2008/1/30
Jasst'08 in Tokyo

NEC
Communication Systems

Risk Based Testingの実践方法と適用事例からの考察
～短期間で要求品質を確保する～

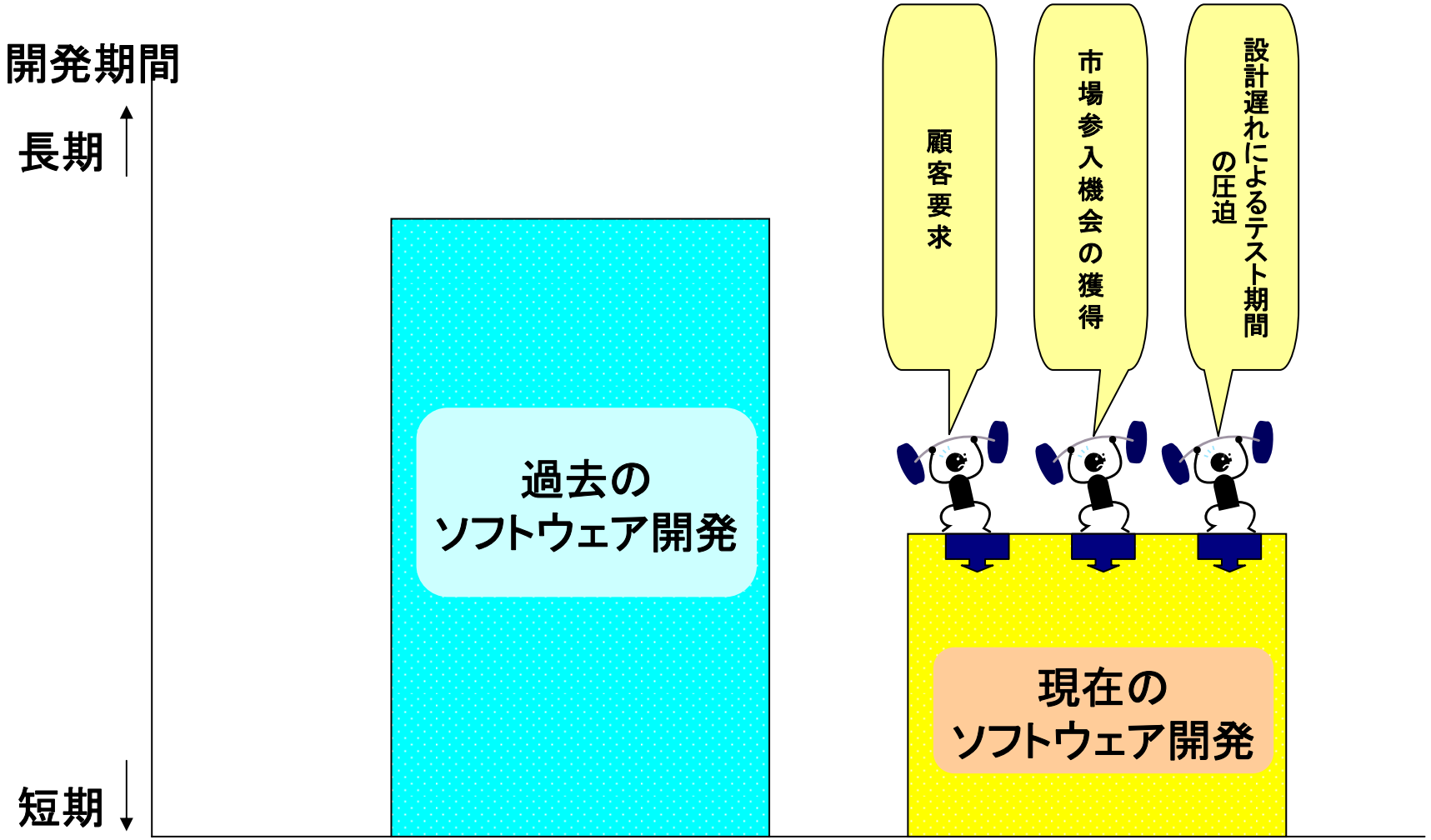
石田　智亮/石山　康介/米田征弘(NEC通信システム)
西　康晴(電気通信大学)

Agenda

1. Introduction(リスクベースドテストってなんだろう？)
2. Risk based Testingの適用手順
3. 適用結果
4. 適用全般に対する考察

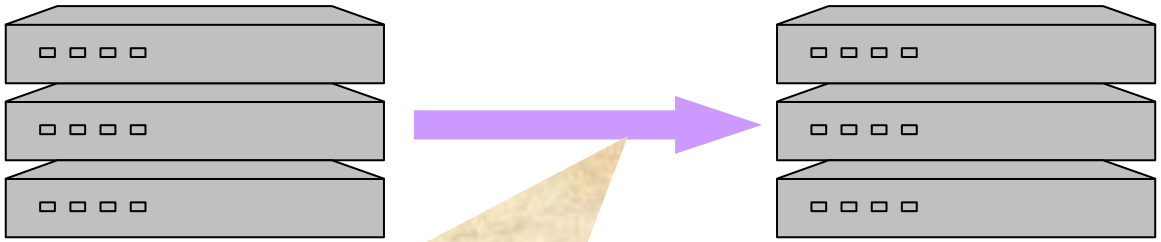
1.Introduction～リスクベースドテストってなんだろう？～

1.Introduction～リスクベースドテストってなんだろう？～



1.Introduction～リスクベースドテストってなんだろう？～

- 質問：全部テストできますか？



ホストAからホストBへのコントロールは、制御信号で行います。

7 6 5	4 3 2	1 0
C(3)	B(3)	A(2)
D(8)		
E(16)		

左の信号について全部テストしたら・・・・・

A（2ビット）	4通り	
B（3ビット）	8通り	4×8＝32通り
C（3ビット）	8通り	32×8＝256通り
D（8ビット）	256通り	256×256＝65，536通り
E（16ビット）	65536通り	65536×65536＝4,294,967,296通り

