ソフトウェア・テスティングについて

- 無形労働の視点から -

岸田 孝一
Kiss cedar, Call witch!
SRA

@ JaSST2013

西さんから依頼された3つのサブテーマ

- Software Testing
- Creativity
- Immaterial Labor

最初の小競り合い

- Software Testing
  - さて、わたしとの接点は？
- Creativity
  - 絵描きとしての経験から何がいえるか？
- Immaterial Labor
  - それがソフトウェアの本質？

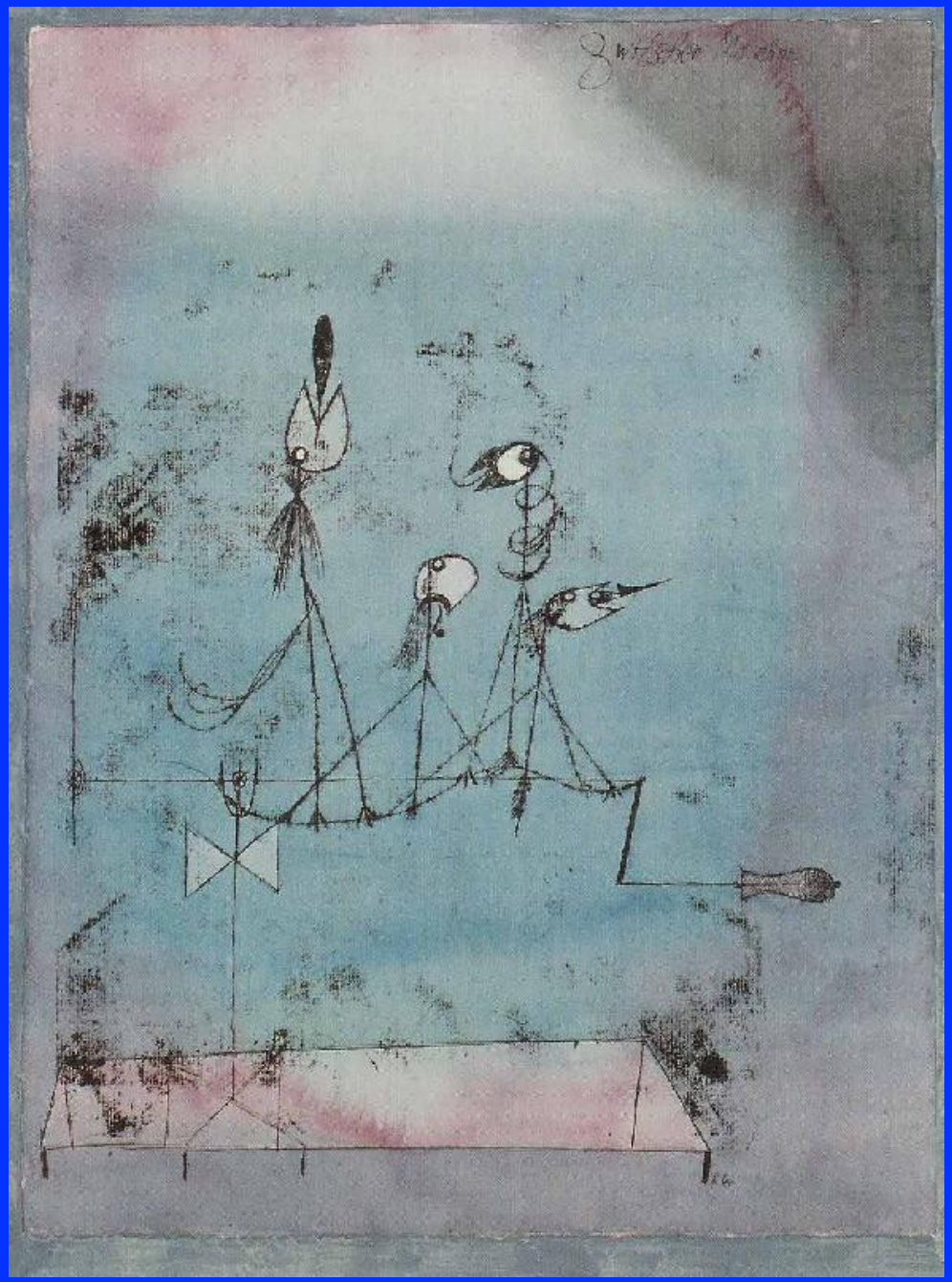
自己紹介
絵描き修業からプログラマへ

パウル･クレーのバウハウス講義

- 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.

