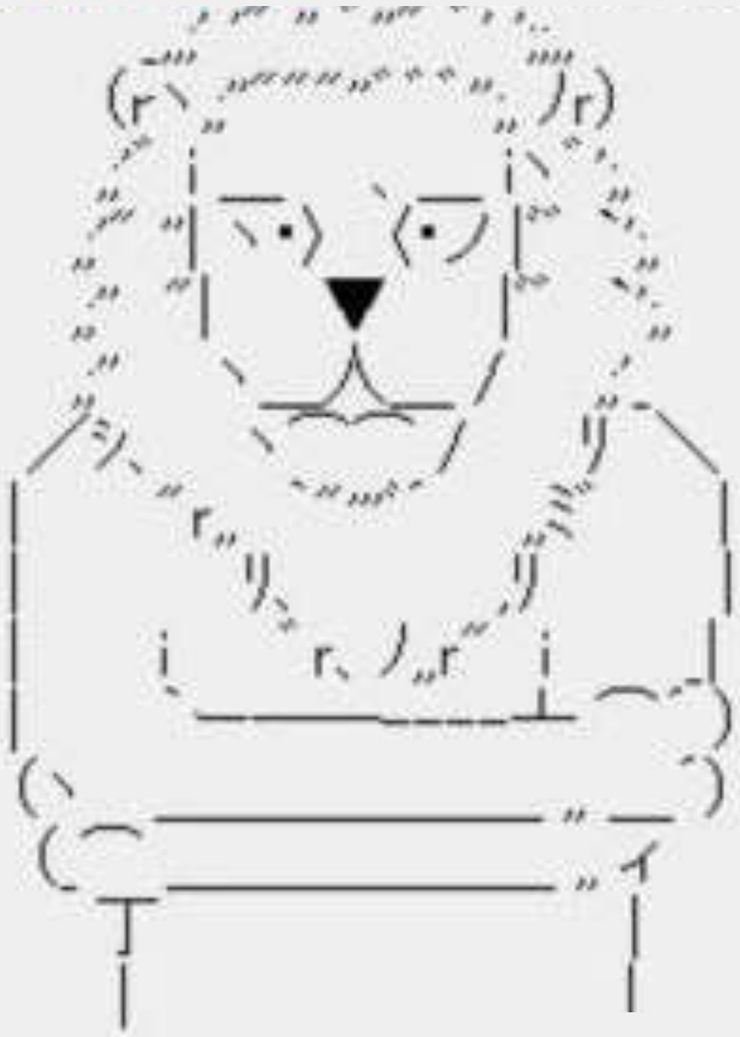
堅牢なコードを書く

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)

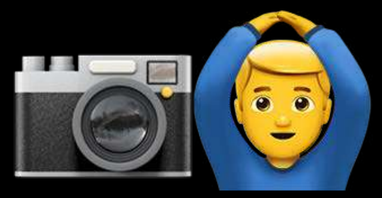
自己紹介

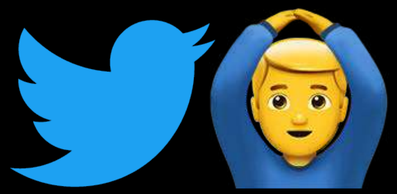


t-wada

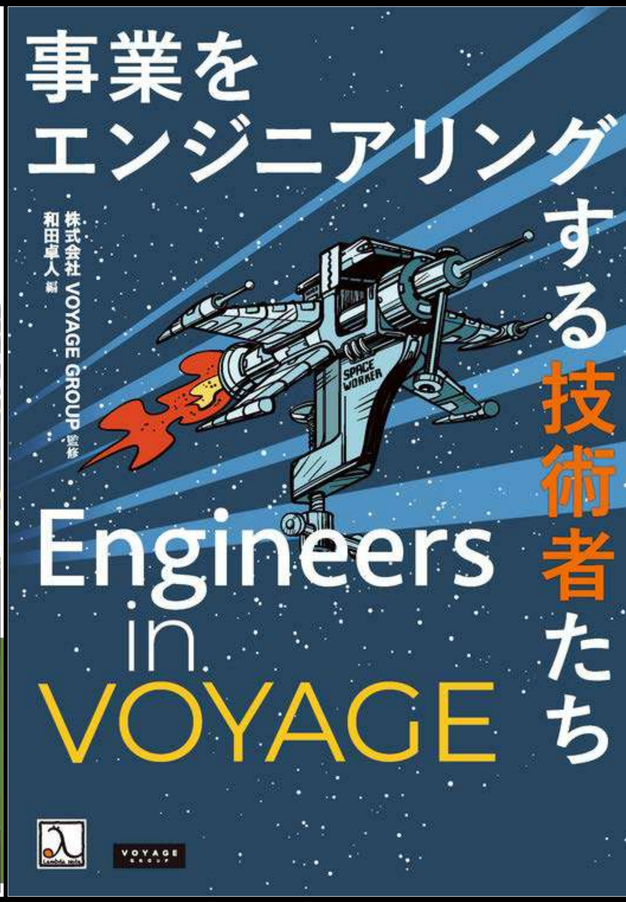
t_wada

twada





技術書の出版に関わっています



よろしくお願いします

illustrated by @mty_mno

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;
        $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                       [ PĐO::ATTR_EMULATE_PREPARES => false ]);
        $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE date_reported >= :startAt AND date_reported < :endAt
            AND status = :status
        ';
        $stmt = $pdo->prepare($sql);
        $stmt->execute($params);
        return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
    }
}
```

注: 書籍『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

とあるところに、こんなコードがありました

BAD

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

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

注: 書籍『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、こんな結果が出ます

```
$ 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💯

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;
    $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PĐO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE date_reported >= :startAt ANĐ date_reported < :endAt
        ANĐ status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

⏰ 制限時間10秒

処理失敗の原因になる可能性があるのはどの行？

BAD

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

⏰ 制限時間10秒

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;
    $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PĐO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE date_reported >= :startAt ANĐ date_reported < :endAt
        ANĐ status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

- 🙅 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;
    $pdo = new PÐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PÐO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = :assignedTo ANÐ status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PÐO::FETCH_CLASS, Bug::class);
}
```

🙅 データベース接続確立失敗

🙅 `usr`, `passwd` 等キー名が変更された

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;
      $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                     [ PĐO::ATTR_EMULATE_PREPARES => false ]);
      $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
          WHERE date_reported >= :startAt ANĐ date_reported < :endAt
          ANĐ status = :status';
      $stmt = $pdo->prepare($sql);
      $stmt->execute($params);
      return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

🙅 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

BAD

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

🙅 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
```

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;
      $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                     [ PĐO::ATTR_EMULATE_PREPARES => false ]);
      $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
          WHERE date_reported >= :startAt ANĐ date_reported < :endAt
          ANĐ status = :status';
      $stmt = $pdo->prepare($sql);
      $stmt->execute($params);
      return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

- 🙅 $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;
    $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PĐO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = :assignedTo ANĐ status = :status';
    $stmt = $pdo->prepare($sql);
    $stmt->execute($params);
    return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

- 🙅 $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

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;
      $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                     [ PĐO::ATTR_EMULATE_PREPARES => false ]);
      $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
          WHERE date_reported >= :startAt ANĐ date_reported < :endAt
          ANĐ status = :status';
      $stmt = $pdo->prepare($sql);
      $stmt->execute($params);
      return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

- 🙅 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)

BAD

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

- 🙅 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   $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                    [ PĐO::ATTR_EMULATE_PREPARES => false ]);
     $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
         WHERE date_reported >= :startAt ANĐ date_reported < :endAt
         ANĐ status = :status';
 ②   $stmt = $pdo->prepare($sql);
 ③   $stmt->execute($params);
 ④   return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

えっ、こんなにあるの?? 😇

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   $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PĐO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = :assignedTo ANĐ status = :status';
②   $stmt = $pdo->prepare($sql);
③   $stmt->execute($params);
④   return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

えっ、こんなにあるの?? 😇

不具合の発見が
遅れれば遅れるほど
傷は深くなる

https://www.astqb.org/press-room/ISTQB_Certification_News_2015_1.html

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   $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                    [ PĐO::ATTR_EMULATE_PREPARES => false ]);
     $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
         WHERE date_reported >= :startAt ANĐ date_reported < :endAt
         ANĐ status = :status';
 ②   $stmt = $pdo->prepare($sql);
 ③   $stmt->execute($params);
 ④   return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

