《　避けて通れないテストと検証　》
"苦労"対効果をあげるには？

Parasoft Japan

野田勝彦

www.parasoft.co.jp

info@parasoft.co.jp

デブサミ2008 Developers Summit

PARASOFT®

We make software work.™

2008年 4月 23日

Developers [Test] Summit 2008

主催：株式会社翔泳社

【Session-4】

避けて通れないテストと検証。“苦労”対効果をあげるには？

みなさんは、ソフトウエアの品質を保障するために何をしていますか？さまざまな手段や方法がありますが、どんな手法を採用しても、テストと結果の検証は実施しなければなりません。本講演では、演者が最も"苦労"(費用)対効果が高いと考える単体テストと開発者同士で確認しあうコードレビューについて、効果的な実施方法をご紹介しようと思います。

講演：　Parasoft Japan 株式会社　野田 勝彦

提供：　テクマトリックス株式会社

ソフトウエア単体テスト基礎講座

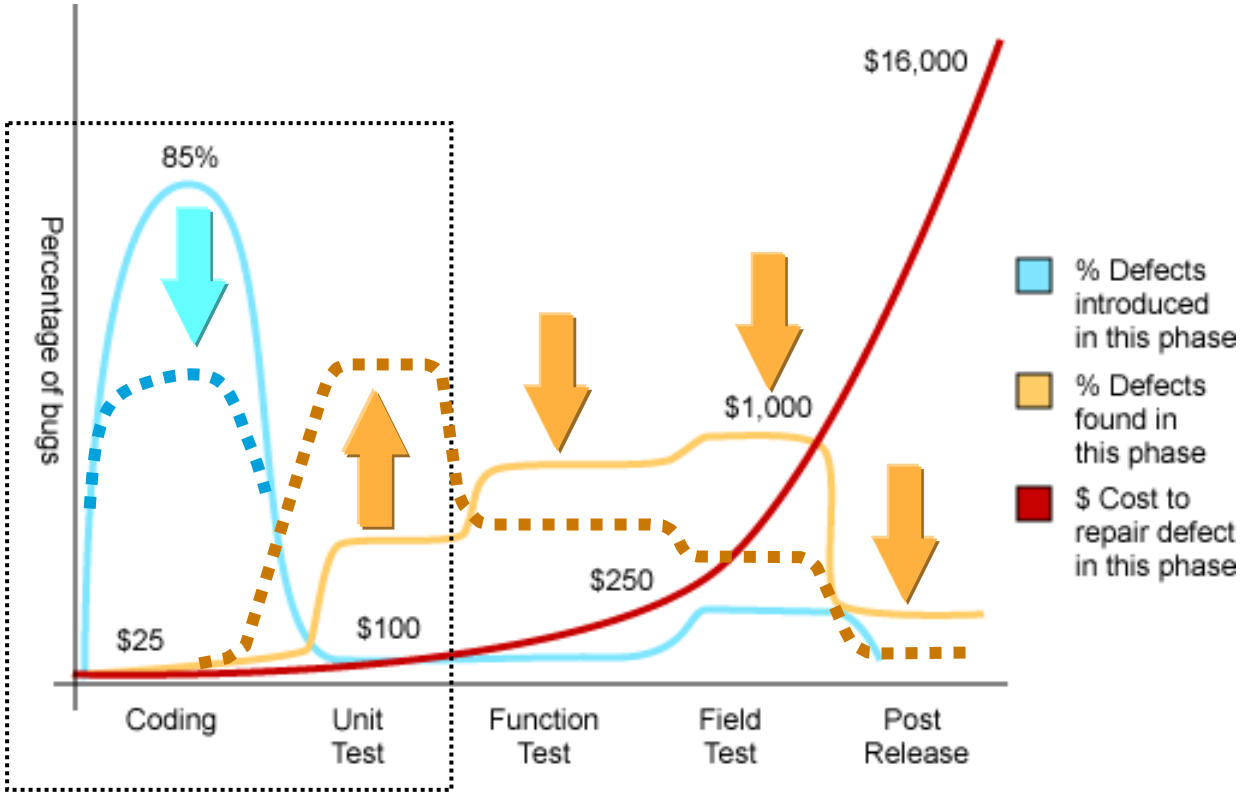
単体テストやコードレビューができていない開発に関わっていらっしゃる方へ

ソフトウエア開発に、魔法やトリックは存在しませんので、本日の講演内容は、とても基本的な内容です。

PARASOFT®
We make software work.™

Parasoft ソリューションのコンセプト

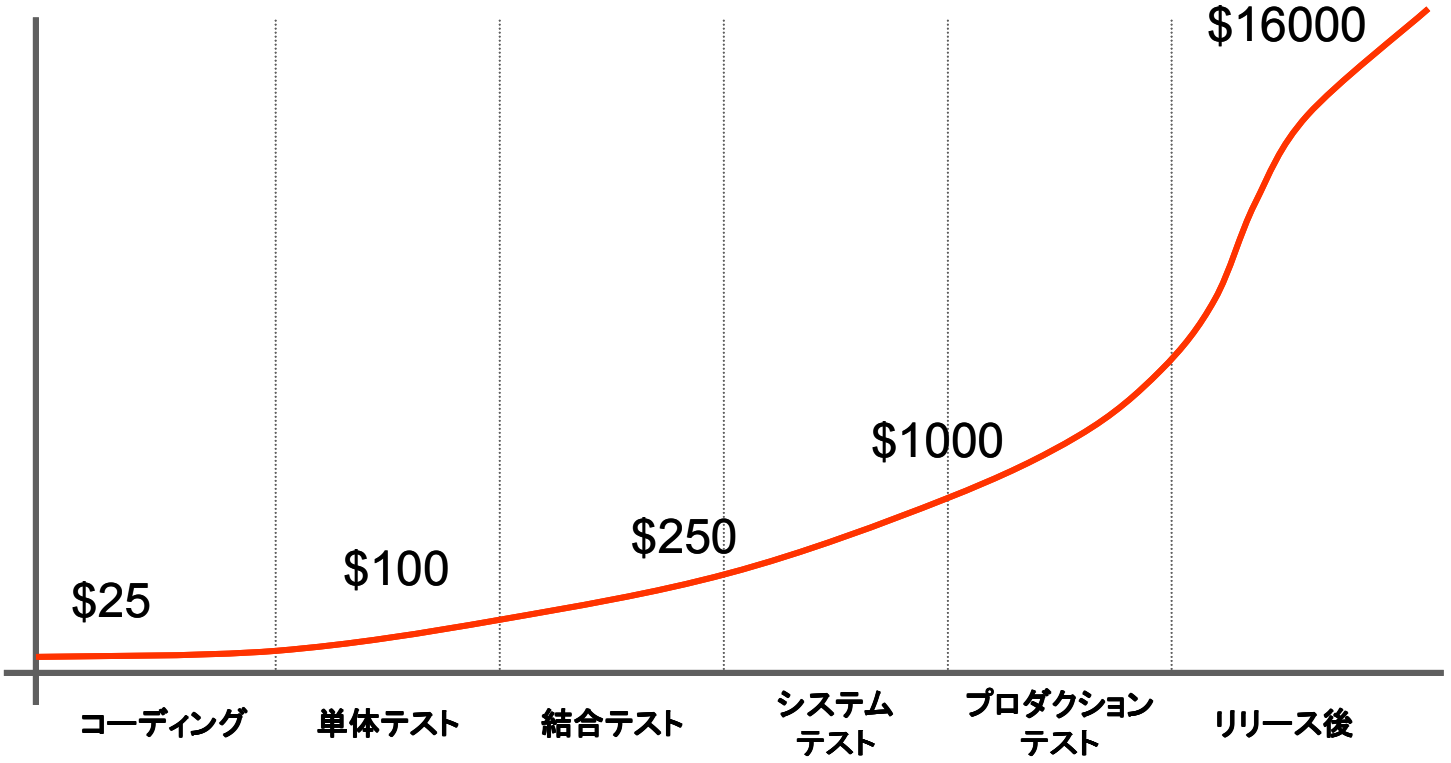
- ADP (Automated Defect Prevention)
  - ソフトウエアの欠陥を未然に防止する。



