私が経験してきたソフトウェアビルドと
マルチスレッドプログラミング

柴田　芳樹

JaSST'16 Niigata

2016年4月15日

自己紹介

- 柴田芳樹　1959年生まれ
- 九州工業大学情報工学科＆大学院（情報工学）
- 職務経歴
  - 富士ゼロックス（株）（Xerox PARCを含め米国ゼロックス社に4年半駐在）
  - 日本オラクル（株）
  - （株）ジャストシステム
  - 富士ゼロックス情報システム（株）（米国ゼロックス社に半年駐在）
  - （株）リコー
- 主な商品化された開発
  - Fuji Xerox 6060 Workstation（1984年～1988年）
    - https://www.youtube.com/v/KGL1PadFa8M
  - Fuji Xerox Global View（1988年～1991年）
  - Fuji Xerox DocuStation IM 200 / AS 200（1993年～1996年）
  - Fuji Xerox DocuCentreシリーズ （2000年～2001年）

製品の概要

Fuji Xerox 6060 Workstation （1986年発売）

- ビットマップディスプレイ、マルチウィンドウ、文書作成、ネットワーク機能、etc
- CPU: 68000, OS: Idris (Unixクローン）、開発言語：C言語
- プロセス間通信によるプログラミング

出典：http://www.fujixerox.co.jp/company/profile/history/product.html

Fuji Xerox DocuStation IM 200（1996年発売）

- コピーとFaxの基本機能に加えてペーパーユーザーインタフェース搭載
- CPU: SPARC, OS: Solaris 2.3、開発言語：C++言語
- **マルチスレッドプログラミング**

出典：http://www.fujixerox.co.jp/company/profile/history/product.html

Fuji Xerox DocuStation AS 200（1995年発売）

- コピーとFaxの基本機能に加えて、
- 自治体窓口証明（戸籍）発行システム
- プレズマディスプレイによるタッチパネル操作
- CPU: SPARC, OS: Solaris 2.3、開発言語：C++言語
- **マルチスレッドプログラミング**
- 1998年にリコーへOEM（「イマジオ 市町村窓口証明システムSG2000」）

出典：https://web.archive.org/web/19980119153537/http://www.fujixerox.co.jp/product/hukugo/dsas200-2.html