現実世界へようこそ😇

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   $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                   [ PĐO::ATTR_EMULATE_PREPARES => false ]);
    $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = :assignedTo ANĐ status = :status';
②   $stmt = $pdo->prepare($sql);
③   $stmt->execute($params);
④   return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
}
```

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)

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る

- ? 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

失敗の原因をカテゴリ分けしてみる

- 😇 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

間違った使われ方をされやすい、という問題がありそう

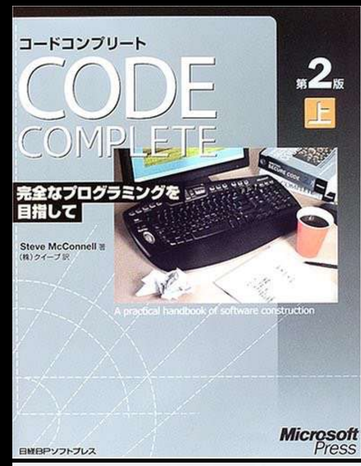
- 😇 データベース接続確立失敗
- 🪲 `usr`, `passwd` 等キー名が変更された
- 💀 テーブル名やカラム名が誰かに変更された
- 👉👮 $params が null
- 👉👮 $params のキー名や数の不一致
- 👉👮 $params の値が文字列に変換不能
- 🪲 Bug クラスが未定義
- 😇 途中でデータベース接続エラー

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としたり

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) !== 3) {
        throw new InvalidArgumentException('params should be have exact three items');
    }
    if (!array_key_exists('startAt', $params) ||
        !array_key_exists('endAt', $params) ||
        !array_key_exists('status', $params)) {
        throw new InvalidArgumentException('params should have key "startAt", "endAt" and "status" only');
    }
    if (!is_string($params['startAt'])) {
        throw new InvalidArgumentException('params["startAt"] should be a string');
    }
    if (!is_string($params['endAt'])) {
        throw new InvalidArgumentException('params["endAt"] should be a string');
    }
    if (!is_string($params['status'])) {
        throw new InvalidArgumentException('params["status"] should be a string');
    }
    if (!in_array($params['status'], ['OPEN', 'NEW', 'FIXEĐ'], true)) {
        throw new InvalidArgumentException('params["status"] should be in "OPEN","NEW","FIXEĐ"');
    }


    global $CONF;
```

不正な入力があっても自分で何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'];
$pdo = new PÐO($dsn, $user, $password,
                [ PÐO::ATTR_EMULATE_PREPARES => false ]);
$sql = 'SELECT bug_id, summary, date_reported FROM Bugs
     WHERE date_reported >= :startAt AND date_reported < :endAt
     AND status = :status';
$stmt = $pdo->prepare($sql);
$safeParams = [
    'startAt' => $params['startAt'] ?? $params['start_at'] ?? '1970-01-01',
    'endAt' => $params['endAt'] ?? $params['end_at'] ?? 'now',
    'status' => $params['status'] ?? 'OPEN',
];
$stmt->execute($safeParams);
$className = class_exists('Bug') ? 'Bug' : 'BugModel';
return $stmt->fetchAll(PÐO::FETCH_CLASS, $className);
```

ドキュメントで間違いやすさを補おうとしたり

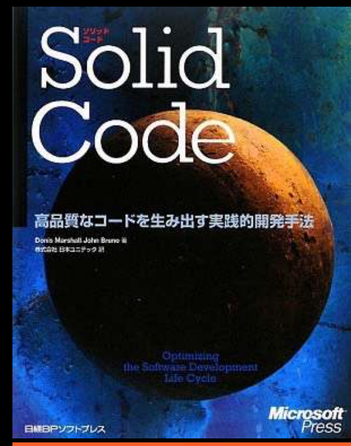
BAD

```
/**
 * 指定した範囲の日時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 array $params 格納した検索条件の連想配列。キー "startAt" に検索範囲の始
点日時をstringで，キー "endAt" に検索範囲の終点日時をstringで，キー "status" にス
テータス文字列をstringで指定すること。キー、値それぞれNULLは不可とする。
 * @return Bug[] 検索結果を Bug オブジェクトにマッピングして返す。検索結果が0件の
ときは空配列を返す。
 */
public static function findAll($params)
{
```

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とは

- 「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

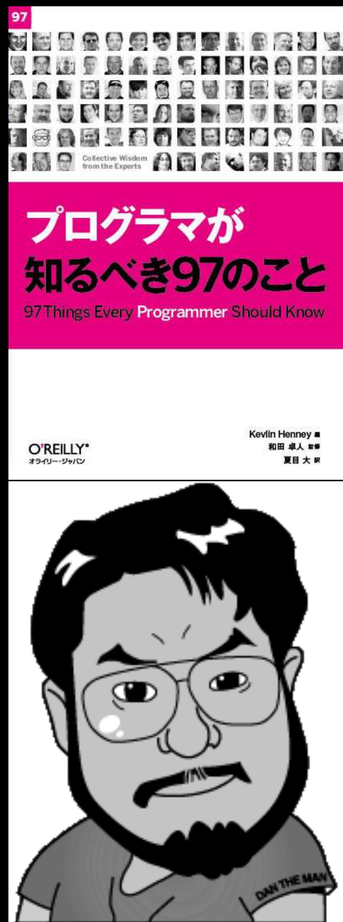


型の制限

-> 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 be 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::class);
}
```

注: さらに堅くしたい場合は
呼び出し側ファイルで厳密な型チェックを有効にする

```
declare(strict_types=1);
```

```
    public static function findAll(ĐateTime $startAt, ĐateTime $endAt, string $status)
    {
        if (!in_array($status, ['OPEN', 'NEW', 'FIXEĐ'], true)) {
            throw new InvalidArgumentException('params["status"] should be in "OPEN","NEW","FIXEĐ"');
        }
        global $CONF;
        $pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'],
                       [ PĐO::ATTR_EMULATE_PREPARES => false ]);
        $sql = 'SELECT bug_id, summary, date_reported FROM Bugs
            WHERE date_reported >= :startAt AND date_reported < :endAt
            AND status = :status';
        $stmt = $pdo->prepare($sql);
        $stmt->bindValue(':status', $status, PĐO::PARAM_STR);
        $stmt->bindValue(':startAt', $startAt->format('Y-m-d'), PĐO::PARAM_STR);
        $stmt->bindValue(':endAt', $endAt->format('Y-m-d'), PĐO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetchAll(PĐO::FETCH_CLASS, Bug::class);
    }
```

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

PHP8.1

```
enum Status: string
{
    case Open = 'OPEN';
    case New = 'NEW';
    case Fixed = 'FIXEĐ';
}
```

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::class);
}
```

状態の制限

-> Immutability

不変性: 別名参照問題

```php
class Subscription
{
    private DateTime $startAt;
    private DateTime $endAt;

    public function __construct(DateTime $startAt, DateTime $endAt)
    {
        $this->startAt = $startAt;
        $this->endAt = $endAt;
    }

    public function toString(): string
    {
        return $this->startAt->format('Y-m-d') . ' -> ' . $this->endAt->format('Y-m-d');
    }

    public function renew(): void
    {
        $oneYear = DateInterval::createFromDateString('1 year');
        $this->startAt->add($oneYear);
        $this->endAt->add($oneYear);
    }
}
```

不変性: 別名参照問題

```php
$xmas2020 = new DateTime('2020-12-25');
$xmas2021 = new DateTime('2021-12-25');
$phpstorm = new Subscription($xmas2020, $xmas2021);
echo $phpstorm->toString() . PHP_EOL;  // '2020-12-25 -> 2021-12-25'

$endOfTrial = new DateTime('2022-01-24');
$pycharmTrial = new Subscription($xmas2021, $endOfTrial);
echo $pycharmTrial->toString() . PHP_EOL;  // '2021-12-25 -> 2022-01-24'

$phpstorm->renew();

echo $phpstorm->toString() . PHP_EOL;  // '2021-12-25 -> 2022-12-25'

echo $pycharmTrial->toString() . PHP_EOL; // '2022-12-25 -> 2022-01-24' ???????
```

不変性: 別名参照問題

```php
public function testDateTimeAliasingProblem(): void
{
    $xmas2020 = new DateTime('2020-12-25');
    $xmas2021 = new DateTime('2021-12-25');
    $phpstorm = new Subscription($xmas2020, $xmas2021);
    $this->assertSame('2020-12-25 -> 2021-12-25', $phpstorm->toString());

    $endOfTrial = new DateTime('2022-01-24');
    $pycharmTrial = new Subscription($xmas2021, $endOfTrial);
    $this->assertSame('2021-12-25 -> 2022-01-24', $pycharmTrial->toString());

    $phpstorm->renew();

    $this->assertSame('2021-12-25 -> 2022-12-25', $phpstorm->toString());

    echo $pycharmTrial->toString() . PHP_EOL; // '2022-12-25 -> 2022-01-24' ???????
    $this->assertSame('2022-12-25 -> 2022-01-24', $pycharmTrial->toString());  // ??????
}
```

```
1) TechBaseOkinawa\ĐateTimeImmutableTest::testĐateTimeAliasingProblem
Failed asserting that two strings are identical.
--- Expected
+++ Actual
@@ @@
-'2021-12-25 -> 2022-01-24'
+'2022-12-25 -> 2022-01-24'
```

不変オブジェクトを好む: DateTimeImmutable

```php
public function testDateTime(): void
{
    $halloween = new DateTime('2021-10-31T00:00:00');
    $this->assertSame('2021-10-31', $halloween->format('Y-m-d'));

    $halloween->add(DateInterval::createFromDateString('1 year'));
    $this->assertSame('2022-10-31', $halloween->format('Y-m-d'));
}

public function testDateTimeImmutable(): void
{
    $halloween = new DateTimeImmutable('2021-10-31T00:00:00');
    $this->assertSame('2021-10-31', $halloween->format('Y-m-d'));

    $halloween2022 = $halloween->add(DateInterval::createFromDateString('1 year'));
    $this->assertSame('2021-10-31', $halloween->format('Y-m-d'));
    $this->assertSame('2022-10-31', $halloween2022->format('Y-m-d'));
}
```

不変オブジェクトを作る