（答え）1つの制御信号でさえ、ソフトウェア開発では全てテストできません！

1.Introductin～リスクベースドテストってなんだろう？～

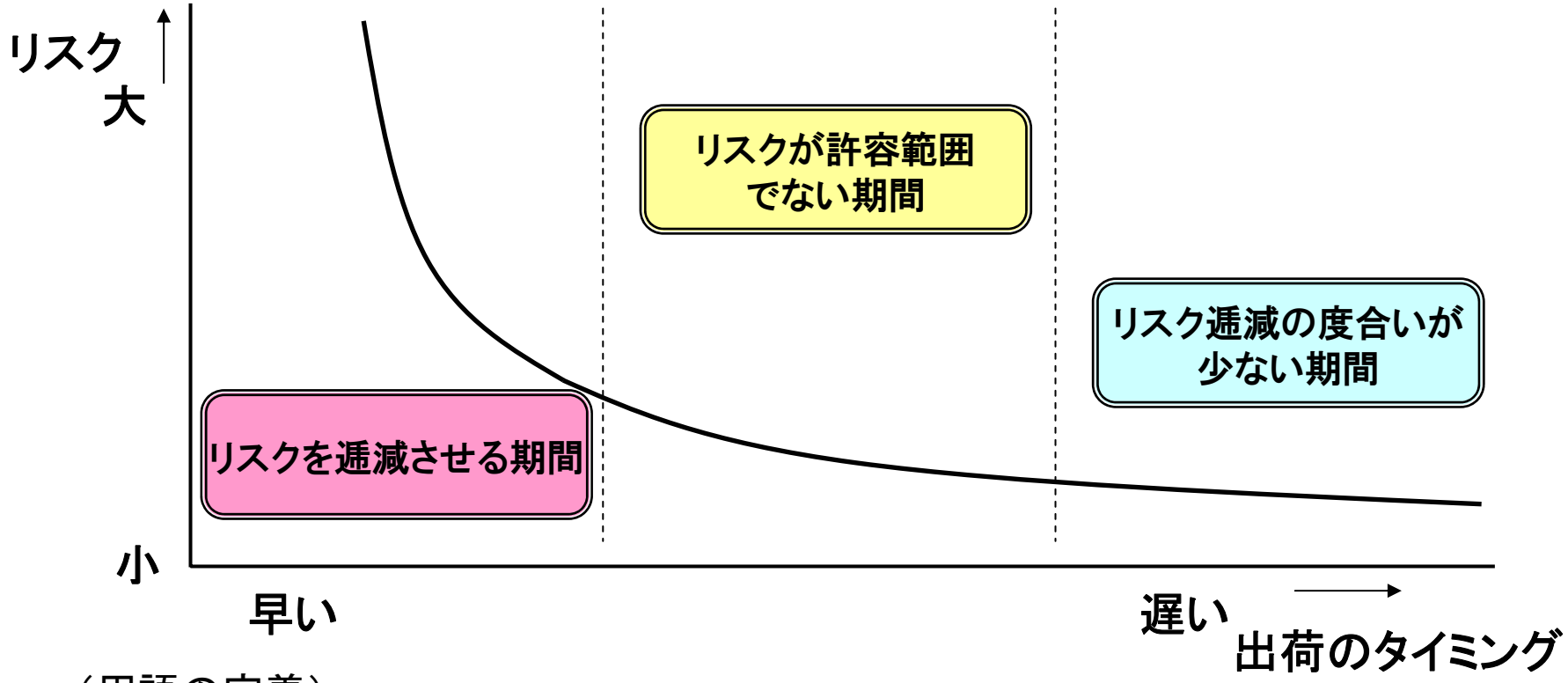
- 課題
  - 短納期
  - 全部テストできない
- 施策
  - リスクベースドテスト
    - 製品にバグが出た場合の「顧客（ユーザ）に対する迷惑度」を品質リスクと捉え、与えられたテスト期間の早い段階で品質リスクを逓減するテスト。
      - 「顧客（ユーザ）の迷惑度」とは、あるテストケースでバグが出た場合の頻度と重要度で考える。
      - この場合のリスクとは、あるテストケースを実施しなかったときにバグが出るかも知れないし、出ないかも知れないということ。

1.Introduction～リスクベースドテストってなんだろう？～

リスクベースドテストの目的(1)

