない	同上	
21	単曲モード	-10円	楽曲	9999曲		設定できない	同上	
22	なし	-10円	コンテンツ	10000曲		設定できない	同上	
23	連曲モード	0円	楽曲	6曲		指定金額で課金される	同上	
24	連曲モード	-10円	コンテンツ	9曲		設定できない	同上	
25	単曲モード	0円	楽曲	1曲		指定金額で課金される	同上	
26	なし	9990円	コンテンツ	1曲		課金されない	同上	
27	単曲モード	9990円	コンテンツ	6曲		指定金額で課金される	同上	
28	なし	0円	楽曲	10000曲		課金されない	同上	
29	なし	9990円	楽曲	7曲		課金されない	同上	

…

リスク•テスト全体を俯瞰

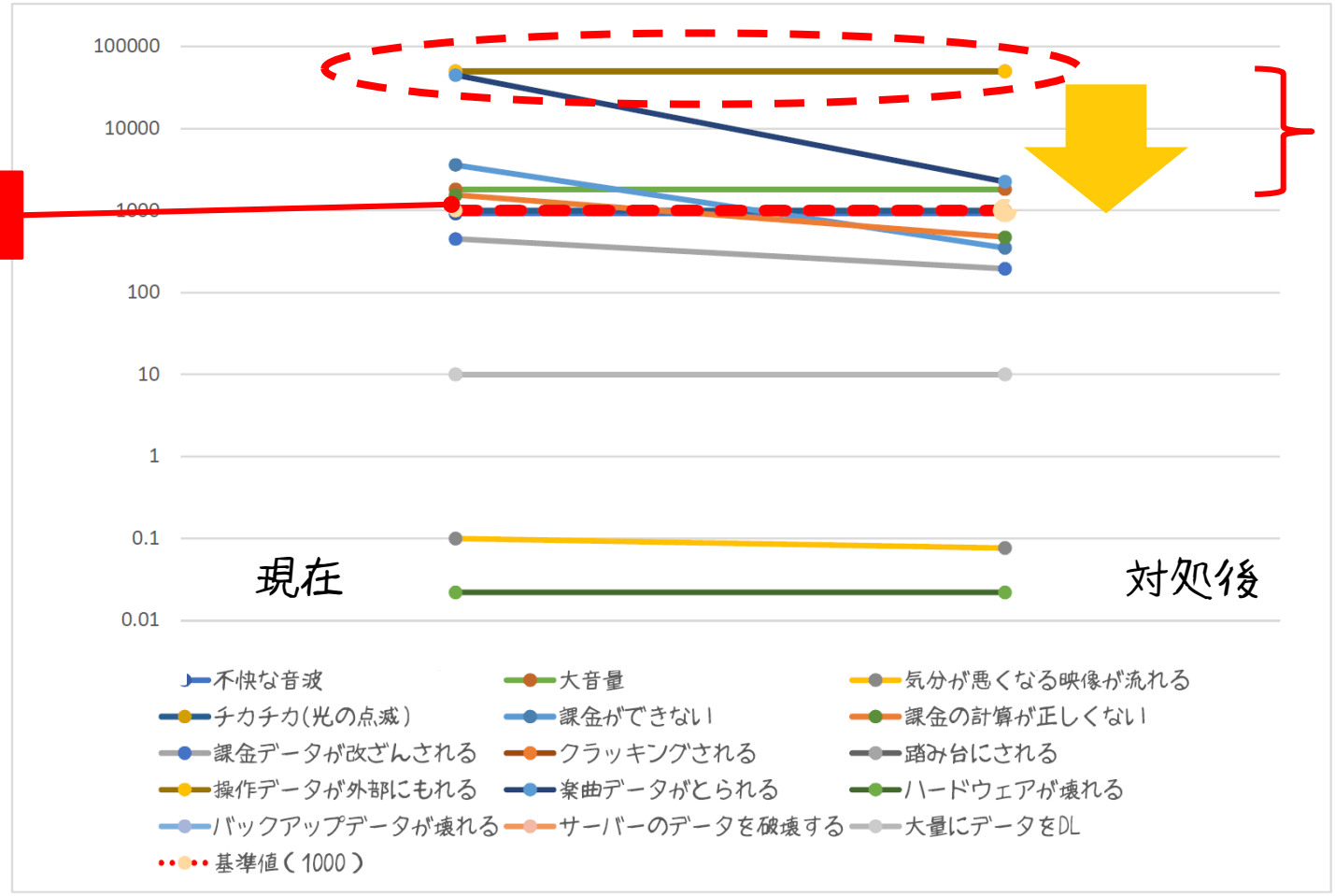
「やばさ」に対して、許容範囲になっているか、何をテストするかを俯瞰する。

テスト俯瞰表

				やばさ							テストタイプ				
カテゴリ	リスクカテゴリ	No.	フォールト項目	Rank	現在	⇒	Rank		低減数	低減率	データ	流れ	条件	イベント/タイミング	構成
顕在化しにくい	身体的リスク	1	不快な音波	9	917.8	⇒	7	917.8	0	0.00%	ー	ー	ー	ー	ー
		2	大音量	6	1809.32	⇒	5	1809.32	0	0.00%	ー	ー	ー	ー	ー
		3	気分が悪くなる映像が流れる	14	0.10	⇒	14	0.10	0	0.00%	ー	ー	ー	ー	ー
		4	チカチカ(光の点滅)	8	1000	⇒	6	1000	0	0.00%	ー	ー	ー	ー	ー
	金銭的リスク(課金)	14	課金ができない	5	3600.00	⇒	9	352.08	3247.92	90.22%	○	○	○	○	ー
		15	課金の計算が正しくない	7	1544.58	⇒	8	472.5	1072.08	69.41%	ー	ー	ー	○	○
		16	課金データが改ざんされる	10	450.00	⇒	10	194.58	255.42	56.76%	ー	○	ー	ー	ー
	社会的要請	17	クラッキングされる	1	50000	⇒	1	50000	0	0.00%	ー	ー	ー	ー	ー