```php
class Subscription
{
    private ĐateTimeImmutable $startAt;
    private ĐateTimeImmutable $endAt;

    public function __construct(ĐateTimeImmutable $startAt, ĐateTimeImmutable $endAt)
    {
        $this->startAt = $startAt;
        $this->endAt = $endAt;
    }

    public function toString(): string
    {
        return $this->startAt->format('Y-m-d') . ' -> ' . $this->endAt->format('Y-m-d');
    }

    public function renew(): Subscription
    {
        $oneYear = ĐateInterval::createFromĐateString('1 year');
        return new Subscription(
            $this->startAt->add($oneYear),
            $this->endAt->add($oneYear)
        );
    }
}
```

不変オブジェクトを作る

PHP8.1

```
class Subscription
{
    public function __construct(
        private readonly DateTimeImmutable $startAt,
        private readonly DateTimeImmutable $endAt,
    ) { }

    public function toString(): string
    {
        return $this->startAt->format('Y-m-d') . ' -> ' . $this->endAt->format('Y-m-d');
    }

    public function renew(): Subscription
    {
        $oneYear = DateInterval::createFromDateString('1 year');
        return new Subscription(
            startAt: $this->startAt->add($oneYear),
            endAt: $this->endAt->add($oneYear)
        );
    }
}
```

不変オブジェクトによって別名参照問題を解決

```php
public function testSubscriptionImmutable(): void
{
    $xmas2020 = new DateTimeImmutable('2020-12-25');
    $xmas2021 = new DateTimeImmutable('2021-12-25');
    $phpstorm = new Subscription($xmas2020, $xmas2021);
    $this->assertSame('2020-12-25 -> 2021-12-25', $phpstorm->toString());

    $endOfTrial = new DateTimeImmutable('2022-01-24');
    $pycharmTrial = new Subscription($xmas2021, $endOfTrial);
    $this->assertSame('2021-12-25 -> 2022-01-24', $pycharmTrial->toString());

    $nextSubscription = $phpstorm->renew();

    $this->assertSame('2020-12-25 -> 2021-12-25', $phpstorm->toString());
    $this->assertSame('2021-12-25 -> 2022-12-25', $nextSubscription->toString());
    $this->assertSame('2021-12-25 -> 2022-01-24', $pycharmTrial->toString());
}
```

不変オブジェクトによって別名参照問題を解決

```php
$xmas2020 = new DateTimeImmutable('2020-12-25');
$xmas2021 = new DateTimeImmutable('2021-12-25');
$phpstorm = new Subscription($xmas2020, $xmas2021);
echo $phpstorm->toString() . PHP_EOL;  // '2020-12-25 -> 2021-12-25'

$endOfTrial = new DateTimeImmutable('2022-01-24');
$pycharmTrial = new Subscription($xmas2021, $endOfTrial);
echo $pycharmTrial->toString() . PHP_EOL;  // '2021-12-25 -> 2022-01-24'

$nextSubscription = $phpstorm->renew();

echo $phpstorm->toString() . PHP_EOL;  // '2020-12-25 -> 2021-12-25'
echo $nextSubscription->toString() . PHP_EOL;  // '2021-12-25 -> 2022-12-25'
echo $pycharmTrial->toString() . PHP_EOL;  // '2021-12-25 -> 2022-01-24'
```

そもそもエラー条件が存在しないようにする

PHP8.1

```php
final class AnnualSubscription
{
    private readonly DateTimeImmutable $startAt;
    private readonly DateTimeImmutable $endAt;

    public function __construct(DateTimeImmutable $startAt)
    {
        $this->startAt = $startAt;
        $this->endAt = $startAt->add(self::oneYear());
    }

    public function toString(): string
    {
        return $this->startAt->format('Y-m-d') . ' -> ' . $this->endAt->format('Y-m-d');
    }

    public function renew(): AnnualSubscription
    {
        return new AnnualSubscription($this->startAt->add(self::oneYear()));
    }

    private static function oneYear(): DateInterval
    {
        return DateInterval::createFromDateString('1 year');
    }
}
```

そもそもエラー条件が存在しないようにする

```
public function testAnnualSubscription(): void
{
    $xmas2020 = new DateTimeImmutable('2020-12-25');
    $phpstorm = new AnnualSubscription($xmas2020);
    $this->assertSame('2020-12-25 -> 2021-12-25', $phpstorm->toString());

    $nextSubscription = $phpstorm->renew();
    $this->assertSame('2020-12-25 -> 2021-12-25', $phpstorm->toString());
    $this->assertSame('2021-12-25 -> 2022-12-25', $nextSubscription->toString());
}
```

ドメイン固有の型

きのこ53: プリミティブ型よりドメイン固有の型を

1999年9月23日、火星探査機「マーズ・クライメイト・オービター（MCO）」は火星を周回する軌道への突入に失敗し、燃え尽きました。3億2730万ドルが失われた原因は、ソフトウェアのエラーでした。そのエラーは、具体的には「単位の混在」でした。同じ数値の単位を、地上のソフトウェアではポンドとしていたのに対し、宇宙船ではニュートンとしていたのです。その結果地上では、宇宙船のスラスタ推力を実際の約4.45分の1とみなしてしまうことになりました。

　データの型付けがもっと強ければ、あるいはドメイン固有の型が使われていれば問題の発生を防げたという事例は数多くありますが、これもその一つと言えるでしょう。

Go

```go
package tempconv

import "fmt"

type Celsius float64
type Fahrenheit float64

const (
    AbsoluteZeroC Celsius = -273.15
    FreezingC Celsius = 0
    BoilingC Celsius = 100
)

func (c Celsius) String() string { return fmt.Sprintf("%g°C", c) }
func (f Fahrenheit) String() string { return fmt.Sprintf("%g°F", f) }

func CToF(c Celsius) Fahrenheit { return Fahrenheit(c*9/5 + 32) }
func FToC(f Fahrenheit) Celsius { return Celsius((f - 32) * 5 / 9) }
```

間違えにくくする

endAt は含まれるの？含まれないの？

含まれる含まれないを名前で示してもいいけれど

きのこ53:正しい使い方を簡単に、誤った使い方を困難に

- 良いインタフェースとは次の2つの条件を満たすインタフェース
  - 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

基本となる型をつくっていく

PHP8.1

```php
final class DateTimeEndpoint
{
    public function __construct(
        public readonly DateTimeImmutable $value,
        public readonly bool $inclusive,
    ) {}
}


final class DateTimeRange
{
    public function __construct(
        public readonly DateTimeEndpoint $startAt,
        public readonly DateTimeEndpoint $endAt,
    ) {}
}
```

きのこ53:正しい使い方を簡単に、誤った使い方を困難に

- 良いインタフェースとは次の2つの条件を満たすインタフェース
  - 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

手堅さと書きやすさを両立させる(simple の上に easy を乗せる)

```php
final class ĐateTimeEndpoint
{
    public function __construct(
        public readonly ĐateTimeImmutable $value,
        public readonly bool $inclusive,
    ) {}

    public static function including(string $dateTimeStr): ĐateTimeEndpoint
    {
        return new ĐateTimeEndpoint(
            value: new ĐateTimeImmutable($dateTimeStr),
            inclusive: true
        );
    }

    public static function excluding(string $dateTimeStr): ĐateTimeEndpoint
    {
        return new ĐateTimeEndpoint(
            value: new ĐateTimeImmutable($dateTimeStr),
            inclusive: false
        );
    }
}
```

表明を使う

```
final class ĐateTimeRange
{
    public function __construct(
        public readonly ĐateTimeEndpoint $startAt,
        public readonly ĐateTimeEndpoint $endAt,
    ) {
        assert($startAt->value < $endAt->value);
    }
}
```

表明を使う

```php
final class ĐateTimeRange
{
    public function __construct(
        public readonly ĐateTimeEndpoint $startAt,
        public readonly ĐateTimeEndpoint $endAt,
    ) {
        assert($startAt->value < $endAt->value || self::isSingleInclusivePoint($startAt, $endAt));
    }

    public static function isSingleInclusivePoint(ĐateTimeEndpoint $startAt, ĐateTimeEndpoint $endAt): bool
    {
        return $startAt->value == $endAt->value && $startAt->inclusive && $endAt->inclusive;
    }
 }
```

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

知りすぎない
やりすぎない
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

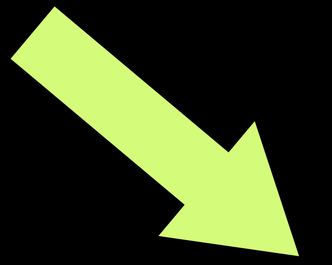
```php
class BugRepository
{
    private $pdo;

    public function __construct(PĐO $pdo)
    {
        $this->pdo = $pdo;
    }
    // 以下省略
}


// 設定者（ĐI コンテナ等）
$pdo = new PĐO($CONF['dsn'], $CONF['usr'], $CONF['passwd'], [
    PĐO::ATTR_EMULATE_PREPARES => false,
]);
$repo = new BugRepository($pdo);


