

PHP7 で堅牢なコードを書く

例外処理、表明プログラミング、契約による設計

和田 卓人 (@t_wada)

Nov 3, 2016 @ PHPカンファレンス2016

#phpcon2016_1



和田 卓人

id: t-wada

@t_wada

github: twada

ひとり歩きするスタンド

名無し募集中。。。 : 2011/12/06(火) 18:32:24.37 0



テスト書いてないとかお前それ
@t_wadaの前でも
同じこと言えんの?

注: でも今日はテストの話はしません

とあるところに、こんなコードがありました

BAD

```
class BugRepository
{
    public static function findAll($params)
    {
        global $CONF;
        $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                       [ PDO::ATTR_EMULATE_PREPARES => false ]);
        $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
                WHERE assigned_to = :assignedTo AND status = :status';
        $stmt = $pdo->prepare($sql);
        $stmt->execute($params);
        return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
    }
}

print_r(BugRepository::findAll([
    'assignedTo' => '12',
    'status' => 'OPEN'
]));
```

注: 『SQLアンチパターン』のひどい
コード例をアレンジして書いています



処理が成功すると、こんな感じ。

```
$ php example.php
Array
(
    [0] => Bug Object
        (
            [bug_id] => 842
            [summary] => 保存処理でクラッシュする
            [date_reported] => 2016-10-26
        )

    [1] => Bug Object
        (
            [bug_id] => 5150
            [summary] => XMLのサポート
            [date_reported] => 2016-10-26
        )

    [2] => Bug Object
        (
            [bug_id] => 6060
            [summary] => パフォーマンスの向上
            [date_reported] => 2016-10-26
        )
)
```

突然ですが
ここで
クイズです ¹⁰⁰
~~100~~

処理失敗の原因になる可能性があるのはどの行？

BAD

```
public static function findAll($params)
{
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
        [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```



制限時間10秒

BAD

```
public static function findAll($params)
{
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```



データベース接続確立失敗



`usr` , `passwd` 等キー名が変更された

BAD

```
public static function findAll($params)
{
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

🚫 テーブル名やカラム名が誰かに変更された

🚫 (ここで) データベース接続エラー

Fatal error: Call to a member function execute() on a non-object

BAD

```
public static function findAll($params)
{
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

- 🚫 \$params が null
- 🚫 \$params のキー名や数の不一致
- 🚫 \$params の値が文字列に変換不能
- 🚫 (ここで) データベース接続エラー

BAD

```
public static function findAll($params)
{
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

🚫 Bug クラスが未定義

🚫 (ここで) データベース接続エラー(※ 設定による)

処理失敗の原因になる可能性があるのはどの行？

BAD

```
public static function findAll($params)
{
    global $CONF;
    ① $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                    [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    ② $stmt = $pdo->prepare($sql);
    ③ $stmt->execute($params);
    ④ return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

えつ、こんなにあるの?? 😊

このなかでも特に手強い状況はどれ？

- ✖️ データベース接続確立失敗
- ✖️ `usr` , `passwd` 等キーワードが変更された
- ✖️ テーブル名やカラム名が誰かに変更された
- ✖️ \$params が null
- ✖️ \$params のキーワードや数の不一致
- ✖️ \$params の値が文字列に変換不能
- ✖️ Bug クラスが未定義
- ✖️ 途中でデータベース接続エラー

私見では: \$params のキーナンや数の不一致

BAD

```
public static function findAll($params)
{
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
        [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

🚫 \$params のキーナンや数の不一致

キー名や数を間違えて実行するとどうなる?

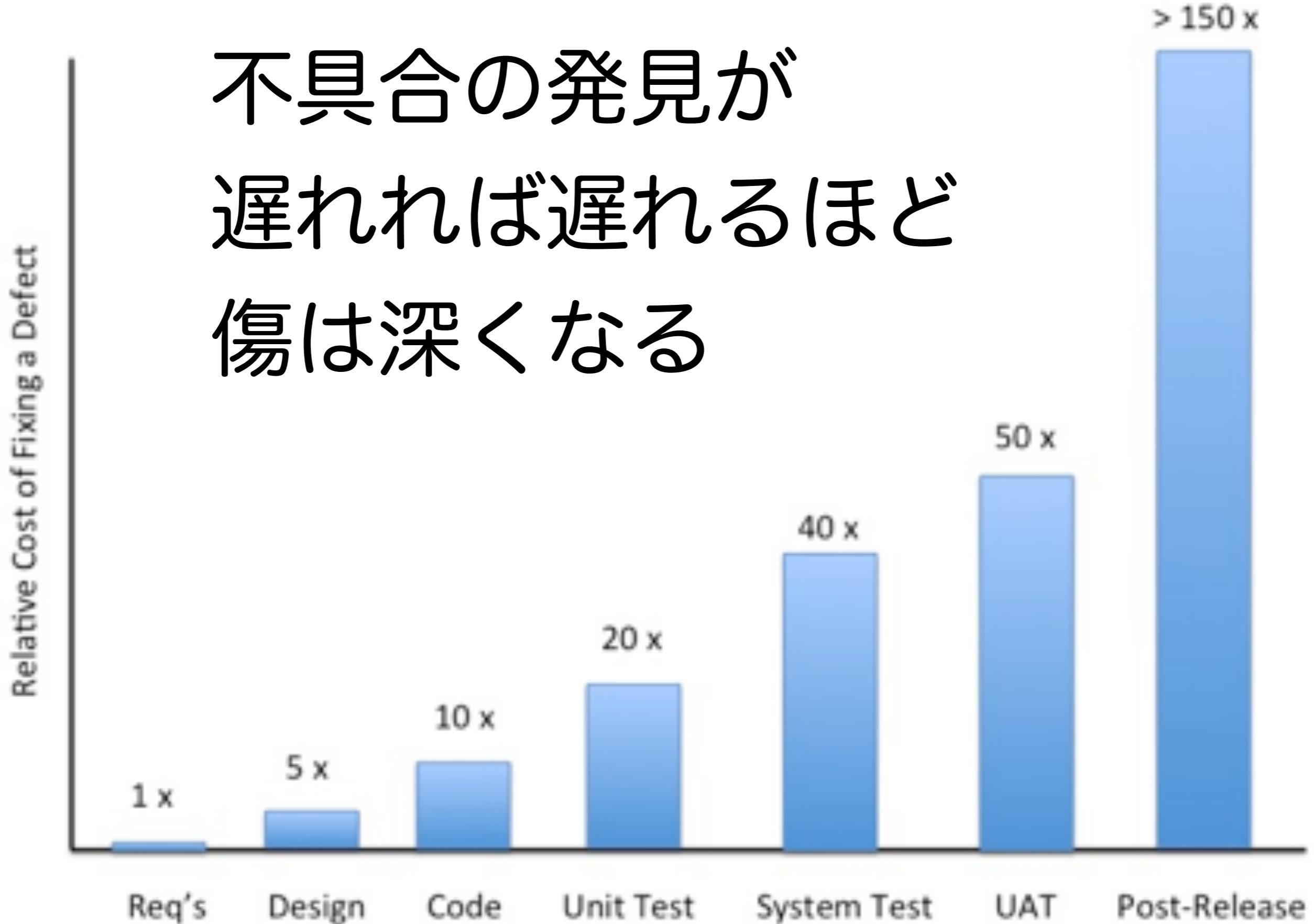
!?

```
$ php example.php
PHP Warning: PDOStatement::execute(): SQLSTATE[HY093]: Invalid
parameter number: parameter was not defined in example.php
Array
(
)
```

裏でこっそり警告が出るだけなの? 😕

空配列が返ってくるの!? 😳

正常系と見分けがつかない? 😭



処理失敗の原因になる可能性があるのはどの行？

BAD

```
public static function findAll($params)
{
    global $CONF;
    ① $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                    [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    ② $stmt = $pdo->prepare($sql);
    ③ $stmt->execute($params);
    ④ return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

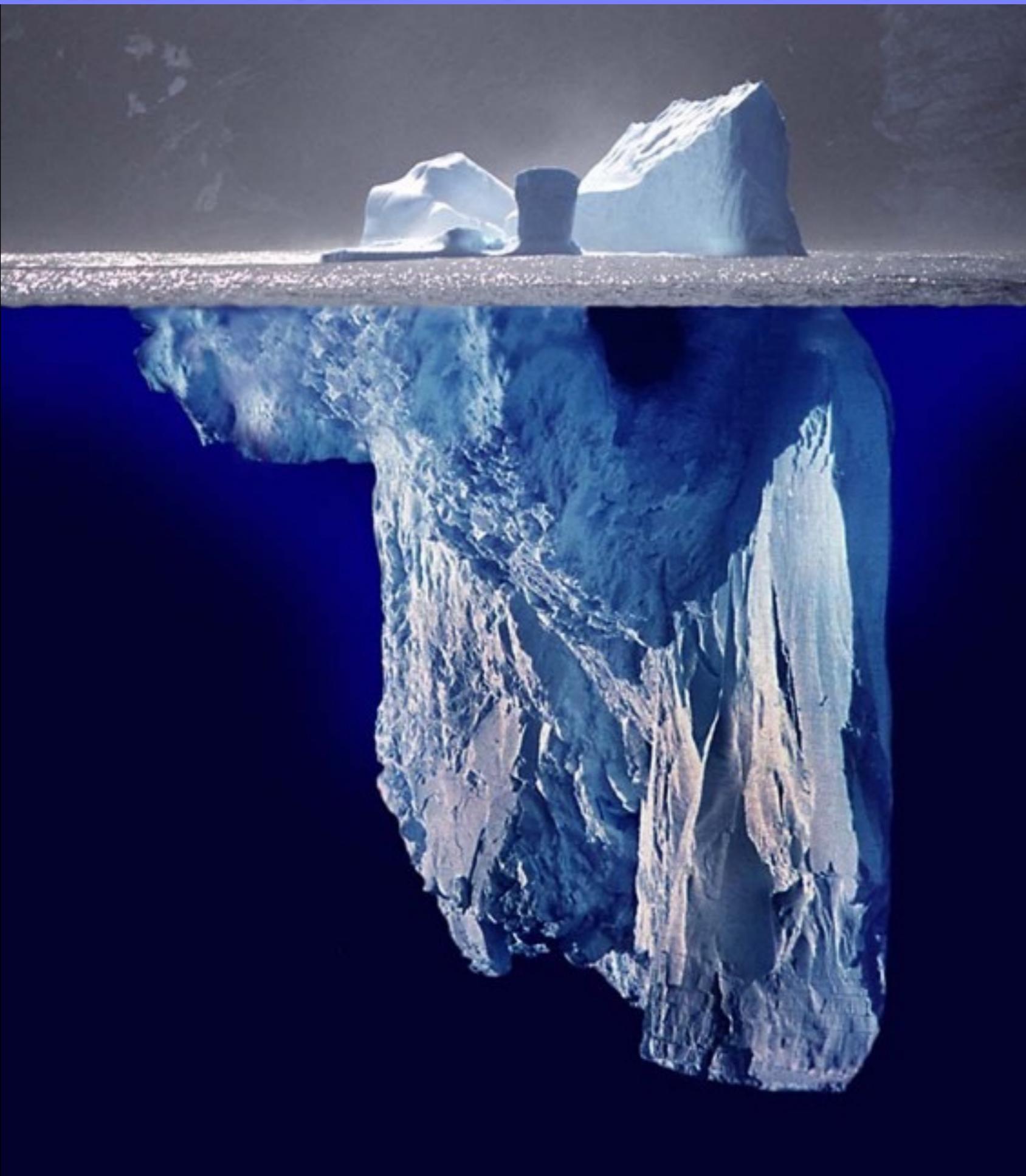
現実世界へようこそ

“賢明なソフトウェア技術者になるための第一歩は、動くプログラムを書くことと正しいプログラムを適切に作成することの違いを認識すること”

— M.A.Jackson (1975)



