Modeling Forum 2011

複雑化するAndroidアプリに対する設計の重要性

2011/10/19

株式会社 豆蔵
岡田 真一

自己紹介

- 氏名：岡田 真一
- 年齢：42歳
- 経歴：
  - 2001年に豆蔵入社
  - 主に業務系アプリケーションのためのフレームワーク開発を担当
  - 自動車業界向け企業に転職
  - ２年半豆蔵から離れていたが4月に復豆。それからAndroidの担当に
- 趣味：昨年から自転車に乗ってます

本日の内容

- （簡単に）Androidの紹介
- Androidの動向/現状の課題
- 大規模/複雑化により問題となること
- Android開発が大規模化の波が来る前に
- Androidの現状の課題への対応例
- 便利なツールのご紹介
- おわりに

Androidとは

Androidとは

- Googleを中心としたOpen Handset Allianceにより開発されるスマートフォン/タブレットといった携帯情報端末向けのプラットフォーム
- カーネルにLinux、アプリケーションフレームワークにJava言語を採用。その他OSSのツール/ライブラリで構成されている
- 主な機能
  - 電話(GSM、CDMAなど)
  - ネットワーク(無線、Bluetooth)
  - 各種センサー(GPS、コンパス、加速度)
  - Webブラウザ(WebKitベース)
  - メッセージ(SMS、MMS)
  - マルチメディア(H.264、MPEG4など)

豆蔵 mamezou

Androidの動向/現状の課題

1.5 Cupcake

1.6 Dunut

2.0/2.1 Eclair

2.2 Froyo

2.3 Gingerbread

3.0/3.1/3.2 Honeycomb

Androidプラットフォームの動向

- より大規模/複雑化
  - 現状は小規模体制
  - スマートフォン/タブレットのH/Wスペックが年々あがってきている
    - 年末には4コアCPU搭載機が発売予定？
    - 1GBのメモリは当たり前？
- スマートフォン/タブレット以外のデバイスにも拡大
  - デジタル家電
    - Android@Home, Android Open Accesary , Google TV
  - カーナビなど車載情報端末
- ビジネスフィールドへの活用が広がる
  - 既存Webアプリケーションのフロントエンドとしての利用
- 日本独自進化？
  - ワンセグ、赤外線通信、お財布

Androidの頻繁なバージョンアップ

- 3-6ヶ月毎リリース
  - 新しいAPIが追加される
  - APIの変更
    - 基本はDeprecatedでしばらく使えるが、いきなり使えなくなる事もある
- スマートフォン、タブレット用のAPIとして分岐中
- 複数のバージョンがサポート対象

Version	API level	Distribution
3.x.x *Honeycomb*	11	0.6%
2.3.x *Gingerbread*	10	9.2%
2.2.x *Froyo*	8	64.6%
2.0.x/2.1.x *Eclair*	7	21.2%
1.6 *Donut*	4	2.5%
1.5 *Cupcake*	3	1.9%

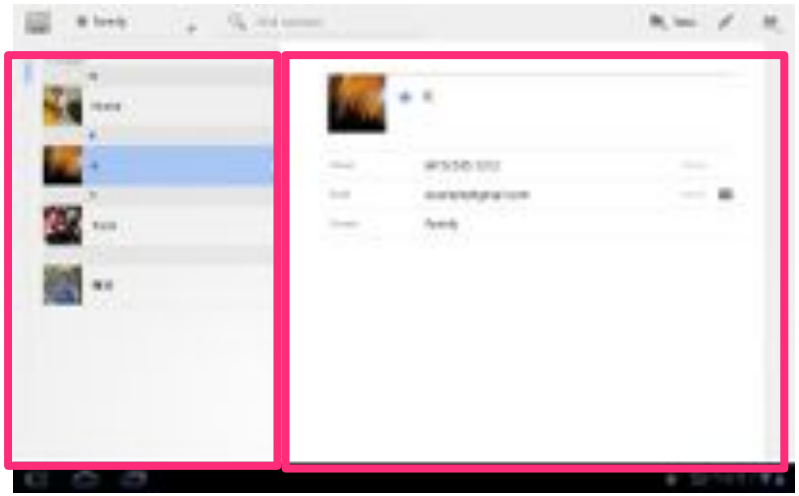
2011年6月1日現在のAndroid Marketへのアクセス統計による バージョンごとの世界シェア。日本のシェアとは異なる。

豆蔵 mamezou

ターゲットが複数ある

- スマートフォン
  - Android 1.X, 2.X系
- タブレット
  - Android 3.X系

4.0? Ice Cream Sandwich で統合の予定



- タブレット版では、**Fragment**を用いて1つの画面に複数の区画を表示することができる
- スマートフォンでは1つの画面には一つの区画しか表示できない

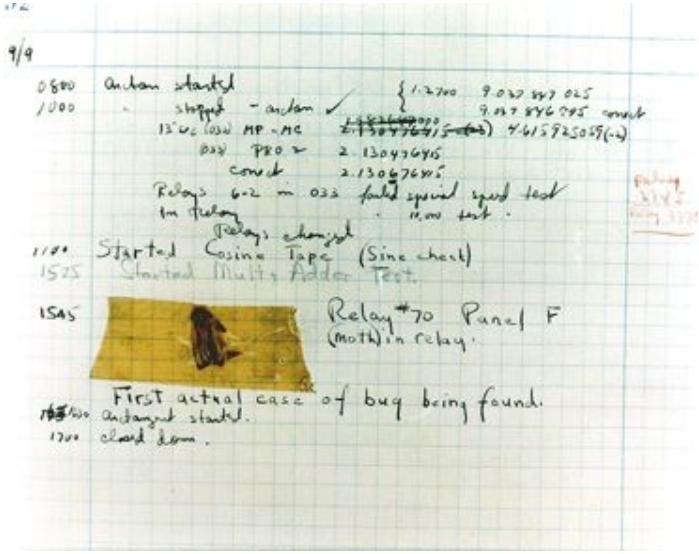
http://developer.android.com/intl/ja/sdk/android-3.0-highlights.html

デバイス毎に色々と違う

- 画面サイズ
- 入力装置
  - トラックボール/ハードウェアキーボードの有無
- 搭載機能
  - 電話3G/Wi-Fi/Bluetooth/カメラ/GPS/コンパス/赤外線通信/NFC…
- 機種に依存した箇所がある
  - カメラのパラメータなど

Androidのバグ

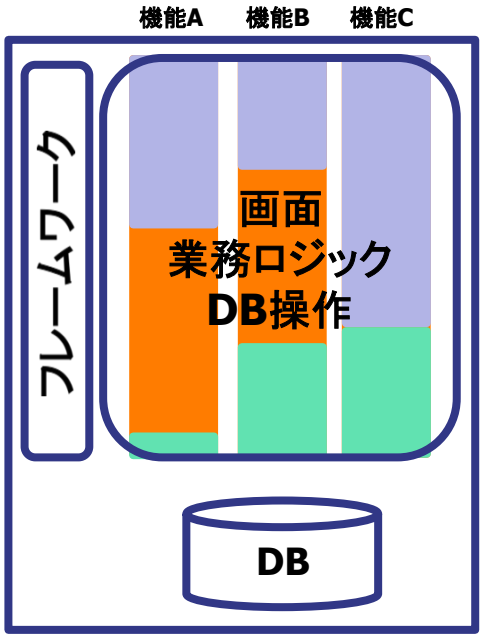
- バージョンによって同じAPIなのに振る舞いが違う
- JNIのパラメータのサイズが大きいと渡したデータが壊れる



大規模/複雑化により問題となること

豆蔵 mamezou

業務系Webアプリケーションの例（1）



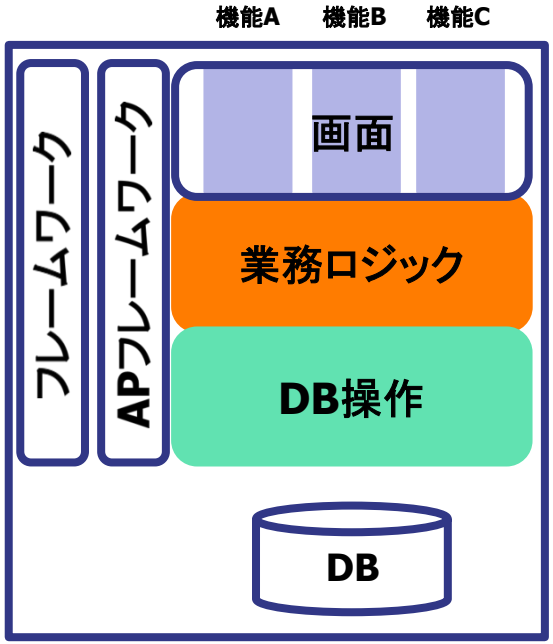
- 画面、業務ロジック、DB操作が全て同じモジュール内で実装。モジュール分割がされていても境界があいまい。
- 担当毎に作り方が異なる
- 類似機能は各担当毎に実装