Jasst'06 in Tokyo
Rick Craig発表資料より

- 出荷判断に使う
  - コストを掛けてテストしても、そのコストに見合う品質リスクの逓減が、もはや、できなくなったら出荷する。



（用語の定義）

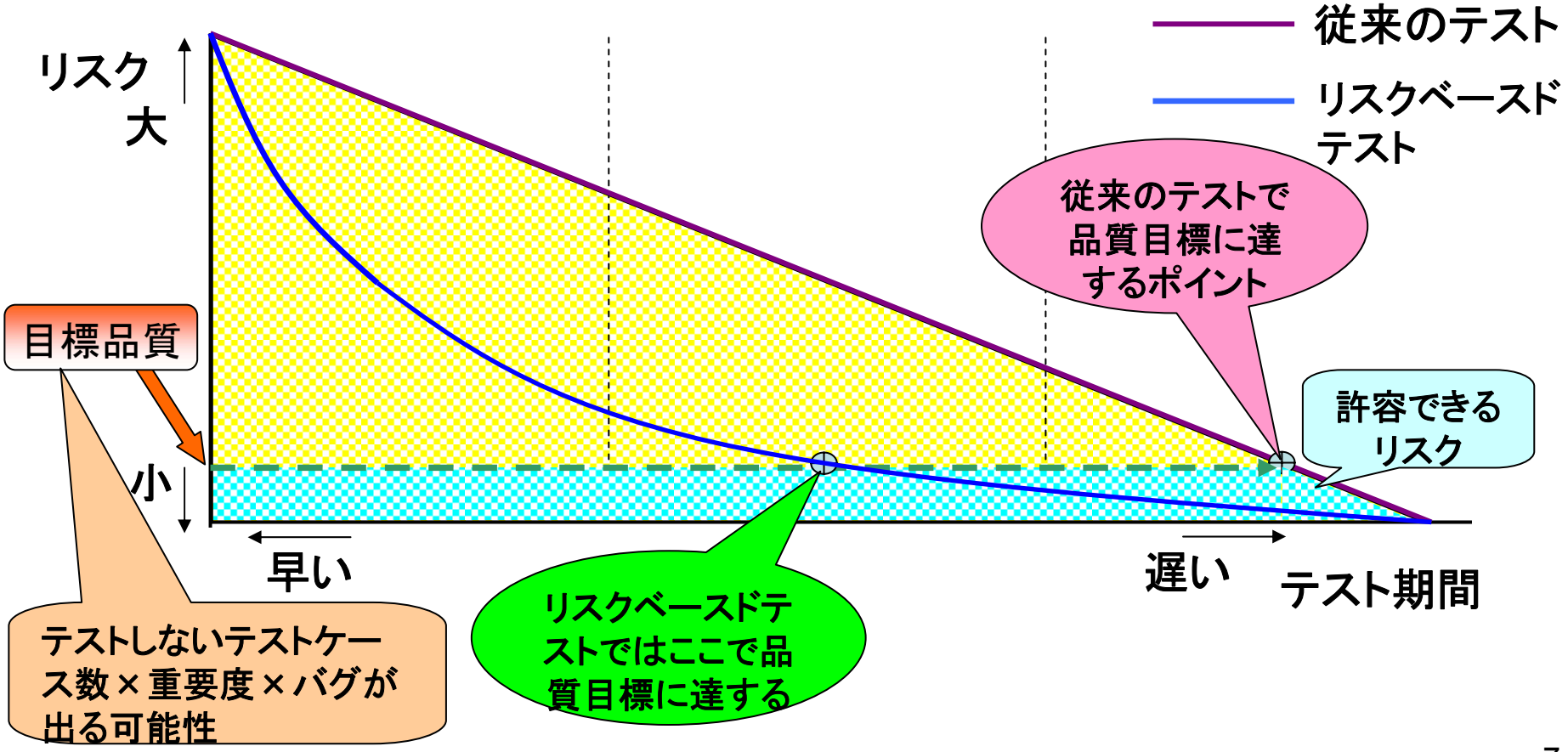
リスク逓減グラフ：上のグラフを指す。

1.Introduction～リスクベースドテストってなんだろう？～

- リスクベースドテストの目的(2)

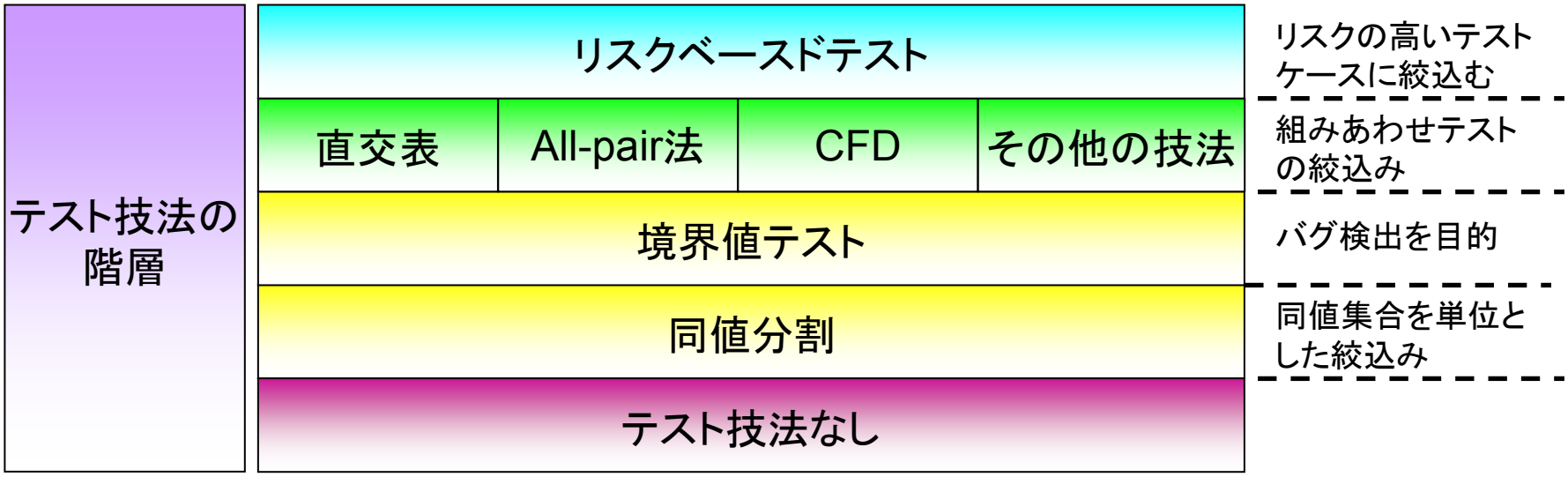
我々の使い方

  - 従来のテストより短期間で同等の品質を確保する。
    - 許容できる品質リスクの位置を品質目標とする。
    - 品質リスクの高いテスト項目を優先的にスケジューリングしテストすることで短期間で品質目標に到達する。



1.Introduction～リスクベースドテストってなんだろう？～

- 他のテスト技法との位置関係
  - テスト項目を絞り込む技法として、下記の技法がある。
  - リスクベースドテストは、これらのテスト技法と並列の位置関係ではなく、これらのテスト技法の上にさらに適用できるテスト技法である。



2.Risk based Testingの適用手順

2.Risk based Testingの適用手順

- リスクベースドテストの手順
  - Rick　Craigのリスクベースドテストをベースに幾つかの工夫を行っている。

No.	手順		Rickとの比較	工夫点
1	リスクマトリックスを作成	機能毎の重要度を決める	☆	1,2
		重要度の高い機能から降順に並べ換える	○	
2	教育	本手法の導入教育を行う	★	3
3	ステークホルダ間でリスクマトリックスをレビュー	リスクマトリックスを評価する	☆	4
		テストの範囲を決める	☆	5
4	設計との連携（今後追加）	よく分からないものから早く作る	★	
5	リスクベースドテストの見直し	リスクマトリックスの見直し	★	6
		テスト計画の見直し	★	6

○：Rick　Craigと同じ、☆：Rick　Craigも提唱しているが違いあり、★：追加タスク

※リスクマトリックスとは？
　テスト対象の機能に対し「バグの発生する可能性」と「バグ発生時の影響度」から機能毎の重要度を割り当てるマトリックス

2.Risk based Testingの適用手順

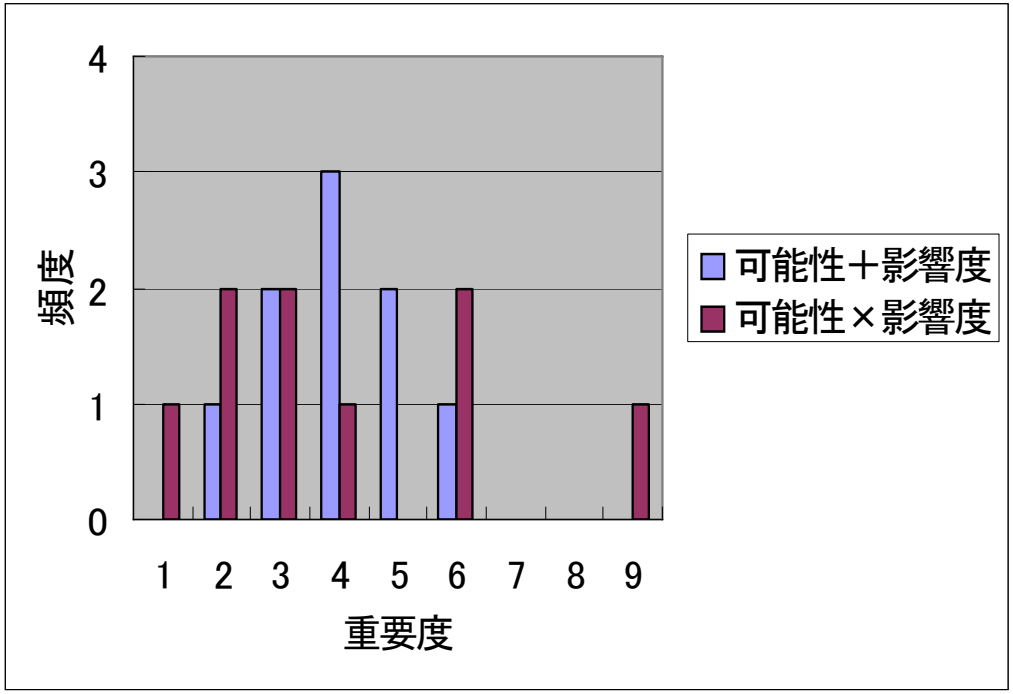
- 工夫点1:バグの発生する可能性因子の判断
  - Rick　Craig
    - (バグの発生する)可能性の評価値は単独の値として決める。
  - 適用事例
    - (バグが発生する)可能性を決定する際、バグが混入する因子を要求分析からシステムテストまでの工程を分析して詳細に定義し、その値をもとにレビューで可能性の評価値を決める。