Paul Klee



分析という概念

- 自然科学(たとえば化学)における分析
  - 与えられた対象(食品あるいは毒薬)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跡する.

プログラミングについての最初の認識

- 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.

抽象表現主義
- 何も抽象せず，何も表現しない -

- 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．

ー ヒルトン・クレーマー

先駆者
- Jackson Pollock -

絵画の3要素

- かたち
- いろ
- 肌触り

さて，ソフトウェアの場合は？

かたち

いろ

肌触り（マチエール）

とりあえず
「かたち＝構造」ということで出発

問題と解決法

- 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)


PROGRAMMING
FOR DIGITAL COMPUTERS

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)
  - Michael Jackson (イギリス)
  - Jack Warnier (フランス)
  - 岸田孝一(日本)

GOTO Letter

- Edgar Dijkstra, “GOTO Statement Considered Harmful”, March 1968. CACM.

- プログラムは，機械が読む以前に人間が読んで理解するものである.
  GOTO 文はそうした理解の妨げになる.

Edgar Dijkstra

あの時代の関心事

- ソフトウェアが含む2つの基本概念
  - 「プロセス」と「構造」
- ハードウェアの内側で起こっているプログラムの実行「プロセス」
- 紙の上に書かれたプログラムの「構造」をそれと一致させたい.

ダイクストラの指摘

- プログラム・テスティングはバグの存在を示すのに使うことはできるが，しかし，バグの不在証明（アリバイ）には使えない．
  - これがテスティングの宿命？
  - テスターはハードボイルド探偵か！？

