

C/C++ ソースプログラムのアーキテクチャを見える化





株式会社東陽テクニカ
ソフトウェア・ソリューション

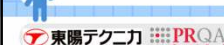
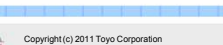
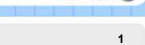
Static Analysis Solutions





目次

- ◆ アーキテクチャの崩れとテスト
- ◆ アーキテクチャの崩れに気づくには
- ◆ Structure101の紹介
- ◆ 設計図と実コードの比較

アーキテクチャとは(1/2)

ソフトウェアの全体構成 / 枠組み

GUI(ユーザ I/F)




- UI制御
- UI描画
- UI入力
- UI出力

通信

- ネットワーク
- データベース
- 外部サービス

汎用計算

- OS
- 言語
- ライブラリ

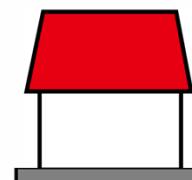
アーキテクチャとは(2/2)




家に例えると...

屋根

壁




土台



目次

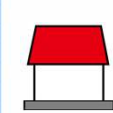
- ◆ アーキテクチャの崩れとテスト
- ◆ アーキテクチャの崩れに気づくには
- ◆ Structure101の紹介
- ◆ 設計図と実コードの比較

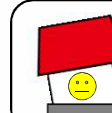
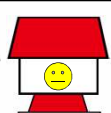
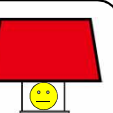




アーキテクチャ崩れの例(1/3)

アーキテクチャが崩れると、テストが大変になる

◆ 家の場合





⇒

家の中にいる人は、崩れに気づきにくい

||

ソフトウェア開発者は**家の中にいる**状態

アーキテクチャ崩れの例(2/3)

◆ ソフトウェアの場合

設計

```

graph TD
    UI[userInput()] --> COM[communication()]
    COM --> ENC[encryption()]
    subgraph BlackBox [ブラックボックス化]
        COM
        ENC
    end
            
```

実装

```

graph TD
    UI[userInput()] --> COM[communication()]
    UI --> ENC[encryption()]
            
```

6

アーキテクチャ崩れの例(3/3)

◆ 機能拡張時

設計

```

graph TD
    UI[userInput()] --> COM[communication()]
    DI[deviceInput()] --> COM
    COM --> ENC[encryption()]
    subgraph BlackBox [ブラックボックス化]
        COM
        ENC
    end
            
```

実装

```

graph TD
    UI[userInput()] --> COM[communication()]
    DI[deviceInput()] --> COM
    UI --> ENC[encryption()]
            
```

このような実装を繰り返すことで
アーキテクチャはさらに崩れる

7

機能拡張時のテスト範囲

◆ deviceInput() 追加時

設計

```

graph TD
    UI[userInput()] --> COM[communication()]
    DI[deviceInput()] --> COM
    COM --> ENC[encryption()]
            
```

実装

```

graph TD
    UI[userInput()] --> COM[communication()]
    DI[deviceInput()] --> COM
    UI --> ENC[encryption()]
            
```

deviceInput() → encryption()
I/F確認の分、テスト範囲が大きい

8

仕様変更の範囲

◆ 通信フォーマット仕様変更時

設計

```

graph TD
    UI[userInput()] --> COM[communication()]
    DI[deviceInput()] --> COM
    COM --> ENC[encryption()]
            
```

実装

```

graph TD
    UI[userInput()] --> COM[communication()]
    DI[deviceInput()] --> COM
            
```

影響範囲が大きい

9

アーキテクチャの崩れとテスト

アーキテクチャが崩れていると、影響範囲が大きくなりすぎる

↓

テストが大変になる

- ◆ 本来、必要な箇所以上のテストが必要になる
- ◆ 無関係と思っていた箇所に影響があり、テグレードが発生する
- ◆ 問題が発生した場合、問題の絞込みに時間がかかる