- 機能（画面）の独立性が高い
- アプリケーション全体で一貫性がない
- 共通機能が少ないため、各担当者の実装量が多い
- ロジックが分散しているため
  - バグが収束しない
  - チューニングが難しい

大規模/複雑化の課題

- 大量の開発者を集め同時期に開発を行う
  - ウォーターフォール型開発プロセス
  - スキルのばらつきが大きい
    - 低スキルの開発者に環境を合わせる傾向にある
- （複数の開発会社が入っているため）機能の共通化が進まない
- 再利用の仕組みがない
- テストが難しい
- 厚いフレームワーク層は敬遠される
  - 階層が深くなり、構造が複雑になるため、開発者への教育に時間がかかる
  - フレームワーク層自体が安定していない
    - バグが存在する→切り分けが大変
  - 薄いフレームワーク層の代表：Struts

業務系Webアプリケーションの例（2）



- 画面、業務ロジック、DB操作がそれぞれレイア毎のモジュールとして実装
- 共通部分はAPフレームワークとしてまとめる
- 業務ロジック、DB操作は共通機能と捉える

- 機能（共通部品）の独立性が高い
- アプリケーション全体で一貫性がある
- 共通部品が多くなるため、個々の機能の実装量が減る
- 機能毎にまとまっているため、不具合に対する修正箇所が局所化される

フレームワーク　：Web, DB用
APフレームワーク：業務/アプリケーション構造に特化したフレームワーク

Android開発に大規模化の波が来る前に

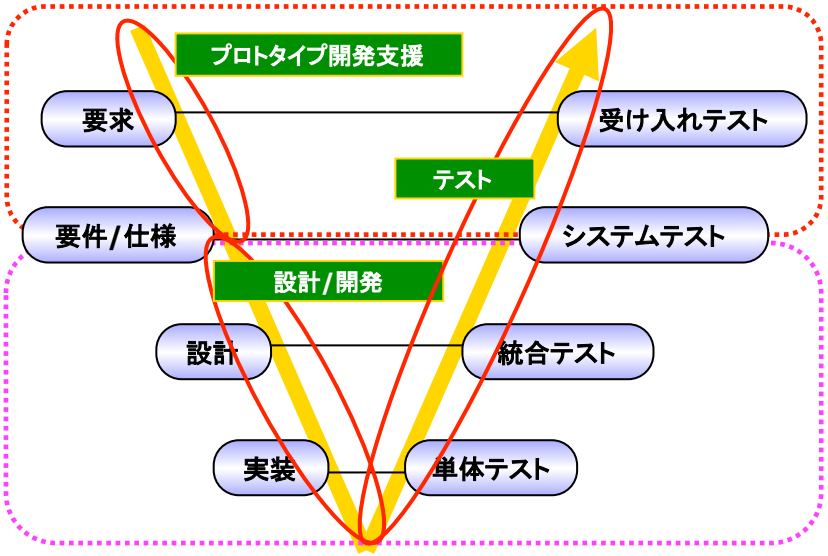
- 開発プロセスを見直す
- 設計を重要性を見直す
- 情報共有の仕組みを考える
- ツールの活用