		18	踏み台にされる	1	50000	⇒	1	50000	0	0.00%	ー	ー	ー	ー	ー
		19	操作データが外部にもれる	1	50000	⇒	1	50000	0	0.00%	ー	ー	ー	ー	ー
		20	楽曲データがとられる	4	45000	⇒	4	2250	42750	95.00%	ー	ー	ー	ー	ー
	物理的破壊リスク	21	ハードウェアが壊れる	15	0.022	⇒	15	0.022	0	0.00%	ー	ー	ー	ー	ー
		22	バックアップデータが壊れる	11	10	⇒	11	10	0	0.00%	ー	ー	ー	ー	ー
		23	サーバーのデータを破壊する	11	10	⇒	11	10	0	0.00%	ー	ー	ー	ー	ー
		24	大量にデータをDL	11	10	⇒	11	10	0	0.00%	ー	ー	ー	ー	ー
顕在化しやすい	うまく使えないリスク	5	ハウリング	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		6	音が出ない	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		7	映像が出ない	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		8	字幕がずれる	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		9	途中で止まる	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		10	テンポが変わる	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		11	字幕が切れる	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		12	となりの音が割り込む	ー	ー	⇒		ー			ー	ー	ー	ー	ー
		13	起動しない	ー	ー	⇒		ー			ー	ー	ー	ー	ー
					204351.82	⇒		157026.4	47325.4	23.16%					