堅牢なコードは 50% 以上がエラーハンドリング



Agenda

第1部: 予防的プログラミング

第2部: 攻撃的プログラミング

第3部: 契約プログラミング

失敗の原因について考え始めよう

- ? データベース接続確立失敗
- ? `usr`, `passwd` 等キーワードが変更された
- ? テーブル名やカラム名が誰かに変更された
- ? \$params が null
- ? \$params のキーワードや数の不一致
- ? \$params の値が文字列に変換不能
- ? Bug クラスが未定義
- ? 途中でデータベース接続エラー

失敗の原因について考え始めよう

- 💡 データベース接続確立失敗
 - 🐛 `usr`, `passwd` 等キーワードが変更された
 - 💀 テーブル名やカラム名が誰かに変更された
 - 👮 \$params が null
 - 👮 \$params のキーワードや数の不一致
 - 👮 \$params の値が文字列に変換不能
 - 🐛 Bug クラスが未定義
 - 💡 途中でデータベース接続エラー

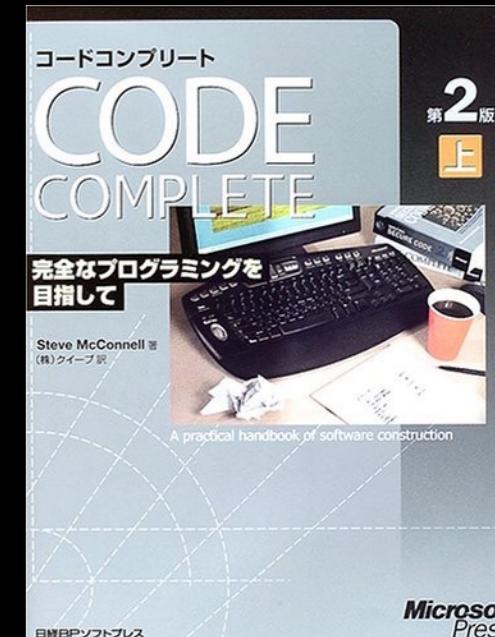
間違った使われ方をされやすい、という問題がありそう

- 👉 😇 データベース接続確立失敗
- 👉 🐛 `usr`, `passwd` 等キーワード名が変更された
- 👉 💀 テーブル名やカラム名が誰かに変更された
- 👉 ⚙️ 👮 \$params が null
- 👉 ⚙️ 👮 \$params のキーワード名や数の不一致
- 👉 ⚙️ 👮 \$params の値が文字列に変換不能
- 👉 🐛 Bug クラスが未定義
- 👉 😇 途中でデータベース接続エラー

対処 <<< 予防
予防的プログラミング

防御的プログラミング

- 「防御的プログラミング」とは、プログラミングに対して防御的になること、つまり「**そうなるはずだ**」と決めつけないこと
- 防御的プログラミングの根底にあるのは、ルーチンに不正なデータが渡されたときに、**それが他のルーチンのせいであったとしても、被害を受けない**ようにすること
- このため、
 - 外部ソースからのデータの値をすべて確認する
 - ルーチンのすべての入力引数の値を確認する
 - 不正な入力を処理する方法を決定する



(狭義の)防御的プログラミングへの誤解

ただひたすら入力をチェックしようつたり

BAD

```
public static function findAll($params)
{
    if (is_null($params)) {
        throw new InvalidArgumentException('params should not be null');
    }
    if (!is_array($params)) {
        throw new InvalidArgumentException('params should be an array');
    }
    if (count($params) !== 2) {
        throw new InvalidArgumentException('params should have exact two items');
    }
    if (!array_key_exists('assignedTo', $params) ||
        !array_key_exists('status', $params)) {
        throw new InvalidArgumentException('params should have key `assignedTo` and `status` only');
    }
    if (!is_int($params['assignedTo'])) {
        throw new InvalidArgumentException('params[`assignedTo`] should be an integer');
    }
    if (!is_string($params['status'])) {
        throw new InvalidArgumentException('params[`status`] should be a string');
    }
    if (!in_array($params['status'], ['OPEN', 'NEW', 'FIXED'], true)) {
        throw new InvalidArgumentException('params[`status`] should be in `OPEN`, `NEW`, `FIXED`');
    }

    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
        [ PDO::ATTR_EMULATE_PREPARES => false ]);
}
```

不正な入力があっても自分でなんとかしようとしたり

BAD

```
global $CONF;
$dsn = $CONF['dsn'] ?? $CONF['ds'] ?? $CONF['dataSource'];
$user = $CONF['usr'] ?? $CONF['user'];
$password = $CONF['passwd'] ?? $CONF['password'];
$pdo = new PDO($dsn, $user, $password,
                [ PDO::ATTR_EMULATE_PREPARES => false ]);
$sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = :assignedTo AND status = :status';
$stmt = $pdo->prepare($sql);
$safeParams = [
    'assignedTo' => $params['assignedTo'] ?? $params['assigned_to'],
    'status' => $params['status'] ?? 'OPEN',
];
$stmt->execute($safeParams);
$className = class_exists('Bug') ? 'Bug' : 'BugModel';
return $stmt->fetchAll(PDO::FETCH_CLASS, $className);
```

ドキュメントで間違いやすさを補おうとしたり

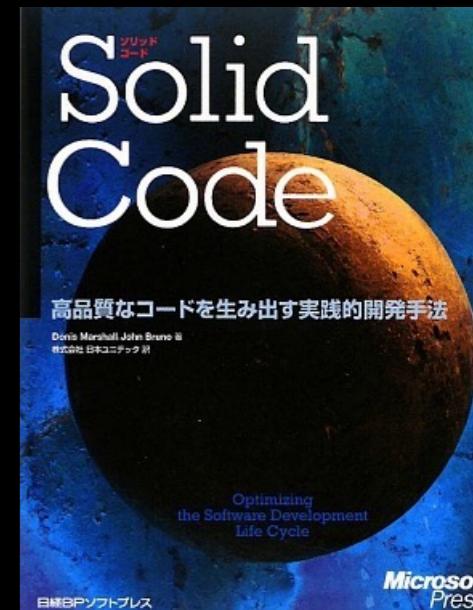
BAD

```
/**  
 * 担当者、ステータスに合致する Bug を検索し、ヒットした全件を Bug オブジェクト  
の配列として返す。  
 *  
 * @param array $params 格納した検索条件の連想配列。キー 'assignedTo' にユーザID  
をintで、キー 'status' にステータス文字列をstringで指定すること。キー、値それぞ  
れNULLは不可とする。  
 * @return Bug[] 検索結果を Bug オブジェクトにマッピングして返す。検索結果が0件  
のときは空配列を返す。  
 */  
public static function findAll($params)  
{  
    global $CONF;  
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],  
        [ PDO::ATTR_EMULATE_PREPARES => false ]);
```

防御的プログラミングとは
悪いコードに絆創膏をあて
ることではない

防御的プログラミング

- 「防御的プログラミング」とは、問題発生を事前に防ごうというコーディングスタイル
 - 可読性の高いコードと適切な命名規則
 - 全ての関数の戻り値をチェック
 - デザインパターンの採用
- 要するに、良識ある実践の積み重ねである
- 防御的プログラミングは、正しいコード作成のための規律をプログラマが一貫して適用するための一種のコーディング標準



きのこ53: 正しい使い方を簡単に、 誤った使い方を困難に

- ・良いインターフェースとは次の2つの条件を満たすインターフェース
 - ・正しく使用する方が操作ミスをするより簡単
 - ・誤った使い方をすることが困難



型の制限

→ 型宣言

きのこJ6: 見知らぬ人ともうまくやるには

“「出来てはならぬことを禁じる」のではなく、はじめから「出来ていいことだけを出来るようにする」と考えるのです”



型宣言によって「出来ていいことだけを出来る」ように

PHP7

```
public static function findAll(int $assignedTo, string $status)
{
    if (is_null($params)) {
        throw new InvalidArgumentException('params should not be null');
    }
    if (!is_array($params)) {
        throw new InvalidArgumentException('params should be an array');
    }
    if (count($params) !== 2) {
        throw new InvalidArgumentException('params should have exact two items');
    }
    if (!array_key_exists('assignedTo', $params) ||
        !array_key_exists('status', $params)) {
        throw new InvalidArgumentException('params should have key \'assignedTo\' and
\'status\' only');
    }
    if (!is_int($params['assignedTo'])) {
        throw new InvalidArgumentException('params[\'assignedTo\'] should be an
integer');
    }
    if (!is_string($params['status'])) {
        throw new InvalidArgumentException('params[\'status\'] should be a string');
    }
    if (!in_array($params['status'], ['OPEN', 'NEW', 'FIXED'], true)) {
        throw new InvalidArgumentException('params[\'status\'] should be in
\'OPEN\', \'NEW\', \'FIXED\'');
    }
}
```

想定しなければならない状況が減った

PHP7

```
public static function findAll(int $assignedTo, string $status)
{
    if (!in_array($status, ['OPEN', 'NEW', 'FIXED'], true)) {
        throw new InvalidArgumentException('params['status'] should
be in \'OPEN\',\'NEW\',\'FIXED\'');
    }
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
        [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT);
    $stmt->bindValue(':status', $status, PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

値の制限
→ Enum

きのこ89: 関数の「サイズ」を小さくする

- 言語組み込みの型(int, string 等)を使うと、取り得る値の組み合わせが膨大になる
- 問題領域の知識を活用して固有の型を作ることで、取り得る組み合わせを大幅に減らせる



PHPで列挙型(enum)を作る

89
ストック0
コメント

php PHP 7643



Hirakuが2013/03/08に投稿(2013/03/14に編集) • Gistを開く • 編集履歴(3) • 問題がある投稿を報告する



① この記事は最終更新日から1年以上が経過しています。

Tweet

G+1

Like 5

Pocket

11

列挙型 - Wikipedia

まず列挙型の定義は～となるんだけど、ここでは「あらかじめ定義した値のいずれかしか取らない特殊な型」という感じを想定します。

要はSplEnumみたいなのですが、拡張モジュールの力を借りなくとも、PHPだけで作れます。リフレクションを使うだけ。

```
<?php
abstract class Enum
{
    private $scalar;

    function __construct($value)
    {
        $ref = new ReflectionObject($this);
        $consts = $ref->getConstants();
```



Hiraku

4131 Contribution

+フォロー

人気の投稿

- WebAPIリクエスト仕様書とコマンドのご提案
- コードをまとめる技術としてレータとジェネレータ
- PHPで高速オシャレな配列操作
- ブロックコメントの工夫
- curl_multiでHTTP並行リクエストサンプル

Hiraku さんの Enum 実装を (少しアレンジして) 使ってみる

```
abstract class Enum
{
    private $scalar;

    public function __construct($value)
    {
        $ref = new ReflectionObject($this);
        $constants = $ref->getConstants();
        if (!in_array($value, $constants, true)) {
            throw new InvalidArgumentException("value [{\$value}] is not defined");
        }
        $this->scalar = $value;
    }

    public final function value()
    {
        return $this->scalar;
    }

    public final function __toString()
    {
        return (string)$this->scalar;
    }
}
```

あらかじめ定義された値だけをインスタンス化できる

```
class Status extends Enum  
{  
    const OPEN = 'OPEN';  
    const NEW = 'NEW';  
    const FIXED = 'FIXED';  
}
```

```
$status = new Status(Status::OPEN);  
$status = new Status('OPEN');
```

```
// "InvalidArgumentException: value [HOGE] is not defined"  
$status = new Status('HOGE');
```

想定しなければならない状況がさらに減った

```
public static function findAll(int $assignedTo, Status $status)
{
    if (!in_array($status, ['OPEN', 'NEW', 'FIXED'], true)) {
        throw new InvalidArgumentException('params["status"] should
be in "OPEN", "NEW", "FIXED"');
    }
    global $CONF;
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
        [ PDO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT);
    $stmt->bindValue(':status', $status->value(), PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

誤りやすいインターフェイスに起因する処理失敗を撲滅

- ✖️  データベース接続確立失敗
- ✖️  `usr`, `passwd` 等キー名が変更された
- ✖️  テーブル名やカラム名が誰かに変更された
- ✓  \$params が null
- ✓  \$params のキー名や数の不一致
- ✓  \$params の値が文字列に変換不能
- ✖️  Bug クラスが未定義
- ✖️  途中でデータベース接続エラー