品質向上のための組織作りのチャンス

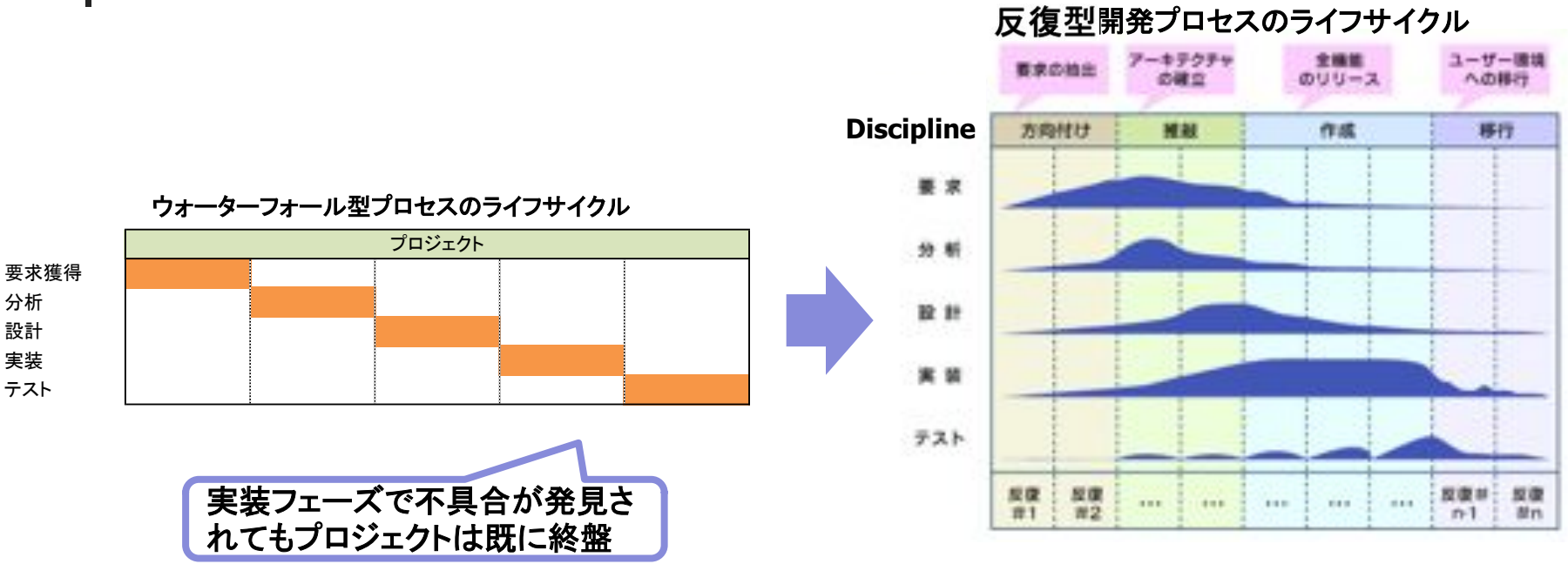
豆蔵 mamezou

開発プロセスを見直す

- 大きな改善
  - 反復型開発プロセスを取り入れる
- 小さな改善
  - レビューを取り入れる
  - ペアプログラミングを取り入れる



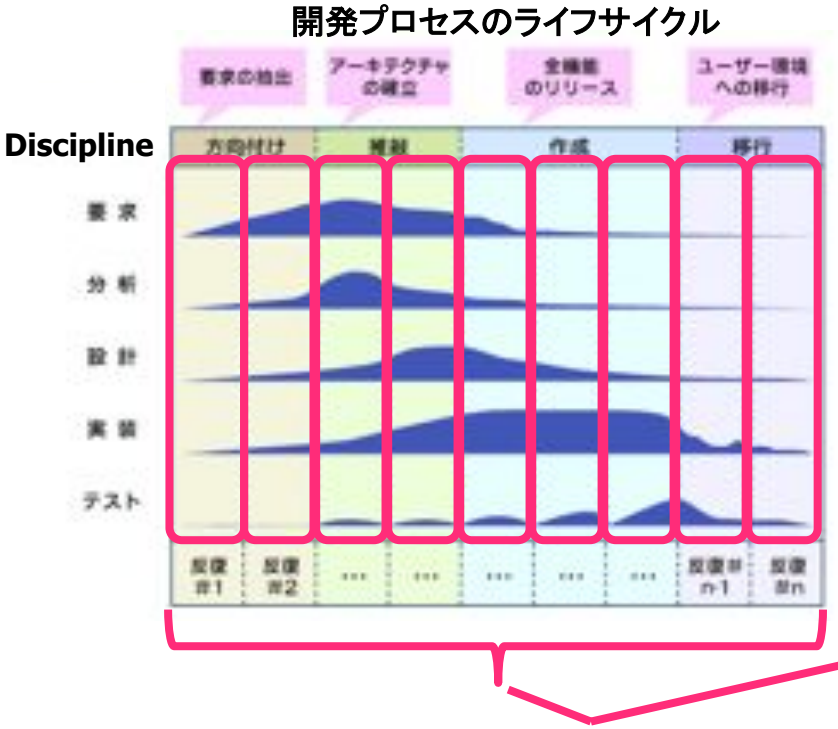
反復型開発プロセスを取り入れる

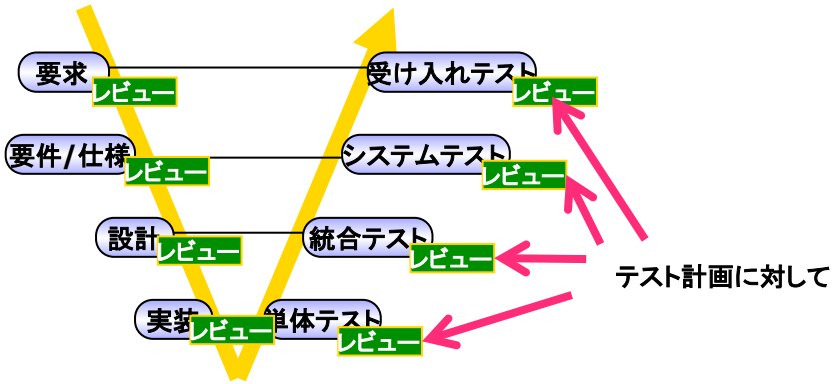


- Androidでは、UIによる操作性が重要視されるため、短いイテレーションで早期に洗練できる
- リスクをプロジェクトの早い段階で抽出できる
- ウォーターフォール開発では開発の途中でAndroidのバージョンがあがった場合対応できない

レビューを取り入れる

- Androidでは、実機を使って簡単に動くアプリケーションを開発することができるため、アーキテクチャ等から逸脱していないか、セキュリティ、品質などがクリアしているかをチェックする





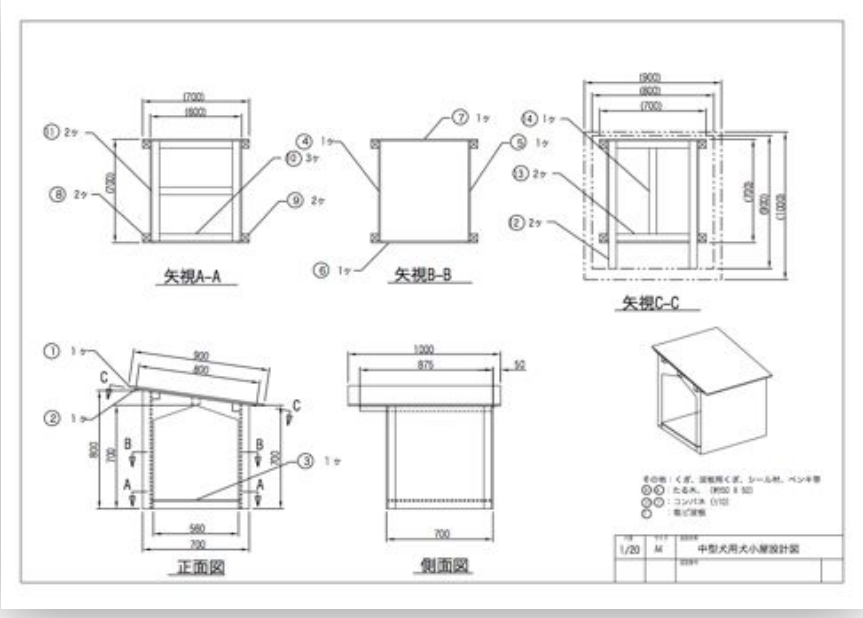
レビューを開発プロセスに組み込む

ペアプログラミングを取り入れる

- リアルタイムレビューに相当するため、作業意識が高まり、品質が向上することが期待される
- 情報の共有化やOJTのような教育的な側面も期待できる
  - Androidフレームワークの使い方の伝授
- 複雑なアルゴリズム/構造が必要な箇所に適用すると、特に効果が高い
- プログラミング時だけではなく、設計などでも適用可能

設計を重要性を見直す

- アーキテクチャを考える
- 扱える程度の規模に分割する
- 再利用を考える
- ドキュメントを考える



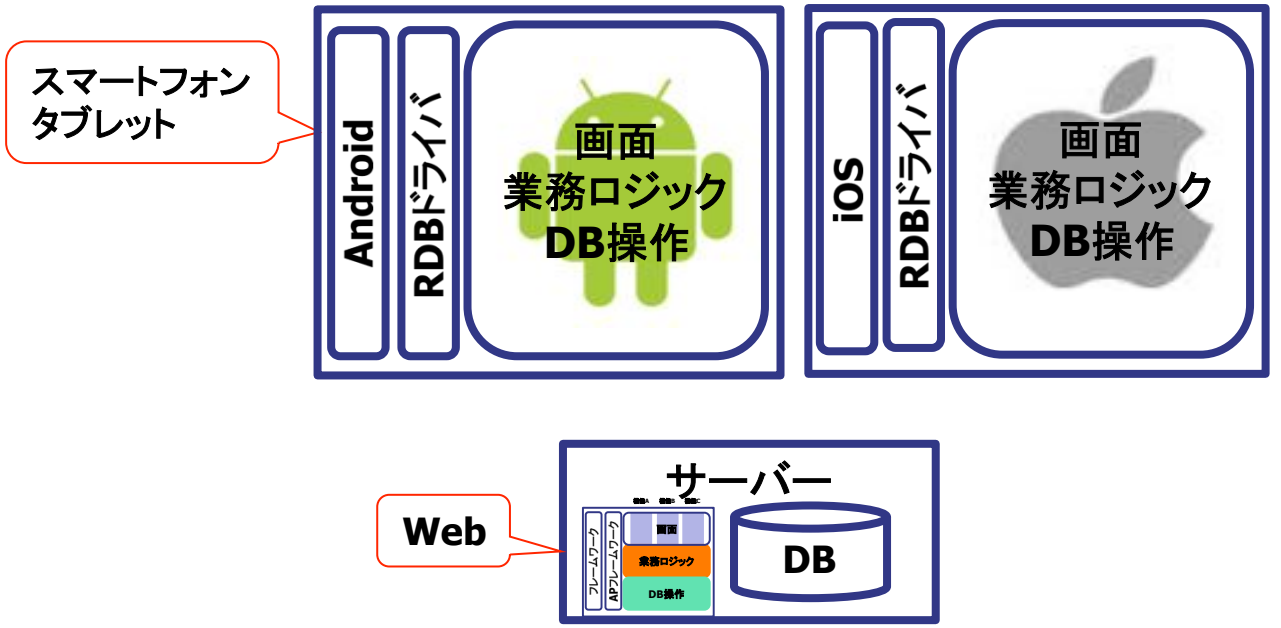
アーキテクチャを考える（1）

- アプリケーションの全体の構造を規定する
  - 論理的なシステム構造の規定
    - サブシステム
    - 論理レイヤー
  - 使用するプラットフォーム、ミドルウェア、市販ソフトなどの利用規定

アーキテクチャを考える（2）

- アプリケーション全体に関わる機能を検討し、共通部品として提供する
  - 入力データ検証
  - フォーマッタ
  - セキュリティ
    - 認証
    - アクセス権
    - 不正アクセス、データ漏洩
  - トランザクション管理
  - データ永続化
  - リソース管理
  - 通信方式
    - 通信データ形式
  - エラー処理