リスク•テスト全体を俯瞰

「やばさ」がどうなるかをそのリリース毎に確認していく。



実現できそうか？

テスト開発コンセプト

「製品のリリースを止めるバグを許容範囲まで減らす！」

課題

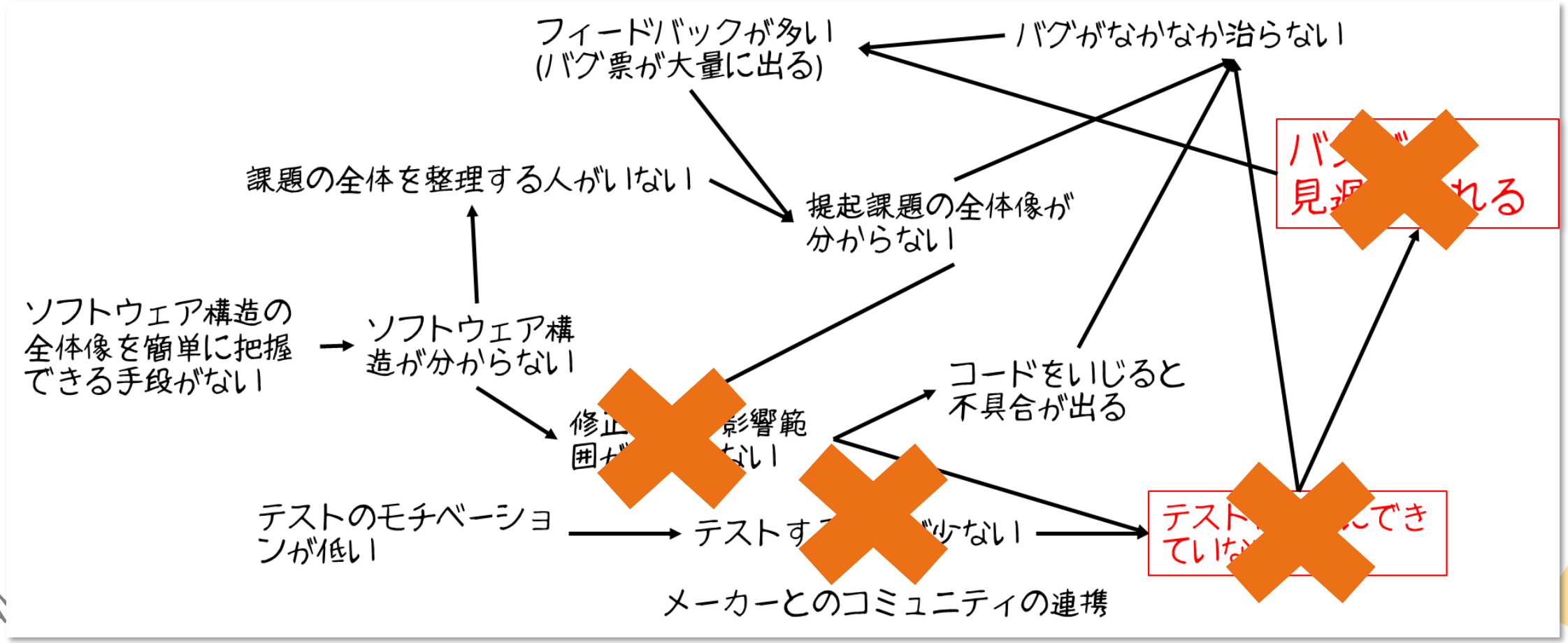
1. 製品のリリースを止める事象およびバグをどう特定するか？
   ⇒フォールトツリー図を用いた事象の分解
2. テストで対処する/しない事象はどのように決めるか？
   ⇒被害金額とコード変更履歴からのリスク値による優先順位
3. 対処するためにどんなテストをどれくらい行うか？
   ⇒テストタイプのリスク低減表による事象の許容範囲への調節

実現できそうか?