10


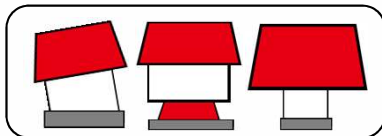
目次

- ◆ アーキテクチャの崩れとテスト
- ◆ アーキテクチャの崩れに気づくには
- ◆ Structure101の紹介
- ◆ 設計図と実コードの比較

11

崩れに気づくには

◆ 家を外から見る

外から見ると崩れていることが分かる

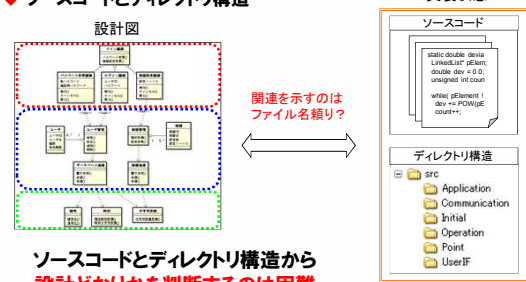
||

ソフトウェア全体として
設計どおりに実装されているかを見る

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 12

設計と実装の比較手段(1/4)

◆ ソースコードとディレクトリ構造



設計図

実装状態

ソースコード

```
static double devia
LinkedList *eElem
double dev = 0.0
unsigned int coun
while (eElem !=
dev += POW(10,
count++;
```

ディレクトリ構造

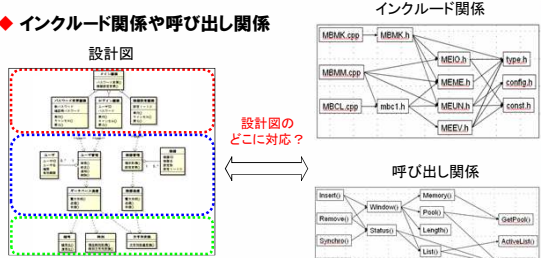
- src
 - Application
 - Communication
 - Initial
 - Operation
 - Point
 - UserIF

ソースコードとディレクトリ構造から
設計どおりかを判断するのは困難

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 13

設計と実装の比較手段(2/4)

◆ インクルード関係や呼び出し関係



設計図

インクルード関係

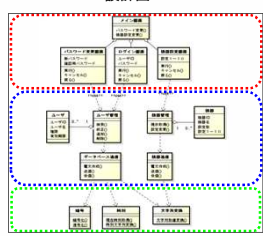
呼び出し関係

設計図のどこに対応?

情報としては必要だが
ファイルのインクルード関係や
関数の呼び出し関係だけでは不十分

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 14

設計と実装の比較手段(3/4)



GUI(ユーザ I/F)

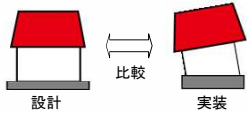
通信

汎用計算

「グループ化」と「グループの上下関係」で
設計図に近づけた形で可視化

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 15

設計と実装の比較手段(4/4)



設計 比較 実装

ソースコードならStructure101を用いると
設計に近づけた形で実装を表示することができる

例えば、次のような問題点が見られる

- ◆ 下位層から上位層への依存がある
- ◆ 様々な箇所と依存関係があり複雑すぎる
- ◆ 階層を跨ぐ依存関係がある

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 16

目次

- ◆ アーキテクチャの崩れとテスト
- ◆ アーキテクチャの崩れに気づくには
- ◆ Structure101の紹介
- ◆ 設計図と実コードの比較

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 17

Structure101とは

**静的解析ツールQA C/QA C++に
アドオンして使用するアーキテクチャ分析支援ツール**

【特長】

- ◆ コードの構成要素間の**依存関係を表示**
- ◆ コードの構成要素の**階層構造を可視化**
- ◆ 構造上、好ましくない**依存関係(循環依存など)を検出**
- ◆ コードの階層構造の**設計図(アーキテクチャ図)と実コードの比較**
- ◆ コードの改版の影響範囲を表示
- ◆ コードの改版で追加された要素や依存関係を**強調表示**
- ◆ Webブラウザから参照可能

Copyright (c) 2011 Toyo Corporation 18

依存関係とは

コードのある**要素**が別の要素に影響を与えるとき、依存関係がある
フォルダ、ファイル、グローバル変数、関数、クラス...