ソフトウェアビルドの経験

- 手動ビルド

  ビルドが必要な時に、手作業でmakeコマンドなどを実行してのソフトウェアビルド

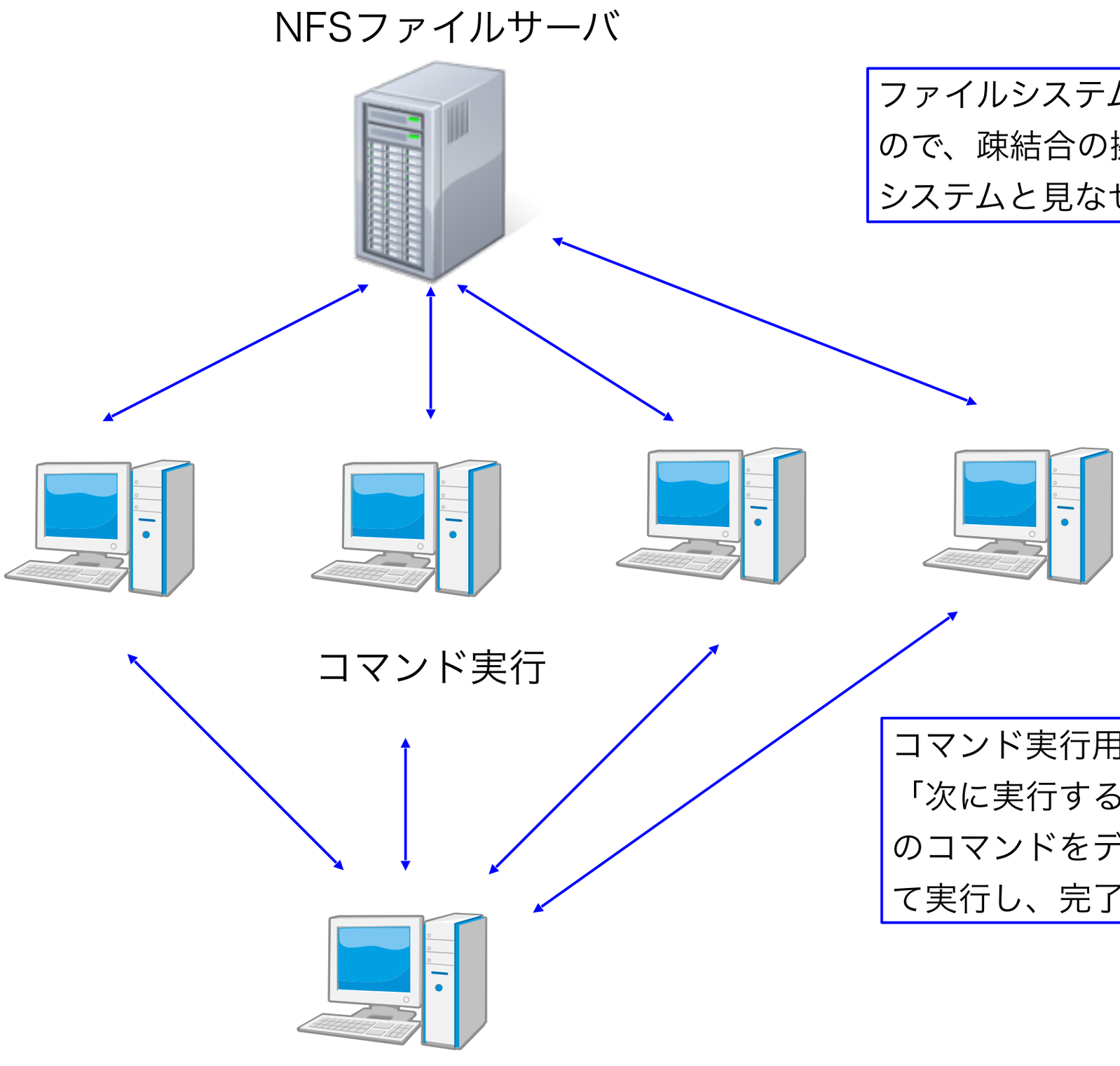
  - 九州工業大学時代
  - Fuji Xerox 6060 Workstation開発
  - Fuji Xerox GlobalView開発
    - 分散コンパイルを開発
  - Fuji Xerox DocuCentre開発（当時）

Windows NTにおける夜間ビルド
が書いてある（1994年12月発売）

- 夜間ビルド
  - Fuji Xerox DocuStation IM 200 / AS 200開発
    - 分散コンパイルも利用
  - プロジェクトN （Linux, C++言語）
    - makeの-jオプションを活用

- 継続的インテグレーション
  - リコーMFP向けJava SE搭載プロジェクト
    - 手作業によるビルド環境をJenkinsを用いた環境へ2名で三ヶ月で移行