テストする人員が少なくても、
修正箇所から影響範囲を見極めてテストを十分できる

テスト開発コンセプト

「製品のリリースを止めるバグを許容範囲まで減らす！」



開発したテスト設計

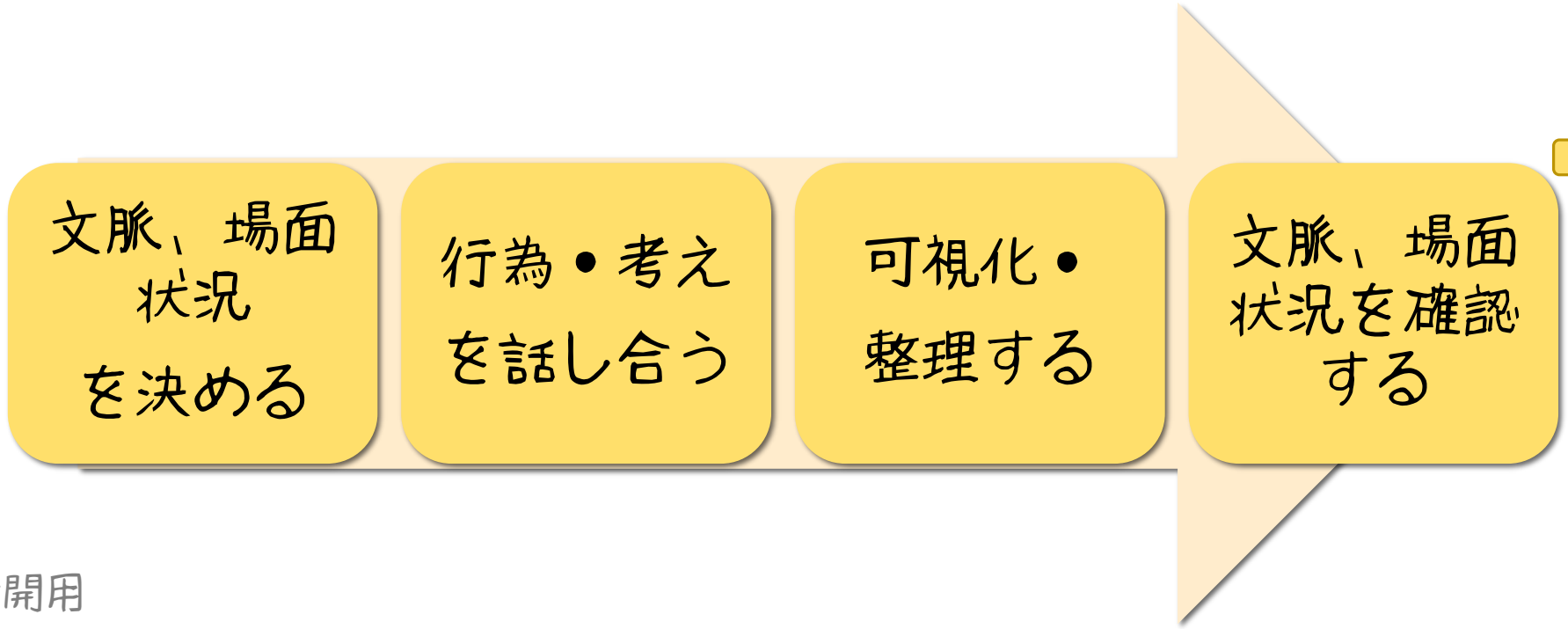
納得できている/できていないところが今後の課題。

◆納得できていること
- 文脈や場面状況にあったテスト設計になった
- 実践できるテスト技術を習得・開発して適用できた

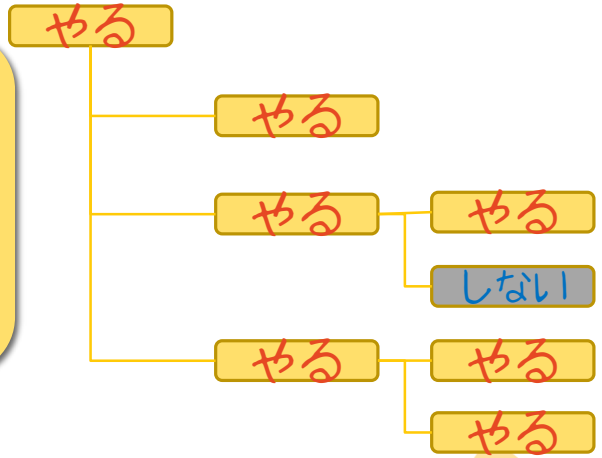
◆納得できていない(未消化な)こと
- テスト設計コンテスト審査員からのフィードバック
  - 数値化したやばさ、被害金額の妥当性
  - テストケースを十分性
- 文脈や場面状況が同じ、実テーマへの実践とフィードバック
  - やりきれない場面が出てくるはず…その際の課題と解決方法の検討