	バグの発生する可能性										
	可能性を決定する因子										
Test Item	開発規模	仕様不明度	要件の変更度合い	設計者のスキル	ソースコードの複雑度	バグ密度(単体)	バグ密度(結合/総合)	機能の利用時期	可能性の評価値	インパクトの評価値	重要度
機能001	H	＊	L	L	＊	L	L	H	H	H	9
機能002	H	＊	L	L	＊	L	L	H	H	M	6
機能003	H	＊	L	L	＊	L	L	H	H	H	9
機能004	H	＊	H	L	＊	L	L	H	H	L	3
機能005	M	＊	M	L	＊	L	M	H	M	M	4
機能006	H	＊	H	L	＊	L	L	H	H	H	9
機能007	L	＊	L	L	＊	L	L	H	L	L	1
機能008	L	＊	L	L	＊	L	L	H	L	H	3
機能009	M	＊	L	M	＊	M	L	H	M	H	6
機能010	M	＊	L	L	＊	L	M	H	M	M	4
機能011	L	＊	L	L	＊	L	L	L	L	L	1

＊:今回未適用

2.Risk based Testingの適用手順

- 工夫点2：機能の重要度を決める（評価値の分散）
  - リスク度の一般的な算出式を重要度の算出に使う
    - （可能性）＋（影響度）・・・・・Rick Craigのリスクベースドテスト
    - （可能性）×（影響度）
  - 重要度をより細分化し、さらに重要性をより明確にできる(可能性）×（影響度）の式を採用した。



可能性と影響度の値は、ともに以下の値を付与している。

H(High)　＝ 3
M(Middle)＝ 2
L(Low)　＝ 1

2.Risk based Testingの適用手順

- 工夫点４:リスクマトリックスを評価する(1)
  - Rick　Craig
    - リスクマトリックスの値をレビュー、修正する。
  - 適用事例
    - リスクマトリックスの作成はテストリーダが行う。
      - まず、誰かが作成したリスクマトリックスをベースにしなければレビューは進まない。
    - このリスクマトリックスの各評価値が妥当か確認する。
      - 「バグの発生する可能性」を決める各因子
      - 「バグの発生する可能性」
      - 「バグ発生時の影響度」
      - 「重要度」

2.Risk based Testingの適用手順

工夫点４:リスクマトリックスを評価する(2)

- レビューにあたっての注意点
  - 「バグの発生する可能性」を決めるにあたり、これを決める各因子間の優先度を予め示しておく。但し、「顧客における利用時期」のように一つの因子で評価値を決めてしまうものがあることに注意する。

2.Risk based Testingの適用手順

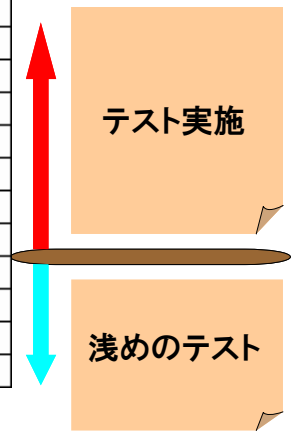
工夫点４:リスクマトリックスを評価する(3)

- 参加メンバ
  - リスクマトリックスを作成することで機能単位の重要度が自動的に決まる訳ではない。主要メンバによるレビューでリスクマトリックスを完成させる。
  - 参加メンバ選択時に考慮すること
    - 顧客（ユーザ）視点で評価できるメンバを選ぶ
    - リスクマトリックスにある要素をレビューできるメンバを選ぶ
    - プロジェクトのテストアーキテクチャについて理解を得ておく必要がある人を選ぶ
      - ・PM（プロジェクトマネージャ）　・テストマネージャ
      - ・顧客（ユーザ）対応SE　・上級管理層
      - ・開発グループリーダ（主要機能ブロック毎）
      - ・テストグループリーダ

2.Risk based Testingの適用手順

- 工夫点5:テストの範囲を決める(1)
  - Rick Craig
    - テスト範囲を絞り込むために、リスクマトリックスに足きり線を設定する。
    - 足きり線の下にある機能は、テストしないか軽くテストを行う。
  - 適用事例
    - テスト範囲を絞り込むときにリスクマトリックス以外で考慮すること
      - テストに対する予算
      - テスト期間
      - 生産性(テスト効率)

	バグの発生する可能性										
	可能性を決定する因子								可能性の評価値	インパクトの評価値	重要度
Test Item	開発規模	仕様不明度	要件の変更度合い	設計者のスキル	ソースコードの複雑度	バグ密度(単体)	バグ密度(結合/総合)	機能の利用時期			
機能001	H	*	L	L	*	L	L	H	H	H	9
機能003	H	*	L	L	*	L	L	H	H	H	9
機能006	H	*	H	L	*	L	L	H	H	H	9
機能002	H	*	L	L	*	L	L	H	H	M	6
機能009	M	*	L	M	*	M	L	H	M	H	6
機能005	M	*	M	L	*	L	M	H	M	M	4
機能010	M	*	L	L	*	L	M	H	M	M	4
機能004	H	*	H	L	*	L	L	H	H	L	3
機能008	L	*	L	L	*	L	L	H	L	H	3
機能007	L	*	L	L	*	L	L	H	L	L	1
機能011	L	*	L	L	*	L	L	L	L	L	1

