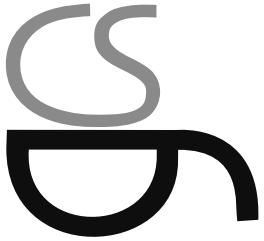
CMSI計算科学技術特論C (2015)

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 前編

Shigeru Chiba, U. Tokyo

講師について

- 専門　　計算機科学
  - プログラミング言語の設計と実装
  - ソフトウェア工学

- 計算科学との関わり
  - XEROX PARC
    - 現IBM X10責任者の Vijay Saraswat が在籍
  - 筑波大計算科学研究センター
  - JST CREST “Modularity for supercomputing”
    - ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出

シミュレータ開発改良の依頼主の事情

（大学・国研の研究者の事情）

- プログラム開発が好きなので自分で開発して研究したい。
- 多忙（教育、公募、学科、学会、雑用）のため開発の時間は不足。
- ポスドク・学生に開発させたいが研究課題には向かない。
- 学生がプログラミング言語を修得するのは容易ではない。
- 学生に開発させたプログラムの開発ノウハウが学生が卒業して消失。
- 新しいプログラミング技術を追いかける時間が不足。
- 単独で開発してきたが、長年の複雑化で開発が行き詰まり。
- プログラム開発者としての評価は論文著者としての評価より低い。
- 公募の研究予算を獲得した。計算機を買うか、ポスドクを雇うか迷う。

上記理由によりプログラム開発の業者（当社）への外注を考え始める。

渡辺尚貴、みずほ総研、CMSI計算科学技術特論C(第8回)2015年11月19日
http://www.cms-initiative.jp/ja/events/others-event/c9pj29/z5wqs8

計算機科学者≠ソフトウェア開発者

- Myth (俗説)
  - 実用的ソフトウェアの開発は論文になる
    - 「研究とは、基本的にはもの作りではなくて、ものを作るための新しい方法を見いだすこと」
      - 千葉滋「ハッカーと研究者」、オープンソースマガジン、ソフトバンククリエーティブ、pp.102-103、July 2006.
      - http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/site/?Columns
    - 「先生」と呼ばれるようになるとプログラムを書かなくなる
      - 自分は、希少人種「（歳食っても）コードを書く研究者」の一人だと思う。

この講義について

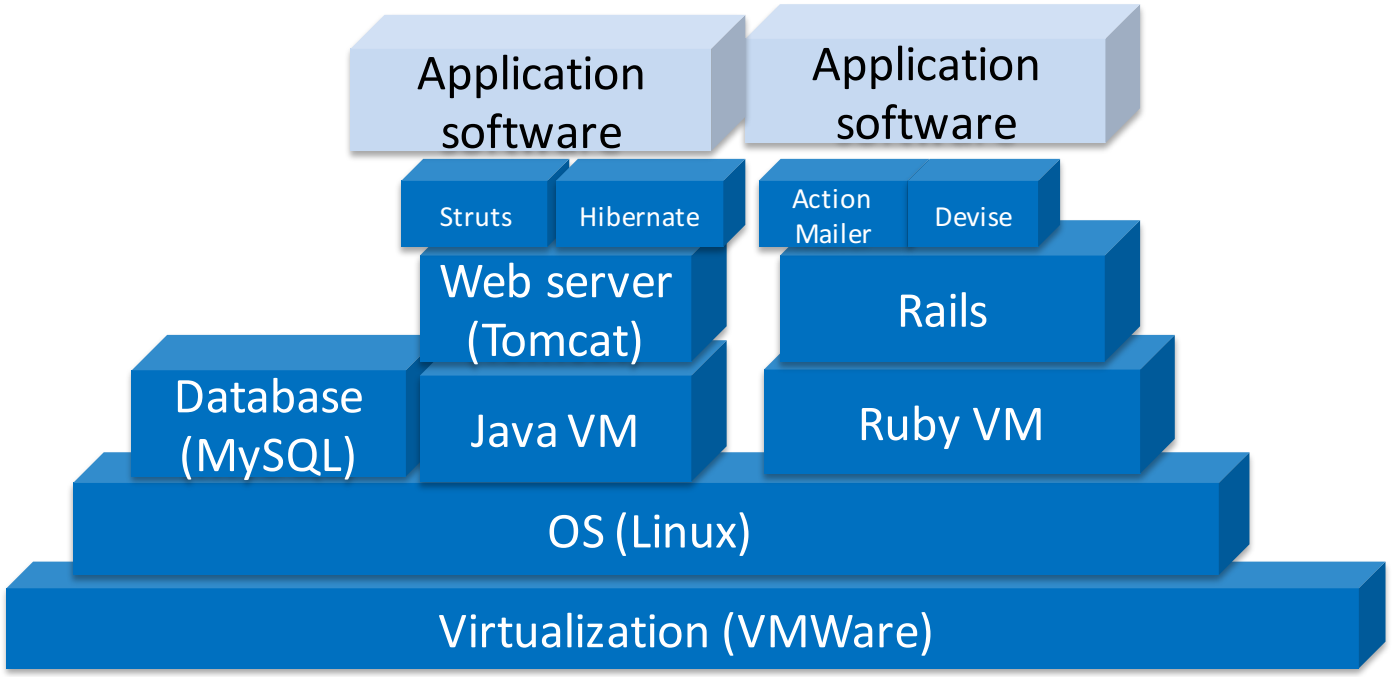
- Web 分野（に限りませんが）のソフトウエア開発
  - ソフトウェアスタック
    - いかにして「本来」巨大なシステムが構築されるか
    - 大半のコードは再利用され、開発者は独自部分に注力
  - その中で世に広く使われる「部品」をいかに売り込むか
    - 私自身の、私の周辺の、経験談
      - 前編: Javassist というライブラリについて