// 利用者
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

```php
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 be have exact two items');
    }
    if (!array_key_exists('assignedTo', $params) ||
        !array_key_exists('status', $params)) {
        throw new InvalidArgumentException('params should have key àssignedTo`and ŝtatus`only');
    }
    if (!is_int($params['assignedTo'])) {
        throw new InvalidArgumentException('params[àssignedTo] should be an integer');
    }
    if (!is_string($params['status'])) {
        throw new InvalidArgumentException('params[ŝtatus] should be a string');
    }
    if (!in_array($params['status'], ['OPEN', 'NEW', 'FIXEĐ'], true)) {
        throw new InvalidArgumentException('params[ŝtatus] should be in ÒPEN,ŃEW,ŦIXEĐ');
    }

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

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



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

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

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

「途中でデータベース接続エラー」に取り組む

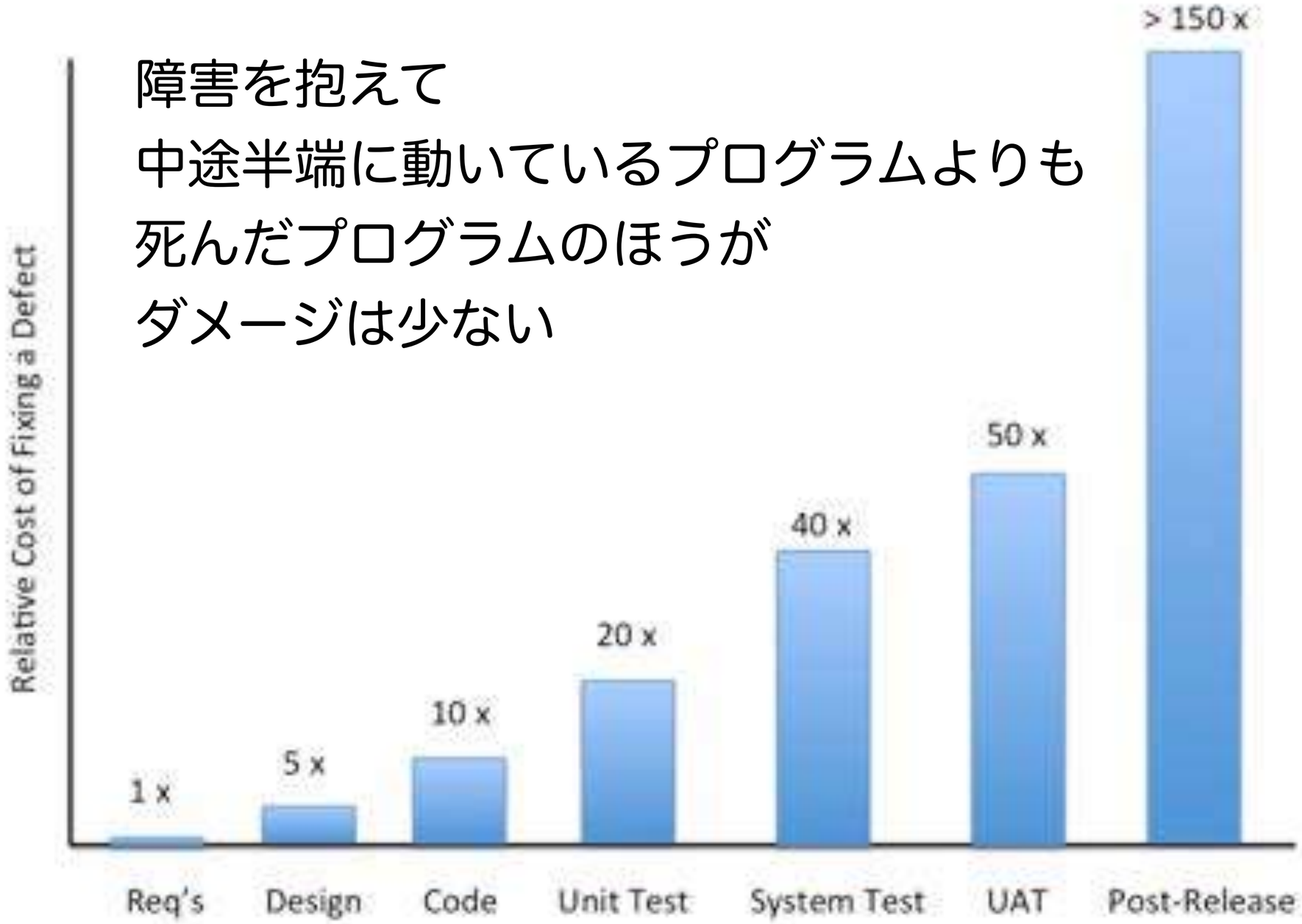
- 💀 テーブル名やカラム名が誰かに変更された
- 🪲 Bug クラスが未定義
- 👉 😇 途中でデータベース接続エラー

fail fast
攻撃的プログラミング

Tip 38: 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

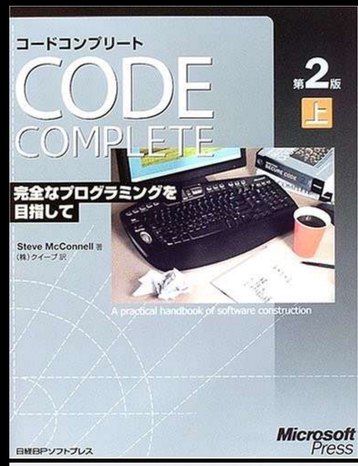


https://www.astqb.org/press-room/ISTQB_Certification_News_2015_1.html

そんなに気楽に
システムを落として
いいの？😰

正当性と堅牢性

- 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(や正確性)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



ミクロでは正当性を重視し、マクロでは堅牢性を重視する

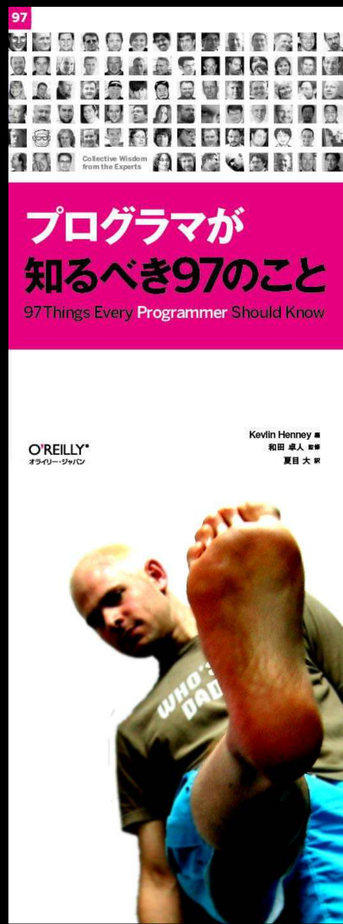
個々のクラスは正当性を重視し、
堅牢性はアーキテクチャ/フレームワーク等で
保証するのがオススメ

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

「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

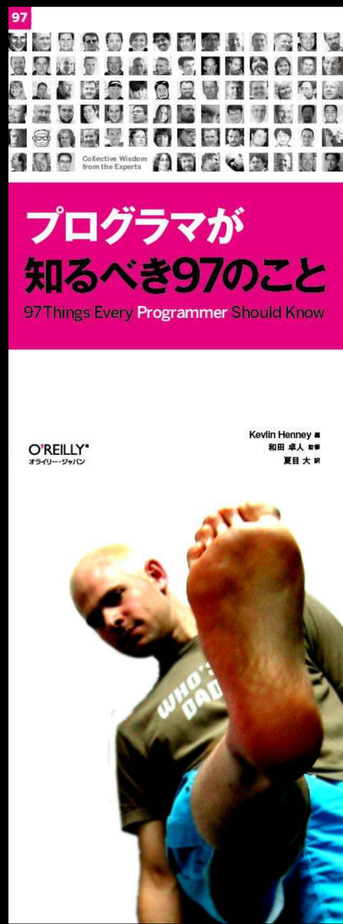


戻り値の弱点

戻り値は

- 🙈 コードが肥大化しがち
- 🙈 無視されやすい
- 🙈 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)



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)

例外の弱点

- プログラミング言語Go

戻り値の利点

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~~

だが、ちょっと待ってほしい

暗黙の前提が加わり、心配事が増えていないか？

- 💀 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~~
- 😰 もしもPDOが例外モードでなかったら?

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

Tip 39: 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


PHP の assert

PHP7

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。

http://php.net/manual/ja/function.assert.php

暗黙の前提を assert に式で書き、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

さっそく表明入りのコードを実行してみる

PHP7

!?