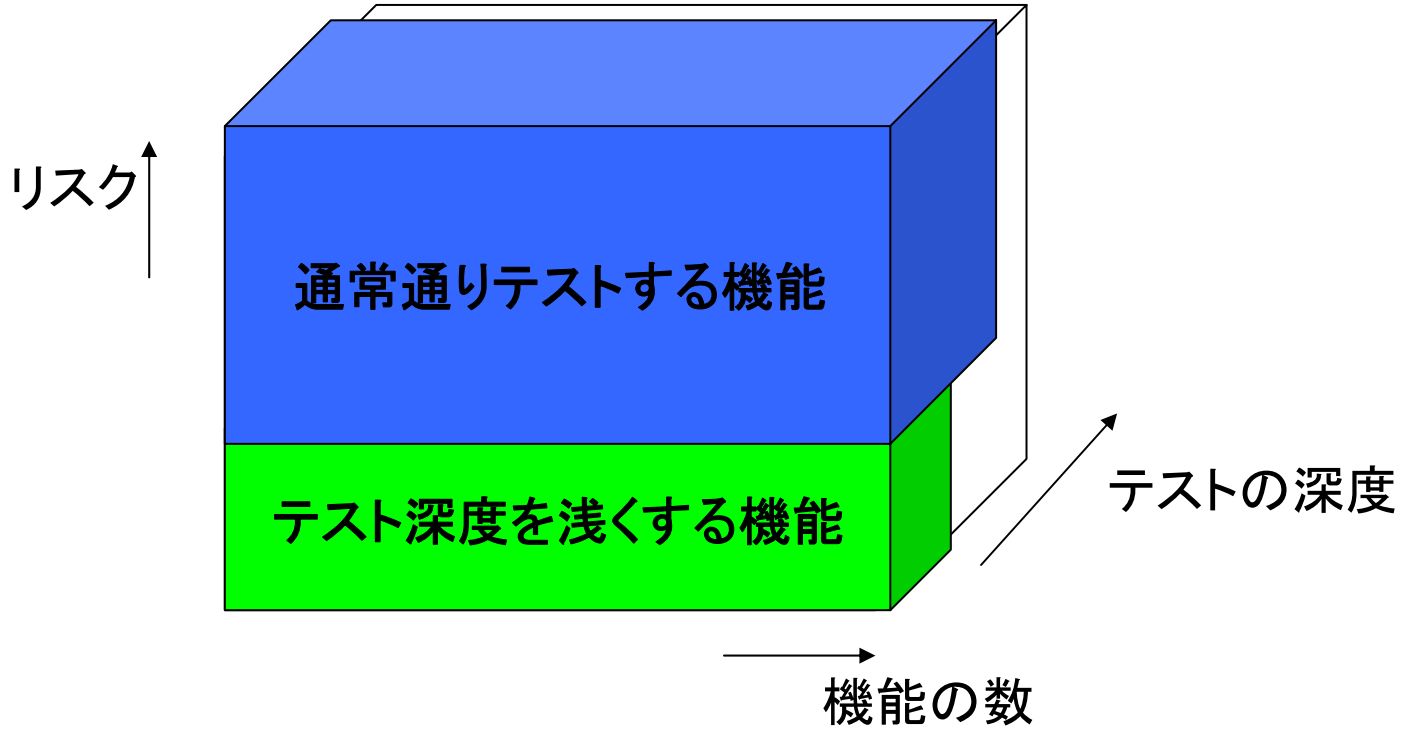


*:今回未適用

2.Risk based Testingの適用手順

工夫点５:テストの範囲を決める(2)

- テスト範囲を絞り込んだ結果、テストの深度を浅くする、またはテストを打ち切る機能についてリスクが少ない根拠を確認する。
  - 今回の開発では手を加えていない
  - テストが既に実行されていて既に合格基準をクリアしている。



2.Risk based Testingの適用手順

- 工夫点６:リスクベースドテストの見直し
  - プロジェクトが進行する中でリスクマトリックスにあるリスク決定のための各因子の状態が変化することを想定する。
    - 単体テスト、結合テスト、およびシステムテストの計画段階で見直す。
    - テスト中も状況監視を行う(ダイナミックに監視)。
  - リスク決定の因子状態が変化しテストスケジュールに影響する場合、その対応は以下のパターンで考える。
    - テストの順序を変える
    - テストの範囲を変える

2.Risk based Testingの適用手順

「バグの発生する可能性」の判断に使用する因子の適用時期

因子	計画	見直し	テスト中	備考
開発規模	○	○	○	見直し時は最新の実績規模を使う
設計者のスキル	○	○	○	設計者の変更があるとき見直し
仕様不明度	○	○	○	
顧客における利用時期	○	○	○	
ソースコードの複雑度	×	○	○	
要件の変更度合い	×	○	○	
バグ密度(単体テスト)	×	○	○	
バグ密度(結合テスト)	×	○	○	
バグ密度(システムテスト)	×	×	○	

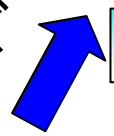
3.適用結果

3.適用結果

- 評価期間
  - 2.0ヶ月→1.0ヶ月
- バグ検出率（件/テスト項目）
  - リスクベースドテスト不適用フェーズ
    - 95/2288 = 0.04
  - リスクベースドテスト適用フェーズ
    - 36/638 = 0.06
- バグ効率（Σ検出バグの重要度/テスト項目）
  - リスクベースドテスト不適用フェーズ
    - 547/2288 = 0.24
  - リスクベースドテスト適用フェーズ
    - 263/638 = 0.41

50%短縮

効率UP→工数削減



1.5倍向上

効率UP→品質向上

1.71倍向上

効率UP→品質向上

4.適用全般に対する考察

4.適用全般に対する考察

- リスクベースドテスト導入プロジェクトのスケジュール管理
  - テスト項目消化実績
  - 信頼度成長曲線
  - リスクの高い機能を優先したスケジュールに対する予実
    - リスクマトリックスにある機能の重要度の高いものから優先してスケジューリングする。この際、人員リソースのスキルとマシン環境による制約を考慮する。
    - スケジュール変更が必要な場合、リスクの高い機能のテストを優先したスケジューリングを極力崩さないようにする。
  - リスク逓減グラフ
    - テストの早い段階でリスクが逓減できているかを見る。
    - 計画通りリスク逓減ができているかを見る。
    - テストの終盤でリスク逓減の度合いが少ない段階かを見る。
      - 出荷判断
      - テスト終了判断