Source: Applied Software Measurement, Capers Jones, 1996

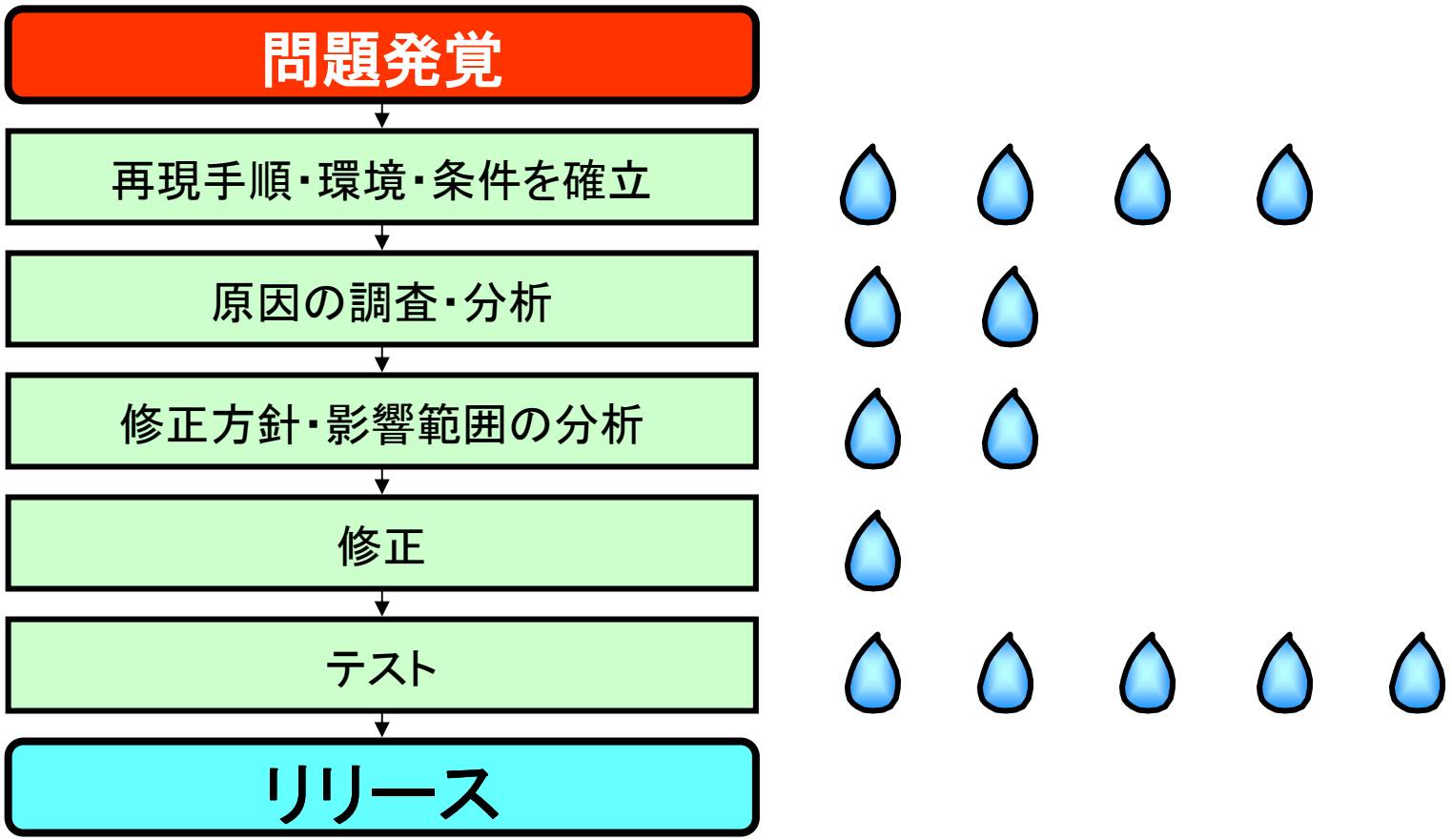
1:10:100の法則

- ソフトウエアのバグ修正コストは、発見されるタイミングによっては、10倍、100倍に成り得る。

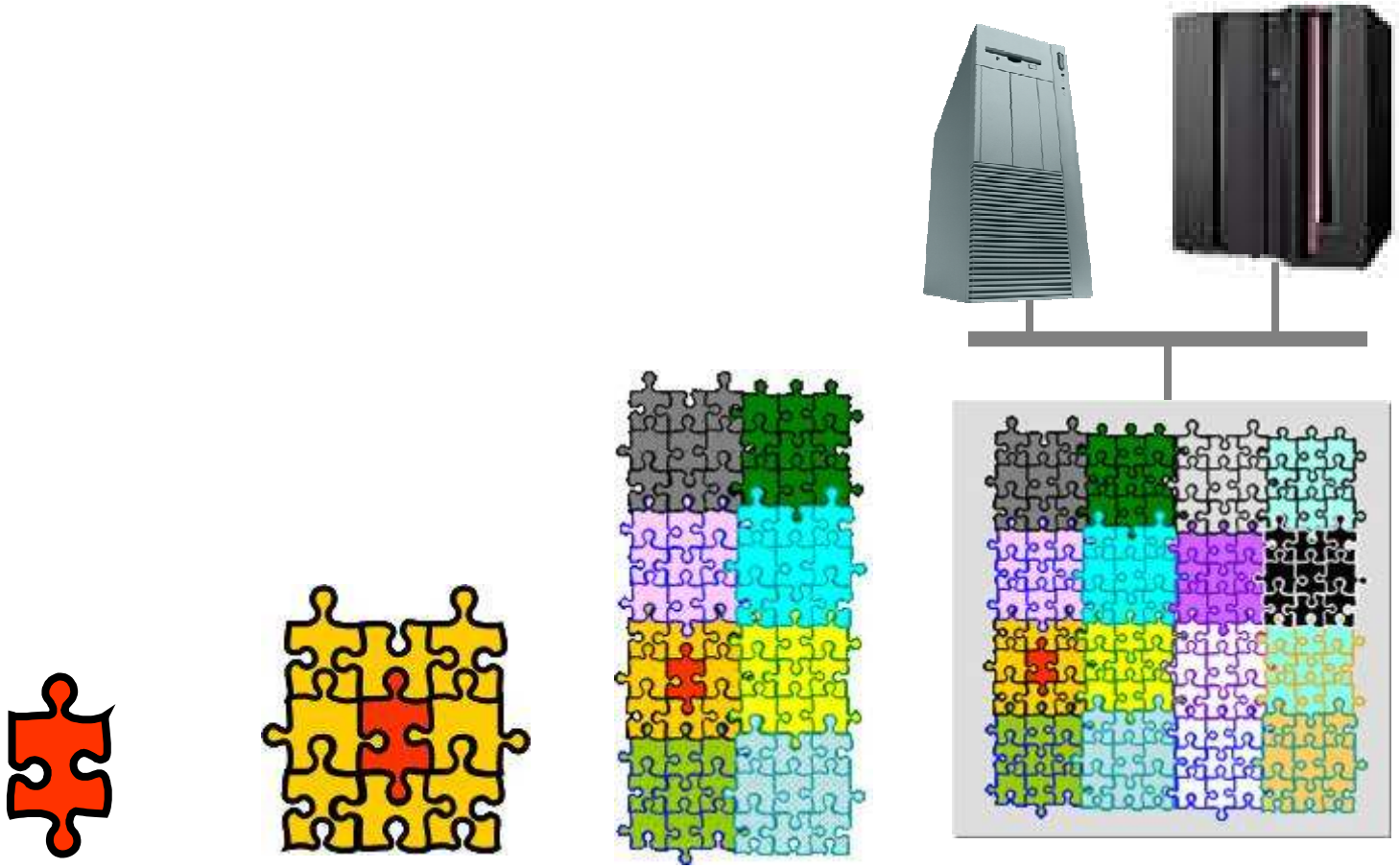


開発者の視点：バグ修正の”苦労”

- バグ発見～バグ修正～リリースまでの手順



再現手順の確立と原因調査の“苦労”



コーディング	単体テスト	結合テスト	システムテスト	プロダクションテスト

バグ修正後の再テストの“苦労”

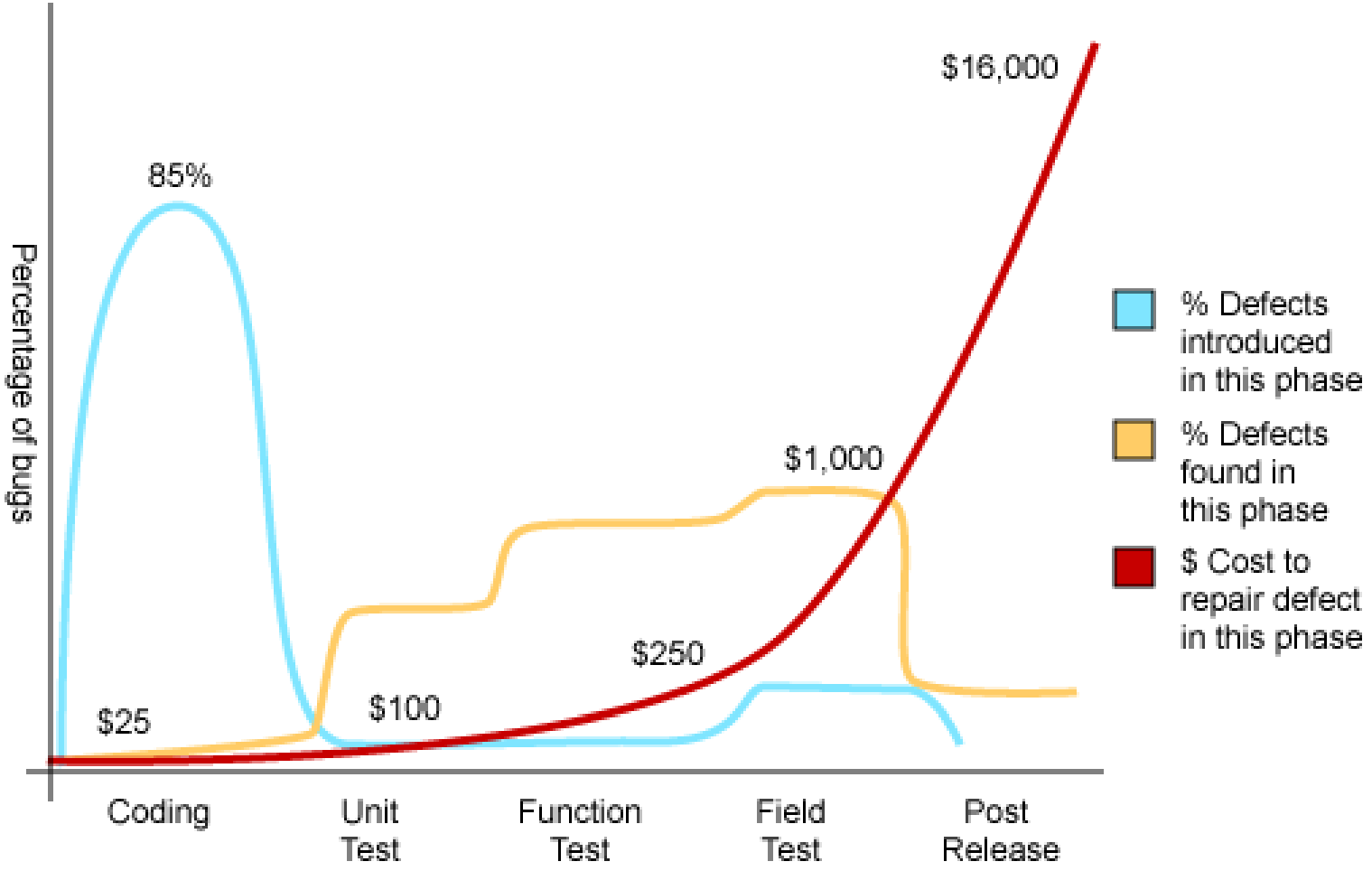
- 単体テスト中に発見

- システムテスト中に発見

- リリース後発見

-

ひと苦労:十苦労:百苦労の法則



Source: Applied Software Measurement, Capers Jones, 1996

早期にソフトウエアの品質向上を実現するための ベストプラクティス

- コーディング規約
  - 一定の規約・ルールに従いコーティングすることにより、ソフトウエアの品質や保守性が向上する。
  - 命名規則、業界標準のコーディング標準
- 単体テスト
  - ソフトウエアを実際に動作させ、入力に対する出力が、設計されたとおりであるかを検証する。
  - ブラックボックステスト、ホワイトボックステスト
- ソースコードレビュー
  - コーティング規約に従っているか、設計書どおりにコーディングされているか、第三者が目視確認する。
  - 集合レビュー、1x1レビュー、ペアプログラミング