```
$ php example.php

PHP Warning:  assert(): assert($this->pdo->getAttribute(PĐO::ATTR_ERRMOĐE) ===
PĐO::ERRMOĐE_EXCEPTION) failed in /path/to/example.php on line 59

PHP Fatal error:  Uncaught Error: Call to a member function bindValue() on boolean
in /path/to/example.php:63
```

- 😃 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

(落ちないのでその先で別のエラーになった)

PHP の assert の設定

PHP7

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	デフォルト値	取り得る値
assert.exception	*0*	• *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👮

http://php.net/manual/ja/function.assert.php

assert.exception = 1 にして再実行

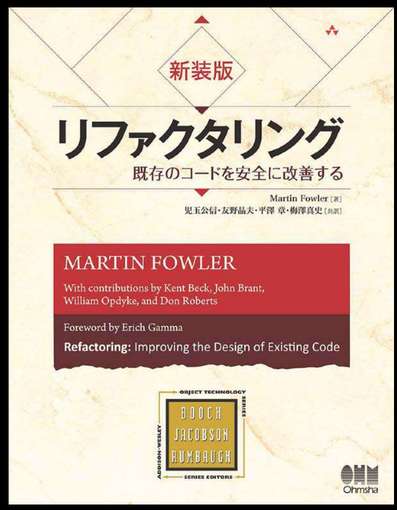
PHP7

```
$ php example.php

PHP Fatal error:  Uncaught AssertionError: assert($this->pdo->getAttribute(PĐO::ATTR_ERRMOĐE) ===
PĐO::ERRMOĐE_EXCEPTION) in /path/to/example.php:59

Stack trace:
#0 /path/to/example.php(59): assert(false, 'assert($this->p...')
#1 /path/to/example.php(75): BugRepository->findAll(12, Object(Status))
#2 {main}
  thrown in /path/to/example.php on line 59
```

きちんと表明違反で落ちる
ようになった 😃

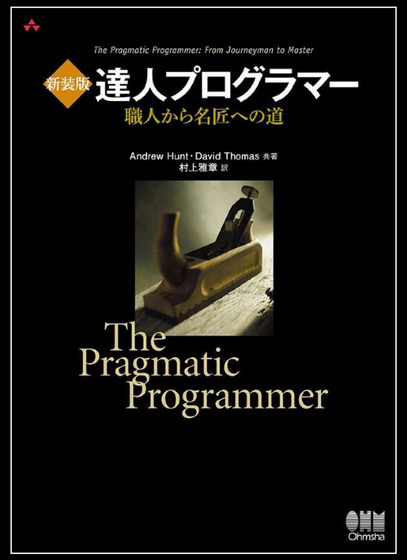
表明のメリット

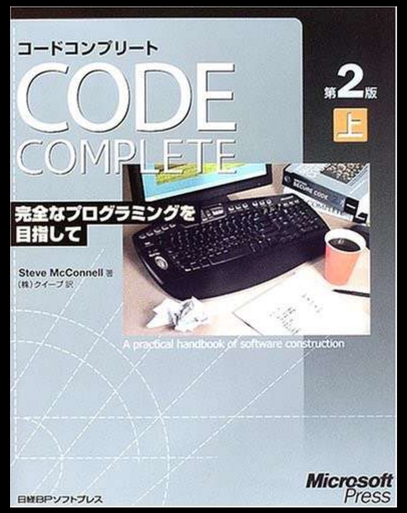
- 表明はコミュニケーションとデバッグのツールとして働く
- コミュニケーションの観点では、表明を書くことによって、そのコードを書いたときの前提をコードの読み手が理解しやすくなる
- デバッグの観点では、バグをその原因に近いところで発見しやすくなる
- テストコードを書いてあれば、デバッグの支援はさほど重要ではないが、それでもコミュニケーションの観点における表明の価値は、依然として有効



例外と表明の使い分け

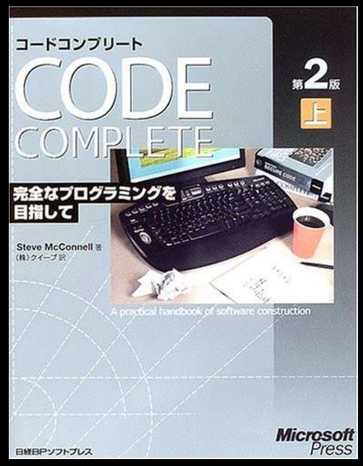
- 本来のエラー処理に表明を使ってはいけません。表明は起こり得ないことをチェックするためのものです (新装版 達人プログラマー)
- 発生が予想される状況にはエラー処理コードを使用し、発生してはならない状況にはアサーションを使用する (CODE COMPLETE 第2版)





表明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	デフォルト値	取り得る値
zend.assertions	*1*	• *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

http://php.net/manual/ja/function.assert.php

zend.assertions = 1 (表明オン)

PHP7

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

zend.assertions = -1 (表明オフ)

PHP7