まとめ

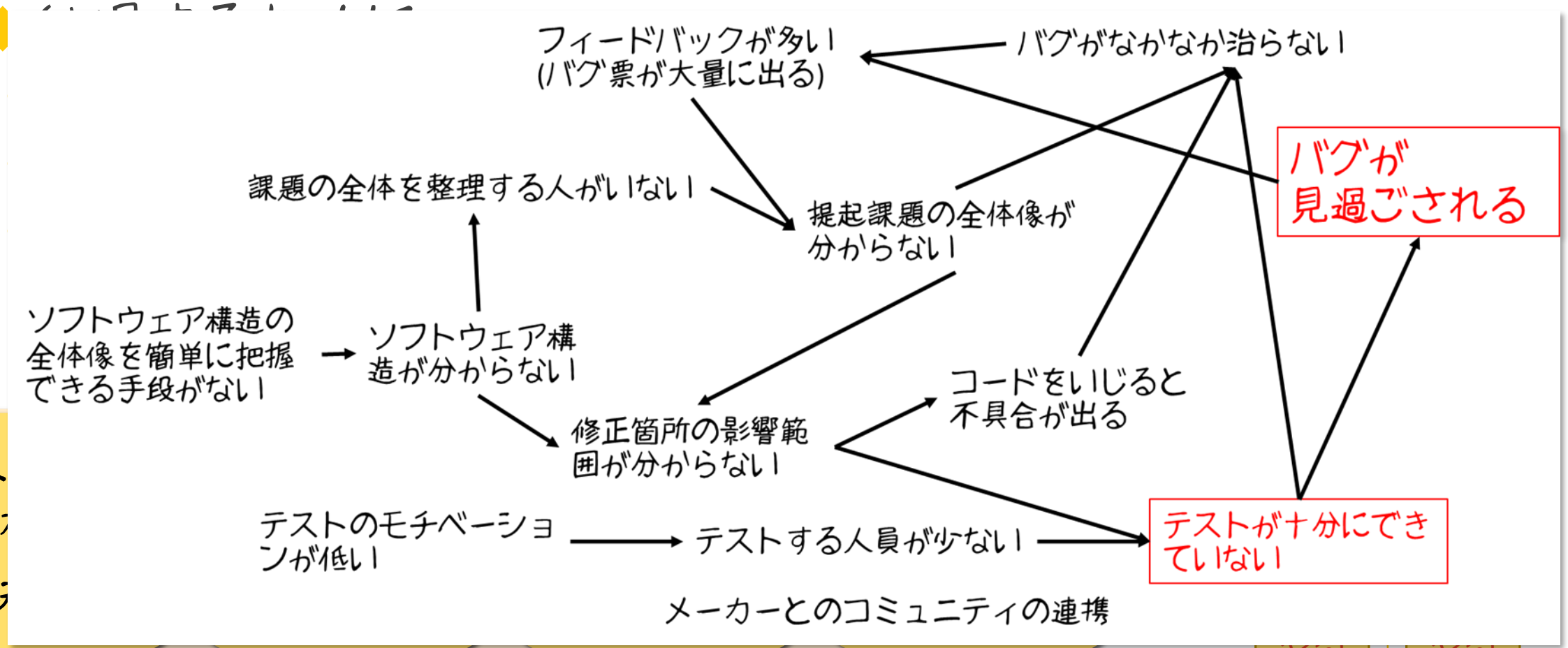
- 納得するために
  - 関連する他人の考えや行為を観察範囲を増やし、整理する
  - 整理したものを通して、関連する他者と議論する
  - 納得できている/できていないものを明確にして取り組む



