ソフトウェア工学のパラダイムシフト

Immaterial Labor の視点から

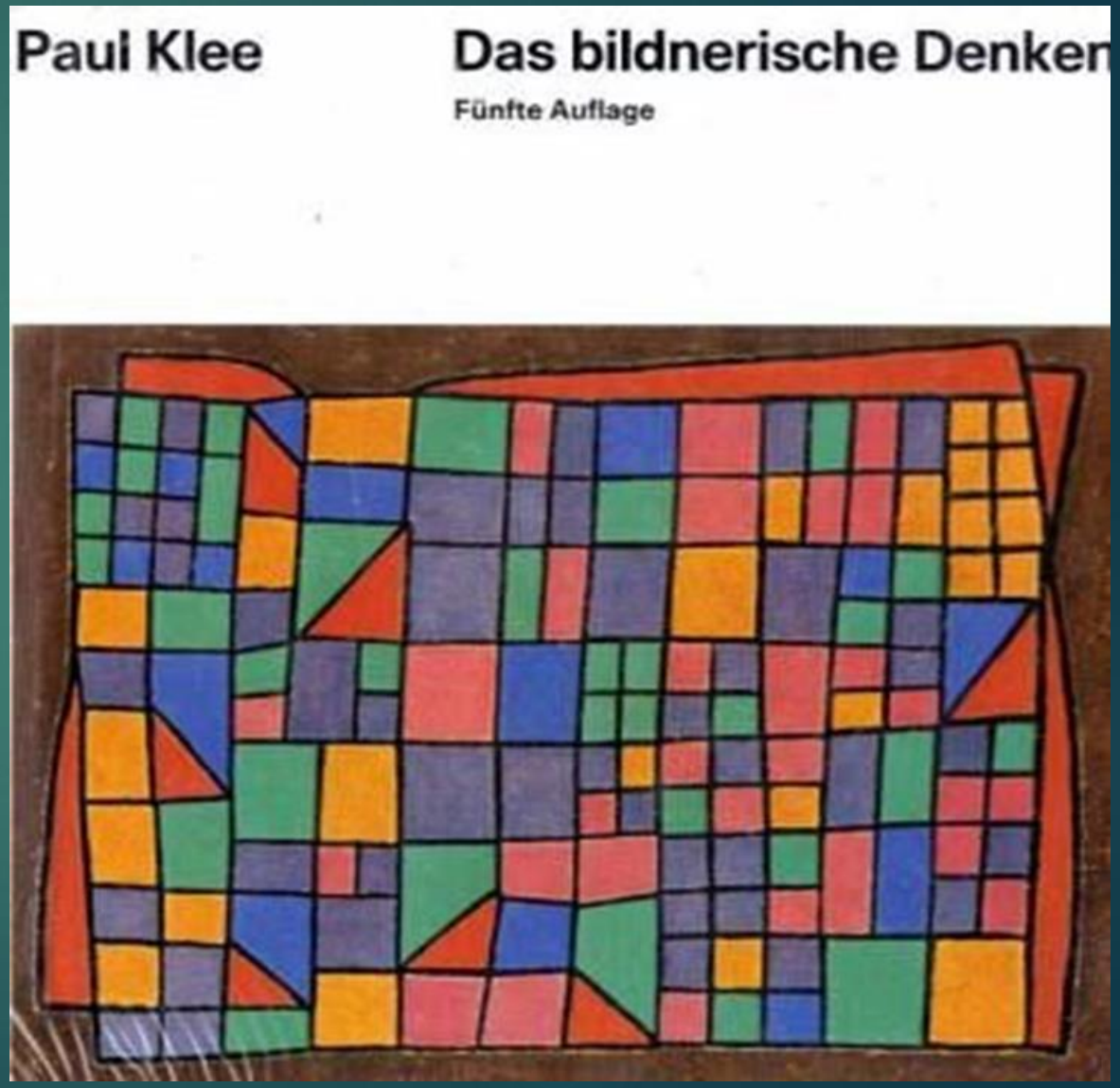
岸田 孝一

SRA

DEC 12, 2014 @ FOSE WS(霧島)

自己紹介

- 1936年，東京生まれ.
- 東京大学理学部物理学科天文学課程中退
- 6年間の大学時代，勉強はせず，
  絵を描くことに夢中だった



パウル・クレーのことば

- 美術とは,
  目に見えるものを
  再現することではない.
  見えないものを見えるようにするのだ.

  その意味で,
  グラフィック・アートは抽象に適している.