ソフトウェア工学の誕生

- NATO Workshop
  - 1968（ガルミッシュ）
  - 1969（ローマ）

  残念ながら日本からは誰も参加していない

- 関心の移動
  - 機械内部のプログラム実行プロセスから
  - 機械の外側でのソフトウェア開発プロセスへ

Friedlich Bauer

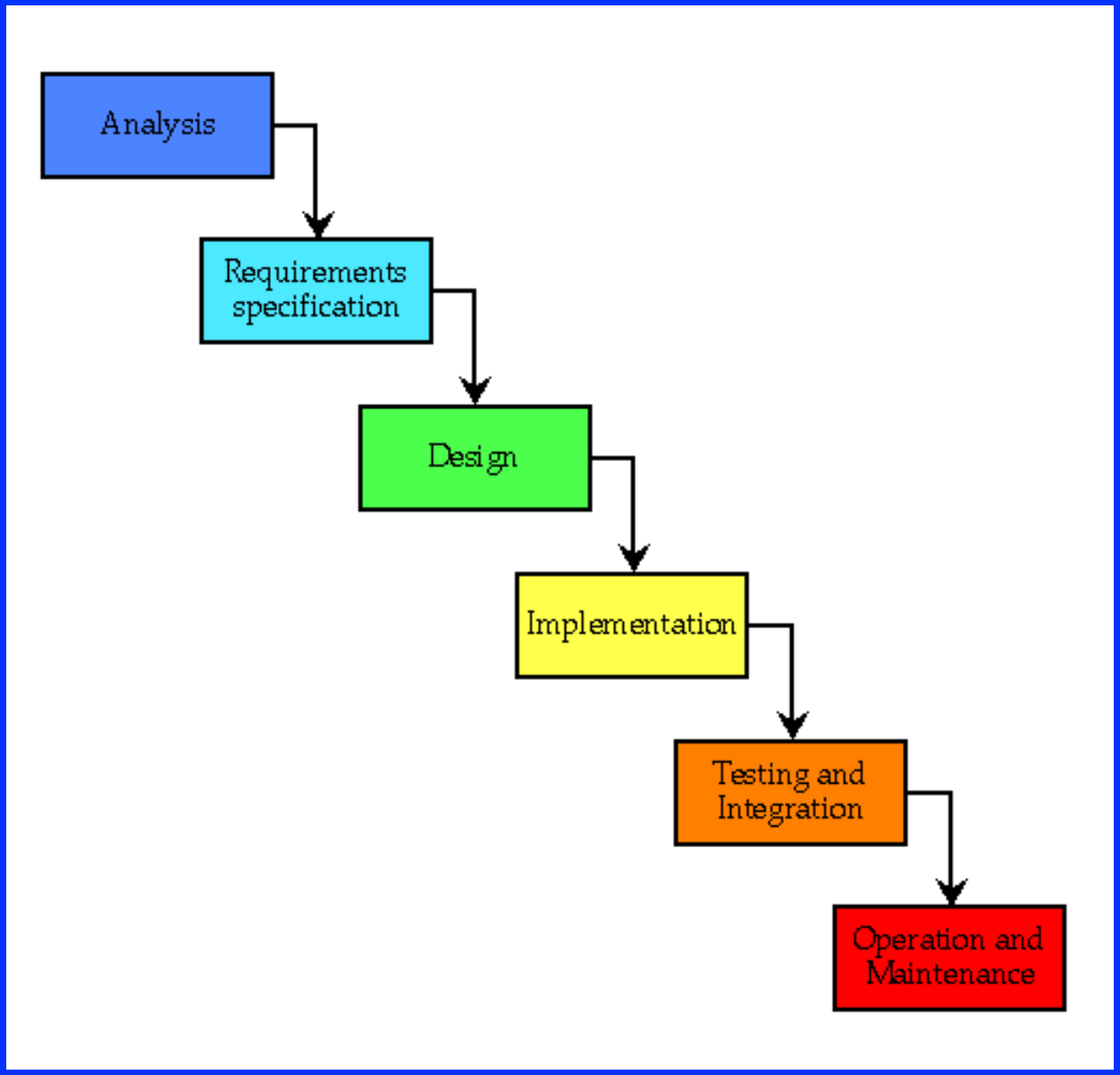
ソフトウェア工学の悲劇

- NATO Workshop での討論の結果としてソフトウェア工学の概念が提唱されたとき，参考にすべき工学としては，電気工学、建築工学、土木工学など，何らかのハードウェアを製造するための工学しか見あたらなかった.

- ソフトウェア＝もの(プロダクト)という認識

Waterfall Model

- Winston Royce のプレゼンテーション
  - 1970 夏 @ WESCON
  - たまたまわたしはそこにいた.
- この論文は 9th ICSE のProceedings に採録されているので, ご一読を！


Analysis
Requirements specification
Design
Implementation
Testing and Integration
Operation and Maintenance

論文の意図と一般の誤解

- Winston Royce さんはハードウェア製造プロセスとソフトウェア開発プロセスとの大きな違いを指摘した.
- しかし Waterfall Model を推奨したものと誤解されてしまった.

JOURNEYS OF
A YOUNG SOFTWARE ENGINEER

http://pragtob.wordpress.com/tag/
winston-royce/

Why Waterfall was
a big misunderstanding
from the beginning
- reading the original paper -

Waterfall の呪縛

- 要件定義がすべての出発点である.
- しかし，完全な要件定義は難しい.
- 大規模で複雑な要件を満足する「正しい」プログラムを作ることも，また難しい.
- それが，われわれがいまここにいる理由.
- また，要件は時間とともに絶えず変化する.

呪縛からの脱出（1）

- マネジメント・サイドの努力
  - プロセス・モデルの改良（Spiral Model他）
  - プロセス・マネジメントの改善
    - TQC, SPICE, CMM, etc.
  - テスト・マネジメントの工夫
    - V Model, W Model, etc.

呪縛からの脱出(2)

- さまざまな開発技法
  - JSD, SA/SD, OO, Agile, etc.
- テスティング技法やツールの工夫
  - テスト・カバレージ分析
  - テスト・データ自動生成
  - 形式手法の応用
  - etc

私的な経験(1)

- テスト･カバレージ
  - サンフランシスコに同じ名前の会社があり，社長の Ed Miller さんがカバレージ技法の専門家であると知った.
  - そこで 1979年夏にこの技法を紹介するセミナーを東京で開催．かなり話題になった.
  - いくつかの大規模アプリケーションのシステム･テストに応用して，モジュールの機能分類とテストの進捗との関係についての知見を得た.