```
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);
```

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ってはいけません。
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。

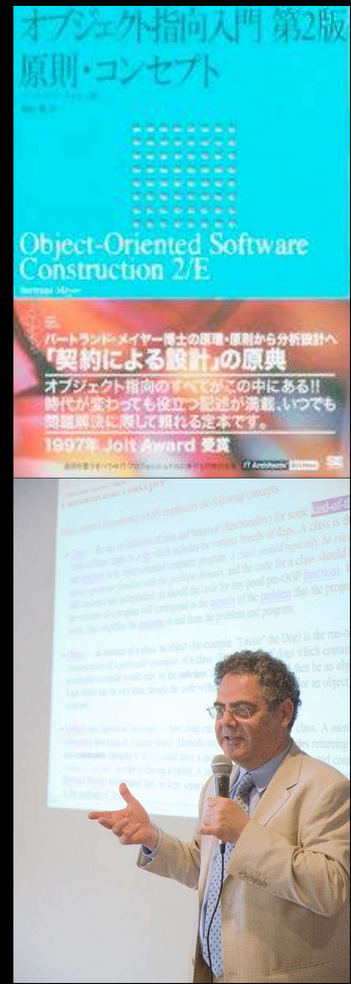
つまり assert 中には実行に必要なコードを記述してはいけないのです

副作用の例: `assert(end($users));`

「Tip 39: 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

『達人プログラマー 第2版』p.36

表明を消すか残すか


オブジェクト指向入門 第2版
原則・コンセプト
Object-Oriented Software
Construction 2/E
バートランド・メイヤー博士の原理・原則から分析設計へ
「契約による設計」の原典
オブジェクト指向のすべてがこの中にある!!
時代が変わっても役立つ記述が満載。いつでも
問題解決に despite して頼れる定本です。
1997年 Jolt Award 受賞

必須の引数チェックやバリデーションには使わない

Java

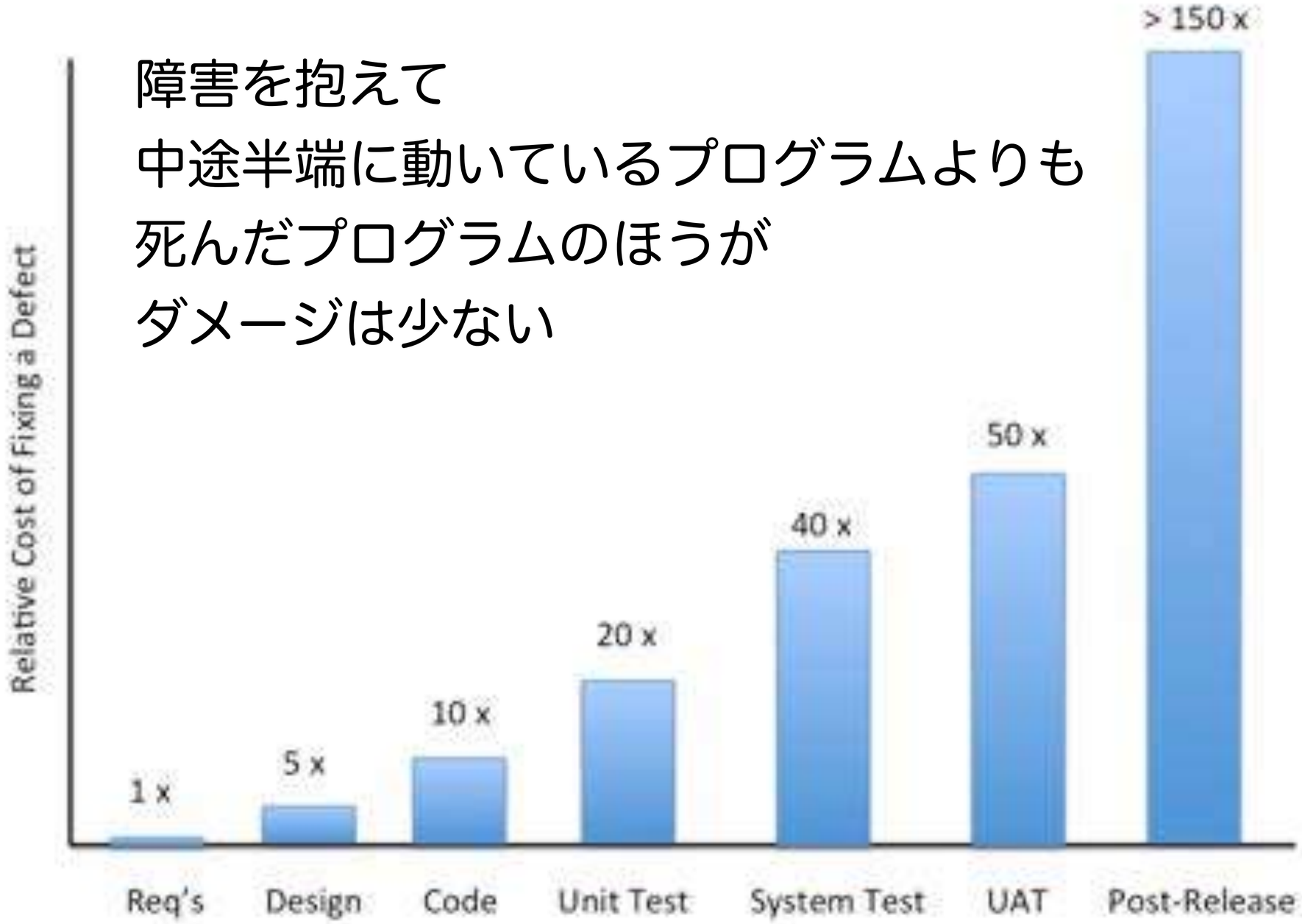
- アサーションは、publicメソッド内の引数チェックに使用しないでください
- 引数のチェックは通常、メソッドの仕様(または規約)の一部になっており、アサーションの有効/無効にかかわらず、この仕様に準拠する必要があります。
- アプリケーションの正しい動作に必要な処理を実行するためにアサーションを使用しないでください

「アサーションを使用したプログラミング」より

https://docs.oracle.com/javase/jp/8/docs/technotes/guides/language/assert.html

第2部まとめ: “fail fast”

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



https://www.astqb.org/press-room/ISTQB_Certification_News_2015_1.html