抽象表現主義

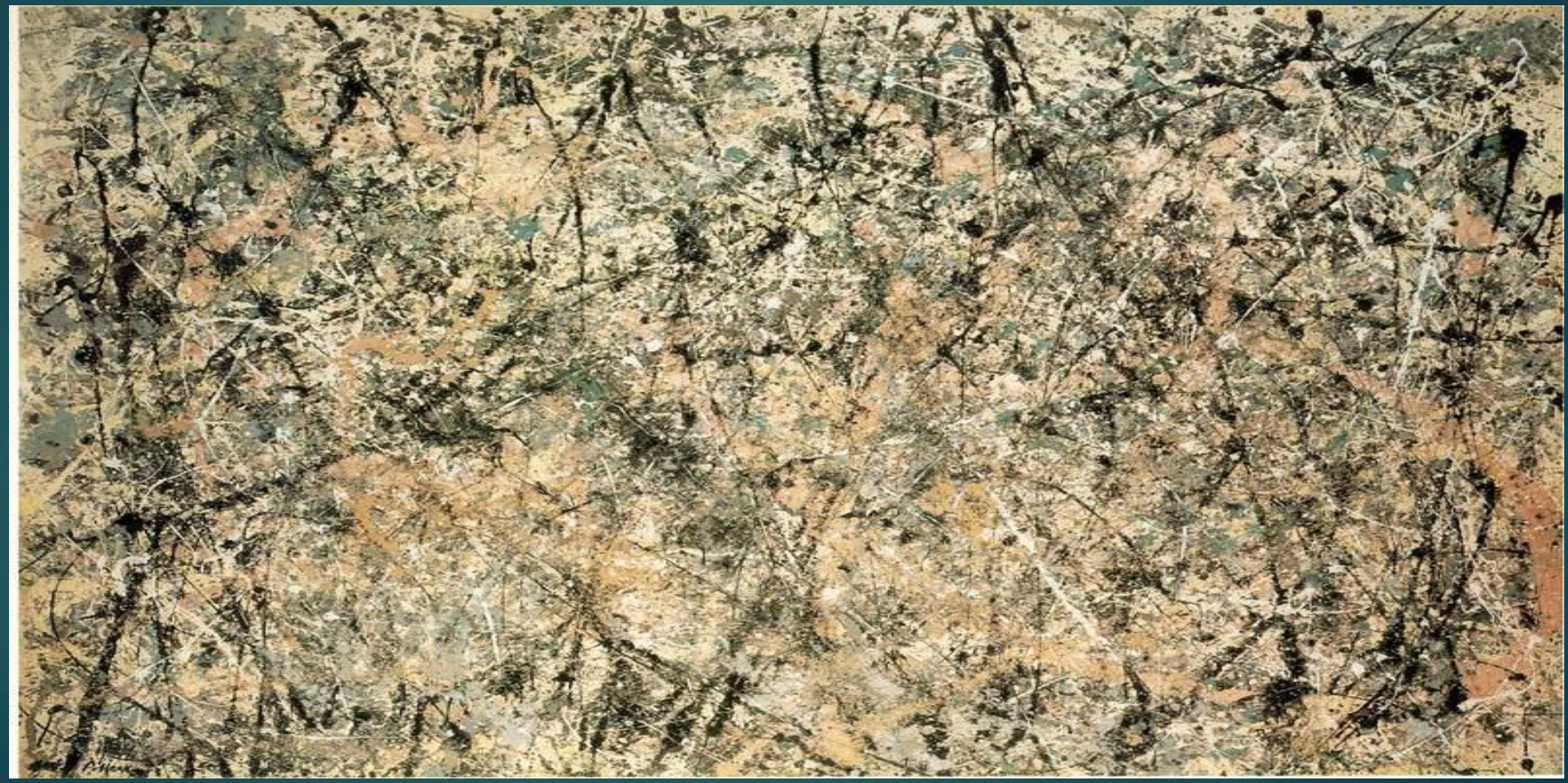
- 「何かを絵に描くのではなく，ただ絵を描く」のだという主張

- 抽象表現主義のほんとうの原動力は，絵画をその純粋な語彙（色や形）に還元し，作品がどう作られたかということ以外の経験が，鑑賞者の心のなかに喚起さ れないようにすることであった．歴史や心理その他のすべてがそこでは排除される． ある意味できわめて手軽なパラダイスが提示され，見るものが，視覚を通じて永遠にそのなかへ逃げこむことを可能にする．

  – ヒルトン・クレーマー

先駆者

- Jackson Pollock -



そのころのわたしの作品

クレーのバウハウス講義における前置き（1921年）

- 自然科学（たとえば化学）における分析
  - 与えられた「プロダクト（食品や薬）」を構成要素に分解し，それぞれが持つ特性を調べ，全体としての対象物の属性を判定する.
- 美術における分析
  - 分析対象の作品を要素に分解したりはしない．その作品がどのような「プロセス」によって創造されたのかを追求する.

プログラミングについての最初の認識

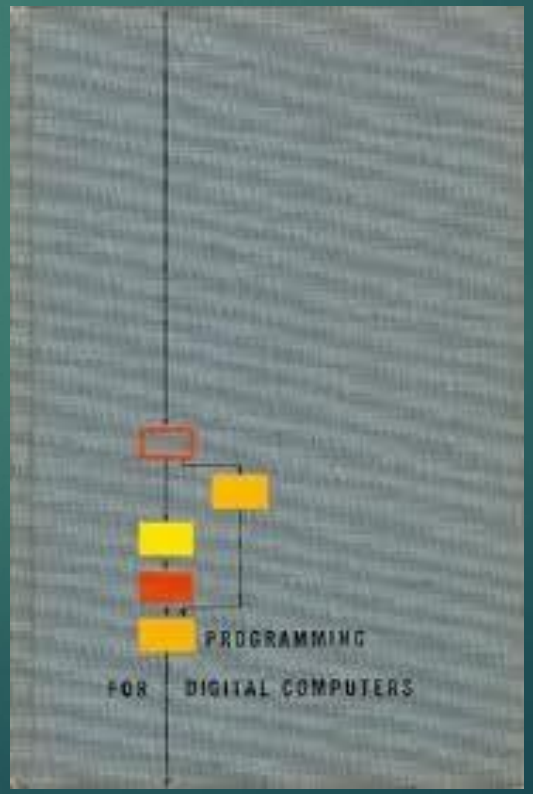
- 偶然のきっかけ（あるセミナー会社での英文資料翻訳のアルバイト）から，プログラマとしての生活を始める

- コンピュータの中で起こるであろう
  目に見えないプログラムの実行プロセスを
  見えるようにすること

- それは抽象絵画によく似ている.