Edward Miller

私的な経験(2)

- モジュール・テストベッド on Unix
  - 上海・宝山製鉄所向け制御プログラムの開発にさいして，実機ではなくUnix 上でのテストを計画し，カバレージやアサーションの機能を盛り込んだツールを作った.
  - Barry Boehm さんたちアメリカTRW社の実践より1年早かった.
  - 結果は大成功．日本でのUnixの普及に貢献した.

Barry Boehm

ソフトウェア進化論の洗礼

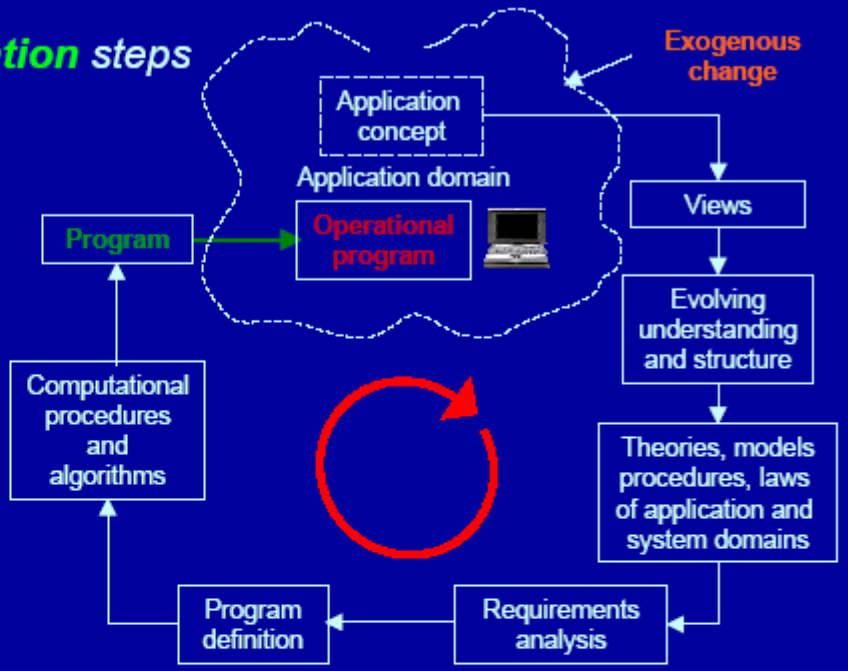
- 1978年秋，イギリス Imperial College の M.M.Lehman 先生を招いて，セミナーを開催した.
- 2種類のソフトウェア：
  - S-Type：仕様が確定でき，変化しない.
  - E-Type：現実世界のアプリケーションに組み込まれていて，時間とともに進化してゆく.