次の相手は、知りすぎ、責務の多すぎに起因する処理失敗

- 👉 😇 データベース接続確立失敗
- 👉 🐛 `usr`, `passwd` 等キーワードが変更された
- 💀 テーブル名やカラム名が誰かに変更された
- 👮 \$params が null
- 👮 \$params のキーワードや数の不一致
- 👮 \$params の値が文字列に変換不能
- 🐛 Bug クラスが未定義
- 😇 途中でデータベース接続エラー

知りすぎない
やり過ぎない
責務の再配置

PDO 生成と設定の責務を外部に出し、コンストラクタで受け取る

```
class BugRepository
{
    private $pdo;

    public function __construct(PDO $pdo)
    {
        $this->pdo = $pdo;
    }

    // 以下省略
}
```

```
// 利用側
$pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'], [
    PDO::ATTR_EMULATE_PREPARES => false,
]);
$repo = new BugRepository($pdo);
print_r($repo->findAll(12, new Status(Status::OPEN)));
```

知りすぎ、責務の多すぎに起因する処理失敗を撲滅

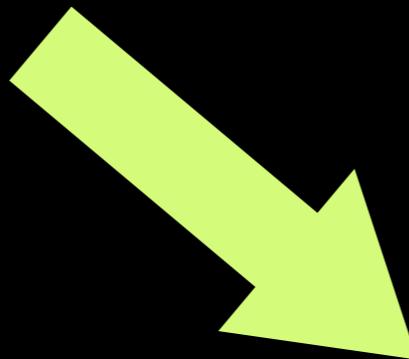
- ✓ 😇 データベース接続確立失敗
- ✓ 🐛 `usr`, `passwd` 等キー名が変更された
- 💀 テーブル名やカラム名が誰かに変更された
- 👮 \$params が null
- 👮 \$params のキー名や数の不一致
- 👮 \$params の値が文字列に変換不能
- 🐛 Bug クラスが未定義
- 😇 途中でデータベース接続エラー

第1部まとめ: 予防に勝る防御なし

```
public static function findAll($params)
{
    if (is_null($params)) {
        throw new InvalidArgumentException('params should not be null');
    }
    if (!is_array($params)) {
        throw new InvalidArgumentException('params should be an array');
    }
    if (count($params) !== 2) {
        throw new InvalidArgumentException('params should have exact two items');
    }
    if (!array_key_exists('assignedTo', $params) ||
        !array_key_exists('status', $params)) {
        throw new InvalidArgumentException('params should have key `assignedTo` and `status` only');
    }
    if (!is_int($params['assignedTo'])) {
        throw new InvalidArgumentException('params[`assignedTo`] should be an integer');
    }
    if (!is_string($params['status'])) {
        throw new InvalidArgumentException('params[`status`] should be a string');
    }
    if (!in_array($params['status'], ['OPEN', 'NEW', 'FIXED'], true)) {
        throw new InvalidArgumentException('params[`status`] should be in `OPEN`, `NEW`, `FIXED`');
    }

    global $CONF;
    if (!isset($CONF['dsn'])) {
        throw new LogicException('config key `dsn` not found');
    }
    if (!isset($CONF['usr'])) {
        throw new LogicException('config key `usr` not found');
    }
    if (!isset($CONF['passwd'])) {
        throw new LogicException('config key `passwd` not found');
    }
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
        [ PDO::ATTR_EMULATE_PREPARES => false ]);

    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    if (!class_exists('Bug')) {
        throw new LogicException('class `Bug` does not exist');
    }
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```



```
public function __construct(PDO $pdo)
{
    $this->pdo = $pdo;
}

public function findAll(int $assignedTo, Status $status)
{
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $this->pdo->prepare($sql);
    $stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT);
    $stmt->bindValue(':status', $status->value(), PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

現在の心配事リスト

- 💀 テーブル名やカラム名が誰かに変更された
- 👉 🐛 Bug クラスが未定義
- 😊 途中でデータベース接続エラー

Agenda

第1部: 予防的プログラミング

第2部: 攻撃的プログラミング

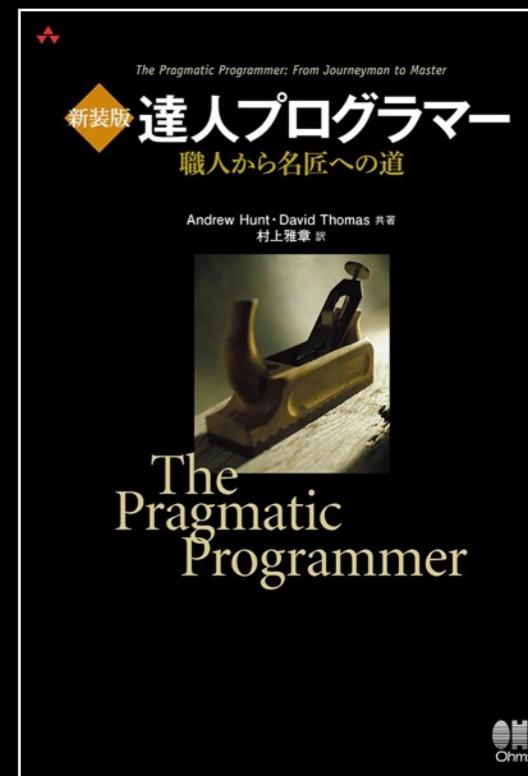
第3部: 契約プログラミング

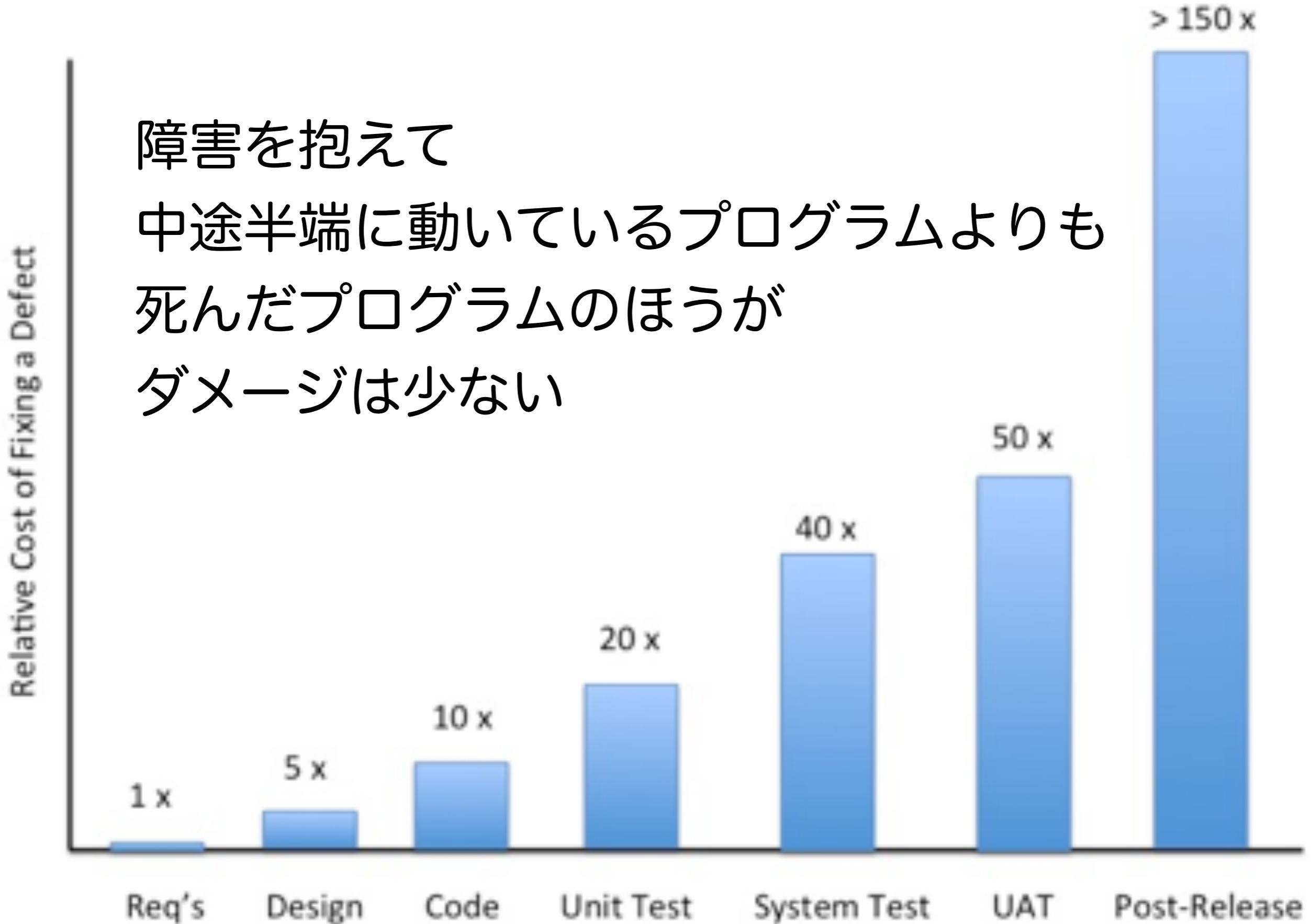
fail fast

攻撃的プログラミング

ヒント32: 早めにクラッシュさせること

- コード中に「あり得ない」と思われる何かが発生した場合、その時点でプログラムはもはや実行可能なものとはなっていない
- 何らかの疑いがあるのであれば、どのような場合でも速やかに停止させるべき。通常の場合、障害を抱えて中途半端に動いているプログラムよりも死んだプログラムのほうがダメージは少ない



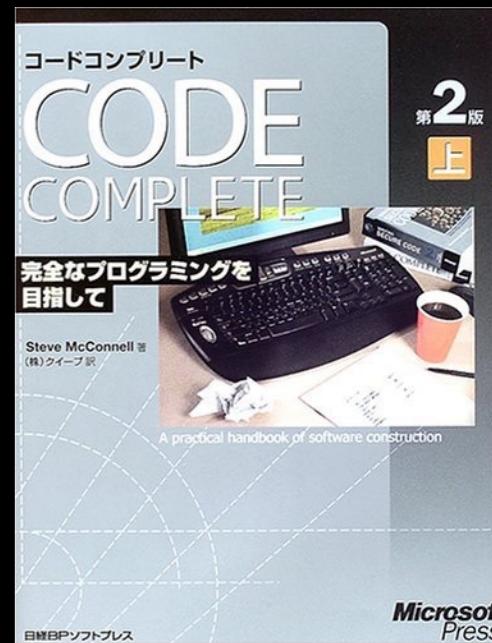


障害を抱えて
中途半端に動いているプログラムよりも
死んだプログラムのほうが
ダメージは少ない

そんなに気楽に
システムを落として
いいの? 😰

堅牢性と正当性

- 最適なエラー処理はエラーが発生したソフトウェアの種類により異なる
- 正当性とは、不正確な結果を決して返さないことを意味する。不正確な結果を返すくらいなら、何も返さない方がましである
- 堅牢性とは、ソフトウェアの実行を継続できるように手を尽くすことである。それによって不正確が結果がもたらされることがあってもかまわない
- 安全性を重視するアプリケーションでは、堅牢性よりも正当性が優先される傾向にある
- コンシューマアプリケーションでは、正当性よりも堅牢性が優先される傾向にある