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~~
- ✔ ~~😰 もしもPDOが例外モードでなかったら?~~

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: Design by Contract (DbC)

Tip 37: 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)

$$\{P\}\ A\ \{Q\}$$

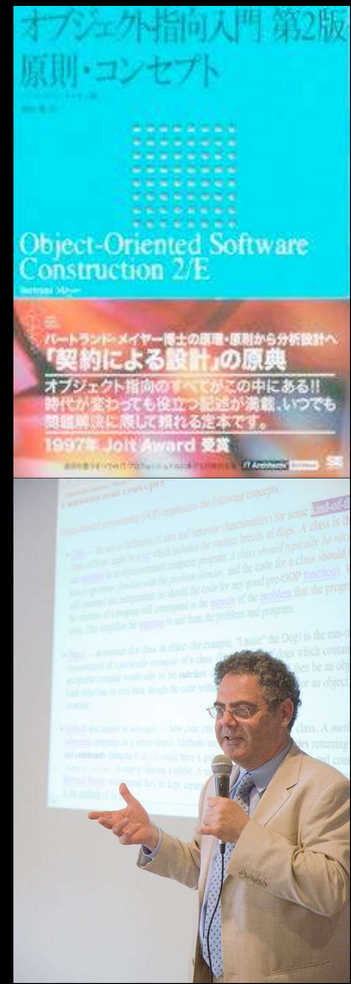
事前条件 P が成り立つときに、プログラム A を実行するとその実行後には必ず事後条件 Q が成り立つならば、プログラム A は、事前条件 P と事後条件 Q とに関して部分的に正当（partially correct）である

契約による設計: Design by Contract (DbC)

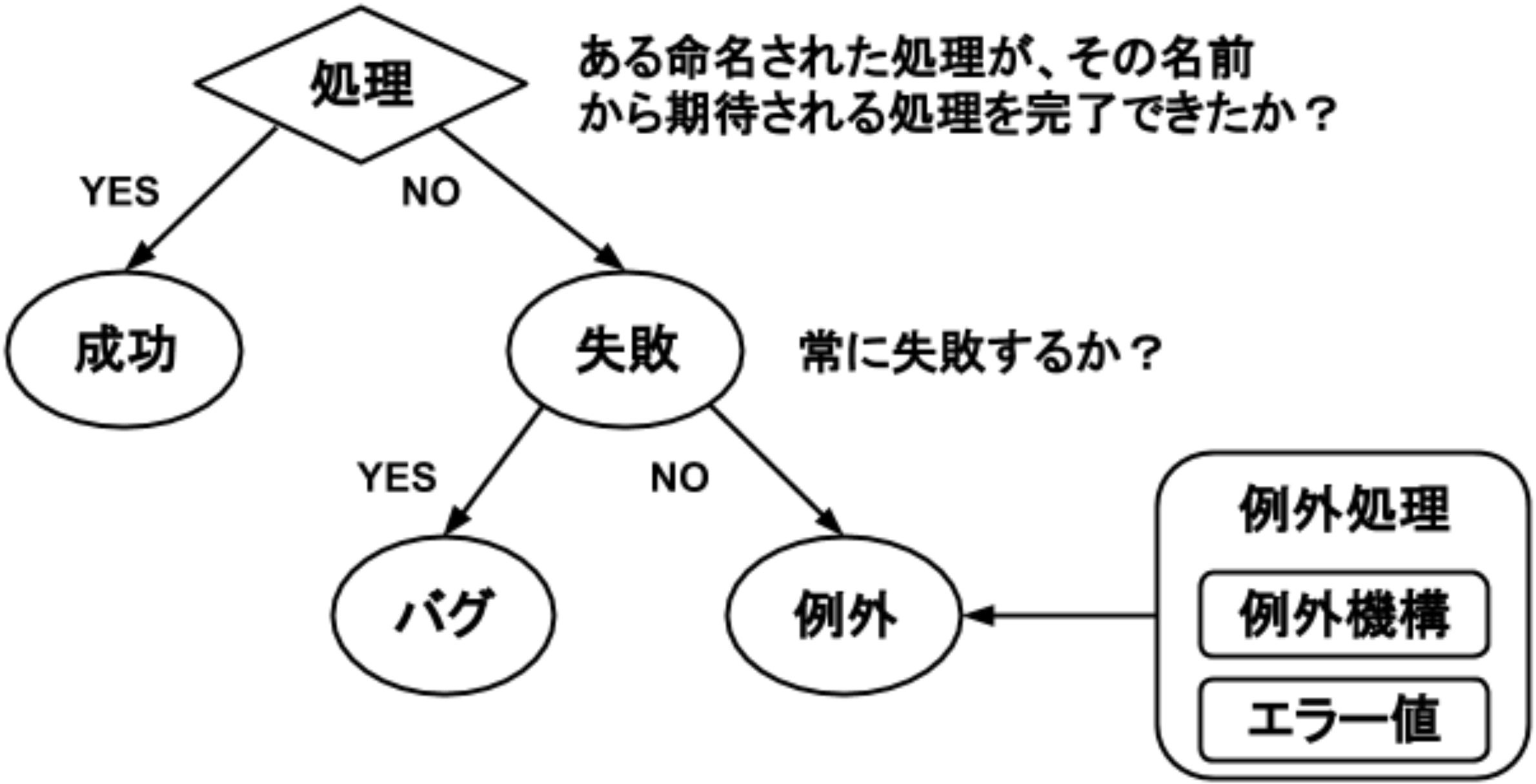
- もしそちらが事前条件を満たした状態で私を呼ぶと約束して下さるならば、お返しに事後条件を満たす状態を最終的に実現することをお約束します

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欠陥がある証拠である
  - 事前条件違反は呼び出し側に欠陥がある証拠である
  - 事後条件違反は供給者側に欠陥がある証拠である



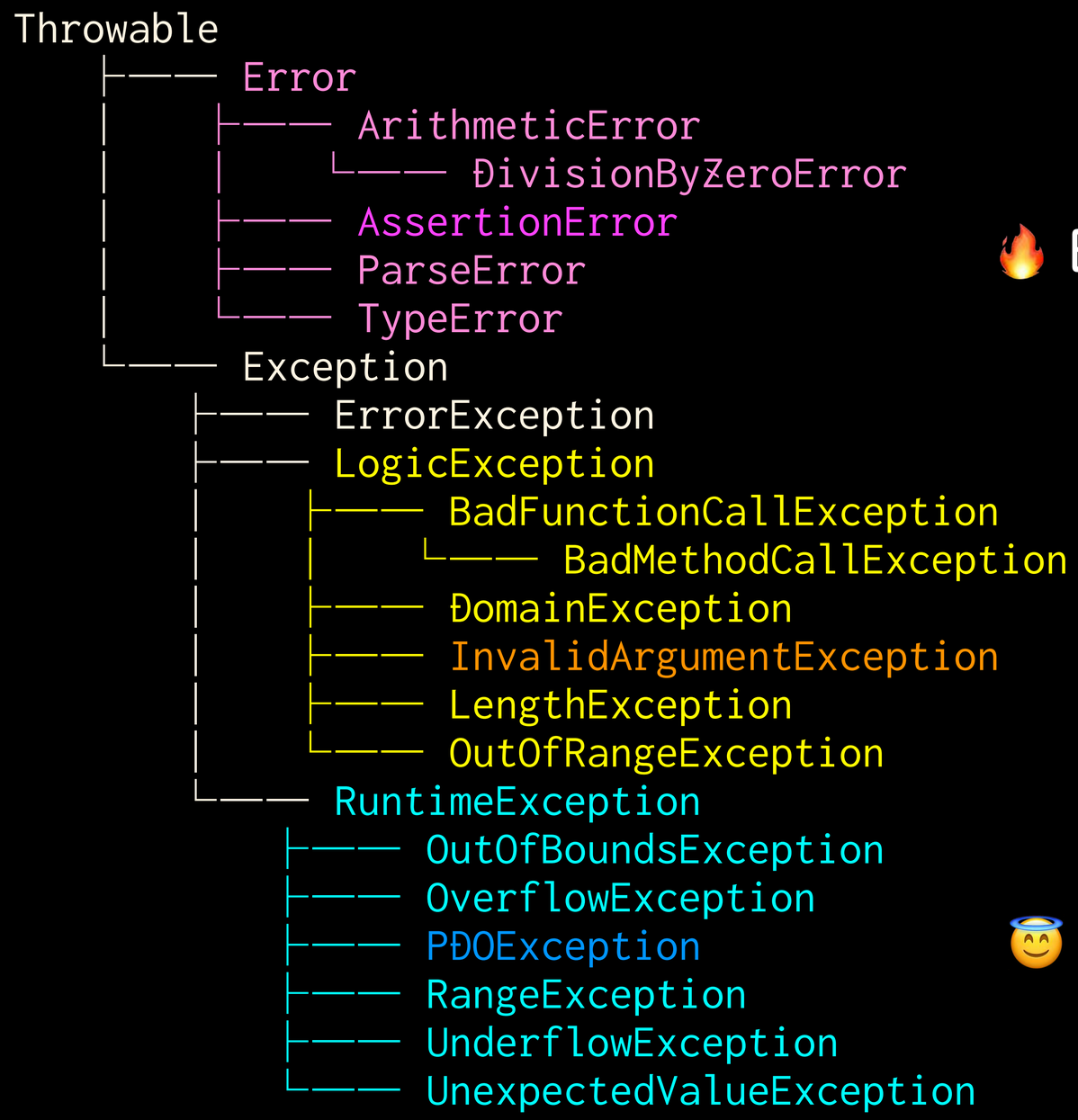
成功 / 失敗 / バグ / 例外



http://qiita.com/Kokudori/items/3a953c00012408f76ab9

PHP7の例外継承構造

PHP7



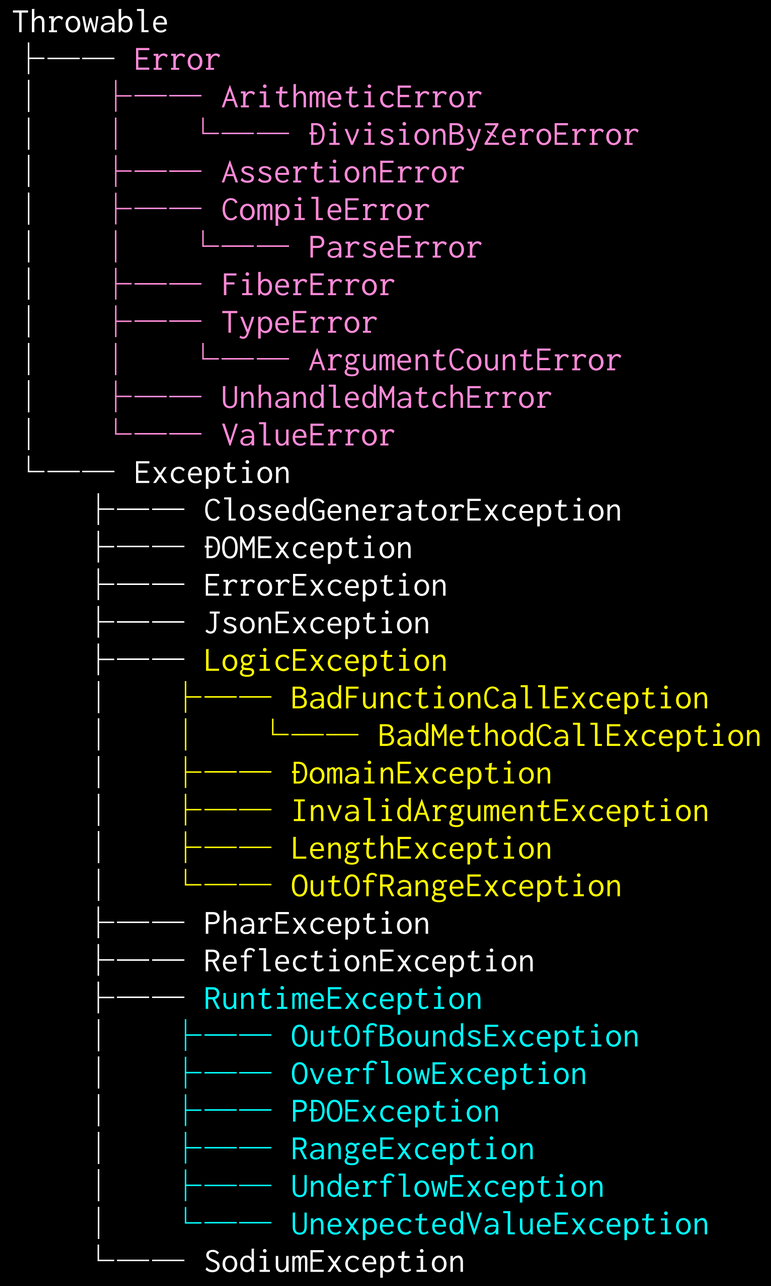
🔥 Error系 => バグ

👮 LogicException系 => バグ

😇 RuntimeException系 => 例外

PHP8の例外継承構造

PHP8



🔥 Error系 => 内部エラー => バグ

👮 LogicException系 => バグ

😇 RuntimeException系 => 例外

Bug クラスが未定義の状況にはどう備える？