今後の課題

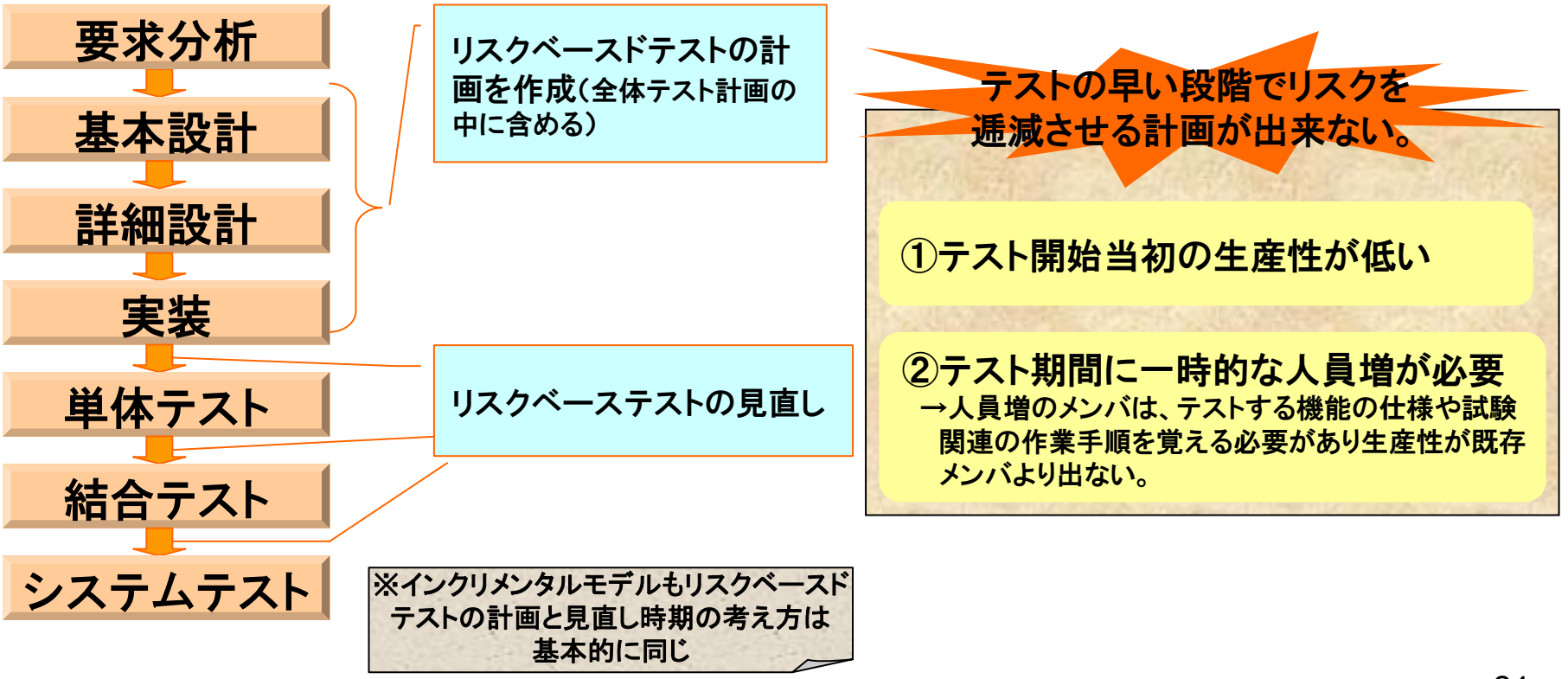
テストスケジュール管理にスプレッドシートを使うと、管理するために確認する項目が多くなり、根気と時間が必要。

解決策

ガントチャートやタスク間の依存関係を表示できる視認性の良いスケジュール管理ツールを使うと有効。CPMやクリティカルチェーンと組み合わせたマネジメントが可能となる。

4.適用全般に対する考察

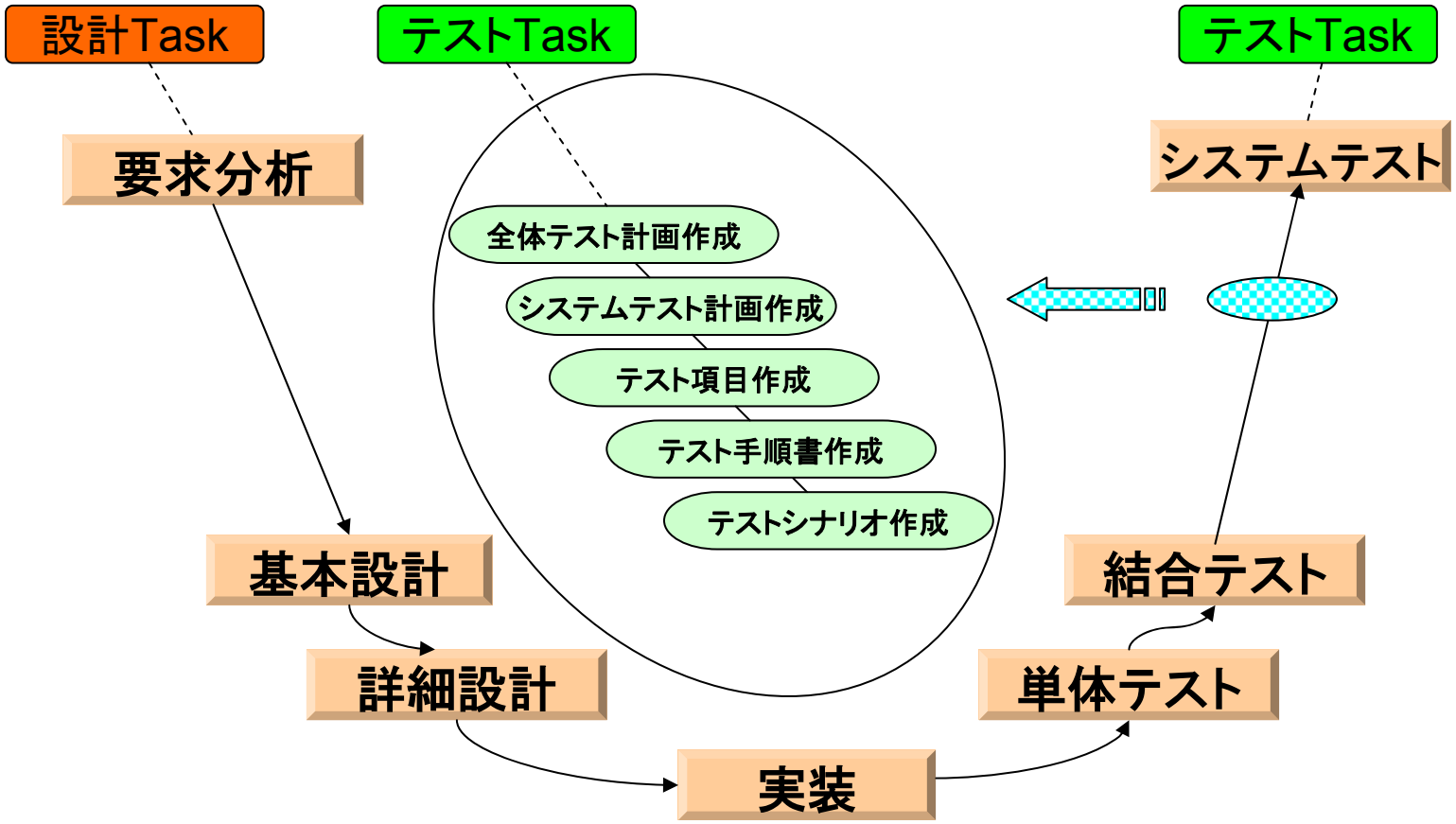
- リスクベースドテストの効果をさらに高める
  - 開発モデル：ウォータフォールモデル
    - テストウェア(*)をテスト期間に作成している。
      - (*)テスト計画、テスト項目、テストシナリオ