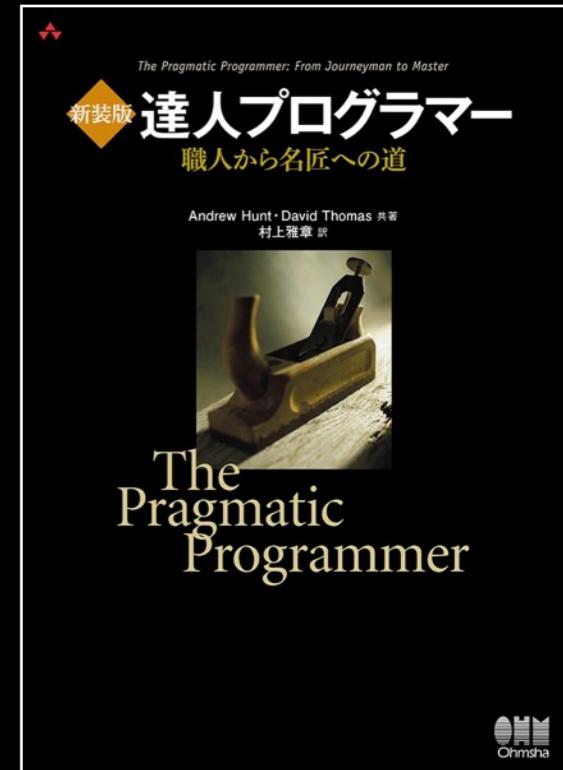
個々のクラスは正当性を重視し、
堅牢性はアーキテクチャ/フレームワーク等で
保証するのがオススメ

例: 個々のクラスは fail fast 原則で書き、Web
フレームワークやグローバルハンドラがキャッチ
して 500 エラー画面等を出す責務を負う

暗黙の前提を明示し
バグをあぶり出す
表明プログラミング

ヒント33: もし起こり得ないというのであれば、 表明を用いてそれを保証すること

「起ころるはずがない」と思って
いることがあれば、それをチェック
するコードを追加してください。
表明(assertion)を用いるの
が最も簡単な方法です



PHP7 の assert

PHP 5 および PHP 7

```
bool assert ( mixed $assertion [, string $description ] )
```

PHP 7

```
bool assert ( mixed $assertion [, Throwable $exception ] )
```

assert() は、指定した **assertion** を調べて、結果が **FALSE** の場合に適切な動作をします。

assertion

アサーション。PHP 5 では、評価対象の文字列か、あるいは boolean 値しか指定できませんでした。PHP 7 ではそれ以外にも、値を返すあらゆる式を指定できます。この式を実行した結果を用いて、アサーションに成功したか否かを判断します。

assert の引数に評価式を書く

```
class BugRepository
{
    public function __construct(PDO $pdo)
    {
        assert(class_exists('Bug'));
        $this->pdo = $pdo;
    }
}
```

!?

```
$ php example.php
```

PHP Warning: assert(): assert(class_exists('Bug'))
failed in /path/to/example.php on line 54

PHP Fatal error: Class 'Bug' not found in /path/to/
example.php on line 66

😊 評価式が出てわかりやすい

😔 でも警告が出るだけで、落ちない



PHP7 の assert の設定

PHP 7 における assert() 用の設定ディレクティブ

デフ

ディレクティブ オル 取り得る値
ト値

- | | | |
|-------------------------|---|---|
| <u>assert.exception</u> | 0 | <ul style="list-style-type: none">• 1: アサーションに失敗した場合には、exception で指定したオブジェクトをスローするか、exception を指定していない場合は AssertionError オブジェクトをスローします。• 0: 先述の Throwable を使ったり生成したりしますが、そのオブジェクト上で警告を生成するだけであり、スローしません (PHP 5 と互換性のある挙動です)。 |
|-------------------------|---|---|

PHP5 との互換性を保つためデフォルトでは警告になっている。
つまり `php.ini` で **assert.exception = 1** にすべし

assert.exception = 1 にして再実行

PHP7



```
$ php example.php
```

PHP Fatal error: Uncaught AssertionError:
assert(class_exists('Bug')) in /path/to/example.php:54

Stack trace:

```
#0 /path/to/example.php(54): assert(false, 'assert(class_ex...')  
#1 /path/to/example.php(74): BugRepository->__construct(Object(PDO))  
#2 {main}  
thrown in /path/to/example.php on line 54
```

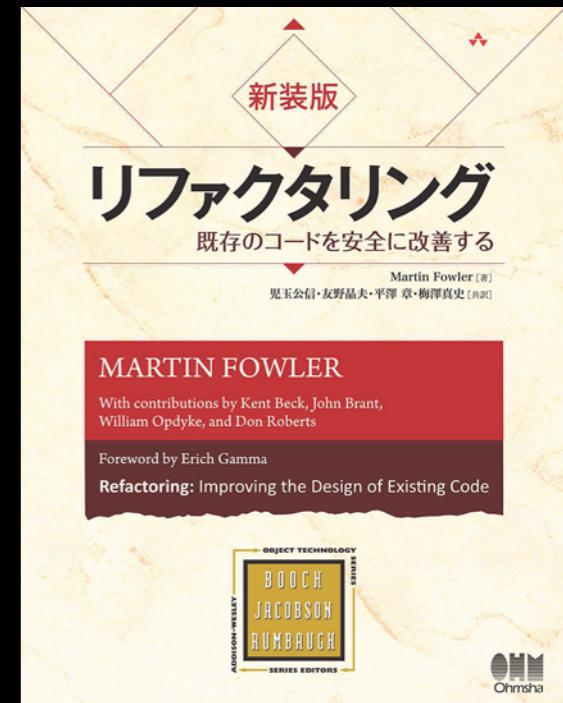
きちんと表明違反で落ちる

ようになった 😊



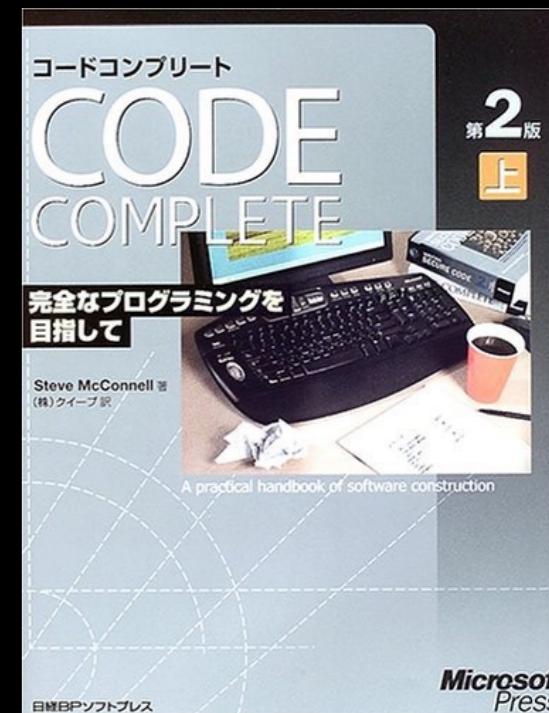
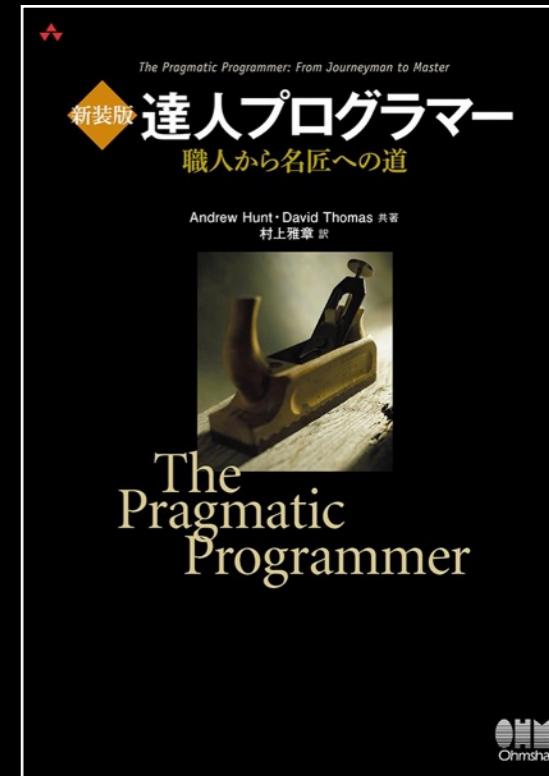
表明のメリット

- 表明はコミュニケーションとデバッグのツールとして働く
- コミュニケーションの観点では、表明を書くことによって、そのコードを書いたときの前提をコードの読み手が理解しやすくなる
- デバッグの観点では、バグをその原因に近いところで発見しやすくなる
- テストコードを書いてあれば、デバッグの支援はさほど重要ではないが、それでもコミュニケーションの観点における表明の価値は、依然として有効



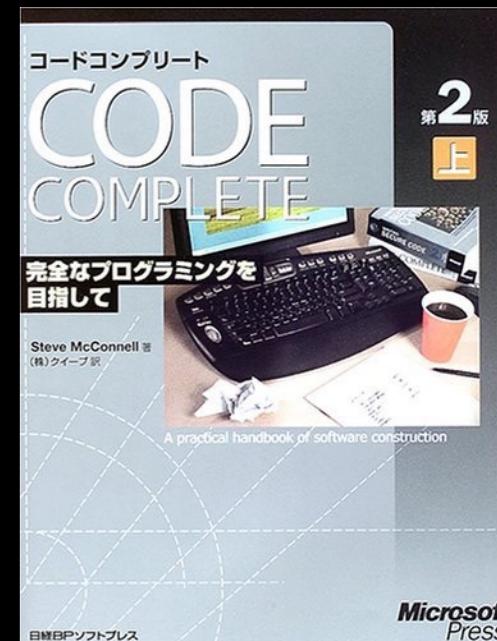
例外と表明の使い分け

- 本来のエラー処理に表明を使ってはいけません。表明は起こり得ないことをチェックするためのものです（新装版 達人プログラマー）
- 発生が予想される状況にはエラー処理コードを使用し、発生してはならない状況にはアサーションを使用する（CODE COMPLETE 第2版）



assertionを入れすぎると 低速になるのでは？

- あまりにも防御的なプログラミングも、それはそれで問題
- 引数を考えられる限りの場所で
考えられる限りの方法でチェックすれば、プログラムは肥大化
し低速になる



「防御的プログラミングに対する防御」より

そこでPHP7ですよ 😊

PHP7

`assert()` は PHP 7 で言語構造となり、`expectation` の定義を満たすようになりました。すなわち、開発環境やテスト環境では有効であるが、運用環境では除去されて、まったくコストのかからないアサーションということです。

PHP 7 における `assert()` 用の設定ディレクティブ

ディレクティブ	デフ オル ト値	取り得る値
---------	----------------	-------

<u>zend.assertions</u>	1	<ul style="list-style-type: none">• 1: コードを生成して実行する (開発モード)• 0: コードを生成するが、実行時には読み飛ばす• -1: コードを生成しない (運用モード)
------------------------	---	---

php.ini の `zend.assertions` で表明のオン/オフを制御できる

```
class BugRepository
{
    public function __construct(PDO $pdo)
    {
        assert(class_exists('Bug'));
        $this->pdo = $pdo;
    }
}
```

```
class BugRepository
{
    public function __construct(PDO $pdo)
    {
        $this->pdo = $pdo;
    }
}
```

php.ini に zend.assertions = -1 と設定すれば表明を除去できる

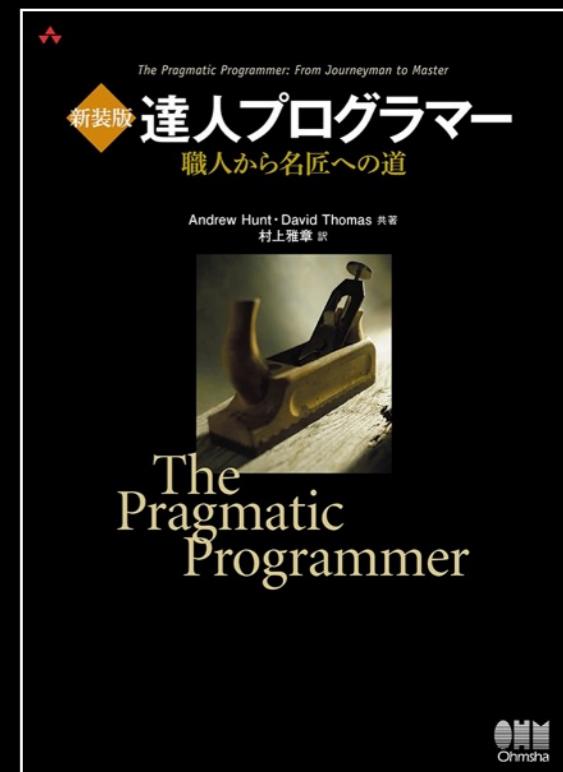
オフにされることを意識して書く

表明に引き渡す条件には副作用があつてはいけません。コンパイル時に表明がオフにされる場合もあるという点を忘れてはいけません。

つまり `assert` 中には実行に必要なコードを記述してはいけないです

副作用の例: `assert(end($users));`

「ヒント33: もし起こり得ないというのであれば、表明を用いてそれを保証すること」より



現在の心配事リスト

- 💀 テーブル名やカラム名が誰かに変更された
- ✓ 🐛 Bug クラスが未定義
- 👉 😊 途中でデータベース接続エラー

「かもしれない」
例外的状況に対処
する

きのこ93: エラーを無視するな

- エラーを無視しても何も良いことは無い
 - 不安定なコード
 - セキュリティ上問題のあるコード
 - 貧弱な構造とインターフェイス
- どうする?
 - 戻り値を使う
 - 例外を使う