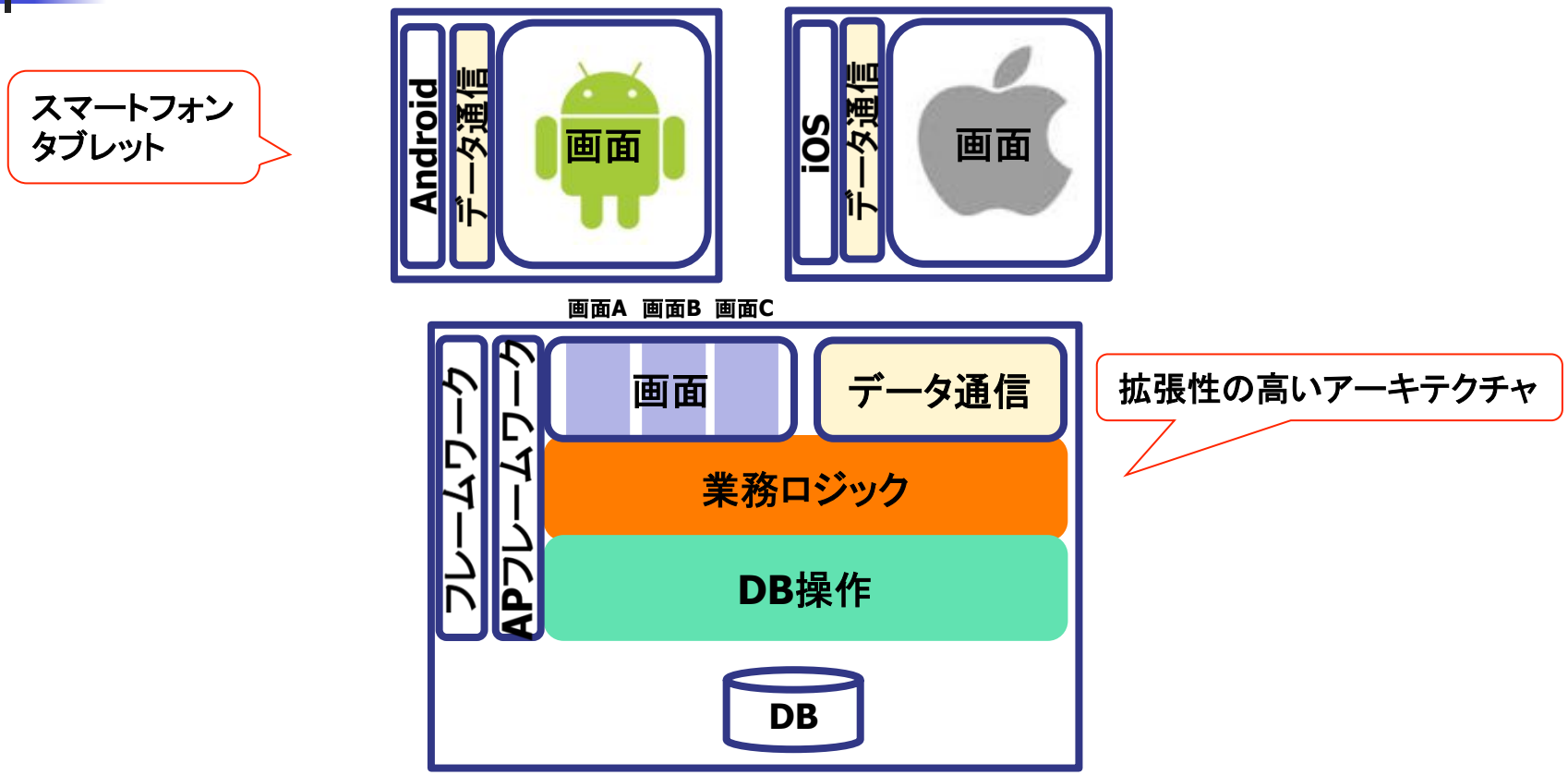
WebアプリケーションのAndroid対応の例（1）



- Android用のRDBドライバがあれば、Webアプリケーションのように全てAndroid上に構築する事も可能
- Webアプリケーションと全て別アプリとなるため、プラットフォーム毎に構築する必要がある

- フレームワーク　:Web, DB用
- APフレームワーク:業務に特化したフレームワーク

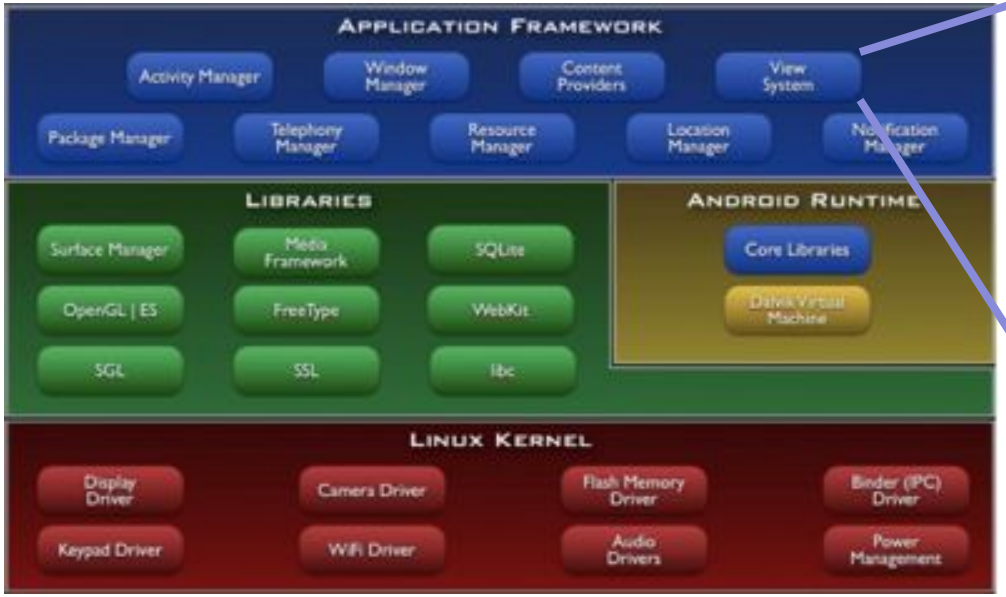
WebアプリケーションのAndroid対応の例（2）

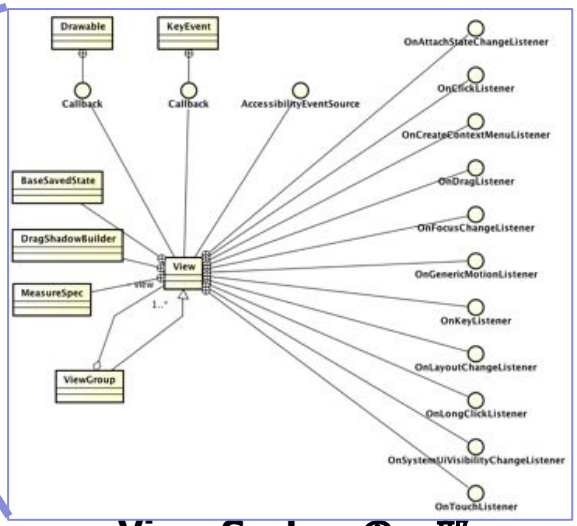


- Webアプリケーション側の変更は、データ通信モジュールの追加のみ
- スマートフォン/タブレット側は画面とデータ通信モジュールが必要

- フレームワーク　:Web, DB用
- APフレームワーク:業務に特化したフレームワーク

扱える程度の規模に分割する

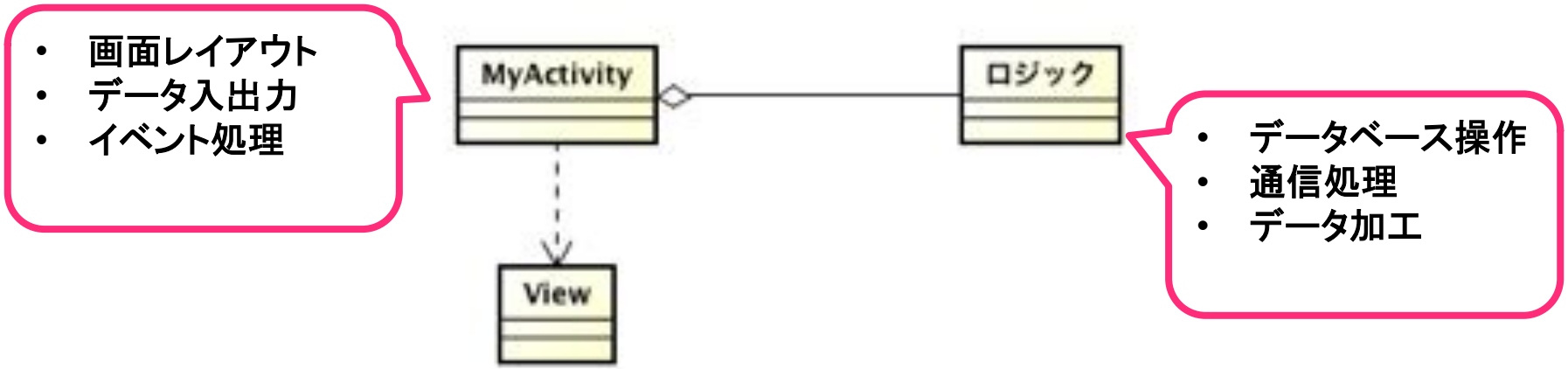




View Systemの一部

- 問題領域が小さい方が扱いやすい
- 理解しやすい
- Androidプラットフォームもサブシステム/モジュールに分割して複雑度を下げている
- 独立したモジュールにすることでテストが容易になる