現実の課題：受託開発企業様

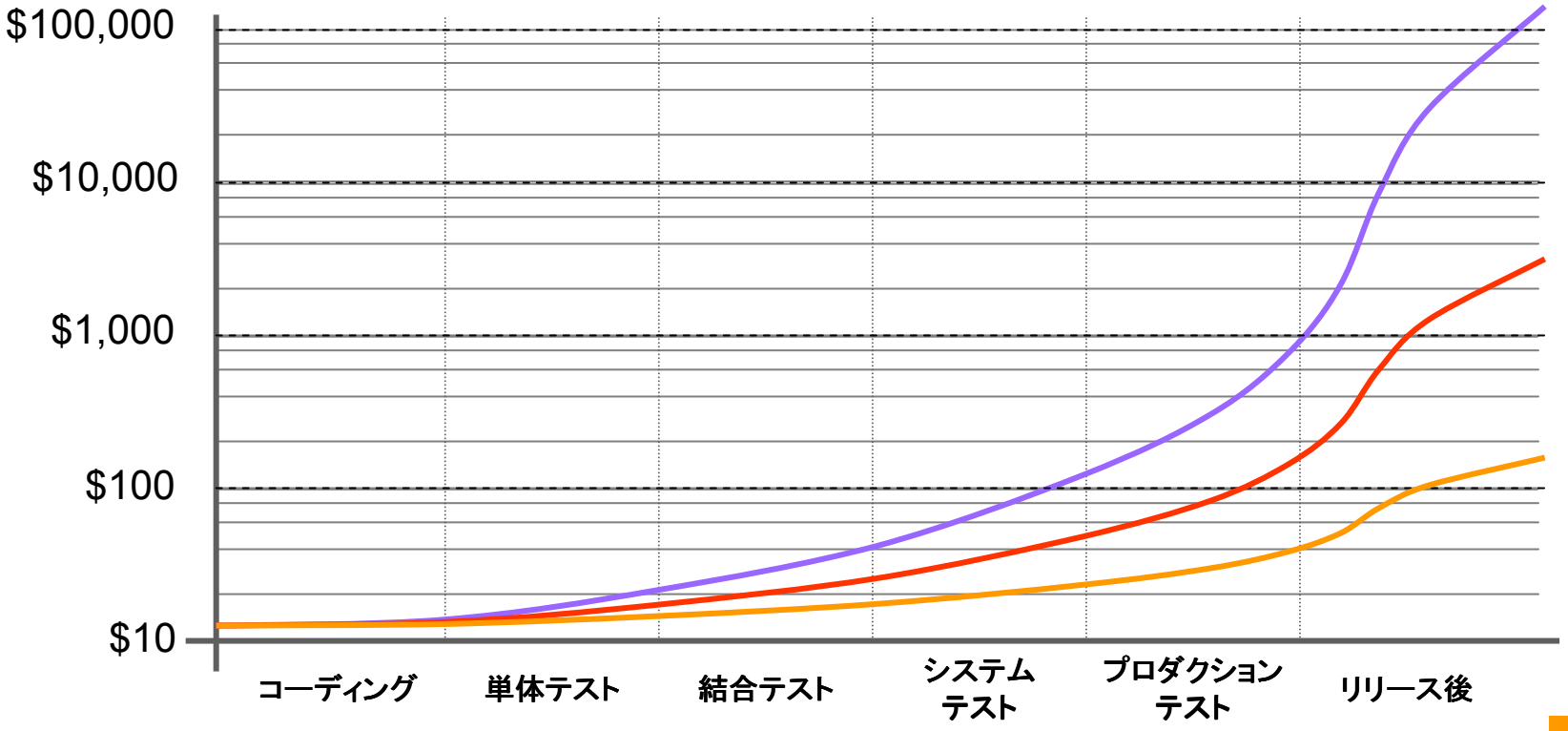
- 単体テストやレビューをするリソース（人・時間）が足りない。
  - 発注元（システムオーナー）のビジネスの競争激化により、より短期間での開発が要求される。
  - 海外などオフショア開発サービスとの価格競争の激化。
  - 単体テストやレビューを手助けしてくれる道具が無い。

現実の課題：発注元（システムオーナー）様

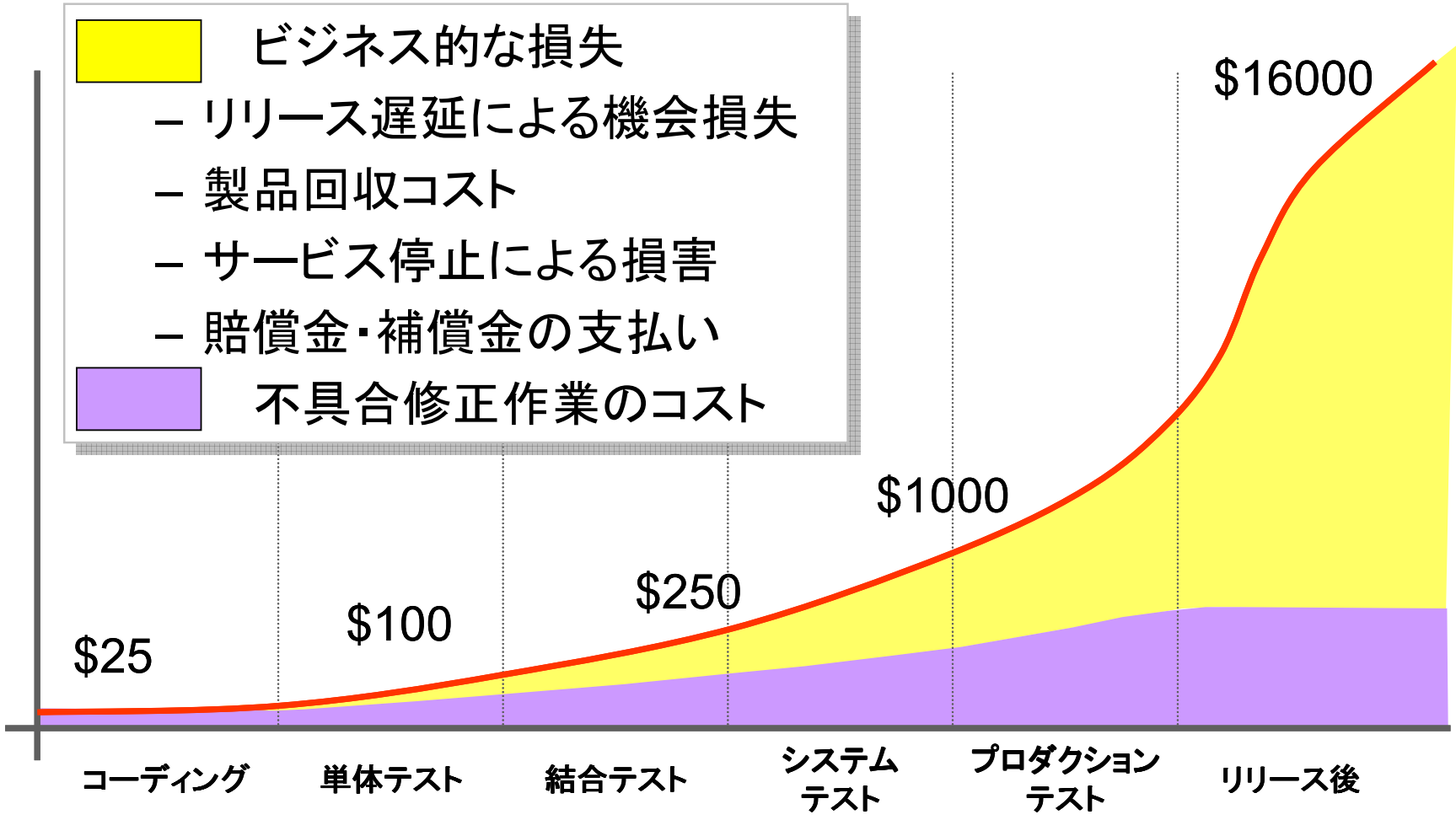
- レビューや単体テストが行われているか検証できない。
  - 成果物責任契約で、開発委託先のプロセスにまで口を出せない。
  - レビューや単体テストが行われたことの確証を要求すると開発委託費が高くなる。

1:100:10000の法則？

- 現代では、バグの対価は、ビジネスの種類、ソフトウエアのミッション、マーケットサイズ、システム規模により、1000倍、10000倍にも成り得る。

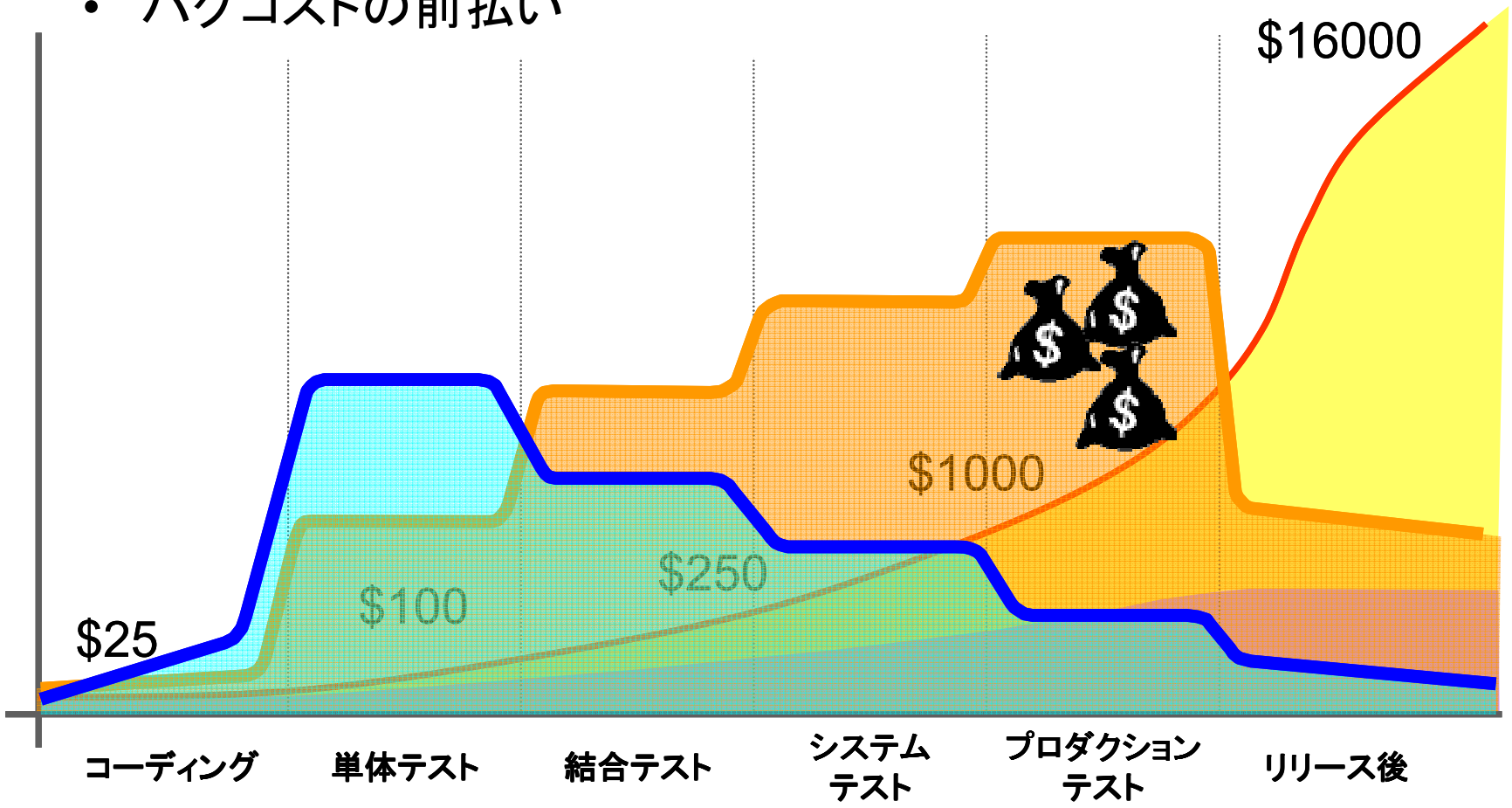


バグコストの内訳


ビジネス的な損失
– リリース遅延による機会損失
– 製品回収コスト
– サービス停止による損害
– 賠償金・補償金の支払い
不具合修正作業のコスト
$25
$100
$250
$1000
$16000
コーディング
単体テスト
結合テスト
システム テスト
プロダクション テスト
リリース後