PDOエラー時の戻り値(false)をチェックする

BAD

```
public function findAll(int $assignedTo, Status $status)
{
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    if (($stmt = $this->pdo->prepare($sql)) !== false) {
        if ($stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT) !== false) {
            if ($stmt->bindValue(':status', $status->value(), PDO::PARAM_STR) !== false) {
                if ($stmt->execute() !== false) {
                    if (($bugs = $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug')) !== false) {
                        return $bugs;
                    }
                }
            }
        }
    }
}
return false;
}
```



mixed 戻し



波動拳

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^([a-zA-Z0-9])+$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be between 2 and 64 characters';
                        } else $msg = 'Password must be at least 6 characters';
                    } else $msg = 'Passwords do not match';
                } else $msg = 'Empty Password';
            } else $msg = 'Empty Username';
            $_SESSION['msg'] = $msg;
        }
        return register_form();
    }
}
```



戻り値(false)をチェックして早期リターン

BAD

```
public function findAll(int $assignedTo, Status $status)
{
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    if (($stmt = $this->pdo->prepare($sql)) === false) {
        $error = $this->pdo->errorInfo();
        report_error('prepare: ' . $error[2]);
        return false;
    }
    if ($stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT) === false) {
        $error = $stmt->errorInfo();
        report_error('bindValue: ' . $error[2]);
        return false;
    }
    if ($stmt->bindValue(':status', $status->value(), PDO::PARAM_STR) === false) {
        $error = $stmt->errorInfo();
        report_error('bindValue: ' . $error[2]);
        return false;
    }
    if ($stmt->execute() === false) {
        $error = $stmt->errorInfo();
        report_error('execute: ' . $error[2]);
        return false;
    }
    if (($bugs = $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug')) === false) {
        $error = $stmt->errorInfo();
        report_error('fetchAll: ' . $error[2]);
        return false;
    }
    return $bugs;
}
```



mixed 戻し



注: report_error はログに出す自作関数

戻り値の弱点

戻り値は

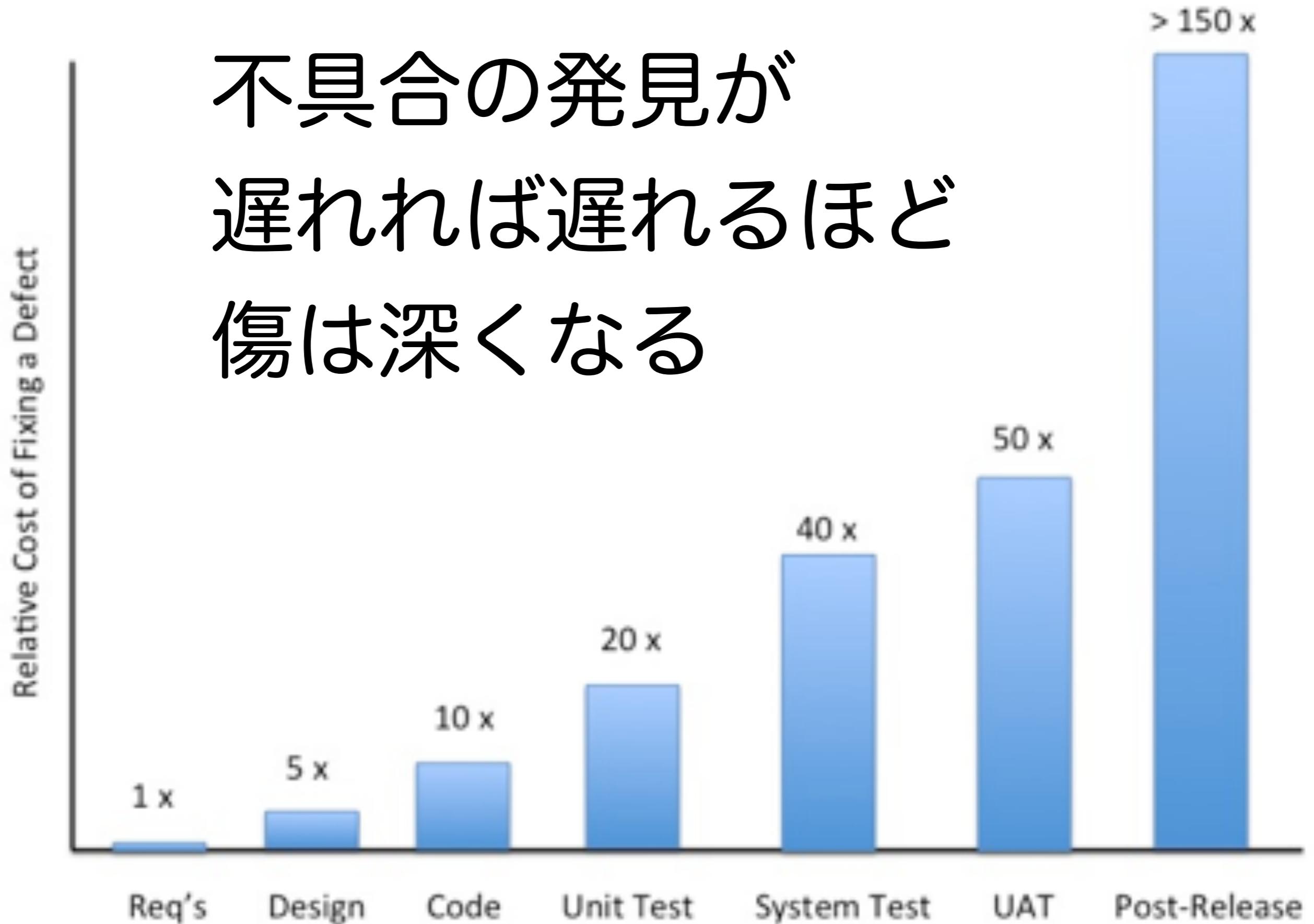
monkey emoji コードが肥大化しがち

monkey emoji 無視されやすい

monkey emoji たとえ重大な問題が潜んでいて
も、戻り値だけでは伝わりにくい

(実際 C 言語の関数の戻り値の中には「無視するのが普通」とされているものさえある)





PDO を例外モードにしよう

- **PDO::ERRMODE_EXCEPTION**

エラーコードを設定することに加え、 PDO は [PDOException](#) をスローします。エラーコードや 関連情報が、クラスのプロパティとして設定されます。この設定もまたデバッグ時に有用で、エラーが発生した時点で スクリプトの実行を停止させることによりコード内の問題点を見つけやすくなります(例外によりスクリプトが終了した際には、トランザクションは自動的に ロールバックされることを覚えておきましょう)。

このモードが有用である理由のひとつとして、伝統的な PHP 形式の警告よりも より明確にエラー処理コードが書けることがあります。例外を発生させず、データベースへのコールのたびに毎回明示的に返り値をチェックすることに比べると、コードの量やネストを減らすことができます。

PHP の例外についての詳細な情報は、 [例外](#) を参照ください。

PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION それだ!! 😊

false の代わりに PDOException が発生するようになる

```
public function findAll(int $assignedTo, Status $status)
{
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $this->pdo->prepare($sql);
    $stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT);
    $stmt->bindValue(':status', $status->value(), PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}

$pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'], [
    PDO::ATTR_EMULATE_PREPARES => false,
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
]);
$repo = new BugRepository($pdo);

$ php example.php

PHP Fatal error: Uncaught PDOException: SQLSTATE[42S22]: Column not
found: 1054 Unknown column 'date_reported' in 'field list' in /path/
to/example.php:63
```

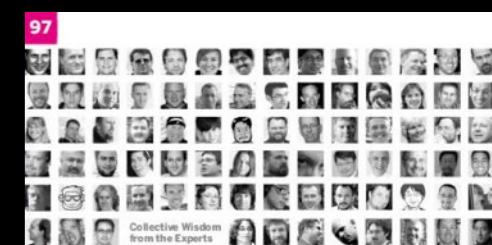
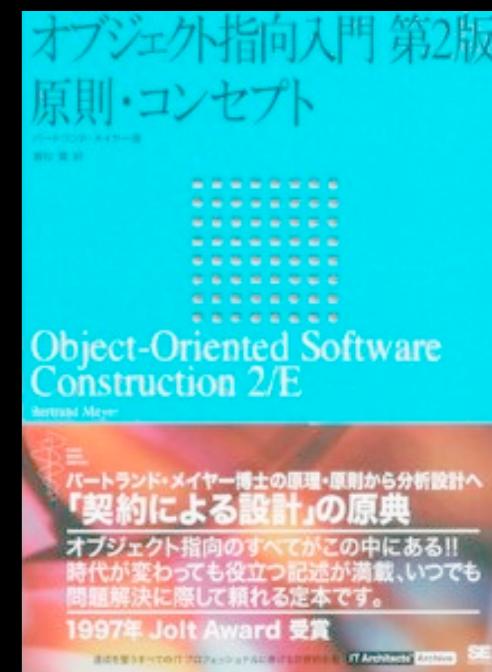
発生した暗黙の前提を assert で明示する