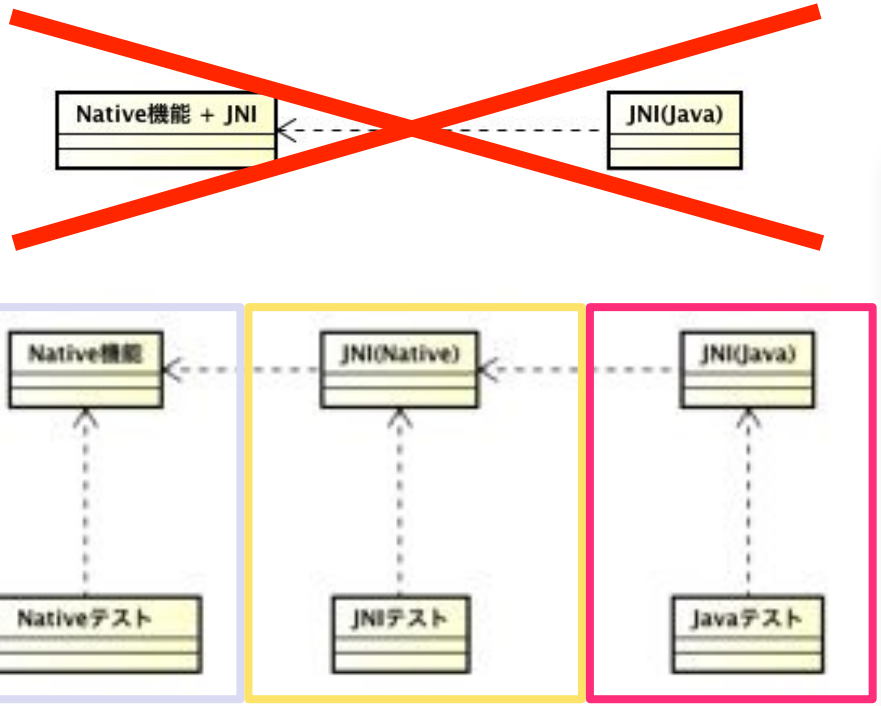
テスト（1）

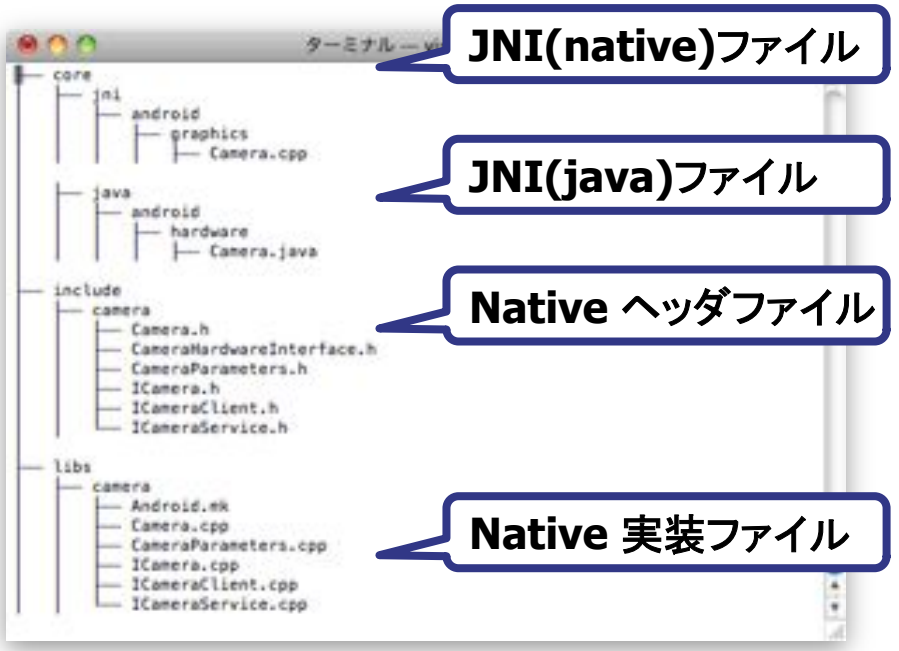


- 画面で必要なデータを取得、加工などを行う処理を画面と別のクラスに分離することで、ロジック部分のみのテストを実施できる
  - 画面操作が不要となるので自動テストが可能