4.適用全般に対する考察

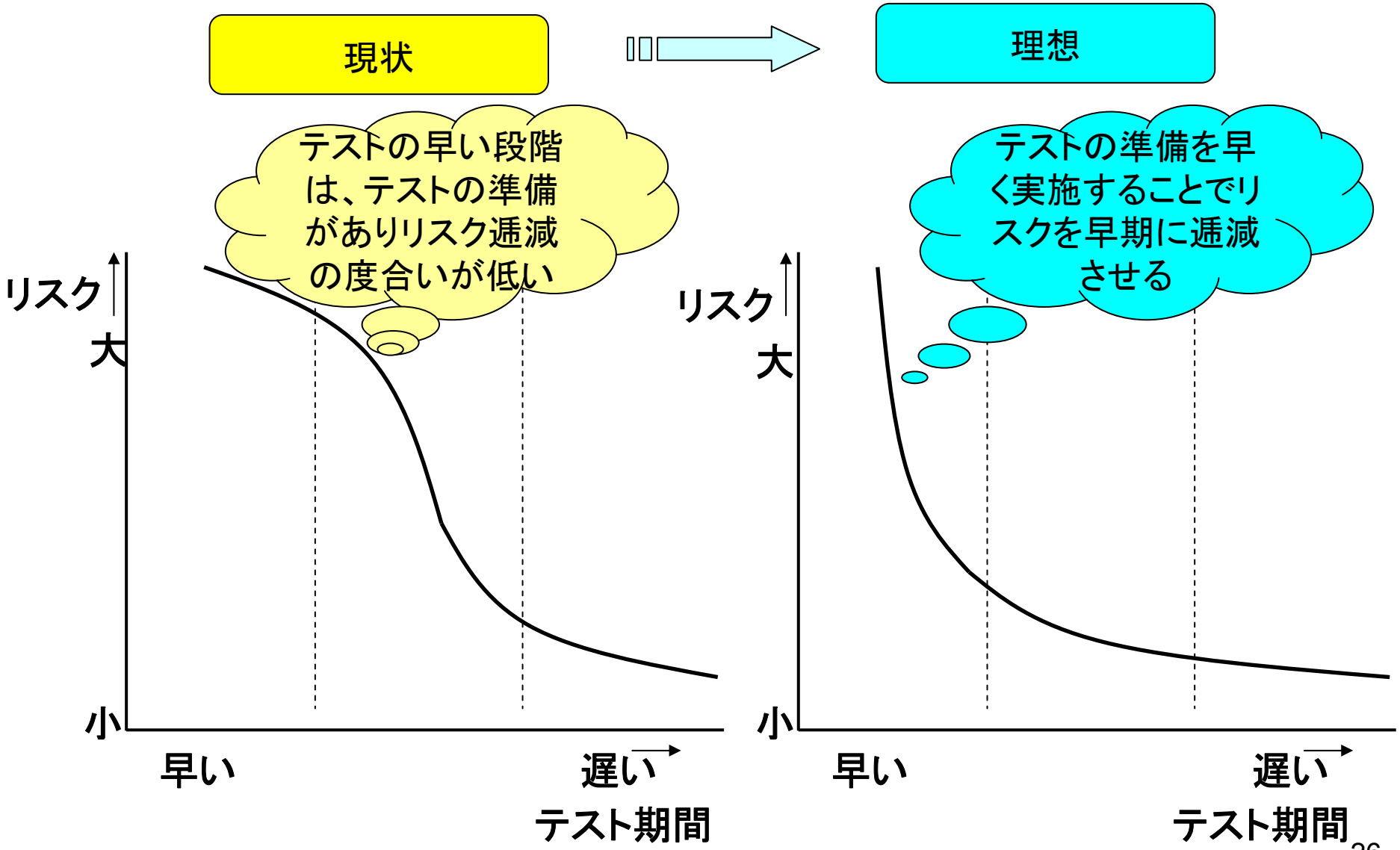
解決策

- 開発モデルにWモデルを適用する。
  - 上流のフェーズでテストウェアを作成する。
- テスト担当者の役割を定義する。
  - テストマネージャ
  - テスト設計者
  - テスター