プログラマとして最初に考えたこと

- スパゲティ・コードや蜘蛛の巣フローチャートの氾濫をどうするか？
- とりあえずの対処法は，モジュールを用いた階層的フローチャート
  - Joachim Jeenel: “Programming for Digital Computers”, (MacGraw Hills, 1959)



そして，構造化の衝撃

- プログラム構造化定理
  - C.Bohm 他，“Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules”, CACM May 1966.
  - １つの入り口と出口を持つプログラムの構造は，すべて，連接・分岐・反復の組み合わせに還元できる.

それぞれの構造化プログラミング

- 構造化定理に触発されたいくつかの動き
  - Edgar Dijkstra (オランダ)
    プログラムの正しさを読んで確認できるようにすべき.
    有名な GoTO Letter
  - Michael Jackson (イギリス)
    プログラムの構造をアプリケーションの構造に一致させる
    データ構造に着目
  - 岸田孝一(日本)
    アプリケーションから独立したプログラム構造の追求
    そのプログラム・ジェネレータがSRAの最初のプロダクト

その後いろいろなことがありました

- 1970～80年代：　ICSEコミュニティに巻き込まれる.
  （当時日本にはソフトウェア工学は存在しなかった）
  4th ICSE から PC member
  9th ICSE Program Co-Chair (1987)
  10th ICSE Keynote Speech (1988)
  IPSW に第１回（1984) から参加
  6th IPSW in 函館の Local Arrangement
- Unix を導入してブームの引き金を引く (1980)
  Unix ユーザ会の設立 (1983)
  Junet & Wide に協力してインターネットの普及に協力.

いろいろなこと（続き）

- SIGMA騒動での政府とのケンカ

  ひまができたので鳥居・片山・斎藤先生たちを誘って，

  国際･産学共同研究プロジェクト SDA を立ち上げる．

  これが日本でのソフトウェア工学運動の出発点だった．

  打倒 SIGMA を旗印にソニーと組んで NEWS Workstation を開発．

  Unix Review 誌のインタビューがきっかけで Richard Stallman と友人になる．

- ソフトウェア技術者協会（SEA）の設立 (1985年)

  1987年から20年以上にわたって毎年中國を舞台に国際会議を実行．

  ACM/SIGSOFT から Distinguished Service Award 受賞 (2001)

さて，ソフトウェア工学のこれから

ACM FoSER Workshop (2010) への Paper

Through the Looking Glass of Immaterial Labor

Yunwen Ye (SRA) Kumiyo Nakakoji (SRA)

Yasuhiro Yamamoto (TITech) Kouichi Kishida (SRA)

われわれの論文の要旨

Immaterial labor, which is a philosophical concept established by Maurizio Lazzarato and others for understanding the post-Fordism industry, refers to the process of producing the informational and cultural contents of a commodity. Through examining software development and software-intensive society with the lens of immaterial labor, this paper aims to make a first step of establishing a new theoretical framework to understand

(1) how to evaluate values of software systems,

(2) how such values are created, and

(3) how software development should be organized to create such values.

不発に終わったメッセージは：

De-“Software Engineering”

- Material Labor のパラダイムに支配されたこれまでのソフトウェア工学には未来はない！
  - 開発作業の細分化
  - 開発工程の流れ作業化
  - 個々の技術者の重要性軽視
- いまや Immaterial Labor へのパラダイム・シフトが時代の流れである
  - ソフトウェア開発のモードもその方向に変わりつつある

Immaterial Labor
（無形労働 または 非物質的労働）

- ネグリ & ハート『帝国』（以文社，2003）

  きわめて政治的
- ラッツアラート『出来事のポリティクス』（洛北出版，2008）

  やや政治的
- その抜粋：

  http://www.generation-online.org/c/fcimmateriallabour3.htm
- 伊藤昌夫さんによる日本語訳：

  http://sea.jp/blog/wpcontent/uploads/2011/03/Lazzarato_Immaterial_Labour_japanese_0.83.pdf

Maurizio Lazzarato

Maurizio Lazzarato, 1996

Immaterial Labour

- 資本主義社会を支配してきたテイラー方式による仕事の形態に変わる新しい生産様式を理解するための思想的枠組み
- 商品に内包される情報あるいは文化を作り出す労働
  - サービス，知識，コミュニケーションの創造
  - サイバネティクスやICTの技能を必要とする
  - 文化的/美学的標準やファッション，趣味，消費行動さらには社会全体の動向に影響を与える
  - 商品を利用するユーザの情緒的ニーズ（感情，興奮，他とのつながり，etc)を満足させる

生産・流通・消費

- フォード方式
  - 部品および工程の標準化による大量生産
  - 製造プロセスの革新がポイント
- トヨタ方式
  - 市場調査にもとづくタイムリーな商品の製造と流通
  - マーケティングとの密接な連携が革新のポイント
- 無形労働における生産
  - 消費者サイドにおける価値観の変革を促す
  - 新しい消費ニーズを作り出すことがポイント