テスト（2）

- Androidのソースツリーでも Native層とJava層の間にJNI層を用意している。→独立性、テスト容易性

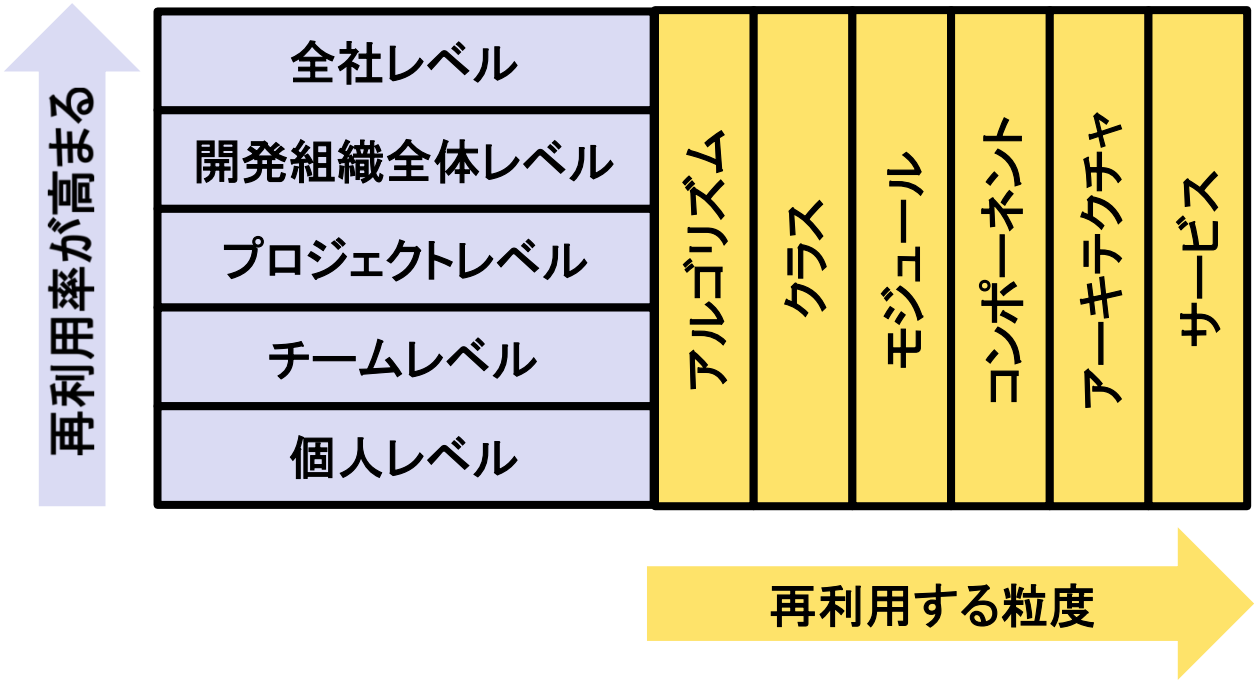




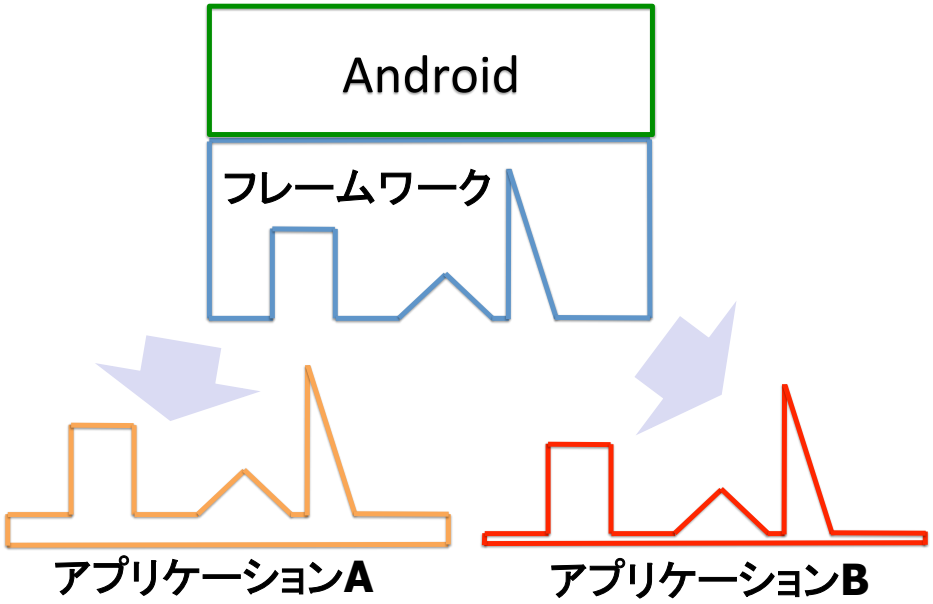
ソースツリーの抜粋

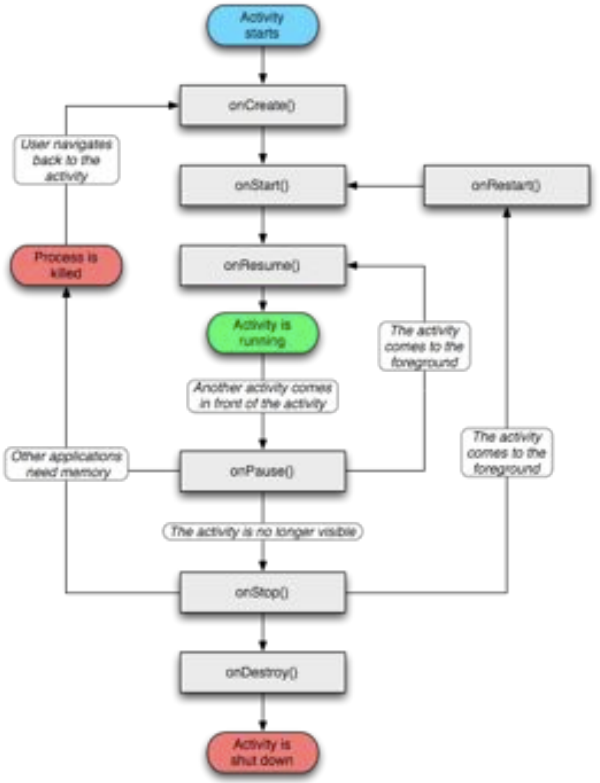
再利用を考える（1）

- ソフトウェア部品を再利用することで、以下のようなメリットがある
  - 品質が高い：　既にテストされているため
  - 生産性が高い：　新しく開発しなくていい



再利用を考える（2）





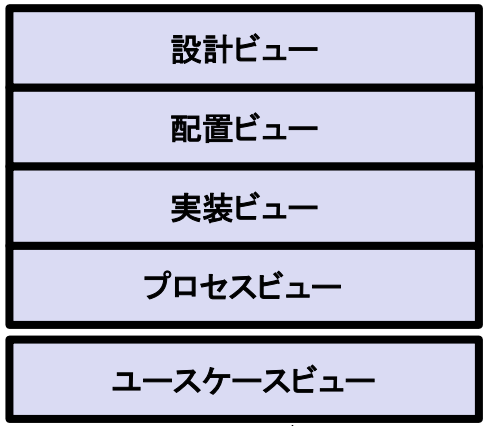
- Androidもフレームワークである
  - Activityなど実装しなければならない箇所が規定されている
- アプリケーションの構造はフレームワークで規定される
- アプリケーション側の実装量を減らす事ができる

ドキュメントを考える

- 複雑なアプリケーションの仕様を、簡潔に表現できる必要がある
  - 日本語による文章では、あいまいな表現となってしまう
- 読み手により解釈が異ならないように厳密な定義が必要
- アプリケーションの静的構造と動的構造が表せる必要がある
  - 静的構造：モジュールの定義、およびその関係
  - 動的構造：アプリケーション実行時におけるモジュール間の相互作用を定義
- 抽象度が異なるモジュールでも、統一的に扱える表記法が必要
  - 複数の表記法を採用すると覚えるのが大変

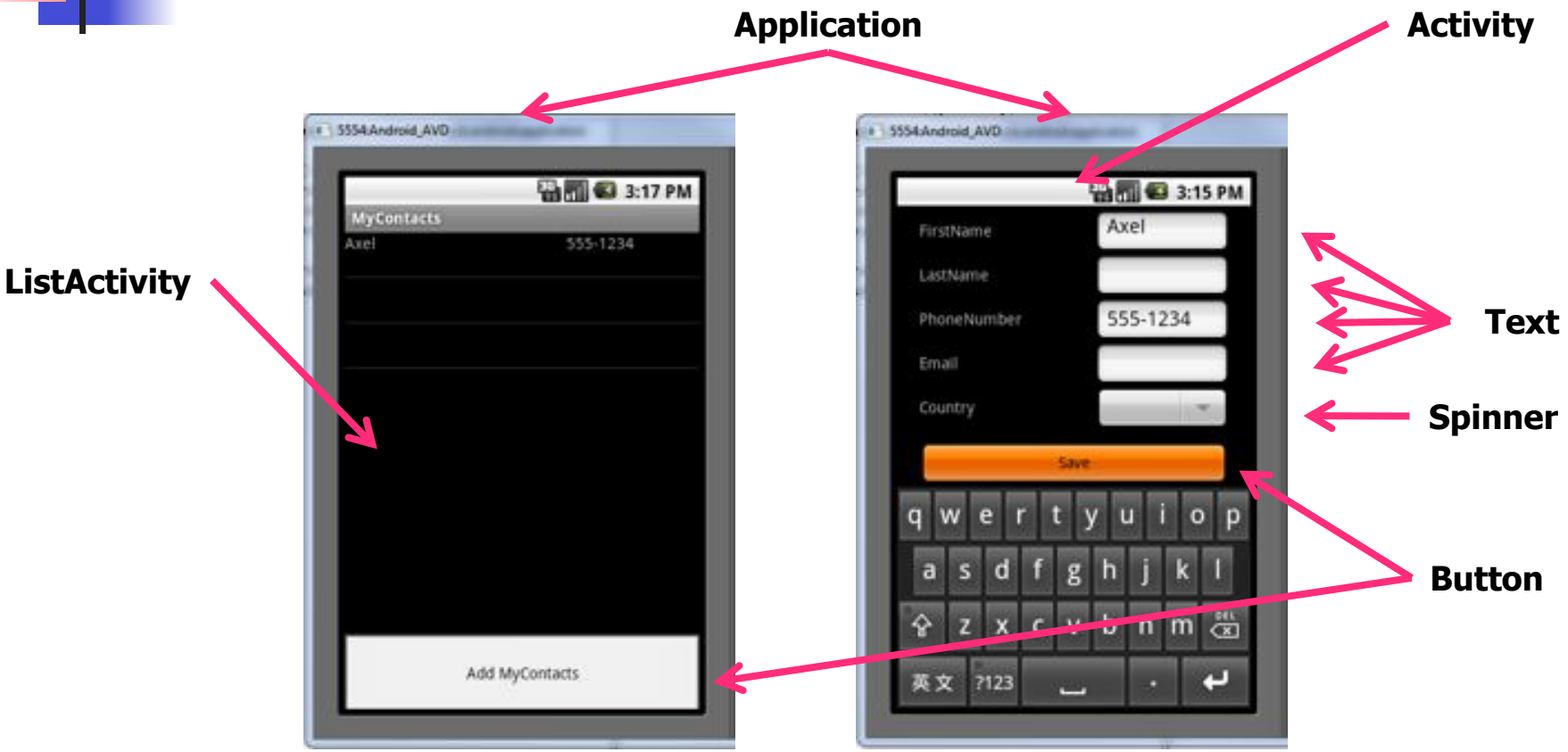
UML

- UMLは業界標準
  - 図を用いたビジュアルな言語
  - 13の図をサポート
    - 静的構造
      - クラス図、パッケージ図、コンポーネント図、コンポジット構造図、配置図、オブジェクト図
    - 動的構造
      - シーケンス図、状態マシン図、ユースケース図、アクティビティ図、コミュニケーション図、タイミング図、相互作用概要図
  - Myルールでは、それぞれの図形の意味があいまいであり、読み手側も混乱する
    - チーム全員に説明しなければならない
  - 全ての図を使う必要はない
    - 静的構造、動的構造が記載できれば良い
      - クラス図、シーケンス図、状態マシン図