M.M.Lehman

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

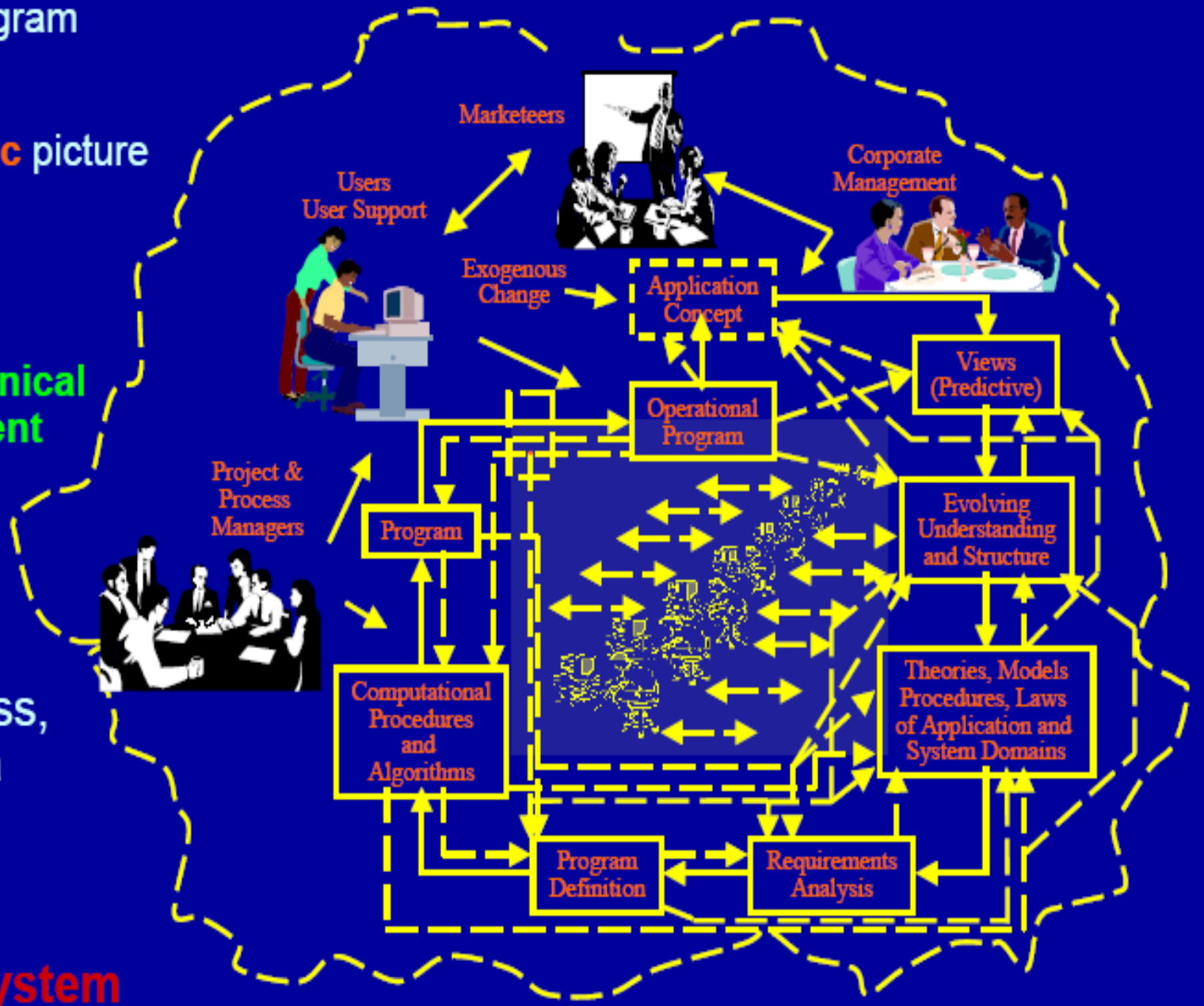


- **Installation** and introduction into **usage** closes major **feedback loop** that drives **iterative process** to **release** sequence of **upgrades** to users

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

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**

組み込みとは？

- すべてのソフトウェア・システムは，対象アプリケーションに組み込まれた E-type システムである．
- 自動車や携帯電話などのハードウェアに組み込まれたソフトウェアは，さらにそれらのハードウェアが利用されるアプリケーションの中に，二重に組み込まれている．

進化論の法則

1. ソフトウェアのサイズは時間とともに大きくなって行く.
2. ソフトウェアの部品は，いくら使っても磨耗しない．しかし，その構造は，メインテナンスが行われるごとに劣化して行く.
3. ソフトウェア・メインテナンスのいわゆる生産性は定常的（不変）である.
4. 以下省略。詳しくは Lehman 先生の Home Pageを参照：
   - http://www.doc.ic.ac.uk/~mml/

ソフトウェアの本質

- ソフトウェアの開発および進化のプロセスは マルチ･レベル，マルチ･ループ．マルチ･エージェントのフィードバック･システムである．
- したがって，要求仕様は永久に確定できないものと覚悟したほうがよい．

いわゆるテスティングの限界

- 要求仕様とプログラムとをつき合わせて行うテスティングには限界がある.
- いわゆる V Model や W Modelをベースに行われるテスティングは，ソフトウェア進化プロセスのマルチ・ループの第1回目のほんの一部分でしかない.

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.
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.