物質的労働パラダイムの普遍化

- コストや生産性などの定量的指標がすべてに適用される.
- 工業のみならず，農業や商業もにも「工場パラダイム」が浸透する.
- さらには，人びとの日常生活も「工場パラダイム」で扱われる.
- たとえば「旅行」：　松尾芭蕉の『奥の細道』や十返舎一九の『東海道中膝栗毛』を，新幹線や飛行機を使った現代のツアーと比較してみれば？

Immaterial な俳句

蝶鳥の
知らぬ花あり
秋の空

松尾芭蕉の作と伝えられる
（ただし，存疑）

この花はいったいどんな花なのか？

ソフトウェアの生産と消費

- ソフトウェア開発は無形労働の典型である.
- 使うことが作ることに繋がる
  - ソフトウェアの価値はそれを使うことによって決まる
  - どんなソフトウェアを作ればよいかは事前に確定できない
- ソフトウェア開発の目的は新しいニーズを作りだすこと
  - ユーザにとっての現実を技術的・社会的・文化的に変革し豊かにする
  - 絶え間なく変化し続ける仕様こそが開発のガイドライン

ソフトウェアの価値と品質

- ３つの次元がある
  - 客観的次元
    - 機能を重視し，予測と制御によって評価する
  - 社会的次元
    - 開放性に注目し，コミュニケーションと解釈によって評価する
  - 主観的次元
    - 感性を重視し，美的あるいは感情的経験によって評価する
- これまでは客観的評価に偏りがちだったのでは？

Immaterial Labor のもう一つの例
工業デザイン

- 「広島が生んだデザイン界の巨匠：栄久庵憲司の世界展」
  広島県立美術館 2014/11/18-12/23
- 工業デザインは「ものづくり」の一環だが，対象とするモノは Material，しかしデザインは Immaterial

SEA での Immaterial 討論

- 2007年７月： IWFST in 中国・杭州
- 2007年１２月： SEA Forum in 東京
- 2008年６月: WG @ SS2008 in 香川
- 2008年７月: Workshop in 前橋
- 2009年６月： WG @ ＳＳ2009 in 札幌
- 2010年６月： WG @ SS2010 in 横浜
- 2011年６月： WG @ SS2011 in 長崎
- 2012年６月：Workshop in 足利

Net 上に公開されている討論の成果

- ラッツアラート論文の日本語訳
  - http://sea.jp/blog/wp-content/uploads/2011/03/Lazzarato_Immaterial_Labour_japanese_0.83.pdf
- SS2011 での WG 討論のレポート
  - http://sea.jp/?p=677#more-677

パラダイム・シフトの予兆

Christiane Floyd :

“A Paradigm Change in Software Engineering”, ACM SIGSOFT News Letter, April 1988.

http://portal.acm.org/citation.cfm?id=43851

２つの視点

- プロダクト指向
  - ソフトウェアを「もの」として扱う
  - 物質的労働の視点
- プロセス指向
  - ソフトウェアはプロセスの道具として考える
  - 無形労働の視点

これらは互いに相補的

プロダクト指向

- ソフトウェアは，それ自体で成立している「もの」である.
- それは，いくつかのプログラムおよび関連ドキュメントから構成されている.
- ソフトウェアの持つ特性は，所与のハードウェア環境のもとで評価される.
- ソフトウェアの利用法はあらかじめ定められており，したがって，その開発仕様は事前に確定されるべきである.

プロセス指向

- ソフトウェアを，人間の行動・学習・コミュニケーションとの関連でとらえる.
- それらの活動は，進化し続ける世界の中で，たえず変化するニーズをともないながら，行われる.
- 開発フェイズでは，ソフトウェアはそうした人間活動をモデル化したものとして扱われる.
- 利用フェイズでは，ソフトウェアはそうした人間活動を支援しまた制約する道具として考えられる.

2つの視点の比較

- プログラムについて
- システムについて
- 品質について
- ソフトウェア開発について
- ドキュメントについて
- メソッドについて

比較のいくつかの例

プロダクト指向

- プログラムは形式化された数学的対象である
- 開発者は対象システムの外側に位置している．開発と利用とは明確に区別される
- 品質はソフトウェア・プロダクトの属性である
- プログラムの理解はドキュメントだけにもとづいてなされるべき．ドキュメントはプロダクトの一部．
- メソッドはそれを利用することによって所定の結果をもたらすプロダクトである

プロセス指向

- プログラムは人間活動を支援する道具あるいは作業環境である
- 開発者はシステムの一部として組み込まれている．開発と利用とは相互に影響しあう．
- 品質はソフトウェアが利用されるプロセスの属性である
- プログラムの理解は人間どうしのコミュニケーションによってなされる．ドキュメントはその補助手段
- メソッドなど存在しない．ただ特定のメソッドが開発・利用されたプロセスだけがある

