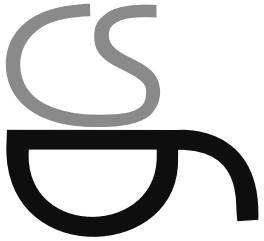
CMSI計算科学技術特論C (2015)

千葉　滋
東京大学情報理工学系研究科
創造情報学専攻



ソフトウェア工学の視点から 後編

Shigeru Chiba, U. Tokyo

最近の潮流

- 開発スピードが重要
- 独自部分に注力、それ以外は極力汎用品 (commercial off-the-shelf) を再利用
  - ソフトウェアのモジュール化が大切
    - 全て内製コードでもモジュール化は有用
  - 車輪の再発明は避ける
- 内製コードは保守性を意識
  - 再利用性、拡張性、bug-fix し易さ
  - Code clone は基本、悪

Shigeru Chiba, U. Tokyo

Lessons

- 再利用性と使いやすさ、で競争力をえる
  - そのソフトウェアを使う人は果たして多いか？
  - 利用機会の多い機能 (function) の提供
  - 再利用性の高いコード
  - 最初のとっつきやすさ
    - Use case を明確に
    - ドキュメント整備が大切
  - 保守の継続性
    - 長寿命なコード
      - 依存技術、配布形態 (CVS, SVN, git, ..)、ライセンス
  - 効果的な宣伝

Feedback

- 化学系では「単体動作アプリ」が売れる
  - 同じような計算をするアプリが溢れかえって互いにお客を奪い合う
  - Webアプリ
    - サイトごとに特注のアプリケーションを開発
    - Wordpress 等は単体アプリといえますが…

Shigeru Chiba, U. Tokyo

保守の継続は難しい

↳ バグfix, 小改良, マシンの変化

- 解？ Open Source Development
  - コミュニティ開発
    - 世界中から無償で人が集まって開発に貢献

ボランティア ?

    - 開発段階なら？
      - 新機能の追加は楽しい
      - 自身の能力の宣伝になる
      - ユーザ目線からも信頼感が高まる

      - ソフトウェアは永遠に進化し続けないといけない

Shigeru Chiba, U. Tokyo

JBoss Inc. のビジネスモデル

- 基幹ソフトウェア
  - オープンソース・ソフトウェアで無償公開
- 基幹ビジネス
  - （法人向け）チュートリアル
  - 保守サービス
    - ２次、３次受け保守サービス会社を配下にもつ
- 開発者（従業員）
  - 世界中に拡散

Apple のビジネスモデル

- 基幹ソフトウェア　MacOS X
  - ほとんど無料
- 基幹ビジネス
  - Hardware 販売
  - iTunes, AppStore, …
  - 昔の Microsoft の基幹ビジネス
    - 基幹ソフトウェアの販売

Shigeru Chiba, U. Tokyo

Low-cost な保守のために

- Modular design
  - 活発に機能拡張し続けるためにも
  - Separation of Concerns
    - 関心事 (Concern) ごとにコードをまとめる
      - 分担開発が可能に（人月の神話）
      - Bug fix も無駄がない
    - どこが将来変わりうるかを、まず分析
      - 差分を探す

Shigeru Chiba, U. Tokyo

Information Hiding

- あるいは Encapsulation
- Module interface
  - モジュールは内部を隠す
    - 変化しうる design decision を隠す
  - 外部とは限られた interface を通じて
    - 実装の自由
    - 利用者にとっての理解しやすさ

Shigeru Chiba, U. Tokyo

Modularity is not free

- 実行速度は遅くなりがち
- Crosscutting concerns の存在
  - いくつかのモジュールに跨がらないと実装できない concern
    - 例えば高速化のための機能
    - 解決策　コンパイル時に織り込む？
      - modular code を tangling code に
      - C++ templates や DSL の利用

Separation of Concerns の技法

- オブジェクト指向
  - 副作用（代入）を用いたプログラミングでは
    - データ（状態）と関数（手続き）は
      セットで提供したい
    - 例

```
MPI_Request req;
MPI_Status status;
MPI_Irecv(buf, len, MPI_DOUBLE, …, &req);
MPI_Wait(&req, &status);
```

Separation of Concerns の技法

- オブジェクト指向
  - データが中心
    - 計算を物理現象として捉える
    - 言語サポートがあるとよい
  - データと関数をセットで提供
    - 組み合わせを間違えない
    - 目的の関数を利用者が探しやすい

```
Req r = new Req();
r.irecv(…);
r.wait();
```

    - 例

```
class Req {
    MPI_Request req;
    MPI_Status status;
    void irecv(…) { … }
    void wait() { MPI_Wait(&req, &status); }
        :
};
```

Shigeru Chiba, U. Tokyo

Fortran

- データを整数で指定
  - c　内部に受信状態を表すデータがあり、
    c　その番号が req

```
Integer req
call MPI_Irecv(array, N, …, req, err)
call MPI_Wait(req, ...)
```

Fortran

- データ中心に名前を変えてみる
  - Ireceiver で関数を探せる？
    - Wait が Ireceiver_get, Isend_get, … と違う名前に
  - Integer irecv
    call MPI_Ireceiver_make(irecv, array, N, …)
    call MPI_Ireceiver_get(irecv, ...)

MPI_Wait

Shigeru Chiba, U. Tokyo

継承

- 関数の分類をクラスで表現
  - 共通部分はスーパークラス(親クラス)
    - 自動的に別名がつく？

```
class MPI_Nonblocking {
   get(Integer i, …)
}
```

```
class MPI_Ireceiver {
   make(Integer i, Real buf(:), …)
}
```

```
class MPI_Isend {
   make(Integer i, Real buf(:), …)
}
```

Shigeru Chiba, U. Tokyo