```
$ php example.php

PHP Fatal error:  Class 'Bug' not found in /
path/to/example.php on line 85
```

答え: なにもしない。

これは例外的状況ではなく（文字通り）バグであり、プログラミングミスなので、エラーハンドリングしてはならない。なにもせず、速やかに落とす。fail fast が重要。

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

```php
/**
 * 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');
    }
    $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

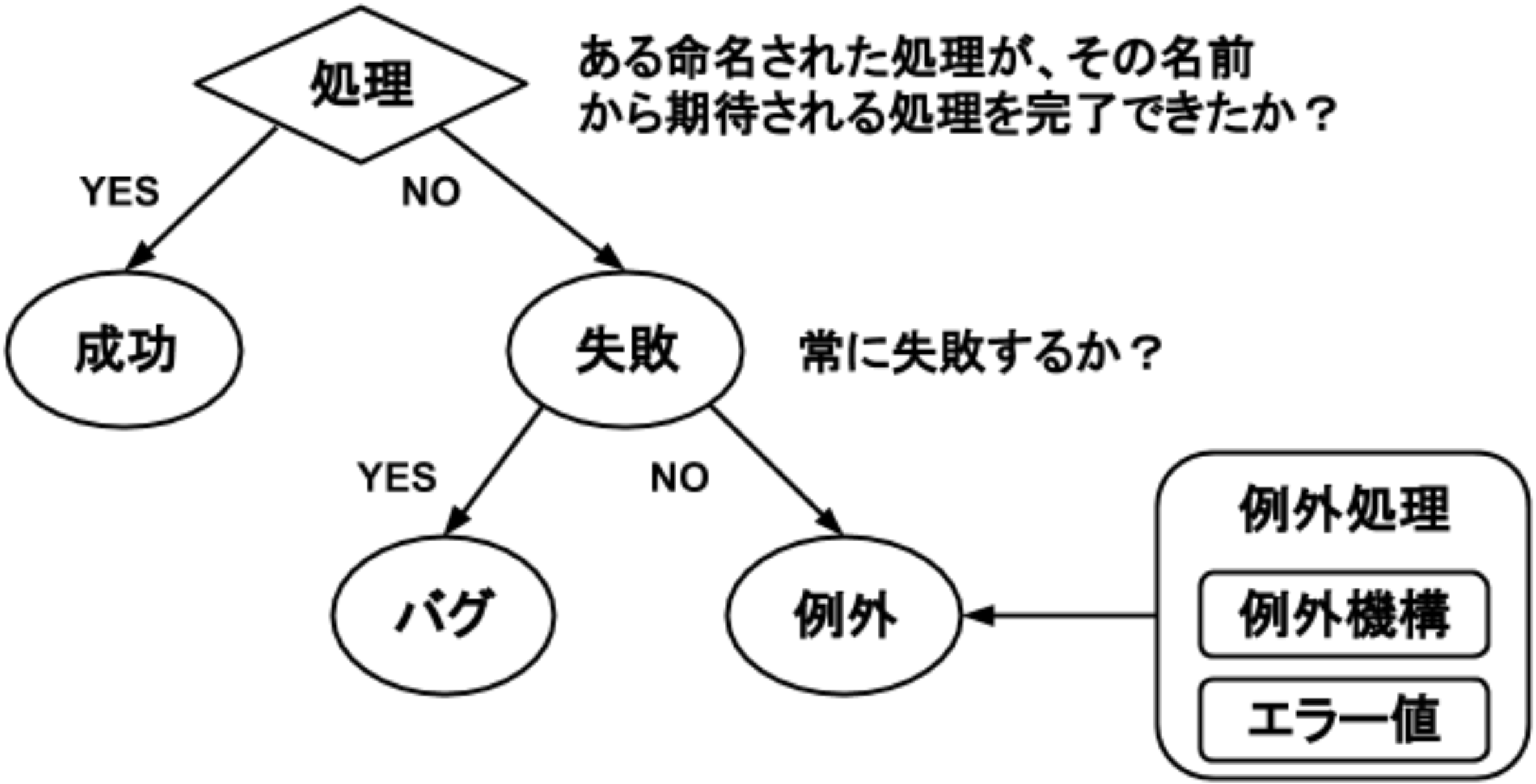
```php
/**
 * 指定された担当者 IĐ およびステータスに合致する Bug を検索し、ヒットした全件を Bug オブジェクトの配列として返す。
 *
 * @param int $assignedTo 担当者IĐ
 * @param Status $status ステータス
 * @return Bug[] 条件に合致した Bug オブジェクトの配列を返す。検索に合致するものがない場合は空配列を返す
 * @throws LogicException カラム名違いや文法エラー等SQLのミスが存在する場合
 * @throws PĐOException データベースとのやりとりに何ら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

SQLSTATE の値を調査し、 PĐOException 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、誰の責任かも見分けられるようにする

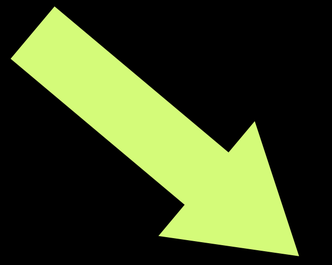


http://qiita.com/Kokudori/items/3a953c00012408f76ab9

まとめ

第1部まとめ: 予防に勝る防御なし

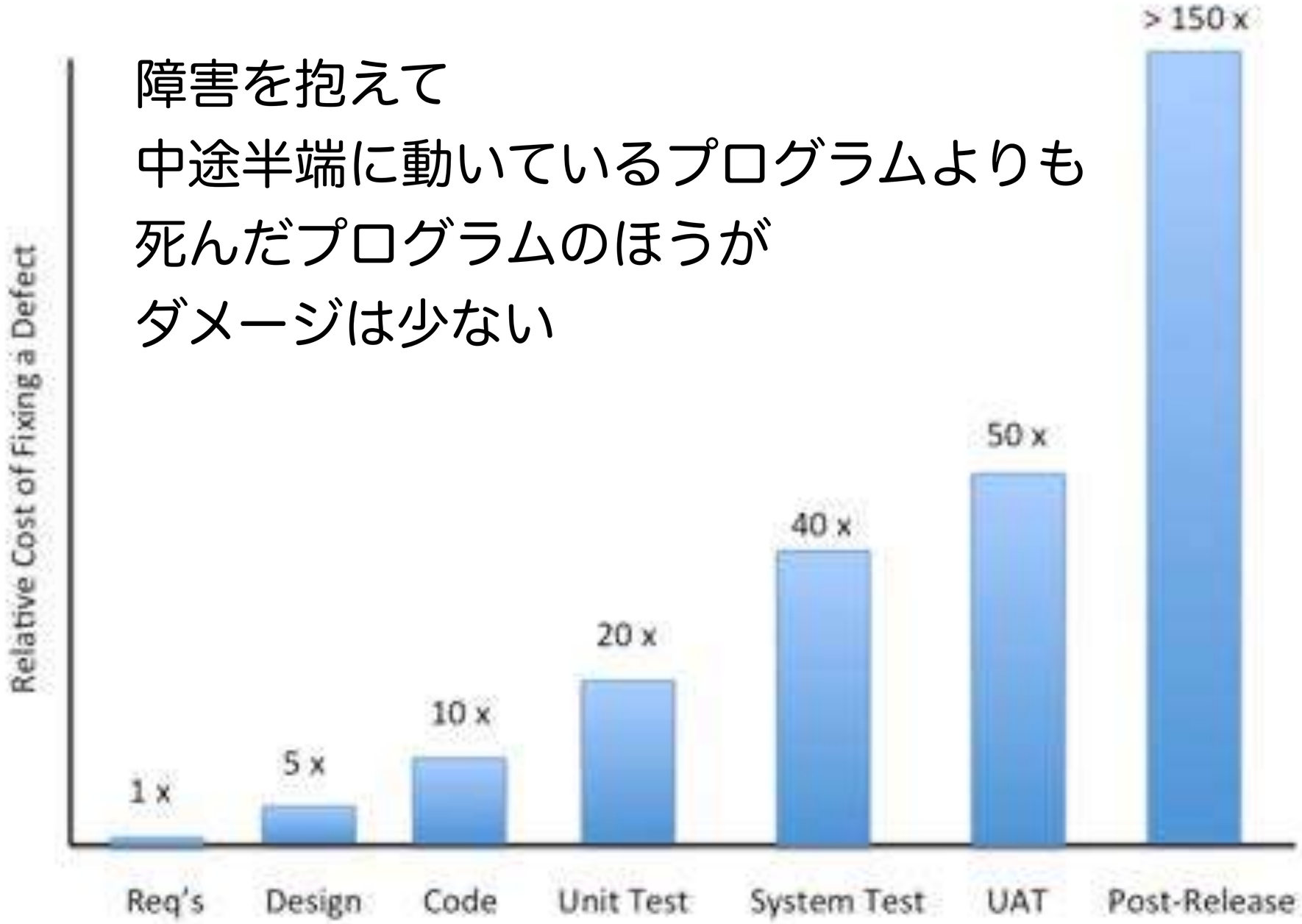
```php
public static function findAll($params)
{
    if (is_null($params)) {
        throw new InvalidArgumentException('params should not be null');
    }
    if (!is_array($params)) {
        throw new InvalidArgumentException('params should be an array');
    }
    if (count($params) !== 2) {
   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

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

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



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

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

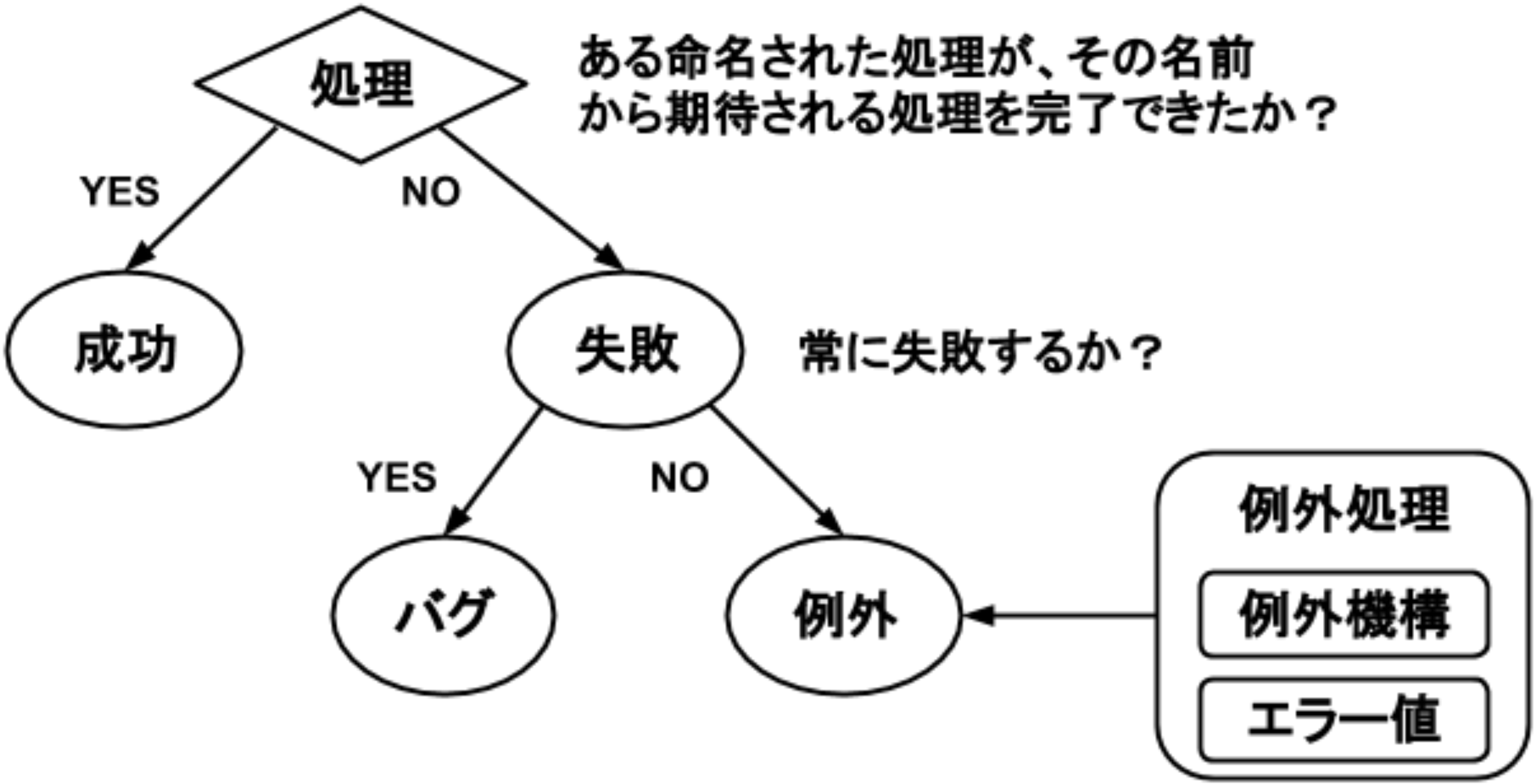
第2部まとめ: “fail fast”

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



https://www.astqb.org/press-room/ISTQB_Certification_News_2015_1.html

第3部まとめ: バグと例外を区別し、誰の責任かも見分けられるようにする



http://qiita.com/Kokudori/items/3a953c00012408f76ab9

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)