テスト要求(目的)モデル



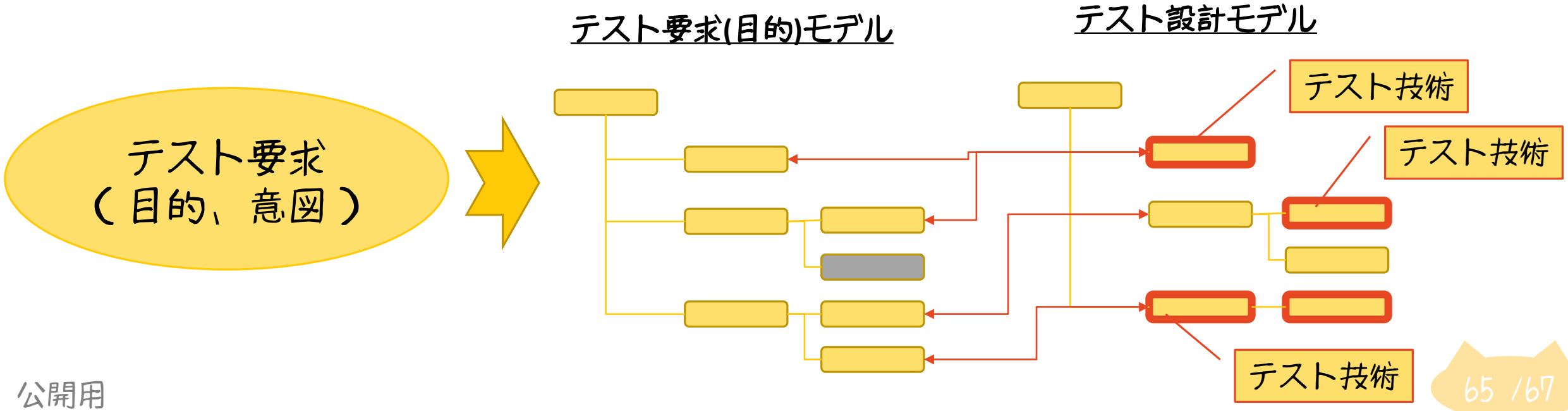
まとめ



自分がやり易いやり方でやってみる

まとめ

- ワンステップするために
  - 関係する自分以外の押さえるべき考えや行為をないがしろにしない
  - 考えや行為の観察範囲を整理して成長、洗練させていく
  - 合わせてテスト技術を取り込んでいく（無暗に使わない）



まとめ

- ワンステップするために
  - 関係する自分以外の押
  - 考えや行為の観察範囲
  - 合わせてテスト技術を

テス

テスト要求
（目的、意図）

一つずつ試していく

```
~/2_code/pykaraoke$ git bugspots .
Scanning . repo
        Found 52 bugfix commits, with 34 hotspots:

        Fixes:
                - Fix window position to top-left for new player windows.
                -  * Fixed colour cycling in the CDG player
                - v0.4 written by Will Ferrell.
                - Fix cosmetic typo in "may be corrupt" message
                - - Request hardware-accelerated and double-buffered displays fr
om SDL when
```

ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)

```
        Hotspots:
                0.0133 - pykdb.py
                0.0093 - pykaraoke.py
                0.0077 - ChangeLog
                0.0024 - pykar.py
                0.0019 - pycdg.py
                0.0018 - performer_prompt.py
                0.0010 - pympg.py
                0.0009 - _pycdgAux.c
                0.0009 - setup.py
                0.0006 - pykmanager.py
                0.0003 - .cvsignore
                0.0002 - README.txt
                0.0002 - pykplayer.py
                0.0002 - pykaraoke_mini.py
                0.0001 - _cpuctrl.c
```

Bugspots（Google のバグ予測アルゴリズム）

まとめ

◆納得するために
- 関連する他者の考えや行為を観察範囲を増やし、整理する
- 整理したものを通して、関連する他者と議論する
- 納得できている/できていないものを明確にして取り組む

◆ワンステップするために
- 関係する自分以外の押さえるべき考えや行為をないがしろにしない
- 考えや行為の観察範囲を整理して成長、洗練させていく
- 合わせてテスト技術を取り込んでいく（無暗に使わない）

"テストの目的や意図"を理解し、押さえる方法確立して
自分なりのテストを開発して、育てていきましょう！

ご清聴
ありがとうございました

参考資料

1. ASTER(2017)『テスト設計コンテスト’18 チュートリアル資料 Open版』ASTER.
2. International Software Testing Qualifications Board 用語集作業班(2014)『ソフトウェアテスト標準用語集(日本語版)Version 2.3.J01』 Japan Software Testing Qualifications Board 技術委員会.
3. JaSST Kansai実行委員会(2016)『「ソフトウェアテストの価値って?」 ～ てすにゃんくえすと～』http://jasst.jp/symposium/jasst16kansai/pdf/S4-1.pdf (2018/06/11 アクセス)
4. Igrigorik (2011)『Bugspots - Bug Prediction Heuristic』 https://github.com/igrigorik/bugspots (2018/06/11 アクセス)