4 + 1 のビュー

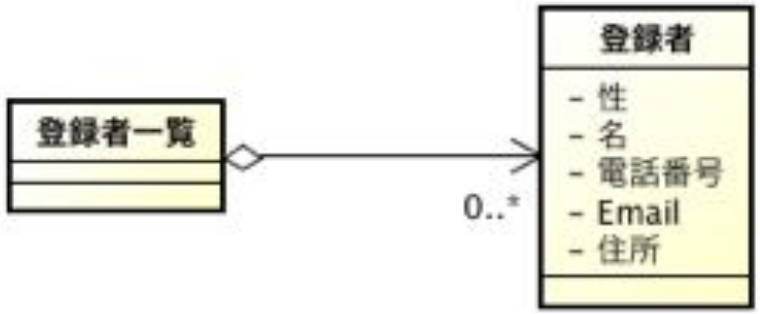
分析・設計の例



- 登録者の一覧を表示
- 選択した登録者の詳細を表示

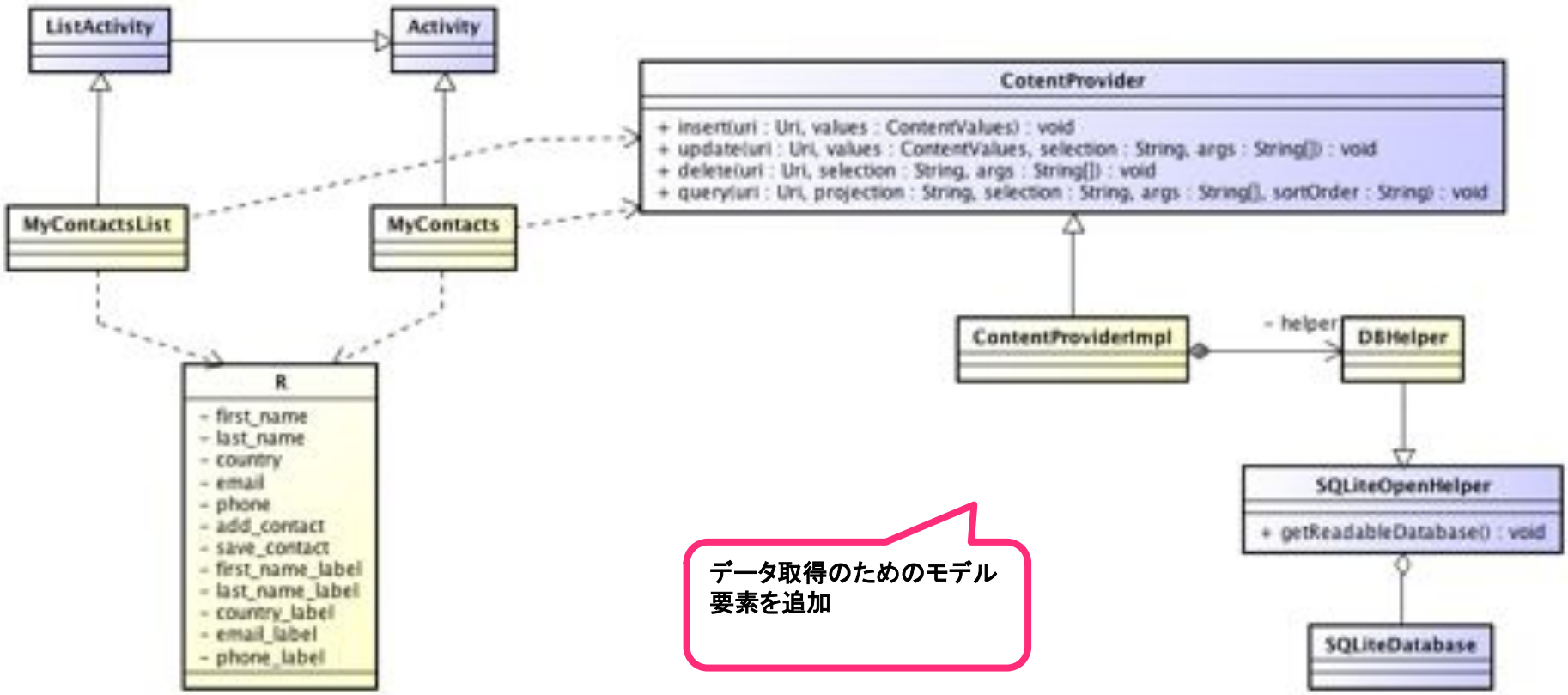
分析モデル

- プラットフォームに依存しないモデル
- Android以外のプラットフォームでも使う事ができる



設計モデル

- プラットフォーム固有の環境を分析モデルに対して追加/変換したモデル



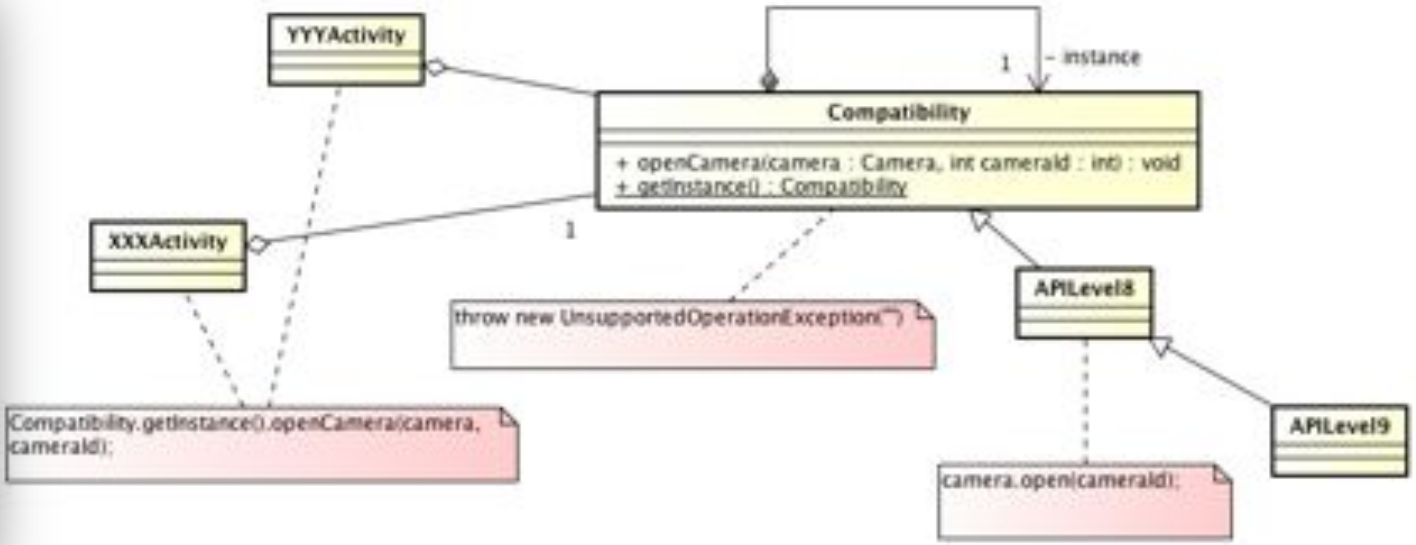
Androidの現状の課題への対応例

- バージョン間の差異の吸収の一例
- 機種依存への対応の例

バージョン間の差異の吸収の一例

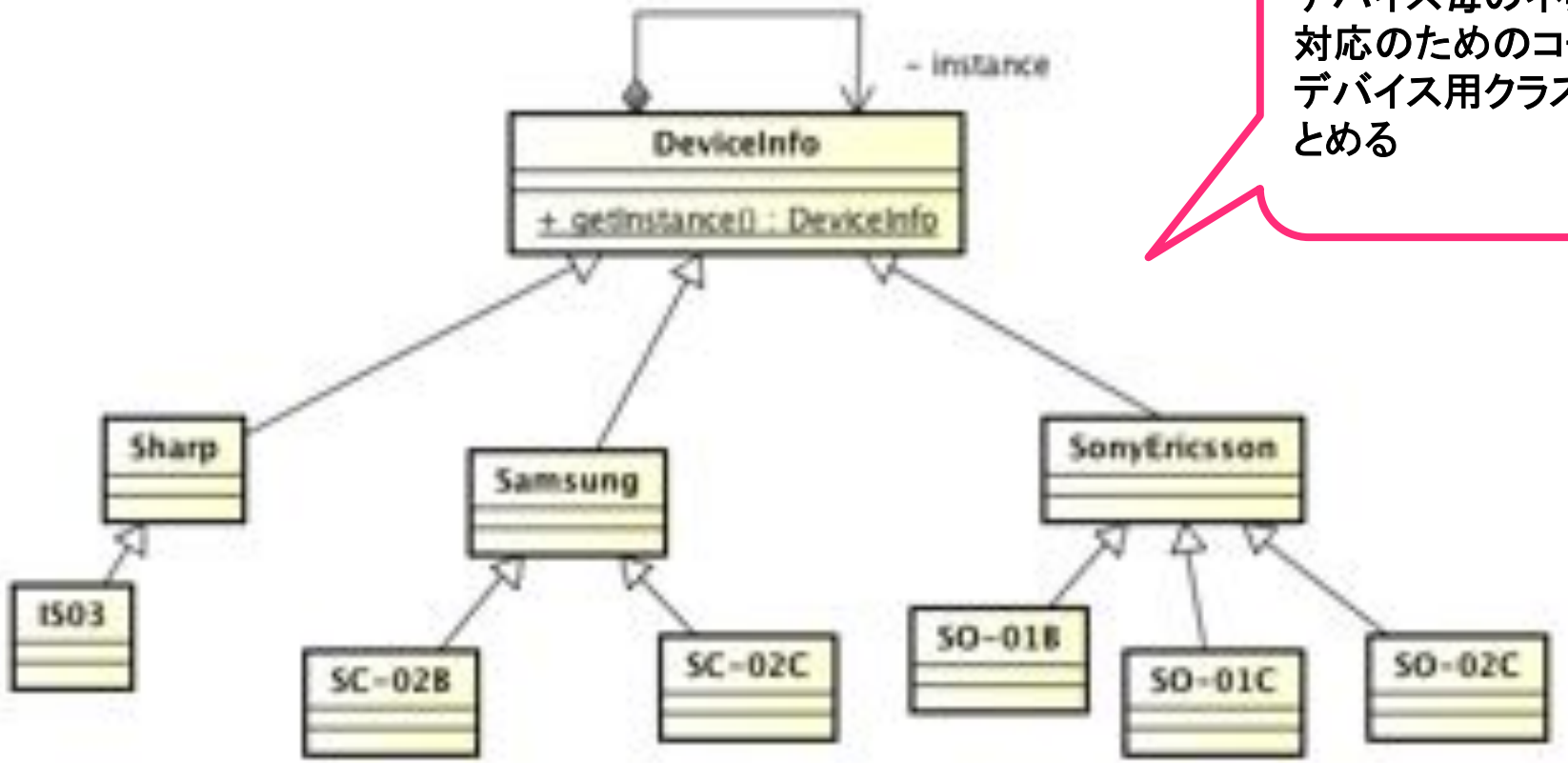
```
int api_level = Build.VERSION.SDK_INT;

// 1.X
if (api_level <= 4) {
    // cupcake or donut
}
// 2.X
else if (api_level <= 11) {
    if (api_level <= 8) {
        // eclair
    }
    else if (api_level == 10) {
        // 2.3.3, 2.3.4のみ
    }
    else if (2.3.2) {
        // gingerbread
    }
    else {
        // froyo
    }
}
// 3.X
else {
    // honeycom
}
```



このようなコードがアプリケーションに
散らばらないように
→ クラスを用いてまとめる

機種依存への対応の例



デバイス毎の不具合対応のためのコードをデバイス用クラスにまとめる

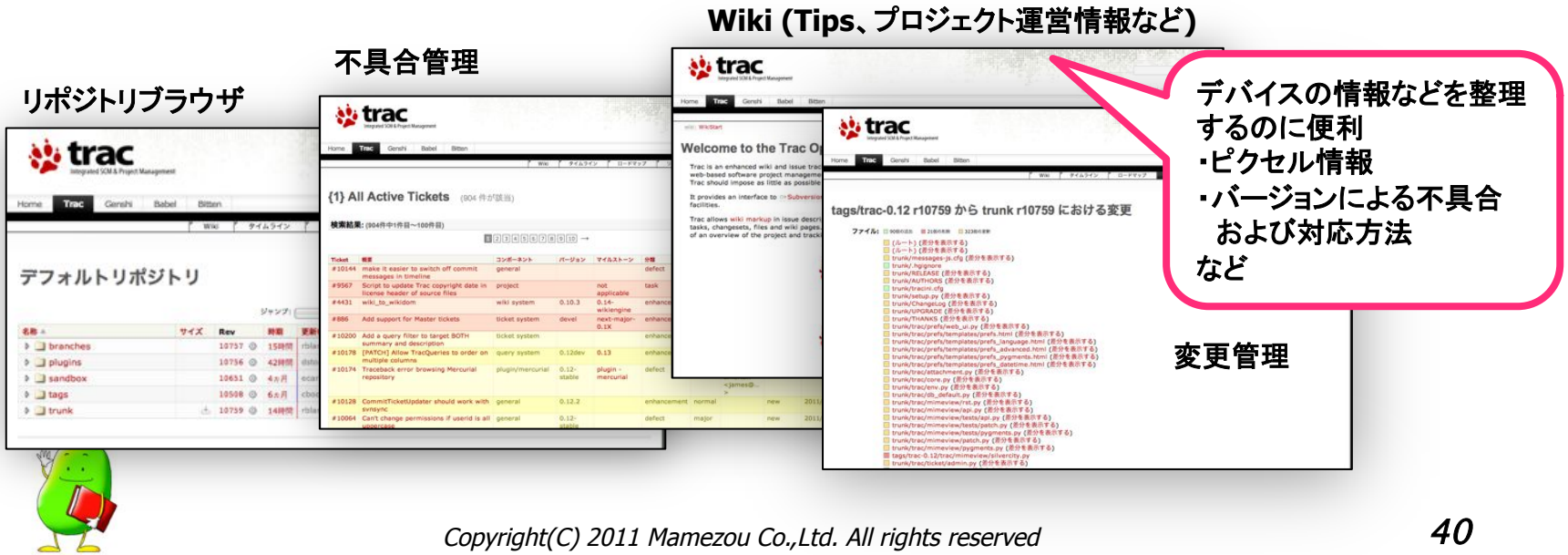
便利なツールのご紹介

- 情報共有を考える