Fuji Xerox GlobalView開発における分散コンパイル（1989年頃）



ファイルシステムはNFSで共有されているので、疎結合の擬似的なマルチプロセッサシステムと見なせる

コマンド実行用のワークステーションは、「次に実行するコンパイルあるいはリンクのコマンドをディスパッチャへ問い合わせて実行し、完了を報告する」を繰り返す

テスト駆動の経験

◆　手動・目視テスト

単体テストも含めてテストをすべて手作業で行い、結果も目視

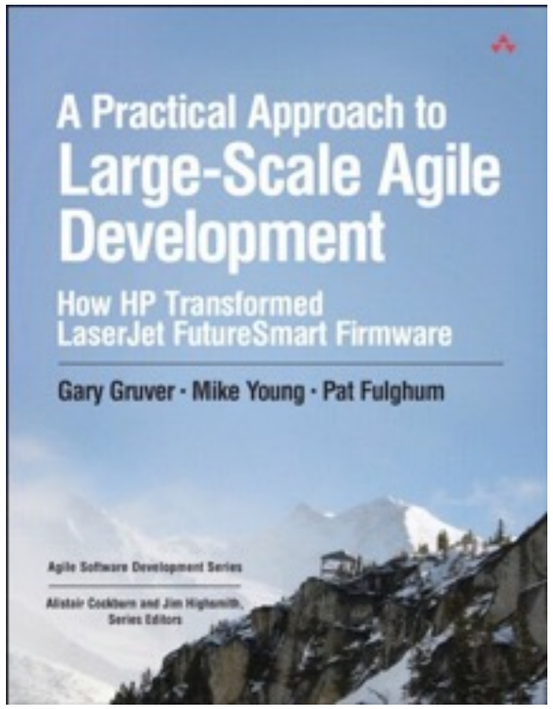
- 九州工業大学時代
- Fuji Xerox 6060 Workstation開発
- Fuji Xerox GlobalView開発
- Fuji Xerox DocuCentre開発（当時）
- Fuji Xerox DocuStation IM 200 / AS 200開発

◆　単体テスト

- Fuji Xerox DocuCentre向けC++ライブラリー開発
- リコーMFP向けJava SE搭載プロジェクト

◆　システム完全自動テスト

- デジタル複合機（MFP)（コピー、ファックス、プリンタ）のコントローラソフトウェアのPC上での完全自動テスト開発（2003年～2009年）



HP社のMFP開発は先端を走っている

マルチスレッドプログラミングの経験

- 非マルチスレッドプログラミング
  - Fuji Xerox 6060 Workstation開発（すべてプロセス間通信）

- マルチスレッドプログラミング
  - 九州工業大学時代（マルチタスクOSであるMP/Mを使用）
  - Fuji Xerox GlobalView開発
  - Fuji Xerox DocuStation IM 200 / AS 200開発
  - Fuji Xerox DocuCentre向けC++ライブラリー開発
  - 他3件

マルチスレッドプログラミングにおける重要な４要件

- きちんとしたレビュー
  - マルチコアおよびマルチスレッドプログラミングのきちんとした経験および知識を持つ人が設計やコードをレビューしていること
    - マルチスレッドプログラミングは、逐次的プログラムと比較して、経験の浅いひとにとっては直感に反する動作をする
- 自動テストによるテスト
  - 自動テストが整備されていて、いつでもテストを自動実行可能であること
    - 手作業で一回動作したとか100回動作したからと言って、プログラムの正しさは保証されない。
    - 機能を確認するためのテスト設計がきちんとされていること
- 様々な環境での長時間ランニングテスト
  - 製品がシングルコアであっても、マルチコア環境で長時間ランニングテストを繰り返す（開発者のPCを総動員して、平日夜間・週末も）
  - 同時にシステムにさまざまな負荷（CPU負荷、HD負荷、メモリ負荷）をかけて動作させる
- ハングしたらその場で徹底して調査
  - ログを入れてもう一度再現させるという対応は避けて、その場で徹底的に調査する

マルチスレッドプログラムのテスト

- 継続的インテグレーションは必要条件ではあるが、十分条件ではない
  - CIサーバでの1回のテストによる合格は、プログラムの正しさを保証しない
  - システム全体を自動でテストできるように最大限の工夫をする
- 開発者の開発用PCを最大限に活用する
  - 夜間、週末に開発用PCを遊ばせるのではなく、自動テストを様々な環境で動作させる
  - 開発用PCのスペックをケチらない。開発しながら仮想環境で自動テストを実行できるぐらいのスペック
- 朝、出社してテストが停止していたら、調査を最優先業務とする
  - 二度と発生しないかもしれないので、最優先で調査させる
  - 原因が判明したら必ず「再現テスト」を作成してから、修正を行わせる
- 起きた障害をきちんと記録して、分析する
  - 開発と並行して、原因分析・対策の検討を開発グループ内で開発活動の一環として実施する
  - プロジェクトが終わってから分析するのでは、開発中に改善に取り組めない