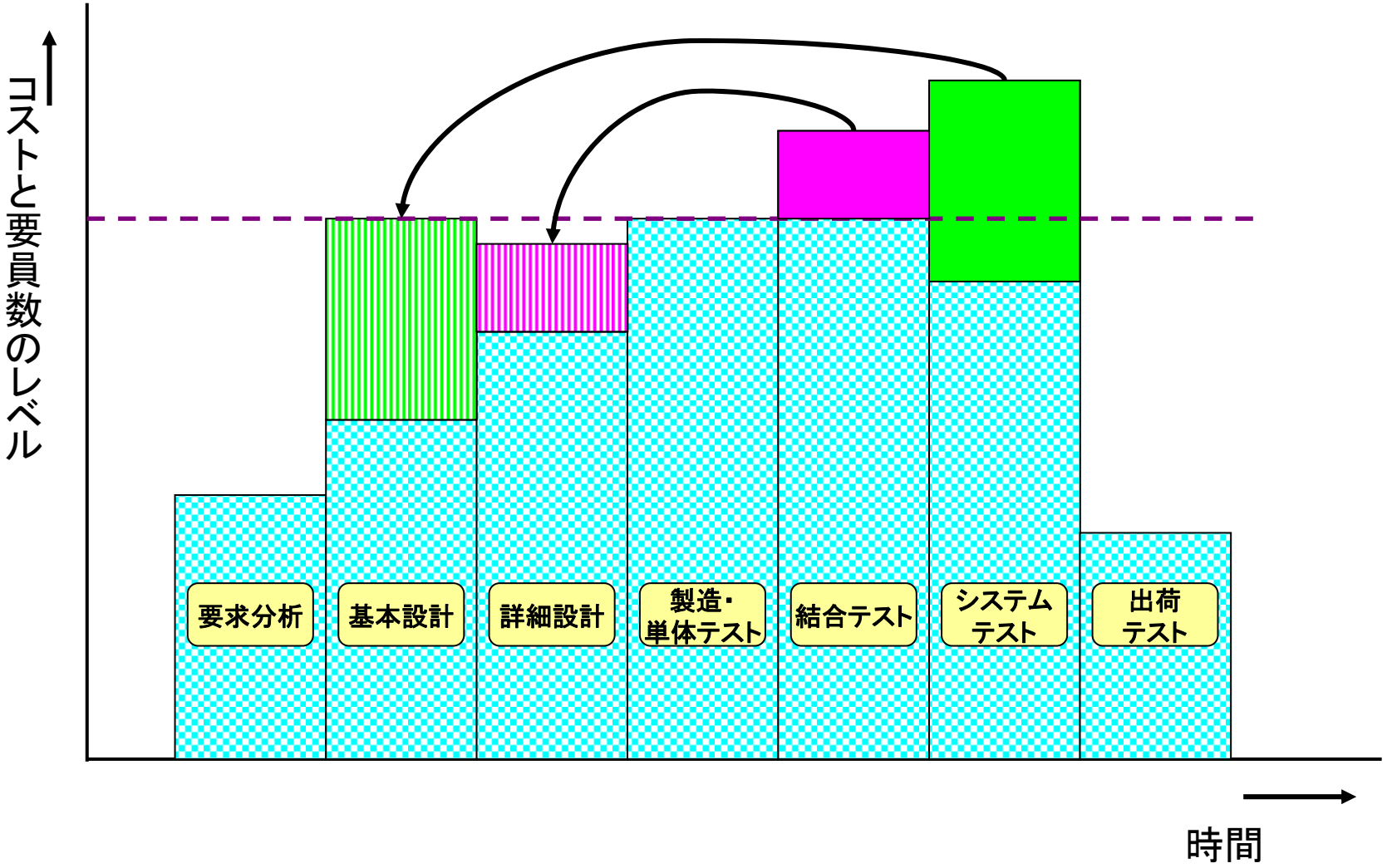
4.適用全般に対する考察

- テスト開始当初の生産性が改善される



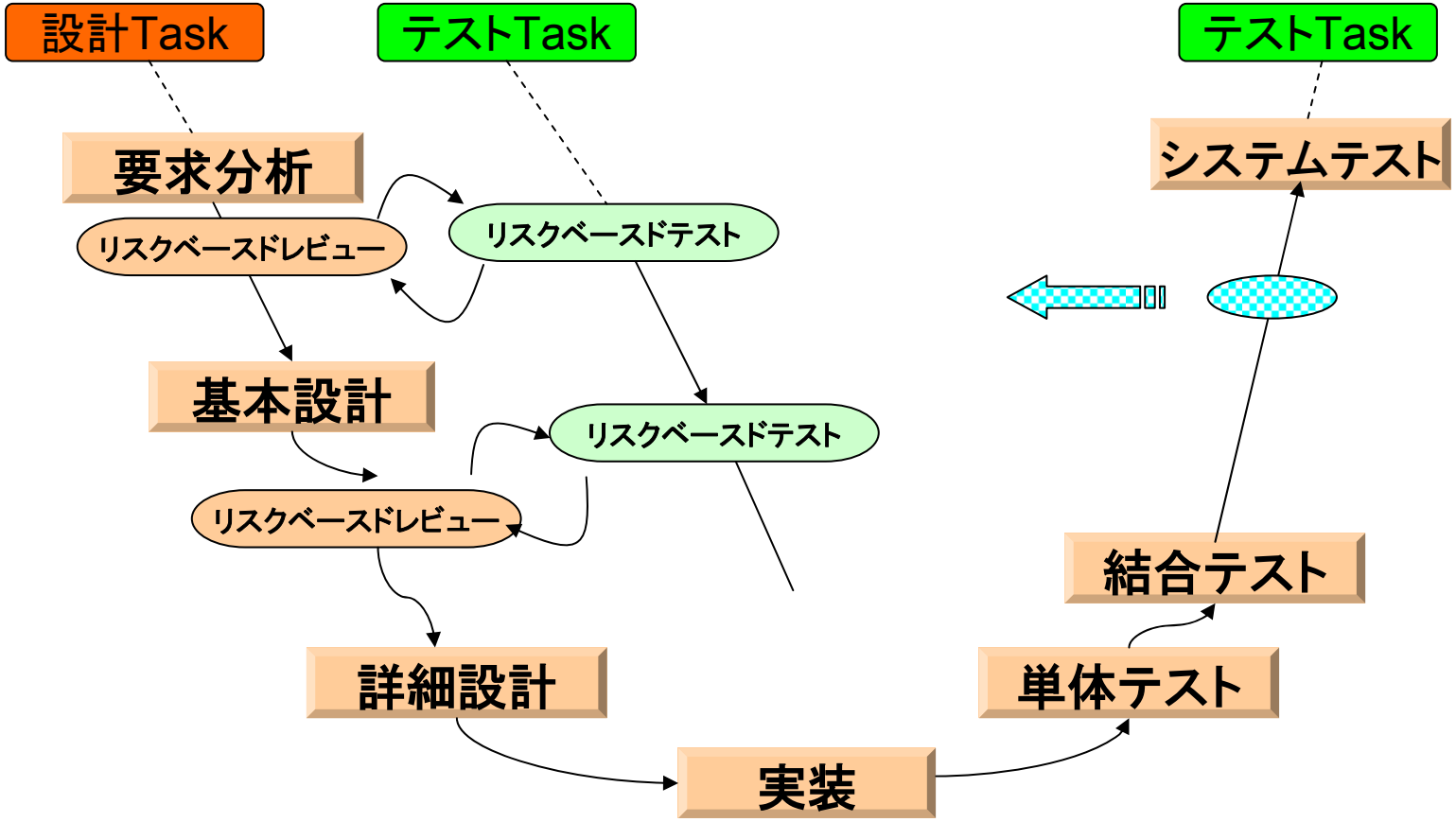
4.適用全般に対する考察

- テスト期間の一時的な人員増を抑える:テストエ数の山崩し



4.適用全般に対する考察

- 品質リスクを上流から考える
  - リスクベースドレビューの実施
    - リスクの高い機能は、レビューを厚くする。
    - リスクが低いと判断した機能が、本当にそうなのかレビューで確かめる。



ー終わりー

ご清聴ありがとうございました。