- 再利用に関するソフトウェア工学的な知見
  - の導入的な内容
    - なぜ新しい言語が生み出されるか
      - なぜ Fortran はバカにされるのか（まみ様ご）
      - Real Programmers don't use Pascal が元談なのか
    - ライブラリ、フレームワーク、DSL を支える要素技術
      - オブジェクト、クラス、継承、functor、overriding、templates、embedded DSLs、boilerplate code、…

Shigeru Chiba, U. Tokyo

ソフトウエア・スタック

- 必要なソフトウェア全体
  - アプリケーションを構築するために、層状に積み上げられたソフトウェア（部品）の全体



Software Development

software-stack awakens

- A long time ago in a galaxy far, far away....

ハードからソフトまで全て一社の製品で統一。
ソフトは何から何までアプリごとに専用品を内製。
選択と集中より、総合商社、百貨店的に手広く開発。
一度開発したアプリは永遠に改造・拡張。
開発者は終身雇用で開発ノウハウは属人化。
開発組織の規模（人数）がものを言う。
職人気質のプログラマが
スパゲティ・コードを量産。
コードは書き捨て、
使い回すより新規に書け。
時間をかけてじっくり開発。
車輪の再発明、上等。

最近の学生との典型的な面接

- 教員「プログラミングは好きですか？」
- 学生「はい」
- 教員「今まで書いたプログラムで
  一番大きなものは？」
- 学生「…」
- 教員「何行くらい書きましたか？」
- 学生「数えたことありませんが、
  500 行くらいだと思います」

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

Javassist

www.javassist.org

- redhat JBoss/WildFly の部品
  - Java バイトコード（仮想機械語）を実行時に書き換えるためのライブラリ
    - Apache BCEL, OW2 ASM, and Javassist
      - 当時は各陣営が重要部品として自陣営に取り込んでいた
  - 論文発表 2000年
  - オープンソース　Apache/LGPL/MPL triple
  - 世界的に非常に？多数の商用・非商用ソフトウェアが利用
    - 開発者(千葉)に直接的な金銭メリットはなし

経緯

- 1998年頃　　開発開始
  - 一応、JST さきがけ研究 (1998-2001) の一環で
- 1999年　　公開　　w/MPL
- 2000年　　論文発表
- 2003年　　JBoss Inc. 傘下に w/MPL+LGPL
  - 論文を読んだ学生が Jboss にインターンに行き、Javassist を紹介したのがきっかけらしい
  - ドキュメントが利用者目線で一番わかりやすかったから、とか
  - 日本のメディアが逆輸入して日本でも知られるように
- 2006年　　redhat が JBoss を買収
- 2011年　　MPL+LGPL+Apache に

License

- Mozilla Public License 1.1
  - Firefox 等。無保証、ソース開示、直接的な派生物も同ライセンス、関連特許も無償利用可
- GNU Lesser General Public License 2.1
  - ライブラリ用。リンク先にはソース開示が及ばない GPL.
- Apache License 2.0
  - 派生物には非適用。法人が歓迎