TCOを削減するための提言

- バグコストの前払い



TCOを削減するための提言

- 早期にソフトウエアの品質向上を実現するためのベストプラクティスを実践する。
  - コーディング規約
  - 単体テスト
  - ソースコードレビュー
- そのために
  - 受託開発企業様
    - TCOの観点で「価格競争」する。
  - 発注元（システムオーナー）様
    - 開発プロセスに関して口を出す。
    - コーディング規約が遵守されていること、単体テストやレビューが実施されたことの確証を要求する。

Parasoft Tools

ツールでできることは、ツールを使い、苦労を減らしたい。

PARASOFT®
We make software work.™

各プログラミング言語用Parasoftツール

- プラットフォーム
  - Microsoft Visual Studio .NET専用
- 対象言語
  - C#, Managed C++, Visual Basic

- プラットフォーム
  - Microsoft Visual Studio .NET
  - Eclipse、Eclipseベースの各社IDE
- 対象言語
  - C言語、C++

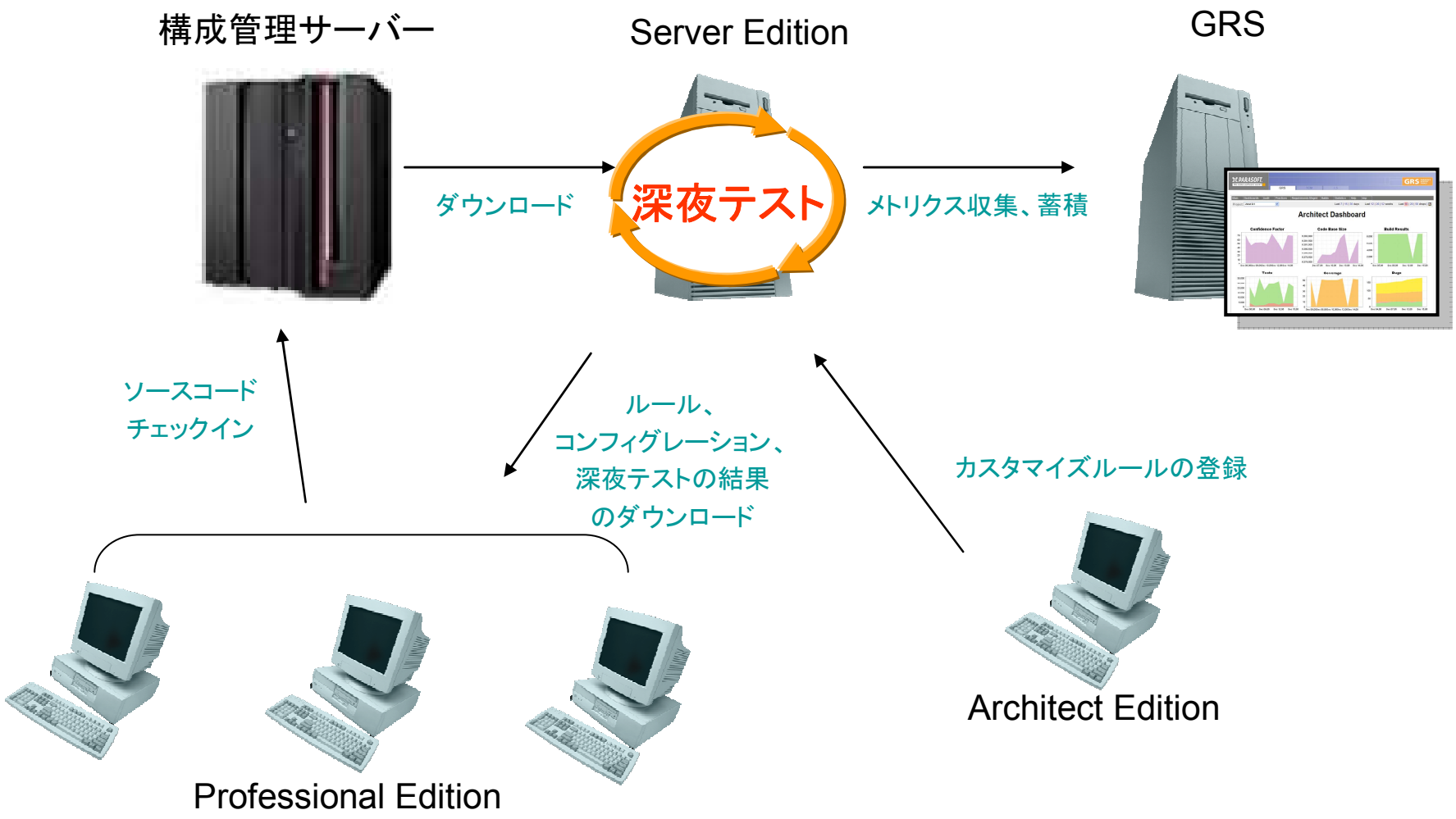
- プラットフォーム
  - Eclipse、Eclipseベースの各社IDE
- 対象言語
  - Java

主な機能（共通）

- コーディングルール・コンプライアンスチェック
- フロー解析
- クイックフィックス
- 単体テスト支援機能
- Out-Of-The-Box 単体テスト
- 単体テストカバレッジ計測
- サーバーサイドカバレッジ計測
- オブジェクトリポジトリ
- GUI操作でのカスタムテスト
- データソースの利用によるバリエーションテスト
- バリエーションテストの自動生成
- インコンテナテスト
- 実行ログからテストプログラムの生成
- コードレビュー
- レポート
- 深夜の無人テスト
  - 継続的に反復して、コードチェック
  - 継続的に反復して、リグレッションテスト
- メトリクス収集、モニター

【注意】プログラミング言語・ツールバージョンによっては、未サポートの機能があります。

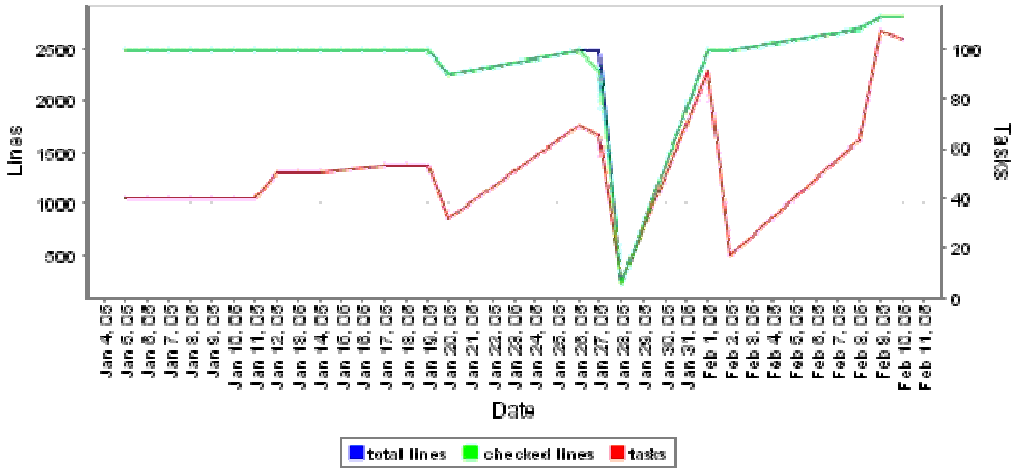
一般的な運用モデル


構成管理サーバー
Server Edition
GRS
ダウンロード
深夜テスト
メトリクス収集、蓄積
ソースコード
チェックイン
ルール、
コンフィグレーション、
深夜テストの結果
のダウンロード
カスタマイズルールの登録
Professional Edition
Architect Edition

コーディング規約のチェック機能

- 組み込みコーティング規約
  - ほとんどの業界標準を網羅

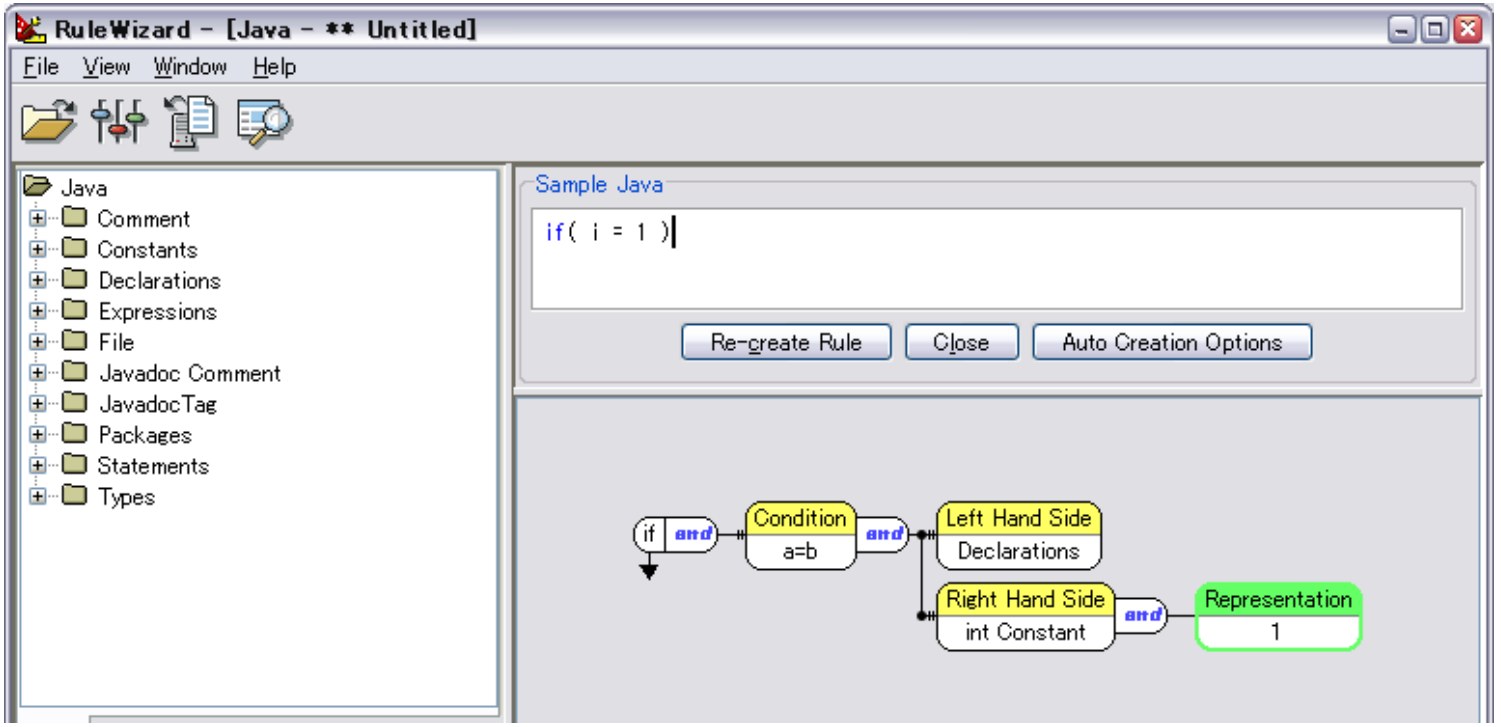
	Ver	ルール数
PARASOFT .TEST™ A Parasoft AEP Technology™	4.1	393
PARASOFT C++test™ A Parasoft AEP Technology™	7.1	851
PARASOFT Jtest® A Parasoft AEP Technology™	8.2	836