パラダイム・シフトの背景：
M.M.Lehman のソフトウェア進化論

2 種類のソフトウェア：

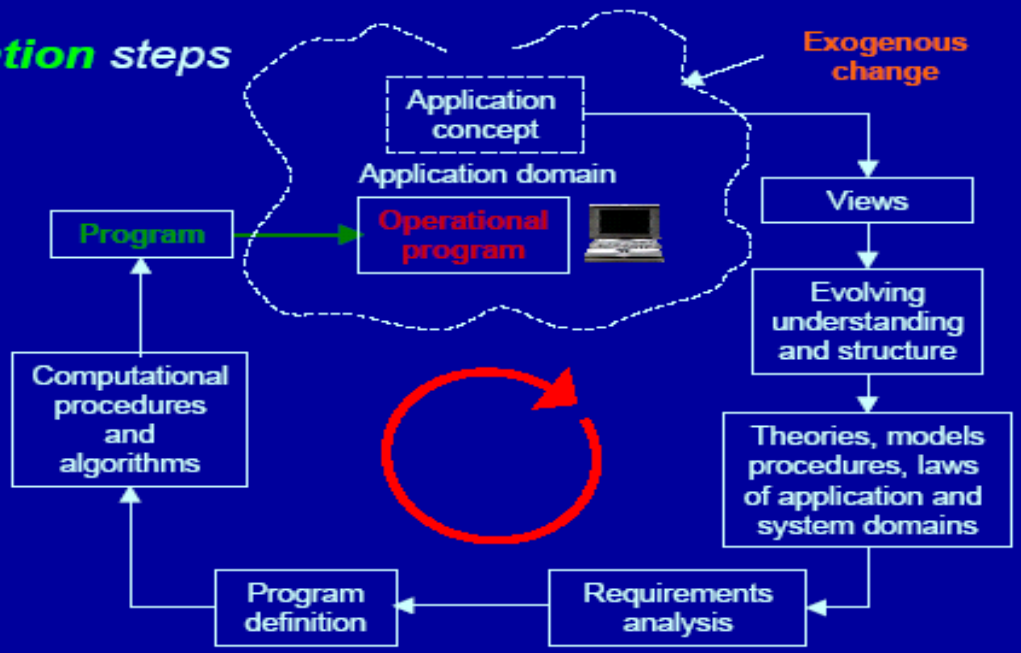
- S-Type: 仕様が確定でき，変化しない.
- E-Type： 現実世界のアプリケーションに組み込まれていて，時間とともに進化してゆく.

Eプログラム

High Level View of E-type Process

*Series of **development** and **implementation** steps*

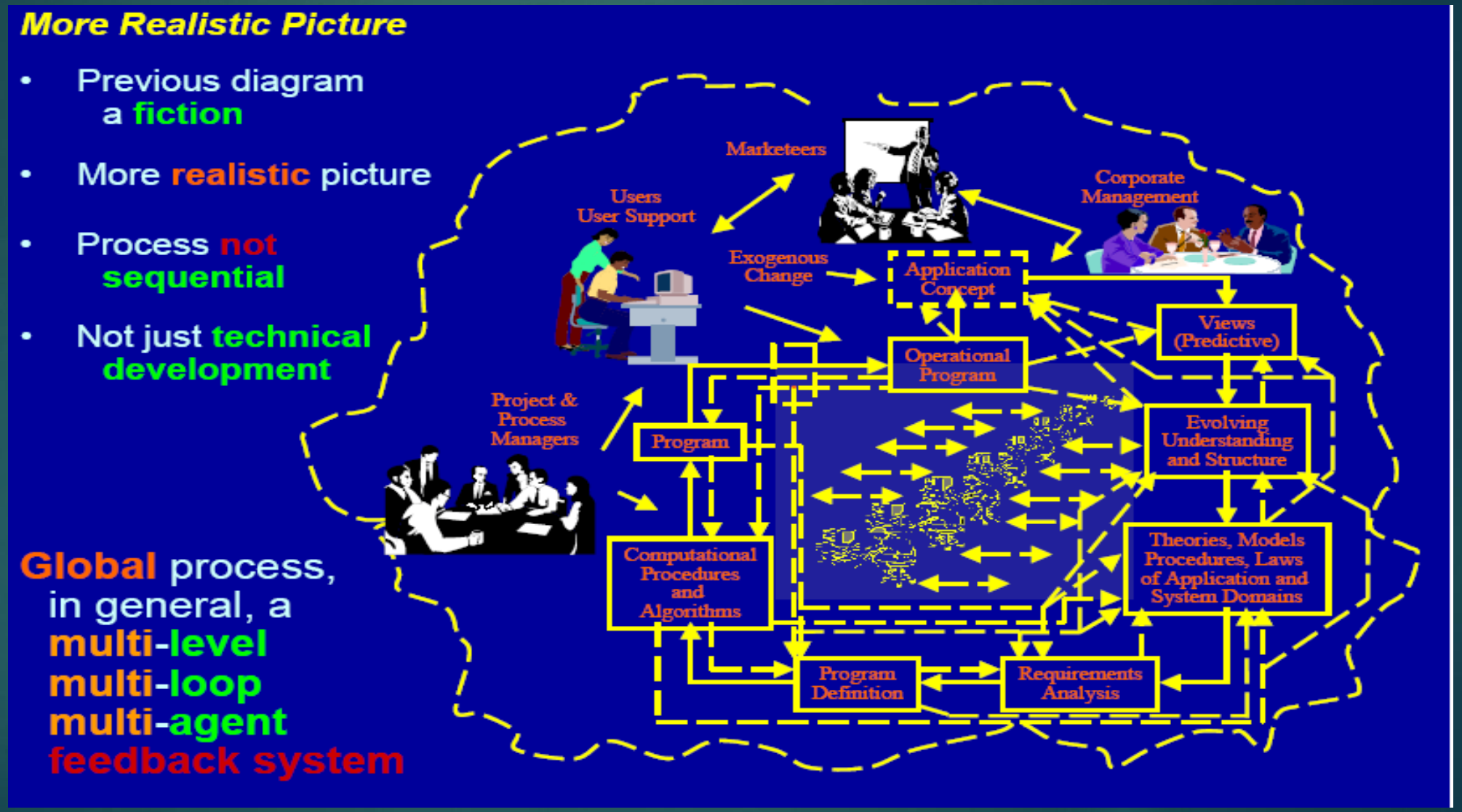
- Culmination: **validated program**
- Final step: **installation** which changes **domain** and **operation** which changes **application**
- All in **changing real world** domain