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

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

コードの再利用性を高める技術

- Application software
  - 基本は as is で利用。設定ファイルでカスタマイズ。
  - ソースコードを読んで自分で直接書き直してカスタマイズ。
- Library
  - 関数、手続き単位で部品として再利用
- Framework
  - 半完成品。
  - 部分的にコードを差し替えて完成品に。
- Domain Specific Language (Code generator)
  - 特定のアプリケーション分野専用の「言語」
  - （部分的に）コードを自動合成

Shigeru Chiba, U. Tokyo

Library での再利用技術

- 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);
```

Library での再利用技術

- 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

Real Programmers …

- Don’t use Pascal.
  - By Ed Post in 1983
  - It compares “real programmers,” who write in Fortran (of course!) with “quiche eaters,” who write in Pascal or other modern languages.
  - “If you can't do it in FORTRAN, do it in assembly language. If you can't do it in assembly language, it isn't worth doing.”

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, ...)
```

Shigeru Chiba, U. Tokyo

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, ...)

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

Framework での再利用技術

- 半完成品
  - 部分的なコード差し替え、追加で完成
    - Main 関数を含むライブラリ？
    - カスタマイズするので再利用の範囲が広い
  - 例
    - 積分サブルーチンに被積分関数を引数で渡す
  - 言語サポートがあるとよい
    - functor, method overriding など
    - カスタマイズ性を上げるには、何でも関数引数に
    - かえって使いにくくなる
      - ほとんどの関数引数は標準品 (default) でよい

Shigeru Chiba, U. Tokyo

差分を探す

- ライブラリ設計の基本
  - 複数のプログラムの共通部分がライブラリ
  - コード中の値が異なる
    - パラメタ化して関数に
  - コードが部分的に異なる
    - 差分を関数引数として受け取るように

Shigeru Chiba, U. Tokyo

Functor

- 関数型言語 ML, Ocaml, Haskell, …
  - いくつかの関数の集まり(組)をまとめて引数として渡せる
    - 差分に対応する関数を事前に何組か用意しておく
    - 差分に対応する関数の組を新たに書く

```
module Stencil(kernels: KernelSig) = struct
    let update grid = function
        ... kernels.halo i j k ...
        ... kernels.func subarray ...
end
```

Shigeru Chiba, U. Tokyo

Method overriding

- オブジェクト指向言語（メソッドの上書き）
  - スーパークラスのメソッドを置換

```
class Stencil {
    void update(Grid g, …) {
        … halo(i, j, k) …
        … func(i, j, k) …
    }
    float halo(int x, int y, int z) { … }
    float func(SubArray a) { … }
}
```

```
class Diffusion extends Stencil {
    float halo(int x, int y, int z) {
        return 0.0F;
    }
    float func(SubArray a) {
        return a.east() + a.west() …
    }
}
```

Shigeru Chiba, U. Tokyo

その他

- 色々なプログラミング技法
  - オブジェクトを module と見なして functor 風にする技法もある
- C++ templates
  - functor 風にも使える
  - ただし型検査はない
  - が、実行が高速になると期待できる

DSL での再利用技術

- DSL: Domain Specific Language
  - コード生成機能つきライブラリ？
  - External DSL
    - 完全に独立した新プログラミング言語
      - プログラムはその言語だけで完結して書ける
  - Embedded DSL
    - 既存言語のプログラム中に埋め込んで書く
    - 専用コンパイラで処理、実行コードを生成
      or 構文だけDSL風だが実体は既存言語のライブラリ

Shigeru Chiba, U. Tokyo

コード生成機能つきライブラリ

- 利用者は boilerplate code を書かずにすむ
  - ライブラリ設計の基本
    - 複数のプログラムの共通部分がライブラリ
    - コード中の値が異なる
      - パラメタ化して関数に
    - コードが部分的に異なる
      - 差分を関数引数として受け取るように
    - コードは異なるが類似性が高い
      - 機械的に自動生成する
      - さもなくば手で書く
        慣れれば単純作業だが、慣れるまでが大変
        単純作業は間違いをおかしやすい

Shigeru Chiba, U. Tokyo

コード生成のための道具

- コンパイラ作成 library/framework
  - ROSE compiler infrastructure
    http://rosecompiler.org
- 言語作成 workbench
  - The Spoofax language workbench
    http://metaborg.org/spoofax/
- プログラム変換 library
  - Javassist
    http://www.javassist.org