プロジェクト / パッケージ名	パッケージ	クラス	インターフェイス	メソッド	オーバーライドメソッド	Static メソッド	Weighted メソッド	DBC メソッド	JUnit メソッド	フィールド	子	Static 属性	行
Best Example	9	37	1	121	2	17	169	1	79	31	11	7	539
examples.bank	n/a	7	0	19	0	1	33	0	9	8	3	0	89
examples.evil	n/a	1	0	0	0	3	9	0	3	0	0	0	15
examples.initialize	n/a	2	0	10	0	0	17	0	0	3	0	0	42
examples.inputs	n/a	2	0	2	0	2	6	0	0	0	0	2	24
examples.leaks	n/a	2	0	4	0	2	5	1	4	3	0	3	8
examples.queue	n/a	3	0	10	1	0	13	0	5	4	0	0	28
examples.singletons	n/a	2	0	6	1	2	13	0	8	3	0	2	24
examples.statemachine	n/a	16	1	66	0	1	71	0	38	9	8	0	313
examples.unit	n/a	3	0	3	0	6	11	0	7	1	0	0	22
合計	9	37	1	121	2	17	169	1	79	31	11	7	539

独自の規約の追加

- RuleWizard （ルールウイザード）
  - 独自ルールの作成
  - 組み込みルールのカスタマイズ



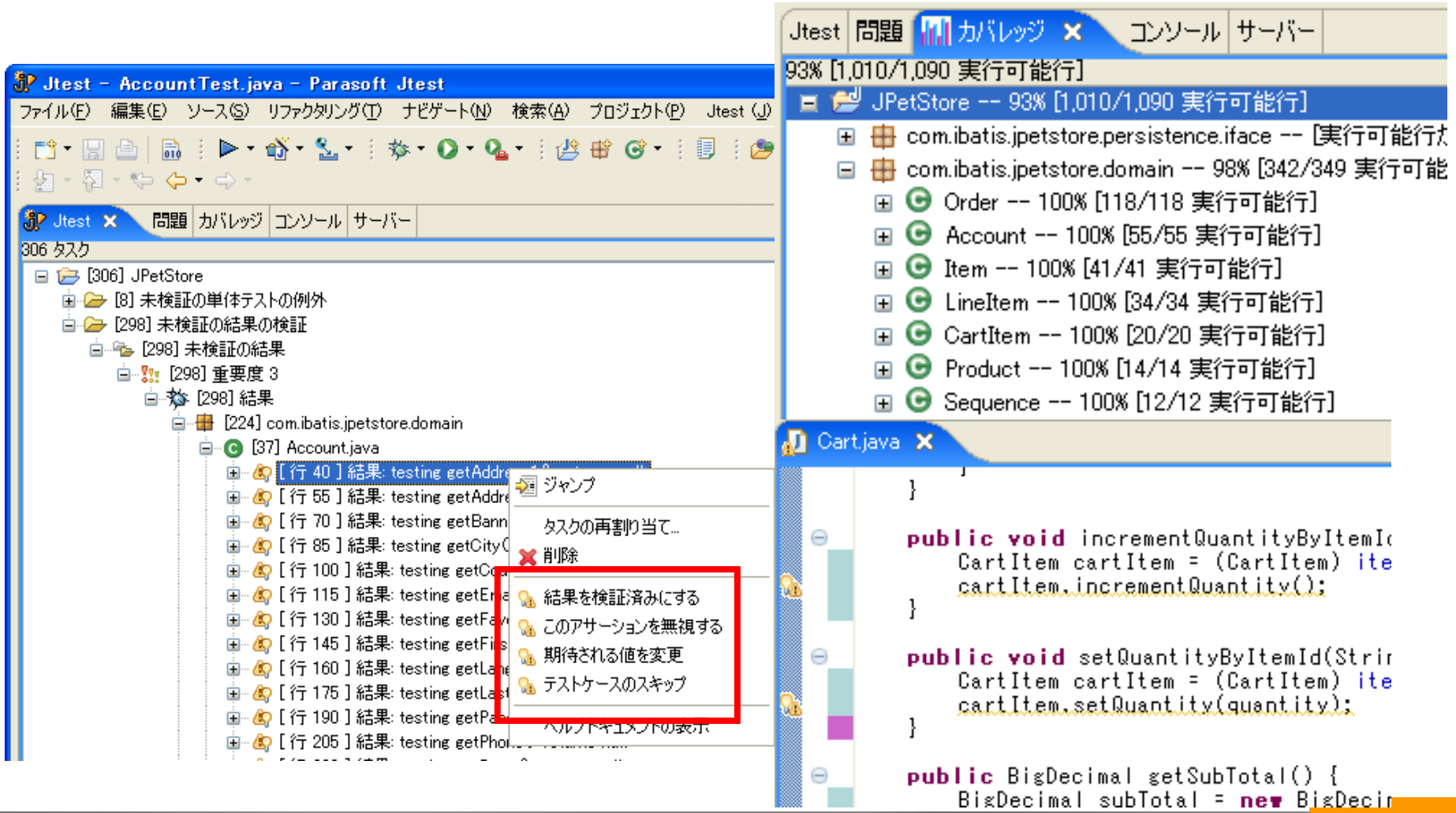
単体テスト支援機能

ワンクリック：Out-Of-The-Boxテスト

- 人手を介さず、自動で単体テストプログラムを生成し、生成されたテストプログラムを実行。

- テスト生成オプション
  - 更新のあったコードのみ再生成／無条件生成。
  - スタブのあり／なし／外部のみ
  - 例外ケースのあり／なし
  - Nullポインタあり／なし
  - Private／Protected／Public
  - 行カバレッジ／ブランチカバレッジ／全て