どうしてミケランジェロになれないんだ

マネージャは、チームの能力が向上することを
ひそかに期待しながらツールを調達する。

ツールのコストは、ツールのコスト以上に高くつく。

— ドロシー・グラハム

「たがねは買ってやった。なのに、どうしてミケランジェロになれないんだ？」すぐに生産性を高めようと必死の組織では、そんな問いかけが聞こえてくるが、そうした組織にかぎって、能力よりも給料の安さで人材を雇う。ミケランジェロ組織にはかならずと言っていいほど、買ったきり積まれたままのツールの山がある。

　ツールが便利なことは言うまでもない。適切な使い手に渡れば、すばらしく生産性を高め、ツールがなければできなかったことを成し遂げられる。しかし、ツールの作り手も言っているはずだが、使いこなすためのスキルがあることが必須条件である。たがねは、ミケランジェロが手にとらなければ、へりの鋭い金属片にすぎない。

『アドレナリンジャンキー』

継続的インテグレーションは「たがね」に過ぎない

失敗する継続的インテグレーション

- マネージャが継続的インテグレーションに関心を払わない
  - ビルド失敗が放置される
    - 開発者がビルド失敗に関心を持つには、マネージャが関心を払わないといけない
    - ビルド失敗については、その原因の調査と対策の検討をマネージャは主体的に行わなければならない
  - 静的解析ツールの結果に関心がない
    - 警告に関心を払って、その原因と対策を考えなければならない
    - 個々のソフトウェアエンジニアが常に無意識に関心を払うようになるまで、マネージャは日々関心を払ってエンジニアに言い続けなければならない
    - スキル不足による警告の増加ならば、スキルアップの施策を検討しなければならない
  - 改善に興味がない
    - さらに改善する余地があれば、そのことを指摘して、改善を指示しなければならない

個々のソフトウェアエンジニアのスキルを向上させるための道具であり、低いスキルを補ってくれるものではない

継続インテグレーションは強みではなくなった

Subversion/Gitなどを使用したソースコード管理、Jenkinsを使用した継続的インテグレーション、様々なxUnitフレームワークを使用した自動テストなどをソフトウェア開発組織として実践することは、今日では、その開発組織の技術的な強みではありません。

それらを実践しないことが、ソフトウェア開発組織の「弱み」なのです。また、組織としてそれらの実践を推進しない、あるいはサポートできないマネージャも「弱み」となります。さらに、大規模なソフトウェア開発組織においては、それらのためのインフラ整備をプロジェクトごとに立ち上げなければならず、サポート部門が存在しないことも弱みとなります。

実践することは、いわゆる「作業」をコンピュータ化することであり、その分、エンジニアは創造的な活動に注力することが可能となります。きちんとしたレビューによるコード品質や設計品質の向上に費やしたり、自動テスト品質向上のための様々な工夫を行う活動に費やしたりすることが可能となります。

従来の(1990年代終わりぐらいまでの）ソフトウェア開発では、「作業」と「創造的な活動」がどちらも人の手で行われていることが多く、「作業」に起因する遅れが「創造的な活動」の時間を圧迫することも起きていました。言い換えると、「作業」部分をコンピュータ化することで、その開発組織はより「創造的な活動」に注力できることになります。技術力が高いとか低いとかは、この創造的な活動の内容や結果を指して言うのであり、継続的インテグレーションを行っていることを指したりはしません。

「作業」を今もって人の手による作業として行っているようなソフトウェア開発組織にとっては、冒頭に述べた事柄を実践しないことが、今日では「弱み」となります。

http://yshibata.blog.so-net.ne.jp/2012-11-02

Q&A

電子メール： yshibata@ca2.so-net.ne.jp
ブログ：http://yshibata.blog.so-net.ne.jp/
Twitter：@yoshiki_shibata