```
public function findAll(int $assignedTo, Status $status)
{
    assert($this->pdo->getAttribute(PDO::ATTR_ERRMODE) ===
PDO::ERRMODE_EXCEPTION);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $this->pdo->prepare($sql);
```

あらたに PDO::ERRMODE_EXCEPTION に
依存するようになったことを assert で明示する

例外の利点

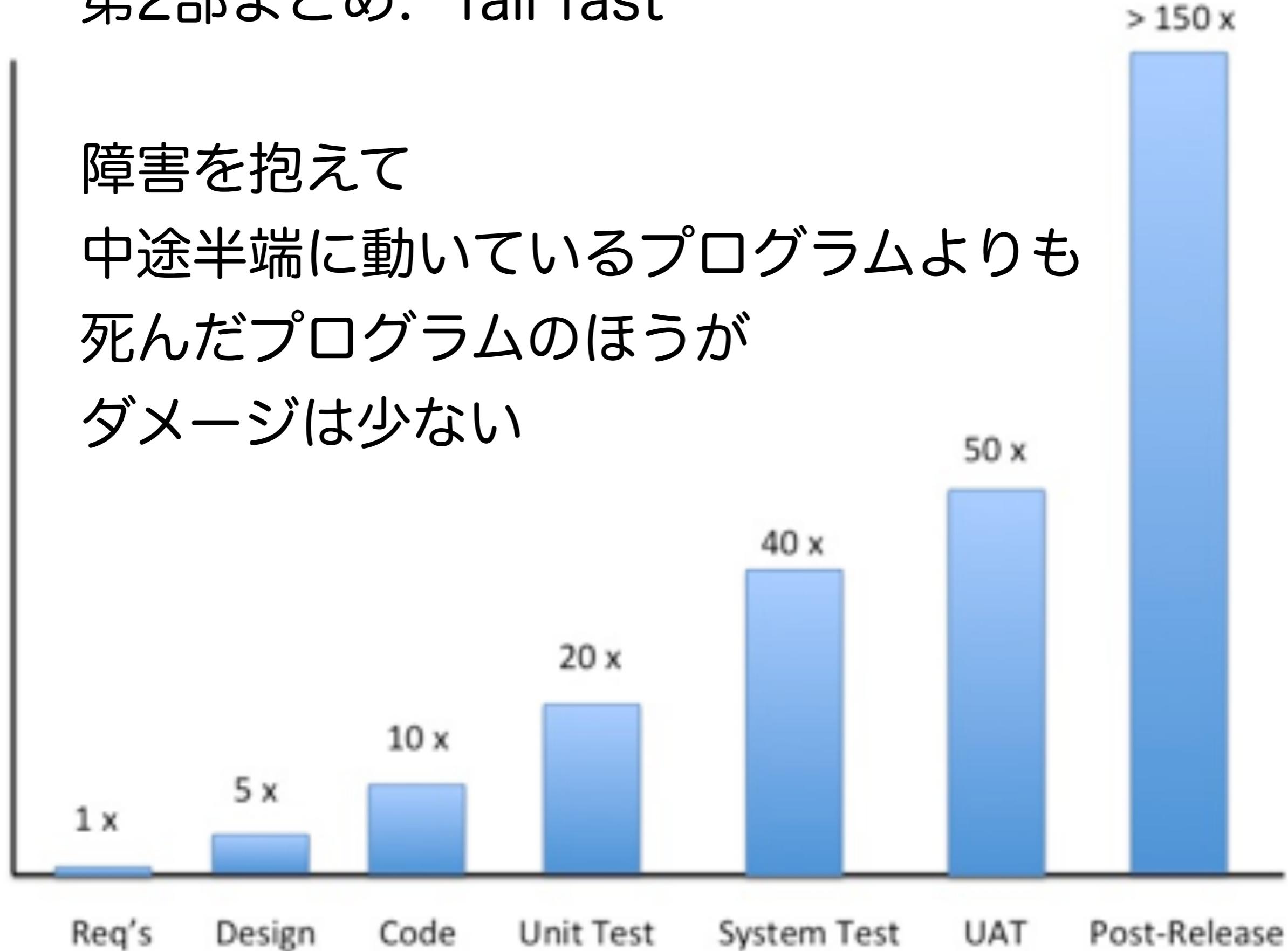
- 例外の最大の利点は、無視できない方法でエラー状態を知らせることである（オブジェクト指向入門第2版）
- 例外を故意に握りつぶすことは可能だが、そういうコードが書かれているときは、書き手の姿勢に問題があるとすぐにわかる（きのこ93）



プログラマが
知るべき97のこと
97 Things Every Programmer Should Know

第2部まとめ: “fail fast”

Relative Cost of Fixing a Defect



現在の心配事リスト

- 👉 💀 テーブル名やカラム名が誰かに変更された
- ✓ 😇 途中でデータベース接続エラー

Agenda

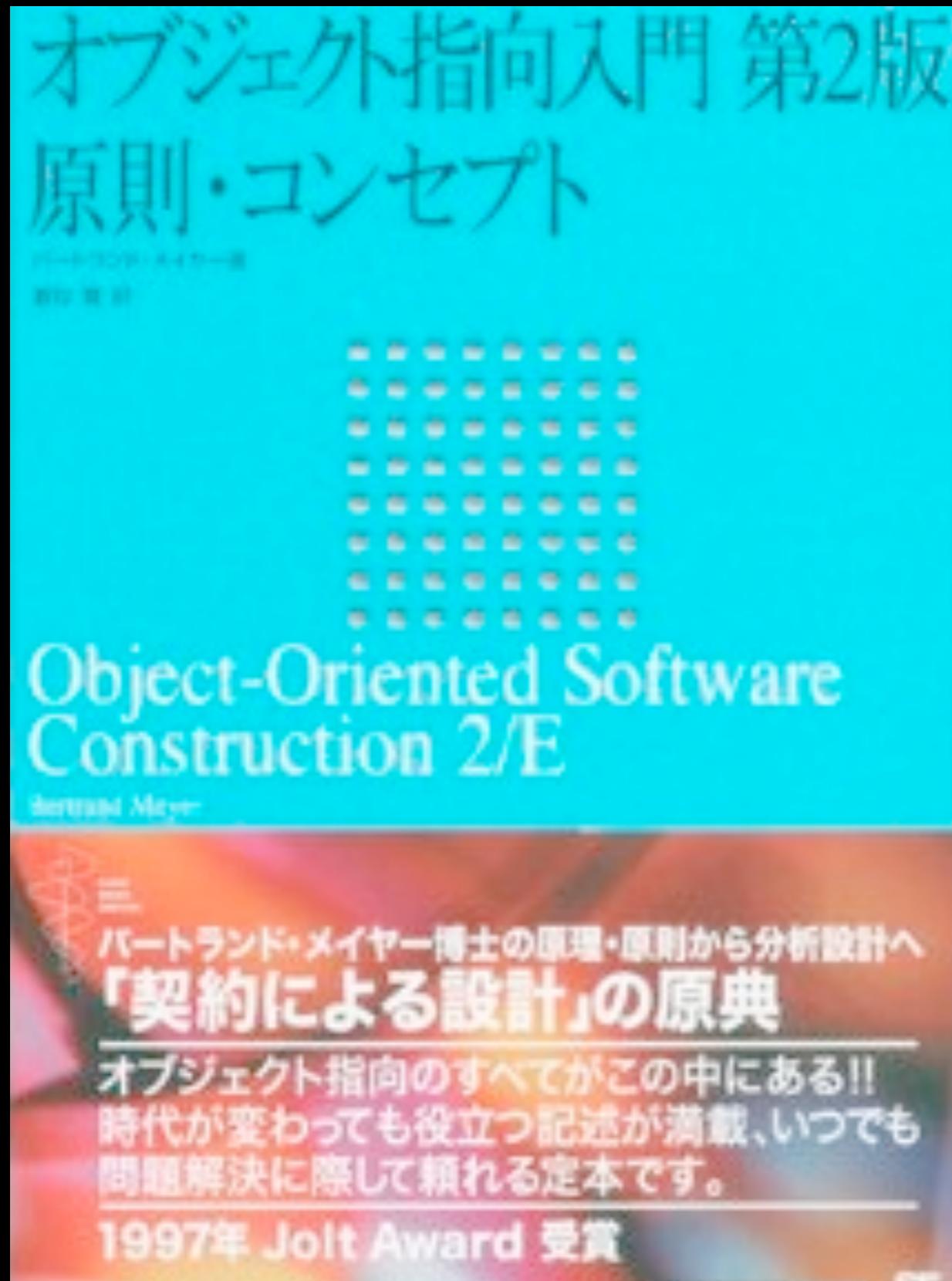
第1部: 予防的プログラミング

第2部: 攻撃的プログラミング

第3部: 契約プログラミング

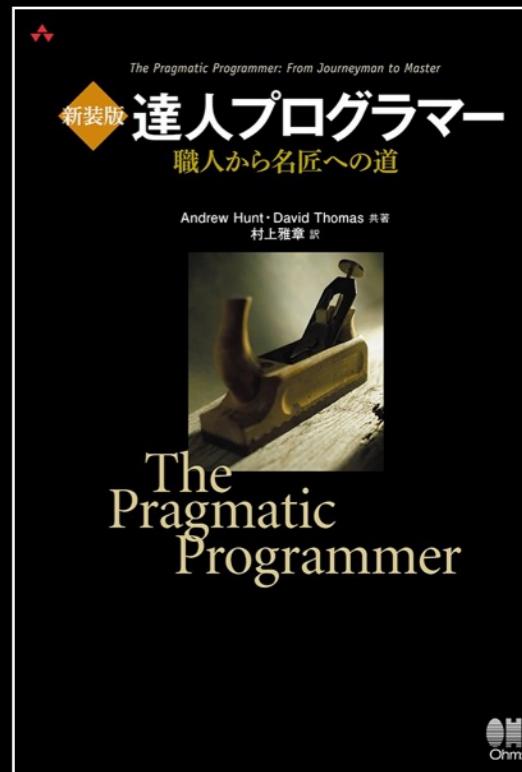
だれの責務か
ハッキリさせる
契約による設計

契約による設計



ヒント31: 契約を用いて設計を行うこと

- 契約による設計とは、ソフトウェアモジュールの権利と責任を文書化（そして承諾）し、プログラムの正しさを保証するための簡潔かつパワフルな技法です
- では、正しいプログラムとは一体何でしょうか？ これは、要求されたこと以上のことも、それ以下のことも行わないというものです



正しさの公式 (Hoare Triple)

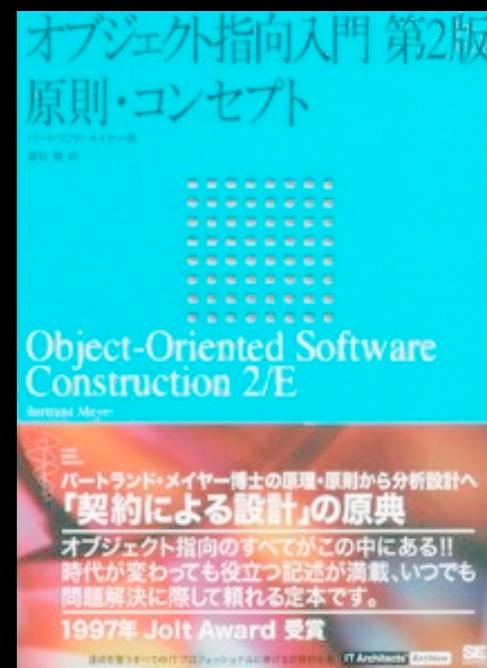
{P} A {Q}

事前条件 P が成り立つときに、プログラム A を実行するとその実行後には必ず事後条件 Q が成り立つならば、プログラム A は、事前条件 P と事後条件 Q について部分的に正当 (partially correct) である

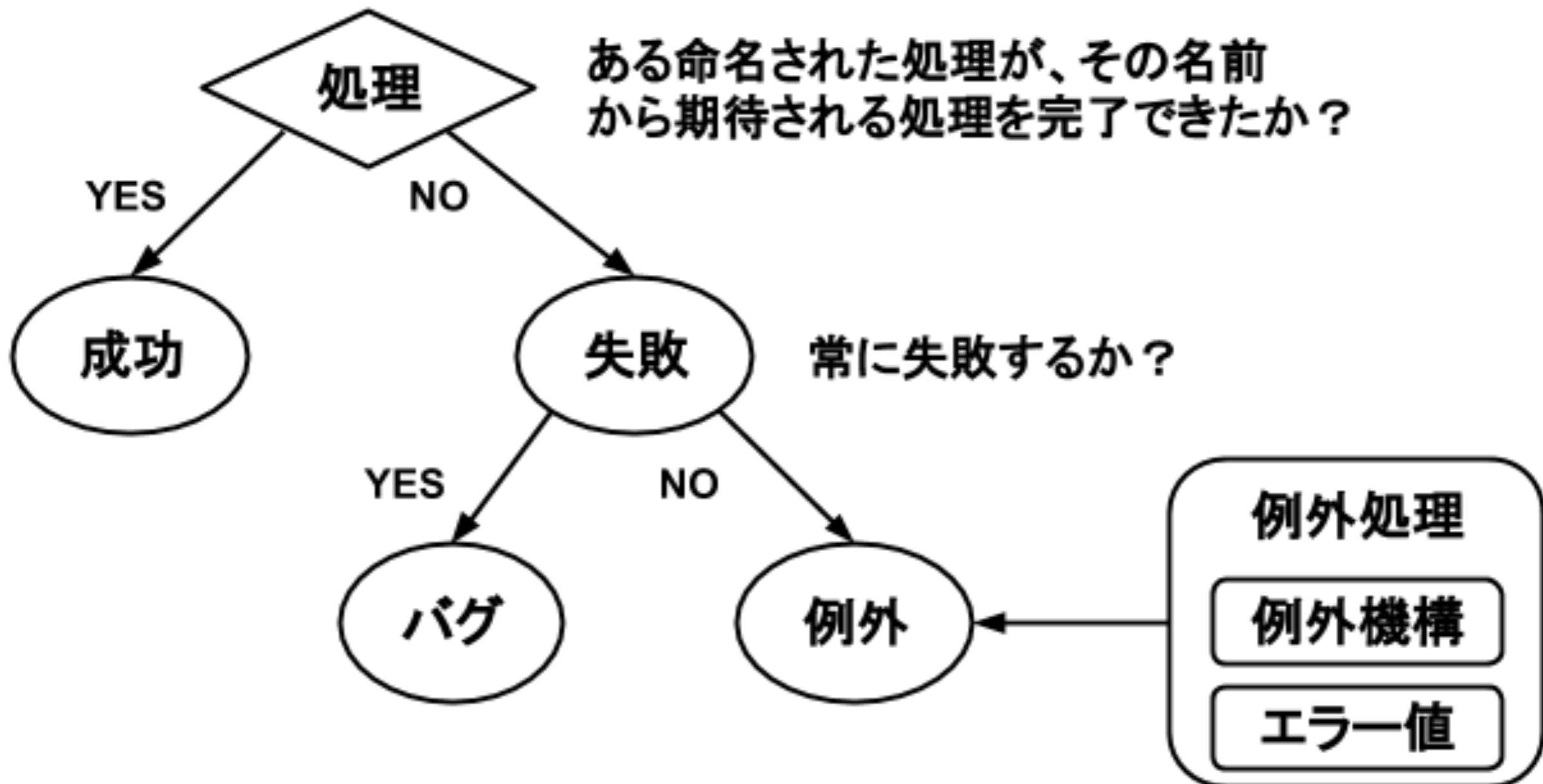
契約による設計

Design by Contract (DbC)

- もしそちらが事前条件を満たした状態で私を呼ぶと約束して下さるならば、お返しに事後条件を満たす状態を最終的に実現することをお約束します



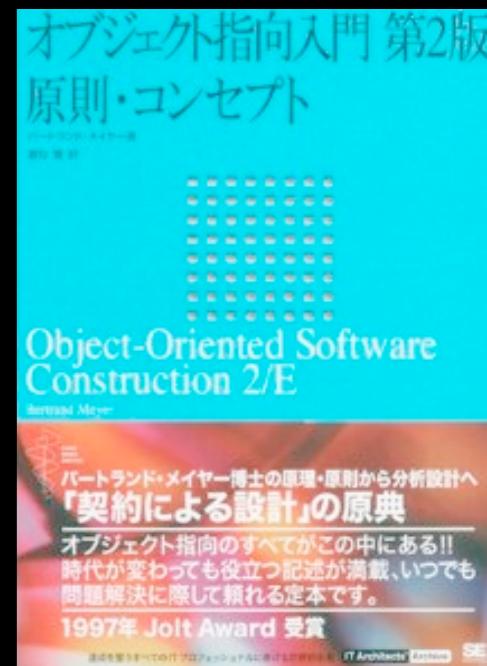
成功 / 失敗 / バグ / 例外



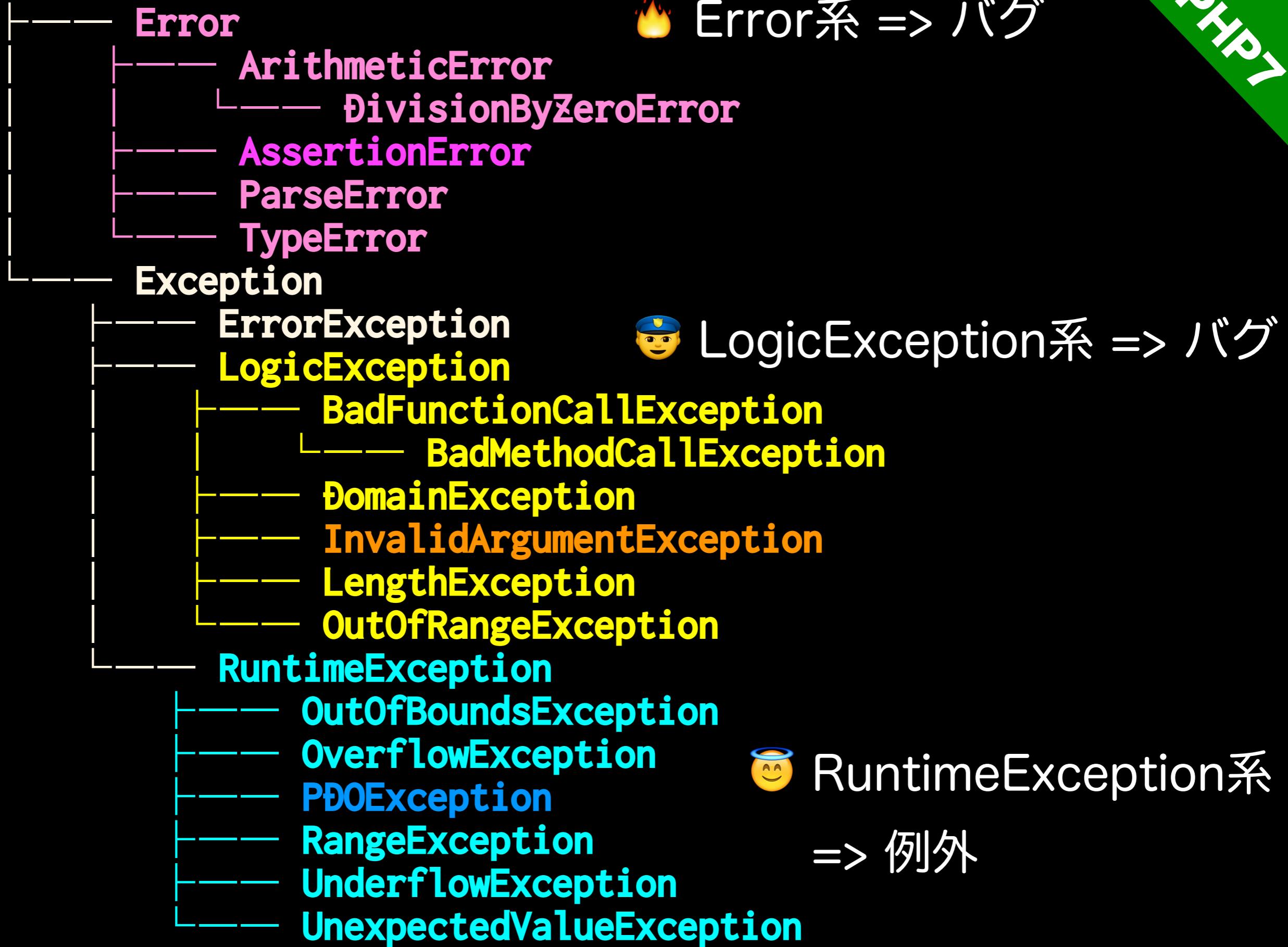
契約による設計

Design by Contract (DbC)

- 実行時の表明違反(や、バグを示すための例外)は、そのソフトウェアにバグがある証拠である
- 事前条件違反は呼び出し側にバグがある証拠である
- 事後条件違反は供給者側にバグがある証拠である



Throwable



コンストラクタの 契約

コンストラクタの事前条件を表現する