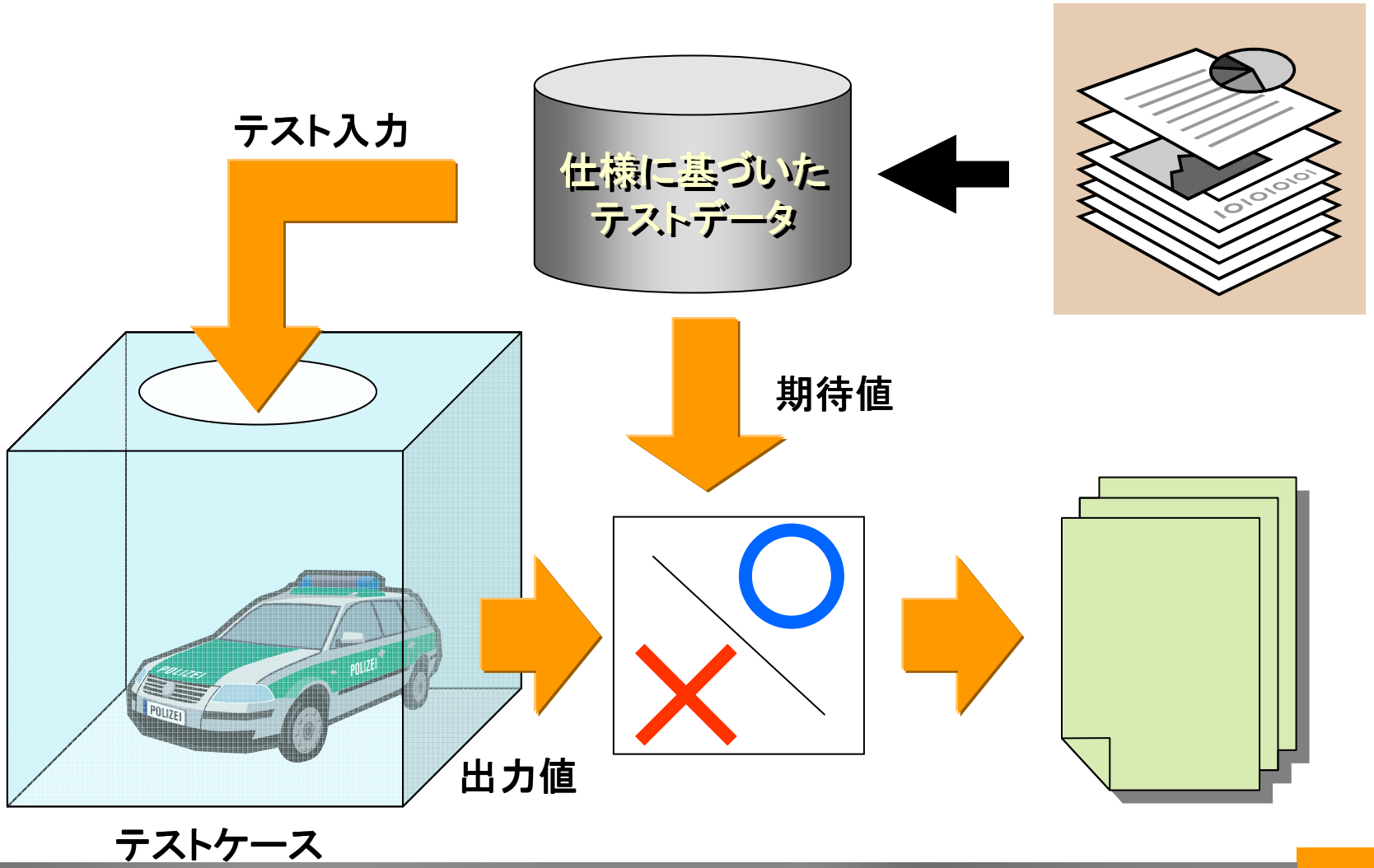
テスト結果と実施率


Jtest - AccountTest.java - Parasoft Jtest
ファイル(F) 編集(E) ソース(S) リファクタリング(T) ナビゲート(N) 検索(A) プロジェクト(P) Jtest (J)
Jtest 問題 カバレッジ コンソール サーバー
306 タスク
[306] JPetStore
[8] 未検証の単体テストの例外
[298] 未検証の結果の検証
[298] 未検証の結果
[298] 重要度 3
[298] 結果
[224] com.ibatis.jpetstore.domain
[37] Account.java
[行 40] 結果: testing getAddre
[行 55] 結果: testing getAddre
[行 70] 結果: testing getBann
[行 85] 結果: testing getCity(
[行 100] 結果: testing getCo
[行 115] 結果: testing getEm
[行 130] 結果: testing getFa
[行 145] 結果: testing getFi
[行 160] 結果: testing getLa
[行 175] 結果: testing getLa
[行 190] 結果: testing getPa
[行 205] 結果: testing getPho
ジャンプ
タスクの再割り当て...
削除
結果を検証済みにする
このアサーションを無視する
期待される値を変更
テストケースのスキップ
Jtest 問題 カバレッジ コンソール サーバー
93% [1,010/1,090 実行可能行]
JPetStore -- 93% [1,010/1,090 実行可能行]
com.ibatis.jpetstore.persistence.iface -- [実行可能行
com.ibatis.jpetstore.domain -- 98% [342/349 実行可能
Order -- 100% [118/118 実行可能行]
Account -- 100% [55/55 実行可能行]
Item -- 100% [41/41 実行可能行]
LineItem -- 100% [34/34 実行可能行]
CartItem -- 100% [20/20 実行可能行]
Product -- 100% [14/14 実行可能行]
Sequence -- 100% [12/12 実行可能行]
Cart.java
}
public void incrementQuantityByItemI
CartItem cartItem = (CartItem) ite
cartItem.incrementQuantity();
}
public void setQuantityByItemId(Strin
CartItem cartItem = (CartItem) ite
cartItem.setQuantity(quantity);
}
public BigDecimal getSubTotal() {
BigDecimal subTotal = new BigDecim

自動生成テスト＝Poor man’s test

- 自動生成の利点
  - 今まで単体テストを全くしていなかった組織に、リグレッションテストスイートをもたらしてくれる最良の方法である。
- 自動生成の問題
  - 仕様に沿った機能テストには成りえない。
  - 誤ったコードが存在しても誤ったコードを実行するためのテストを生成する。
  - カバレッジ100%は、完全なテストを実施した証拠ではない。

パラメタライズ：仕様書に沿ったテストの実施



パラメタライズの例

オリジナルテストプログラム

```
public void testSearch_record1() throws Throwable {
  AddressBook testedObject = new AddressBook();
  testedObject.set_namae("野田");
  testedObject.set_denwa("03-1234-1234");
  testedObject.set_yuubin("100-0001");
  testedObject.set_age(45);
  boolean result = testedObject.search_record();
  assertEquals(true, result); // jtest_unverified
  assertEquals("東京都", testedObject.get_ken()); // jtest_unverified
  assertEquals("１－１－１", testedObject.get_banchi()); // jtest_unverified
  assertEquals("野田", testedObject.get_namae()); // jtest_unverified
  assertEquals("西新宿", testedObject.get_cho()); // jtest_unverified
  assertEquals("03-1234-1234", testedObject.get_denwa()); // jtest_unverified
  assertEquals("100-0001", testedObject.get_yuubin()); // jtest_unverified
  ...bject.get_ku()); // jtest_unverified
  ...get_... jtest_unverified
```

テストデータをExcel等に入力。

Microsoft Excel - AddressBookTest.xls

	A	B	C	D	E	F
1	n	d	y	a	retval0	outcome0
2	野田	03-1234-12	100-0001	45	TRUE	東京都
3	A野田	03-1234-12	100-0001	45		
4	野田Z	03-1234-12	100-0001	45		
5	田	03-1234-12	100-0001	45		
6	野	03-1234-12	100-0001	45		
7	田野	03-1234-12	100-0001	45		
8		03-1234-12	100-0001	45		
9		03-1234-12	100-0001	45		
10	0	03-1234-12	100-0001	45		
11	Hello Worl	03-1234-12	100-0001	45		
12	こんにちl	03-1234-12	100-0001	45		
13	野田	A03-1234-	100-0001	45		

代入文などのデータを引数で受け取るように変更。

```
public void testSearch_record1(String n, String d, String y, int a,
boolean retval0, String outcome0, String outcome1, String outcome2, String
outcome3, String outcome4, String outcome5, String outcome6, String outcome7)
throws Throwable {
  AddressBook testedObject = new AddressBook();
  testedObject.set_namae(n);
  testedObject.set_denwa(d);
  testedObject.set_yuubin(y);
  testedObject.set_age(a);
  boolean result = testedObject.search_record();
  assertEquals(retval0, result);
  assertEquals(outcome0, testedObject.get_ken());
  :
```

```
public static junit.framework.Test suiteSearch_record1() throws ...Exception {
  return jtest.PT.getExcelInputTestSuite(
    AddressBookTest.class, "testSearch_record1",
    AddressBookTest.class.getClassLoader().getResource(
    "co/jp/parasoft/demo/AddressBookTest.xls"), ..., 9,
    new String[] { "n", "d", "y", "a", "retval0", "outcome0", "outcome1",
    "out...2", "outcome3", "outcome4", "outcome5", "outcome6", "outcome7" }
  );
}
```