- プロファイリングを取り入れる
- ツールを導入/どんどん自動化

情報共有を考える

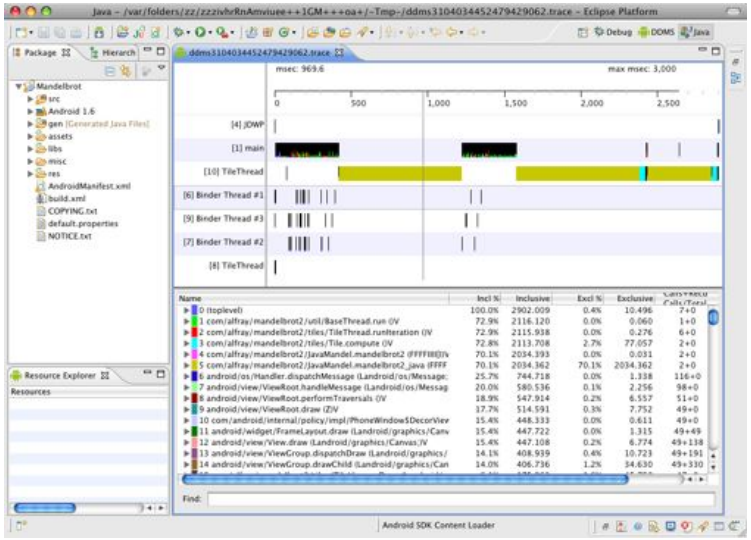
- 構成管理システム(CVS, Subversion, Git,...)
- Trac
  - 構成管理システムのWebインターフェース
    - ソースコードアクセス
    - 不具合管理
    - 変更管理
    - Wiki

複数機種、複数バージョンに対応するためには構成管理が必須



プロファイリングを取り入れる

- 最初から最適化しない
  - 思い込みかもしれない
  - KKD(勘、経験、努力)
  - ボトルネックはどこ？
- 80:20の法則
  - アプリケーションの構造が整理されていないと効果は薄い



traceview

ツールを導入/どんどん自動化

生産性を高めるツールが数多く利用できる環境が整備されてきています。
Eclipseには、サーバー機能が別途必要な項目もあるが、下記項目は、**Eclipse**の標準機能/プラグインがあり、無料で利用することができる

- 静的解析
  - Findbugs
- ソースコード整形、リファクタリング支援
  - JDT, CDT
- ビルド/テスト
  - ant, maven, junit
- 構成管理
  - CVS, SVN, GIT
- bug/issue管理
  - Mylyn, trac
- ソースコード自動生成。。。
  - EMF, Xtext, Acceleo, Xpand

おわりに

- Android 4.0?のリリースが直近に控えているが、しばらくは2.X、3.Xが混在した状態となる
- Android搭載のスマートフォン/タブレットが続々とリリースされている
- Androidの適用領域が増えている

- 開発規模も複雑度も、益々増大することが予想される

- 品質を高めるためには上流行程が重要
- 案件規模が小さい今のうちに開発体制の見直しを

豆蔵 mamezou

ご清聴ありがとうございました。

No Silver Bullet