- **Installation** and introduction into **usage** closes major **feedback loop** that drives **iterative process** to **release** sequence of **upgrades** to users

Driver of **continuing** software ***evolution***

現実のイメージ


More Realistic Picture
• Previous diagram a fiction
• More realistic picture
• Process not sequential
• Not just technical development
Global process, in general, a multi-level multi-loop multi-agent feedback system
Marketeers
Users
User Support
Corporate Management
Exogenous Change
Application Concept
Views (Predictive)
Operational Program
Project & Process Managers
Program
Evolving Understanding and Structure
Computational Procedures and Algorithms
Theories, Models Procedures, Laws of Application and System Domains
Program Definition
Requirements Analysis

Multi-Level, Loop & Agent というメタファ

- ソフトウェア開発・進化のプロセスは大渦巻き（エドガー・アラン・ポーの『メールシュトローム』）.

- だれもがその渦の中に巻き込まれている.
  - 開発者も
  - マネージャも
  - ユーザも
  - そして，モデル化や分析を論ずる研究者も

神の視点は可能か？

システム要件定義について

B.Nuseibeh & S.Easterbrook

“Requirements Engineering: A Roadmap”

Paper presented at ICSE2000

仕様開発のガイドライン

1. 矛盾のない完璧な仕様作りを目指すことはナンセンスである.
2. 仕様のモデル化や分析は，そのシステムが運用される組織あるいは社会環境と独立に行うことはできない.
3. システム機能仕様の確定よりも，運用環境におけるシステムの動作特性に注目すべきである.

システム検証のガイドライン

1. 要求仕様を満足する「正しい」プログラムは第二義的な目標でしかない.
2. ソフトウェアの検証は，それが実際に運用される組織あるいは社会環境と独立に行うことはできない.
3. システムの機能仕様を云々するよりも，その運用環境における動作に注目すべきである.

モデル＆メトリクス

- これまでに提案され議論されてきたプロセス・モデルやメトリクスは，すべて，古典力学的な「神の視点」を前提としている.
- それは，ある種の限定的な（たとえば開発プロジェクトの内部だけを考える)場面では実用的かもしれない.
- しかし，より広い視野に立ってソフトウェア進化プロセス全体を考える上では役立たない.
- 量子物理学における「不確定性原理」をソフトウェアに適用してみたらどうなるか？

神の視点からのプロセス論議

- 上からの目線でプロセスを眺める.
- 類似性または同一性に着目して，プロセスをその構成要素に分解し，全体を木構造で再構成する.
- ここで「同一性」とは，同じような行動が何度も繰り返されるという意味（ex:分析，設計などのサブプロセス，あるいは CMM の KPA)
- しかし，....

差異と反復

- ジル・ドゥルーズ（フランスの哲学者）の指摘：
  - 同一のでまごとが反復されることは決してない.
  - ただ「差異」だけが反復される.
  - 同一性を経由せずに，直接「差異」だけを考えることが重要である.
    - 「差異と反復」上下（河出文庫，2007）

「三物五類」の説

18世紀初め浪速の青年思想家・富永仲基の提唱

「およそ，言に類あり，世あり，人ある，これを言に三物ありと謂ふ．一切の語言，解するに三物を以てする者は，わが教学の立てるなり．いやしくも，これをもってこれを求むるに，天下の道法，一切の語言は，いまだかつて分かれずんばあらざるなり．故に云く，三物五類は立言の紀と，これなり」

参照：宮川康子『富永仲基と懐徳堂 – 思想史の前哨』（ぺりかん社）

ソフトウェア工学の三物五類とは？

- 世（どんな時代？）：

  20世紀も半ばを過ぎて，科学万能主義のパラダイムが爛熟し，そろそろ衰退期に近づいていた.

- 人（だれの提案？）：

  大規模ソフトウェア開発の相次ぐ失敗に悩まされていた欧米の技術者やプロジェクト・マネージャたち.

- 類（レトリック？）：

  それまでに存在していたMaterial Labor における科学的管理工学をソフトウェア開発にも適用しようとした．仲基のいう「張」にあたる．Winston Royce の Waterfall 論文はその失敗を予見していた.

仲基が提唱したもう１つのテーゼ：「加上」の原理

- すべて新しい言説は旧来の説を凌駕しようとして，より古い言説を拠り所にする.
  - 初期のソフトウェア工学：　それまでに成功し確立されていた諸工学をモデルにした.
  - オブジェクト指向：　あるノルウェイの研究者いわく「オブジェクト指向の起源は SIMULA 言語ではなく，マックス・ウェーバーの「理想的官僚制度論」である.
  - いわゆる SPI の指導原理 CMM：　16世紀中國の新儒家・朱子が説いた「修身・斎家・治国・平天下」のパラダイムそのまま.