```
/*
 * 検索処理に使用する PDO インスタンスを渡し、バグリポジトリを初期化する。
 *
 * @param PDO $pdo PDO インスタンス。 PDO::ATTR_ERRMODE が PDO::ERRMODE_EXCEPTION に
 * 設定されていること
 * @throws InvalidArgumentException PDO::ATTR_ERRMODE が適切に設定されていない場合
 */
public function __construct(PDO $pdo)
{
    if ($pdo->getAttribute(PDO::ATTR_ERRMODE) === PDO::ERRMODE_EXCEPTION) {
        throw new InvalidArgumentException('requires PDO::ERRMODE_EXCEPTION');
    }
    assert(class_exists('Bug'));
    $this->pdo = $pdo;
}
```

findAll メソッド の契約

findAll メソッドの事後条件を表現する

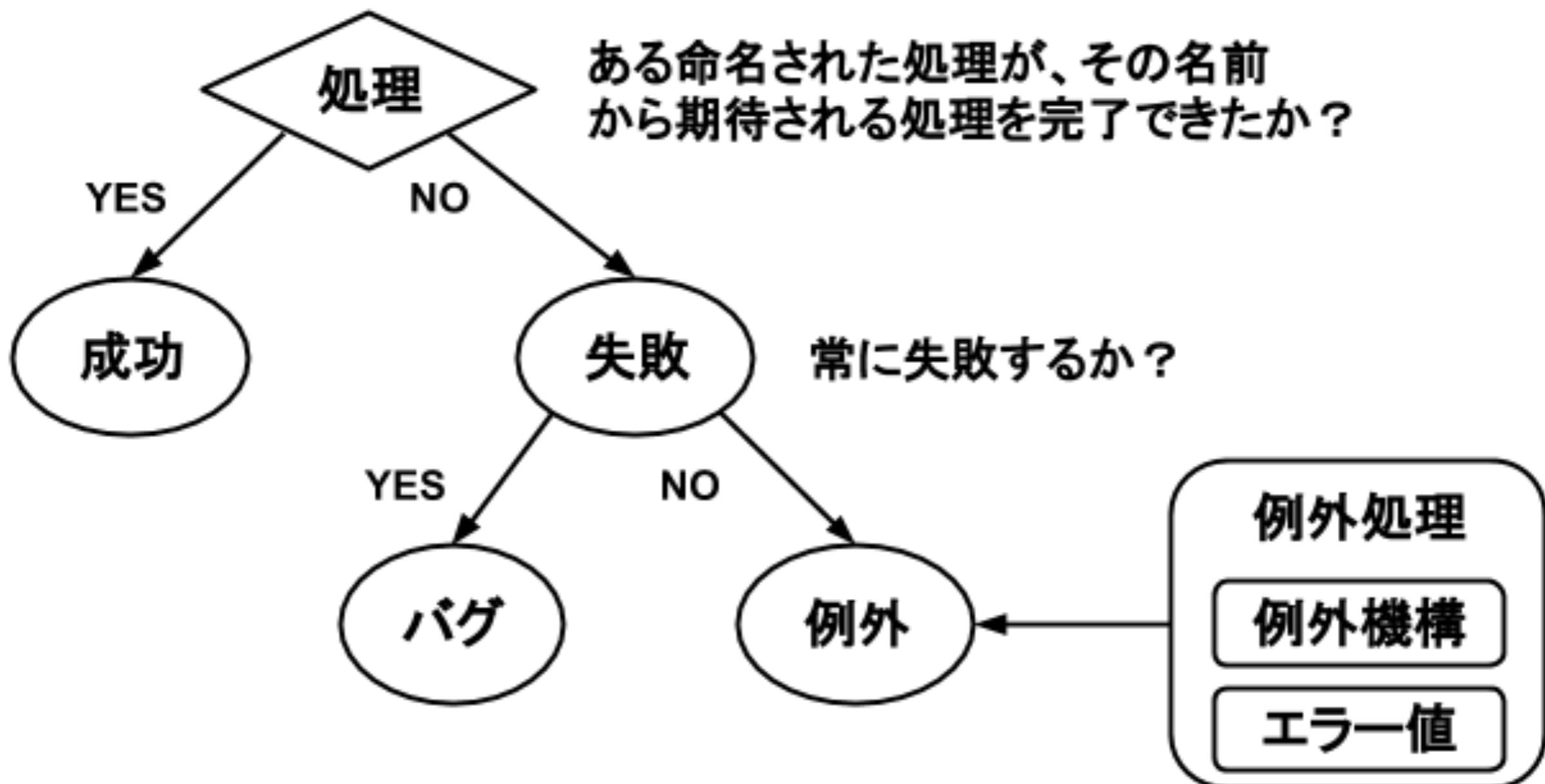
```
/*
 * 指定された担当者 ID およびステータスに合致する Bug を検索し、ヒットした全件を Bug
オブジェクトの配列として返す。
*
 * @param int $assignedTo 担当者ID
 * @param Status $status ステータス
 * @return Bug[] 条件に合致した Bug オブジェクトの配列を返す。検索に合致するものがな
い場合は空配列を返す
 * @throws LogicException カラム名違いや文法エラー等SQLのミスが存在する場合
 * @throws PDOException データベースとのやりとりに何らかの障害が発生した場合
*/
public function findAll(int $assignedTo, Status $status): array
{
    assert($this->pdo->getAttribute(PDO::ATTR_ERRMODE) === PDO::ERRMODE_EXCEPTION);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    try {
        $stmt = $this->pdo->prepare($sql);
        $stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT);
        $stmt->bindValue(':status', $status->value(), PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
    } catch (PDOException $e) {
        if ($this->isGrammaticalError($e->getCode())) {
            throw new LogicException($e->getMessage(), $e->errorInfo[1], $e);
        }
        throw $e;
    }
}
```

findAll メソッドの事後条件を表現する