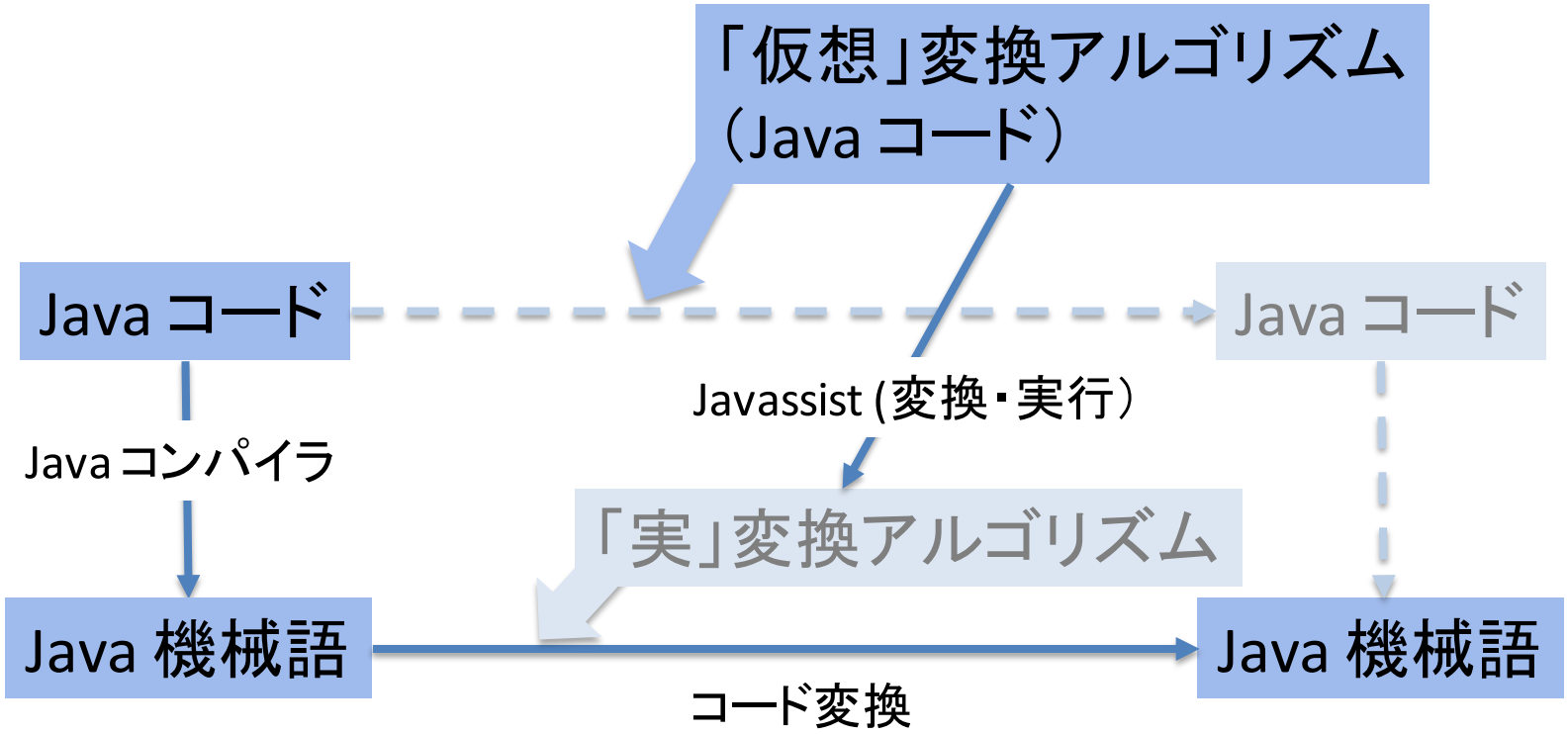
Javassist

- Java 機械語コード変換・生成ライブラリ
- 利用例
  - プログラマは仮想的なインタフェース(API)を提供するライブラリを使って書く
    - 仮想的なAPIは利用が簡単
      - 例: GPU kernel を呼び出しの前後に CPU/GPU 間メモリ転送をしない
  - ライブラリはコンパイル時などにコード変換
    - 本物のAPI向けにプログラムを自動的に書き換える
      - 例: 転送を最適化する位置に CUDAmemcpy() を挿入

Shigeru Chiba, U. Tokyo

Javassist の特徴

- 少ない前提知識で簡単に利用可能
  - Java の知識があれば Java 機械語の知識がなくてもコード変換を記述できる



まとめ

- 広く利用される library/framework
  - 高い再利用性
    - 少々の差分であればカスタマイズして利用可能に
  - 利用の仕方のわかりやすさ
    - ドキュメント整備
    - サブルーチン・関数の探しやすさ
    - 学習の容易さ
      - なるべく少ない前提知識で利用可能に

Shigeru Chiba, U. Tokyo