「世」の移り変わり

- 汎用大型機からワークステーション，NotePCへ，そしてSmart Phone へ
- Local Net から Internet へ，そして Cloud へ
- いくつもの指摘
  - ICSE13 in Honolulu での W.Scacchi や T.Wasserman の発表
  - FSE14 in Hng Kong での A.Wolf や M.Lam の講演

われわれにとって必要な「加上」とは？

- われわれの仕事は単にソフトウェアという１つの有形なプロダクトを製造することではない.
- われわれが従事している仕事の中心的な部分は，開発者とユーザとのあいだの密接なコミュニケーションおよび相互学習のプロセスから構成されている.
- そのような視点に立つことによって，絶えず変化し続けるアプリケーション環境への適応を行うことが可能になる.

CSCWコミュニティからの指摘

Activity Theory の立場から見て

- 次の3種類の開発支援をはっきり区別すべきだろう
  - （１）Control 支援
    - ふつうのプロジェクト管理支援環境
  - （２）Cooperation 支援
    - いわゆる CSCW支援環境
  - （３）Co-Construction 支援
    - ユーザ・開発者一体となったプロセス革新の支援

Floyd Paper の三物五類

- この論文は，次の本の１つの章として書かれた：

  “Computers and Democracy

  – A Scandinavian Challenge”

  Dover Pubishing Co.Ltd, England 1987

- Structured Programming, Object Orientation, Participatory Design, Immaterial Labor, その他の新しい発想にもとづく問題提起がなぜヨーロッパからなされたのか？

その後の注目すべき展開

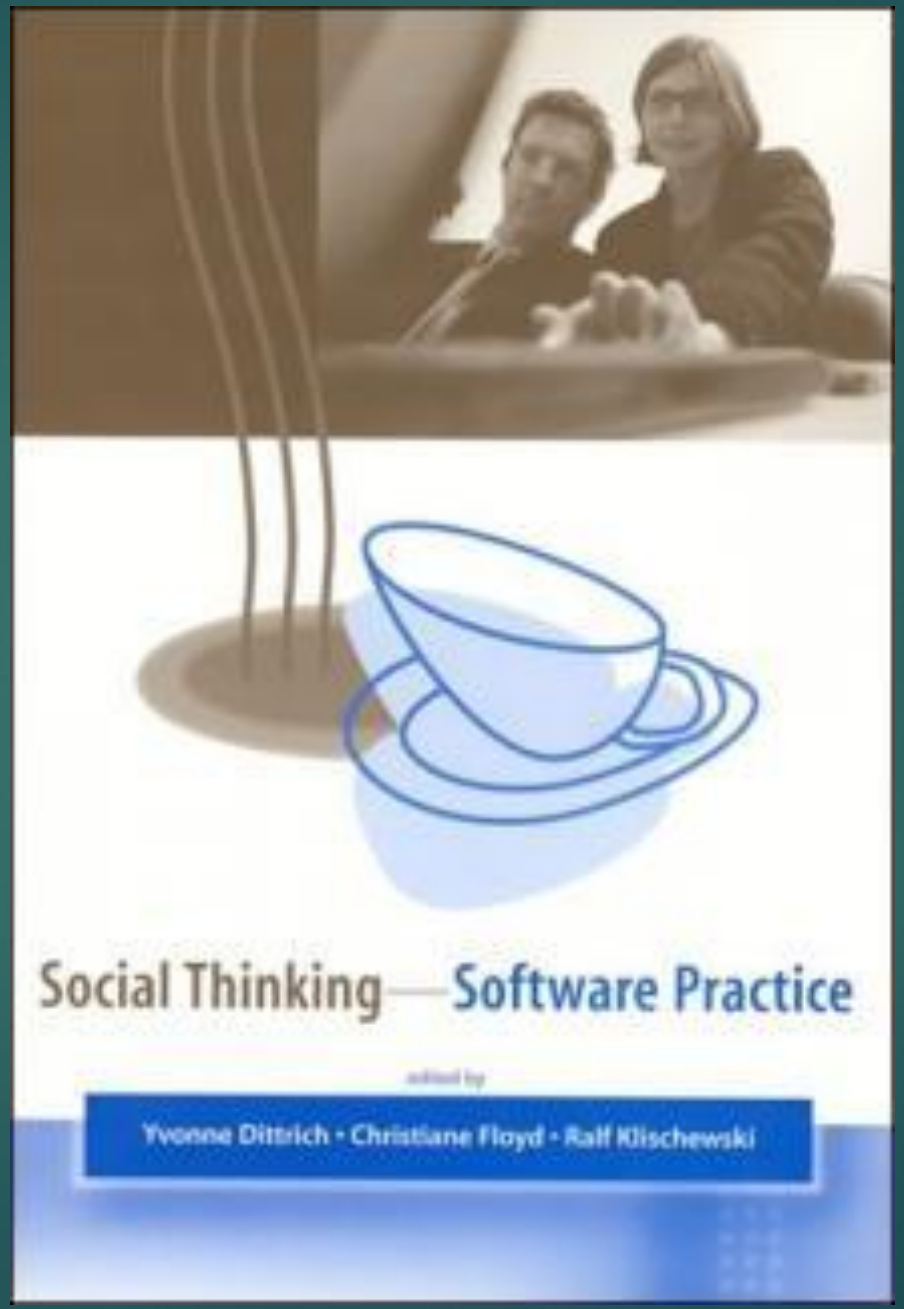
- “Social Thinking - Software Practice”