Excel等を読み取り、行数回繰り返すドライバを作成。

既存アプリのメンテナンス作業の課題

発注元（システムオーナー）

改造作業は少量であっても、新規開発時と同様のテストが必要。

受託開発者

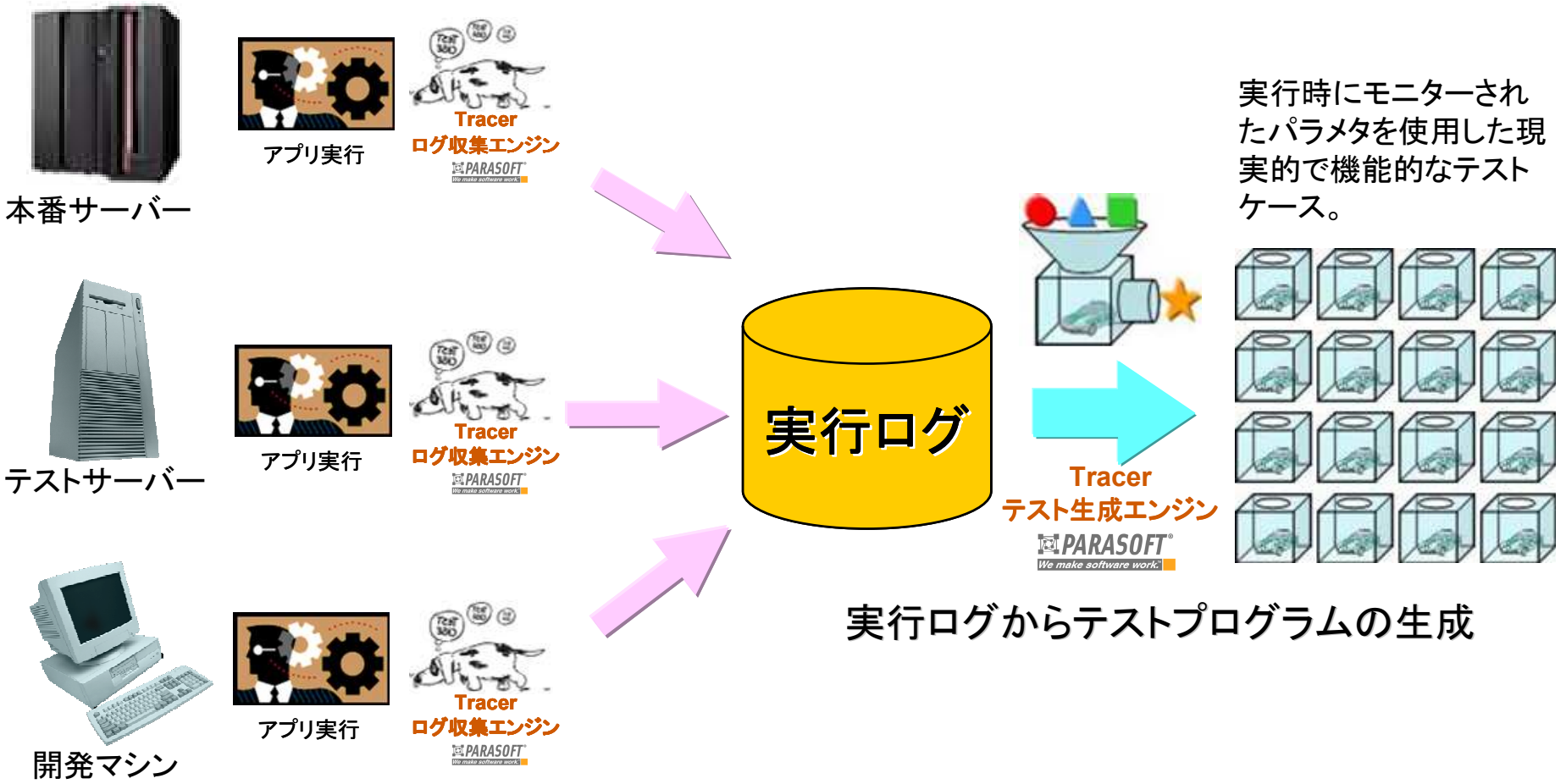
既存アプリ用のテスト資産が無ければ、新規開発時と同様のテストの実施は不可能。

現実は

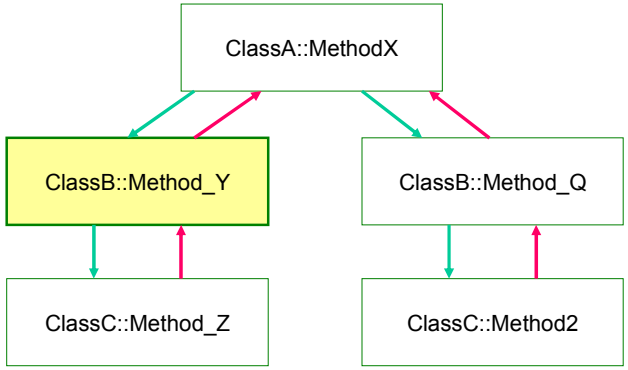
改造した部分のみテストしてリリース

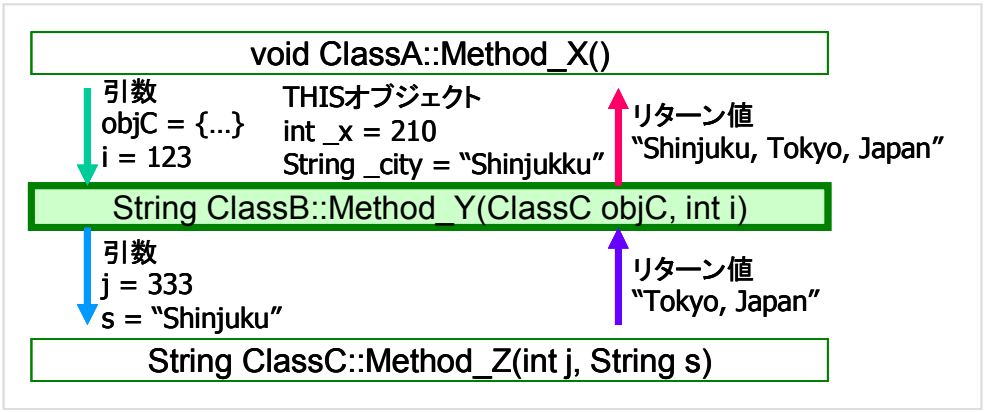
Tracer :既存システム用テスト資産の生成

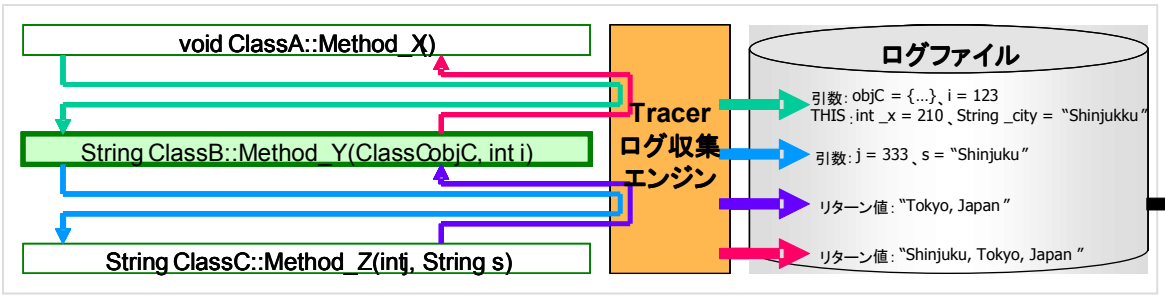
- 既存システム用の現実的で機能的な単体テスト資産を提供

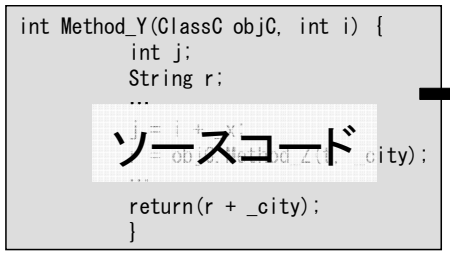


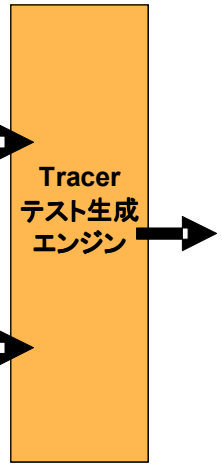
実行モニター　⇒　リグレッションテスト生成

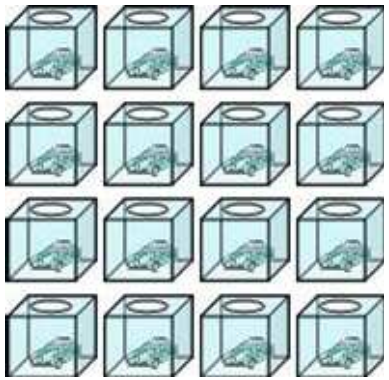












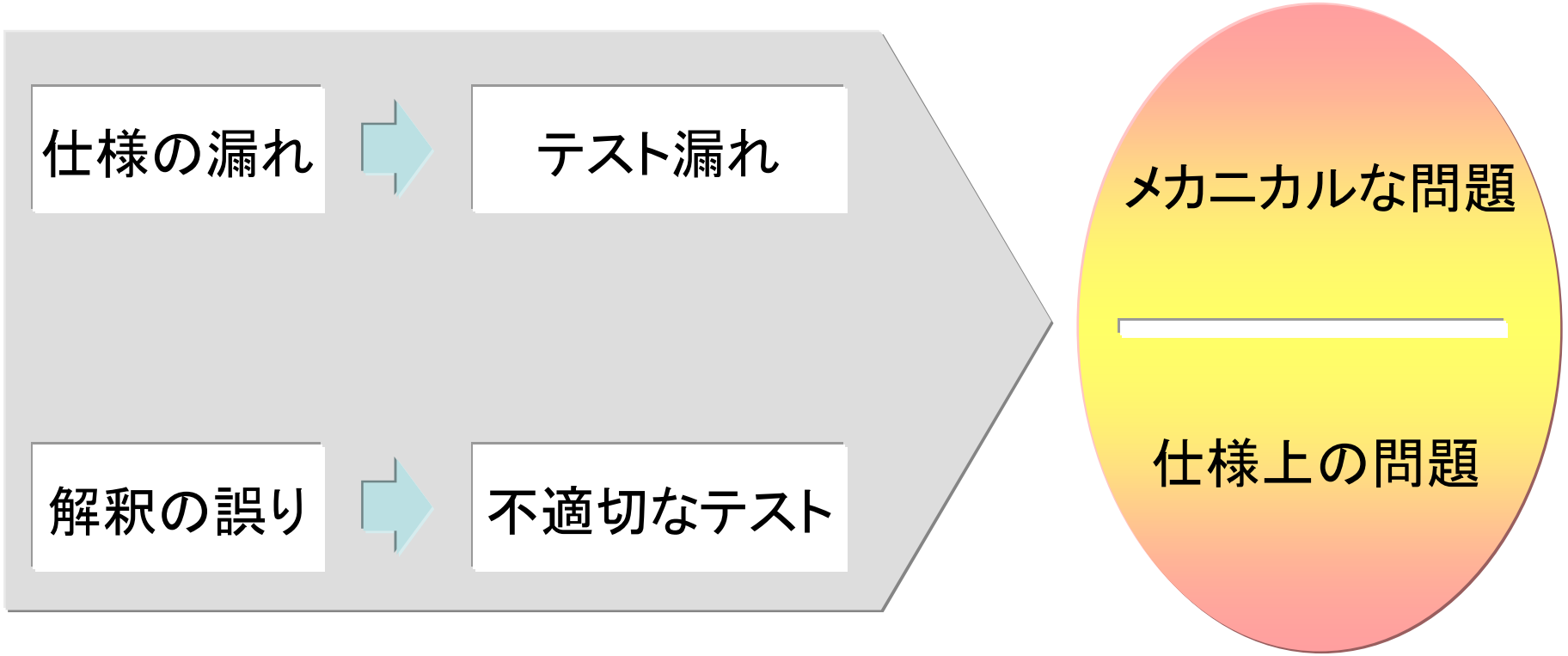
テストケース

テストの落とし穴

- テストとは、仕様書や設計書に記載されたとおりにプログラムが動作するか否かを検証するものである。

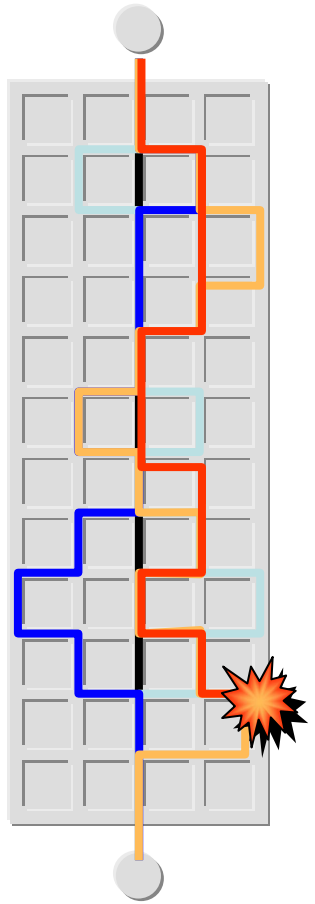
- 仕様書・設計書の矛盾、漏れ
  - 仕様書に記載されていないことは、テストされない。
- 誤った解釈
  - 第三者が指摘しないかぎり、本人は気がつかない。
- 暗黙の仕様（特定のコミュニティー内での常識）
  - 例）ふりがな・フリガナ

テストの落とし穴



バグ探偵:メカニカルな問題は、メカニカルに検出

- バグ探偵(フロー解析による問題の検出)
  - プログラムを実行することなく、フローを解析し、問題を検出。
    - 全ての分岐の組合せを検証
      - 従来のテスト方式では、テストケースの作成に多大な努力が必要。実行にも多くの時間が必要。
    - データフロー解析により
      - プログラムを実行することなく、データの流れを解析し、特定の分岐の組み合わせにより、発生する可能性のある問題を検出。



フロー解析により理論的に検出可能な問題

- 不定、不良データへのアクセス
  - 配列の範囲外へのアクセス
  - 初期化されていない変数などへのアクセス
  - 不正文字入力などのデータの有効性チェック処理の漏れ
  - 有効範囲（上限値～下限値）
- リソース開放処理の問題
  - リソース開放処理の漏れ、二重開放
  - 開放済みリソースへのアクセス
  - 開放と再作成の無駄な繰り返し
- 例外処理、エラー処理の漏れ
  - 例外の受信処理の漏れ
  - エラーチェック漏れ
- 処理手順、フロー制御の問題
  - Open->Read->Write->Closeなどの順序違反
  - 排他制御でのデッドロック
  - 実行されることのないコード、重複したコード、常に同じ結果のif文