- includes ファイルAがファイルBをインクルードする
- calls 関数Aが関数Bを呼び出す
- references 関数Aがクラスまたはグローバル変数Bを使用する
- has クラスAがクラスBをメンバに持つ
- has param 関数AがクラスBをパラメータに持つ
- returns 関数AがクラスBを返す
- specializes 関数AがテンプレートBを特殊化する
- subclasses クラスAがクラスBを継承する

要素Aは要素Bに依存する A → B

Copyright (c) 2011 Toyo Corporation 19

階層構造を可視化(1/2)

ディレクトリ構造 依存関係を元に階層構造を可視化

上位層
↓
下位層

Copyright (c) 2011 Toyo Corporation 20

階層構造を可視化(2/2)

階層構造をグラフで表示する。矢印で依存の方向、数字で依存の数を表す。

プロジェクトのディレクトリ構造

選択した要素内の階層構造と依存関係を表示

循環依存している箇所を赤枠で表示

選択した依存関係の詳細を表示

Copyright (c) 2011 Toyo Corporation 21

設計図の作成(1/3)

既存のコードから設計図を自動生成することが可能

設計図 自動生成

- ◆ 設計図にはフォルダやファイルがセルとしてマッピング
- ◆ セルの追加、削除、移動、グルーピングなどが可能

Copyright (c) 2011 Toyo Corporation 22

設計図の作成(2/3)

下位層から上位層への依存関係を検出

デフォルトでは同階層および下位層から上位層への依存は違反として検出する

ユーザー I/F
↑
通信
汎用計算

Copyright (c) 2011 Toyo Corporation 23

設計図の作成(3/3)

階層を跨ぐ依存関係を検出

上位層から下位層への依存であっても階層を跨ぐ依存を違反として検出する(※)
また、設計図上の任意の実装間の依存に対して、違反とする/しないを設定可能

※設計図のプロパティで、項目StrictをYesに設定する

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 24

目次

- ◆アーキテクチャの崩れとテスト
- ◆アーキテクチャの崩れに気づくには
- ◆Structure101の紹介
- ◆設計図と実コードの比較

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 25

比較のための準備(1/2)

ディレクトリ構造

階層構造

- ◆階層構造はツールが自動的に判断
- ◆ツールがアーキテクチャの設計意図を汲んでくれる訳ではない

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 26

比較のための準備(2/2)

少なくとも初回は設計意図を知っている「アーキテクチャ設計者」が設計図を調整

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 27

問題例 1(1/2)

- ◆下位層から上位層への依存がある

汎用計算(utility)からユーザーI/F(GUI)への依存関係がある
この汎用計算のプログラムを別のアプリケーションで使用する場合、GUI画面まで必要になってしまう
→ 再利用性の低下

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 28

問題例 1(2/2)

実装の誤り

GUIのコード内に本来は汎用計算にあるべき関数が含まれていて下位層から呼び出されている

↓
対象の関数を分離し汎用計算へ

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 29

問題例2(1/2)

◆ 様々な箇所と依存関係があり複雑すぎる

下位層から上位層への依存はないが、このソースは、下位の様々なソースから影響を受け、上位の様々なソースに影響を与える
 → コードが肥大化し、処理が複雑になっている可能性

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 30

問題例2(2/2)

ファイルの分割

コードを分割することで、依存関係を簡潔にする
 → コードの変更時の影響範囲を小さくする

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 31

問題例3(1/2)

◆ 階層を跨ぐ依存関係がある

下位層から上位層への依存ではないが、設計時には階層を跨ぐ依存は意図していない
 → アーキテクチャの崩れ

本来の設計意図

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 32

問題例3(2/2)

放置すると・・・

さらなる意図しない依存を引き起こす

本来の設計意図

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 33

まとめ

アーキテクチャが崩れると、テストが大変になる
 ↓
 崩れに気づくには、外から見る事が重要
 ↓
 「グループ化」と「グループの上下関係」で設計図に近い形でコードの状態を可視化
 ↓
 崩れたアーキテクチャを早い段階で修正

↓

アーキテクチャの維持

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 34

ご清聴ありがとうございました。

株式会社東陽テクニカ
ソフトウェア・ソリューション

東陽テクニカ PRQA Copyright (c) 2011 Toyo Corporation 35