  Edited by R.Klischewski, C.Floyd, and Y.Dittrich

  MIT Press, 2002

- Overview:

  Software practice—which includes software development, design, and use—needs to go beyond the traditional engineering framework. Drawing on a variety of social theory approaches, this book focuses on interdisciplinary cooperation in software practice. The topics discussed include the facilitation of collaborative software development, communication between developers and users, and the embedding of software systems in organizations


Social Thinking—Software Practice
edited by
Yvonne Dittrich · Christiane Floyd · Ralf Klischewski

成立の背景

- C.Floyd さんの主宰で 1999年にドイツの Schloss Dagstuhl で開催されたセミナー参加者たちの共著
- Software practice is everywhere: software development, design, and use as well as related management are shaping today's technology and changing the way we engage in social relations at work and at home, in small groups and in the larger society. Scientific reflection on software practice aims at understanding and improvement; at the same time the motives for understanding and the direction of improvement are also subjects for reflection and discourse. Whereas the discipline of software engineering is mainly concerned with the formal principles, technical basis, and methodological support for software development, reflection on software practice as social activity needs to go beyond a traditional engineering framework. .

全体の構成

- 次の5つの章から成る：
- Deconstructing: Looking for underlying paradigms in software practice and related research; questioning established paradigms for reflecting on software practice; pointing out new concepts and/or new areas for applying existing approaches
- . Informing: Analyzing how social aspects of software use are conceptualized within the development process; creating social science–based representations of the workplace; questioning prescriptions and retaining a sense of critical and reflective appraisal
- . Grounding: Promoting a broader understanding of the software development process by empirical research as well as theoretical concepts; adapting social thinking for developing and improving software development methods
- . Organizing: Relating software practice to organizational change; opening places of thought for reflecting on software management issues; focusing on how to integrate different groups of actors and their pe spectives in information systems development and research
- . Reorienting: Focusing on use-oriented design; providing rich case de- scriptions and concepts for understanding software; evaluating software usability and providing innovative directions for use orientation and design in connection with technological advances

Floyd さんたちの提言

- これからのソフトウェア工学には，社会学や文化人類学，あるいは哲学のパラダイムを取り入れることが必要ではないだろうか？

新しいパラダイムを考える上でのそのほかのヒント

- ゲオルク・ジンメル：水差しの社会学
- ミハイル・バフチン：文学論のメタファ
- ガブリエル・タルド：ネオモナドロジー
- ジル・ドゥルーズ：差異，そしてリゾーム

ジンメルの社会学

- 美術品としての水差しの「取っ手」
- 組織には「取っ手」が必要だ
- 外側の世界とのつながりを失った組織は衰退し 死滅する

バフチン　（1）

- 対話型小説としてのドストエフスキー文学
- 多声と非決定性
- ソフトウェア開発プロジェクトという「物語」
- 技術上/管理上のさまざまな問題についての意見の対立は 最後まで解決されず 次のプロジェクトの持ち越される

バフチン　(2)

- フランソワ・ラブレーの「ガルガンチュア」
- 日常世界とは異なるカーニバルの世界
- 舞台と観客との壁が消える！
- Open Source/Free Software の社会学モデルとしては エリック・レイモンドの「伽藍とバザール」よりも適切では？

タルド：　ネオモナドロジー

- ライプニッツの「窓のないモナド」
- タルドの「窓のあるモナド」
- 神が世界の道筋を決めるのではなく　モナドたちの相互作用によって　歴史が作られて行く
- 香港でいま進行しつつある「雨傘革命」の行方は？
- 個々のモナドの内部構造と相互作用を 活動理論」の三角形モデルのネットワークとして表現・分析できないだろうか？

ドゥルーズ（1）　差異と反復

- 同じものごとが反復されるのではなく 差異が反復される
- 差異の中にこそ 革新の芽がある
- そう考えると CMM モデルは破綻する
- テストの目的は バグを発見し修復するのではなく 仕様と現実との差異を分析し システム進化の道筋を見通すことである

ドゥルーズ（2）　リゾーム

- これまで西欧の思考を支配してきた樹木型階層構造のパラダイムを捨てる
- 樹木ではなく根茎のメタファを！
- 主体・客体 etc といったカテゴリ分けを止める
- 始まりも終わりもなく すべてが中間点であるような「リゾーム」型のプロセス・モデルは可能だろうか？

リゾームの特性

竹やどくだみの根，あるいは蟻の巣

- 連結性・異種混合性
  - 始点も終点もなく，任意のポイント間が連結可能
- 多様性
  - ひとつの規律には統一されない
- 断裂性
  - どこからでも切断でき，再生できる
- 地図性
  - 現実のコピーではなく現実認識そのものであるような「道のない地図」

与えられた挑戦的課題

- 木構造の知識体系をどうやったらリゾーム型に再構成できるのだろうか？
- リゾーム型のソフトウェア・プロセス・モデルとは？
- リゾーム型のプロジェクト組織は可能か？

終わりに

- 精密な基礎研究によってわれわれが手にする成果は，
  未知の大きな X を包囲すること，それだけである.

  Paul Klee

Thank You!