【注意】本リストは、理論的な可能性を示唆ものであり、製品の機能ではありません。

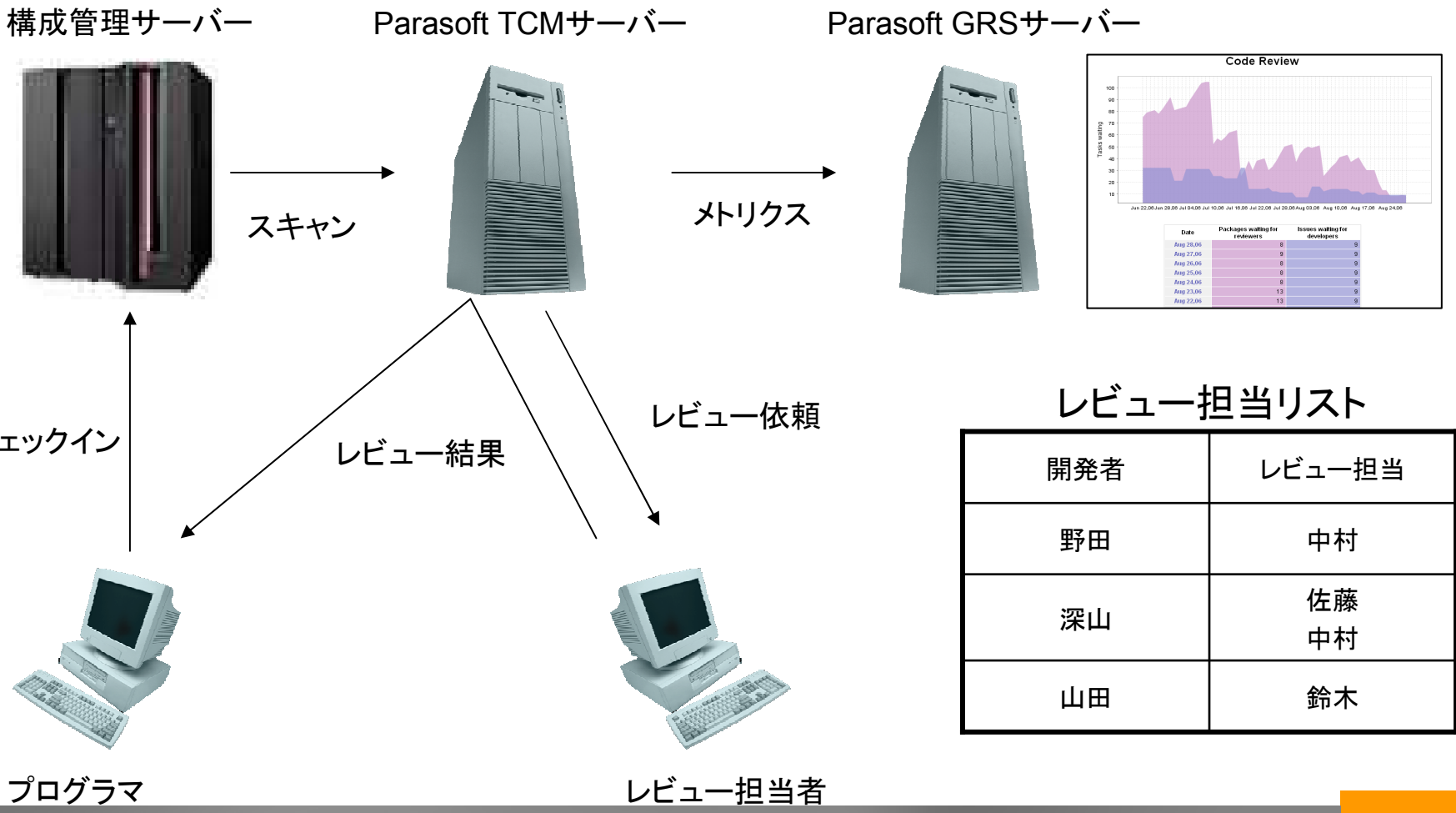
仕様上の問題は？

☆ ☆　ツールは、仕様を理解できない。　☆ ☆

レビューが唯一の発見手段

Parasoftの製品開発チームでも、レビューは最も重要な「品質向上の手段」として、厳密に実施しています。

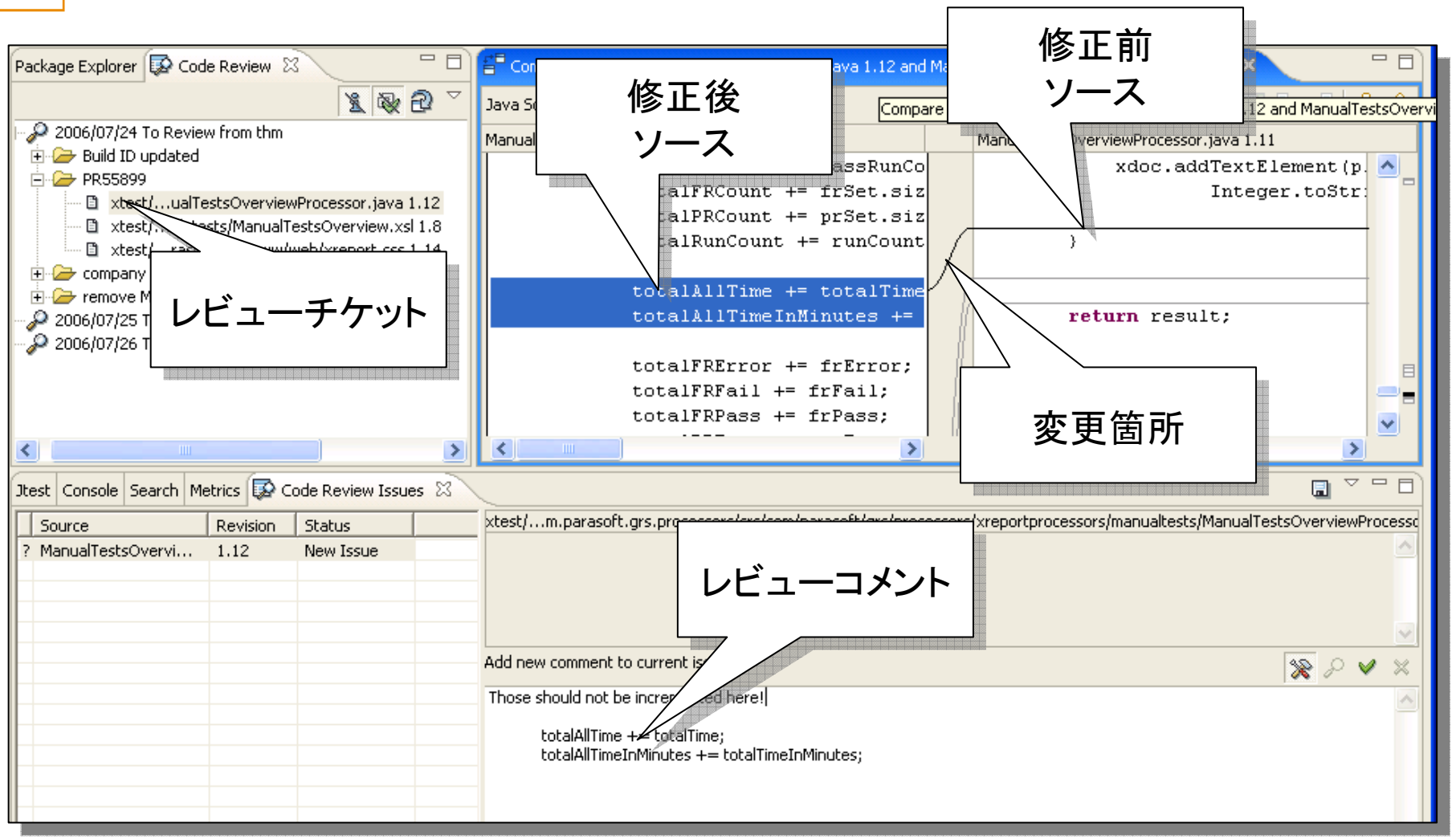
コードレビュー



レビュー担当リスト

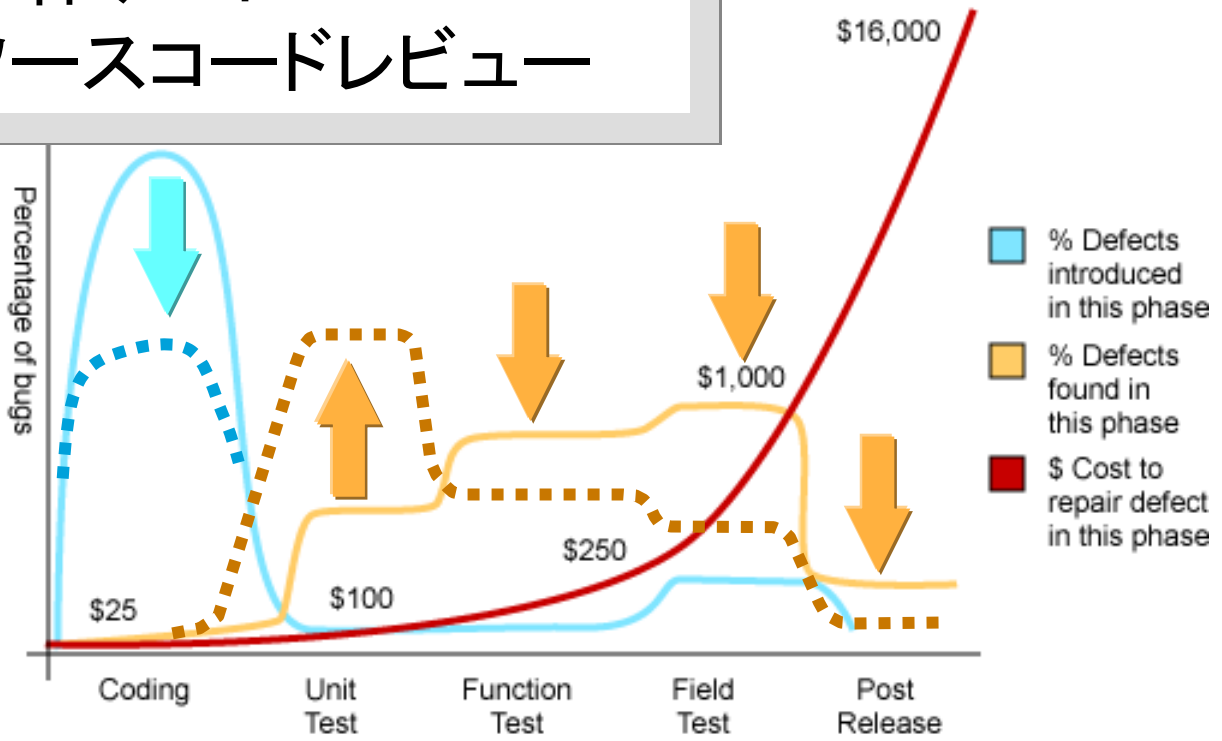
開発者	レビュー担当
野田	中村
深山	佐藤 中村
山田	鈴木

コードレビュー：レビューチケット



“苦労”削減のためのベストプラクティス

- コーディング規約
- 単体テスト
- ソースコードレビュー



Source: Applied Software Measurement, Capers Jones, 1996

THANK YOU

お問い合わせ

テクマトリックス株式会社

システムエンジニアリング事業部

ソフトウェアエンジニアリング営業課

TEL 03-5792-8606

FAX 03-5792-8706

E-MAIL: parasoft-info@techmatrix.co.jp

URL: http://www.techmatrix.co.jp

東京都港区高輪4丁目10番8号 京急第7ビル

Parasoft Japan株式会社

TEL 03-5322-1315

FAX 03-5322-2929

E-MAIL: info@parasoft.co.jp

URL: http://www.parasoft.co.jp

東京都新宿区西新宿1-26-2 新宿野村ビル32F