```
} catch (PDOException $e) {
    if ($this->isGrammaticalError($e->getCode())) {
        throw new LogicException($e->getMessage(), $e->errorInfo[1], $e);
    }
    throw $e;
}
```

SQLSTATE の値を調査し、 PDOException の内容が例外ではなくあきらかにバグの場合は、
バグを示す LogicException で包んで投げ直している。
その際に第3引数を忘れずに設定し、スタックトレースをつなぐ

第3部まとめ: バグと例外を区別し、さらに誰の責任かも見分けられるようにする



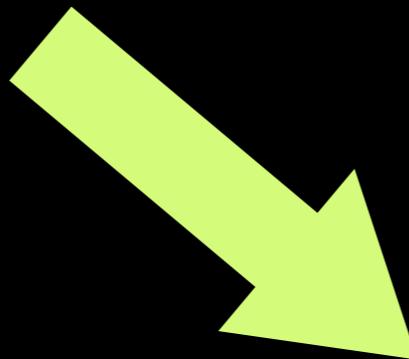
まとめ

第1部まとめ: 予防に勝る防御なし

```
public static function findAll($params)
{
    if (is_null($params)) {
        throw new InvalidArgumentException('params should not be null');
    }
    if (!is_array($params)) {
        throw new InvalidArgumentException('params should be an array');
    }
    if (count($params) !== 2) {
        throw new InvalidArgumentException('params should have exact two items');
    }
    if (!array_key_exists('assignedTo', $params) ||
        !array_key_exists('status', $params)) {
        throw new InvalidArgumentException('params should have key `assignedTo` and `status` only');
    }
    if (!is_int($params['assignedTo'])) {
        throw new InvalidArgumentException('params[`assignedTo`] should be an integer');
    }
    if (!is_string($params['status'])) {
        throw new InvalidArgumentException('params[`status`] should be a string');
    }
    if (!in_array($params['status'], ['OPEN', 'NEW', 'FIXED'], true)) {
        throw new InvalidArgumentException('params[`status`] should be in `OPEN`, `NEW`, `FIXED`');
    }

    global $CONF;
    if (!isset($CONF['dsn'])) {
        throw new LogicException('config key `dsn` not found');
    }
    if (!isset($CONF['usr'])) {
        throw new LogicException('config key `usr` not found');
    }
    if (!isset($CONF['passwd'])) {
        throw new LogicException('config key `passwd` not found');
    }
    $pdo = new PDO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                  [ PDO::ATTR_EMULATE_PREPARES => false ]);

    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    if (!class_exists('Bug')) {
        throw new LogicException('class `Bug` does not exist');
    }
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

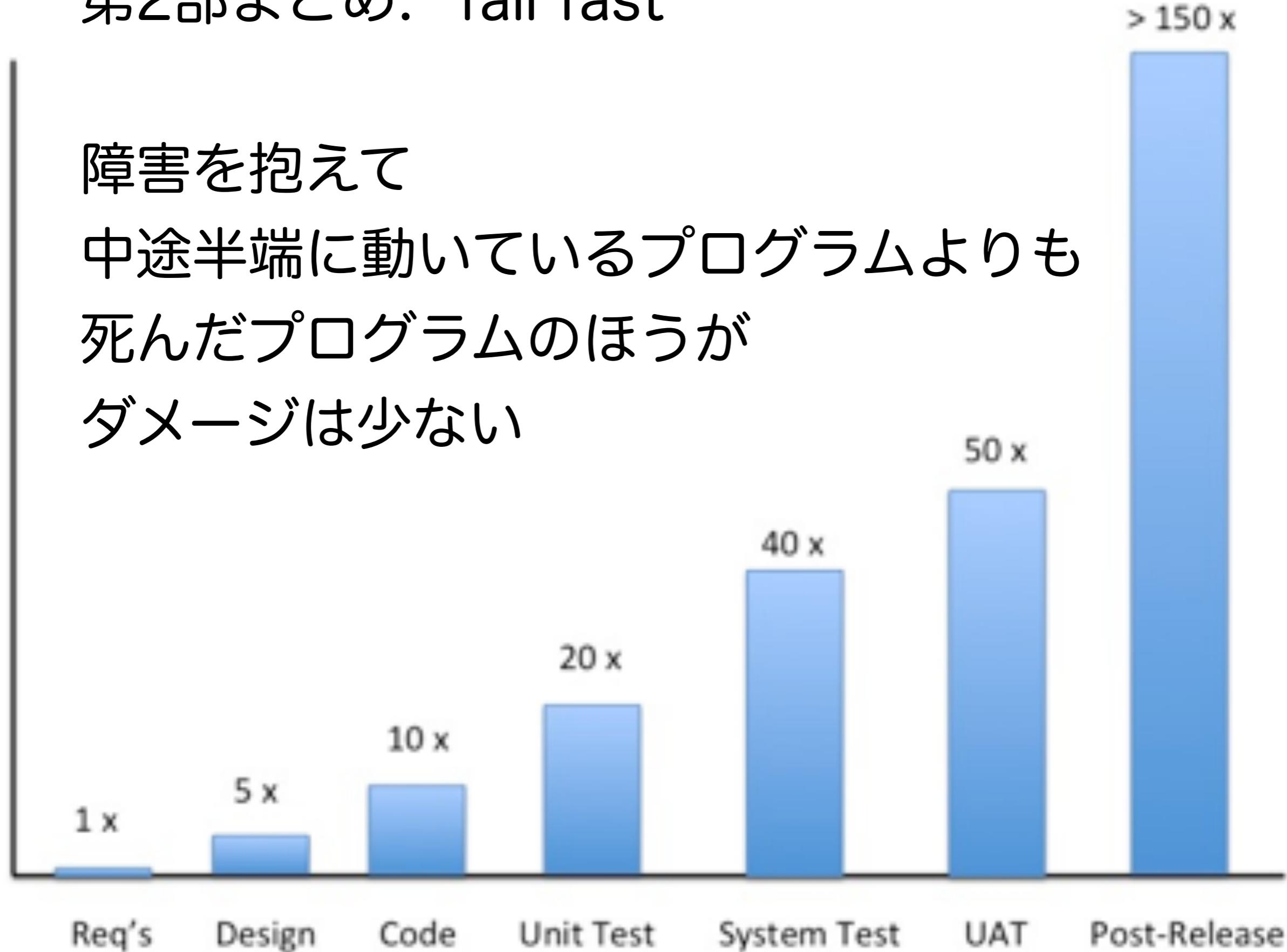


```
public function __construct(PDO $pdo)
{
    $this->pdo = $pdo;
}

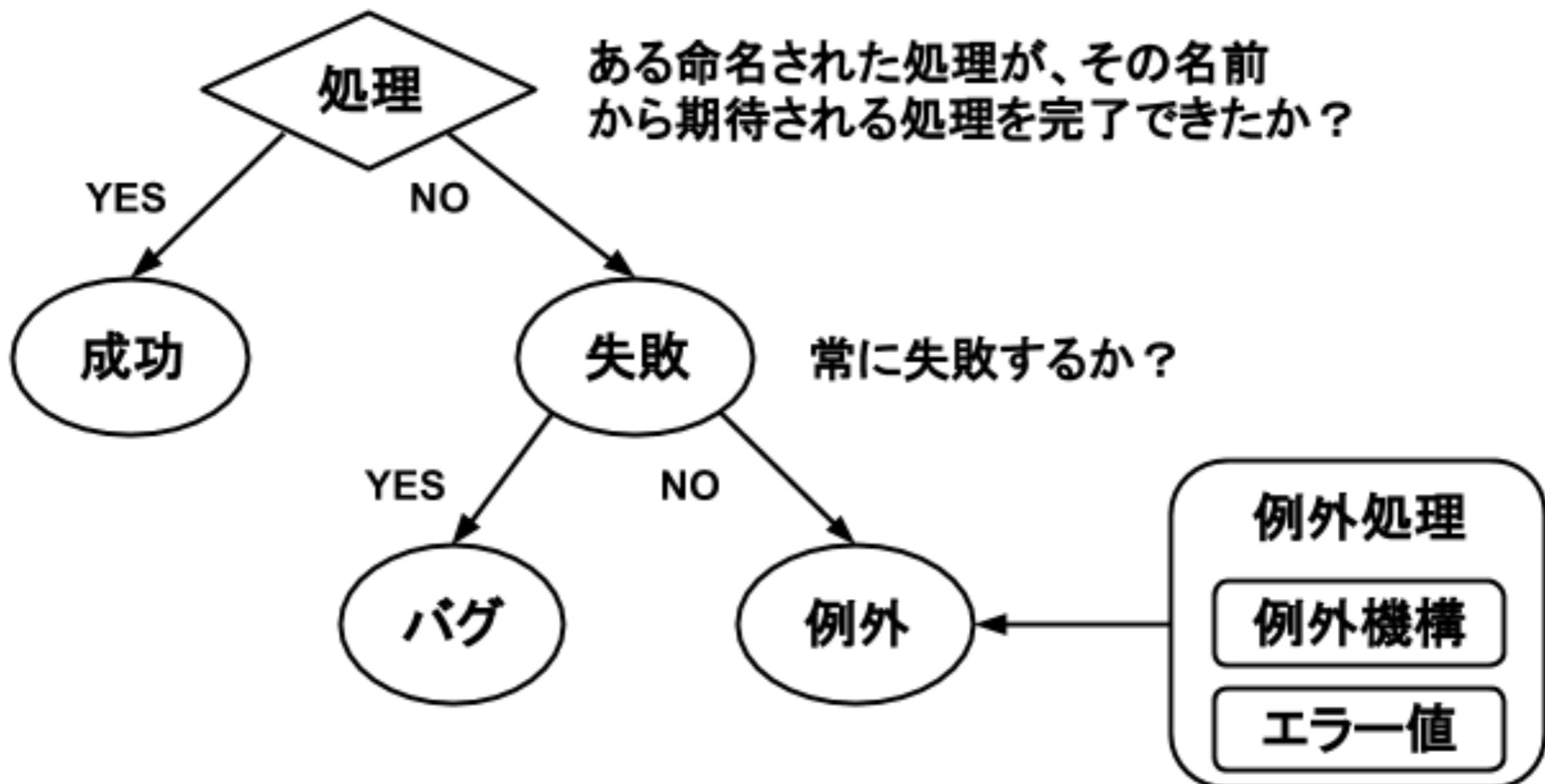
public function findAll(int $assignedTo, Status $status)
{
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE assigned_to = :assignedTo AND status = :status';
    $stmt = $this->pdo->prepare($sql);
    $stmt->bindValue(':assignedTo', $assignedTo, PDO::PARAM_INT);
    $stmt->bindValue(':status', $status->value(), PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_CLASS, 'Bug');
}
```

第2部まとめ: “fail fast”

Relative Cost of Fixing a Defect



第3部まとめ: バグと例外を区別し、さらに誰の責任かも見分けられるようにする



ご清聴ありがとうございました

“賢明なソフトウェア技術者になるための第一歩は、動くプログラムを書くことと正しいプログラムを適切に作成することの違いを認識すること”

— M.A.Jackson (1975)