少し視野を広げてみよう
- 文明史の展開 -

- 農耕社会から工業社会へ
  - 近代化とは，社会全体を工場にすること.
  - われわれはいま，そのパラダイムにしたがって日常の行動をおこなっている.
- 工業社会から情報社会へ
  - 近代化の行き詰まり
  - 工場パラダイムからの脱却の動き

工業社会の変貌

- フォード・システム
  - 製造側の市場調査にもとづく製品仕様
  - 大量生産によるコストダウン
  - そのための部品・プロセスの標準化
- トヨタ・システム
  - 多様な市場ニーズのタイムリーな把握
  - フィードバック重視の多品種生産
  - 製造はサービスだという認識
- そして次は？

ポストモダンの流れ

- 脱工業化，脱工場化
- 工場のサービス化ではなく，無形労働を中心とするサービス産業主体の社会へ
- 標準化された大量生産パラダイムから情報通信ネットワークにもとづく多種生産のパラダイムへ

Immaterial Labor

- イタリア人社会学者 M.Lazzarato さんの論文が英訳され，Internet 上に公開されたのが始まりだった：

“Immaterial Labour”

http://www.generation-online.org/c/cimmateriallabour.htm

その概念を敷衍した書物

マウリツィオ・ラッツアラート 著

（村澤真保呂・中倉智徳 訳）

「出来事のポリティクス：知-政治と新たな協働」

（洛北出版，2008）

- 管理と統制の社会から

ネオ・モナドロジーにもとづく

自由な協働を中心とする社会へ

Maurizio Lazzarato

Immaterial Labor とは？

- 「非物質的」労働あるいは「無形」労働
- もの（プロダクト）を作るのではなく，情報あるいは文化を内に含んだ「かたちのないプロダクト」を作りだす仕事
- たとえば：
  - オーディオ・ヴィジュアル産業
  - ファッション産業
  - ソフトウェア開発

SEA での討論

- 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

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

無形労働としてのソフトウェア開発（1）

- われわれの仕事は単にソフトウェアという有形なプロダクトを製造することではない.

無形労働としてのソフトウェア開発(2)

- 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.

無形労働としてのソフトウェア開発(3)

- 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.

パラダイム・シフト

- 「プロダクト指向」（有形労働の視点）から「プロセス指向」（無形労働の視点）へのゆるやかなシフトが1970年代から始まっている.
  - Christiane Floyd： “A Paradigm Change in Software Engineering”, ACM SIGSOFT News Letter, April 1988.

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

- その背景には Lehman のソフトウェア進化論があった.

Christiane Floyd

開発技法とは？

プロダクト指向の見方
- 開発技法とは汎用のプロダクトである.
- 技法を正しくカスタマイズして利用すれば，常に一定の成果が達成できる.

プロセス指向の見方
- プロダクトとしての技法などは存在しない。だれかが何らかの条件下である技法を考案し利用したプロセスだけが存在する.
- そうした固有のプロセスが，その技法を利用した成果に影響を与えている.

品質とは？

- プロダクト指向の見方
  - 品質とは，ソフトウェア・プロダクトの属性である（バグの少なさ etc）．
  - 開発者の視点
- プロセス指向の見方
  - 品質とは，そのソフトウェアが実際に使われるアプリケーション・プロセスの属性である（仕事との適応性 etc）．
  - ユーザの視点

現実・仕様・プログラム

Lehman の逆V字モデル

２段階のモデル化

（１） 抽象化（逆V字の左脚）:

アプリケーション → 要求仕様

（２） 具体化（逆V字の右脚）:

要求仕様 → プログラム

テスティング

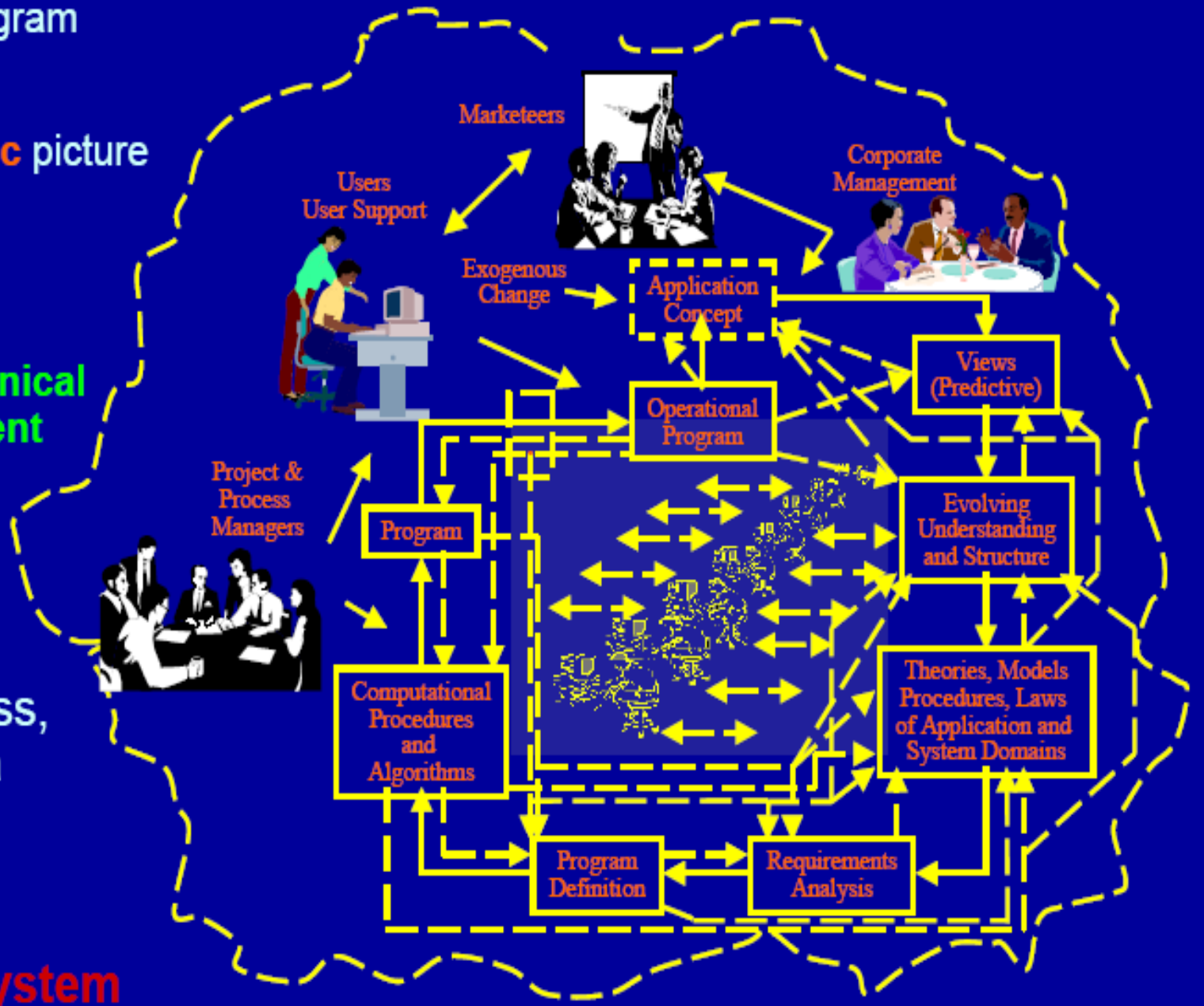
- 逆V字モデルの右脚
- プログラムは仕様を正しく実現（具体化）しているか？
- 常識的なテストのパラダイム
- 全体的なソフトウェア進化プロセスのほんの一部分でしかない

より重要な検証
逆V字の右脚

- 現実世界 vs 仕様
  - 仕様は現実世界をうまく反映しているか？
  - 要求工学の難問
- プログラム vs 現実世界
  - 現実世界のニーズはそのプログラムで十分満足されているのか？
  - ユーザ視点からは最重要の課題

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**

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.

ところで
抽象化（仕様開発）とは？

- ソフトウェアは，「世界」つまり対象アプリケーションを記述した機械である．
- いいかえれば，われわれは「ソフトウェア」という仮想世界の創造者である．

マイケル・ジャクソン：

「ソフトウェア博物誌」

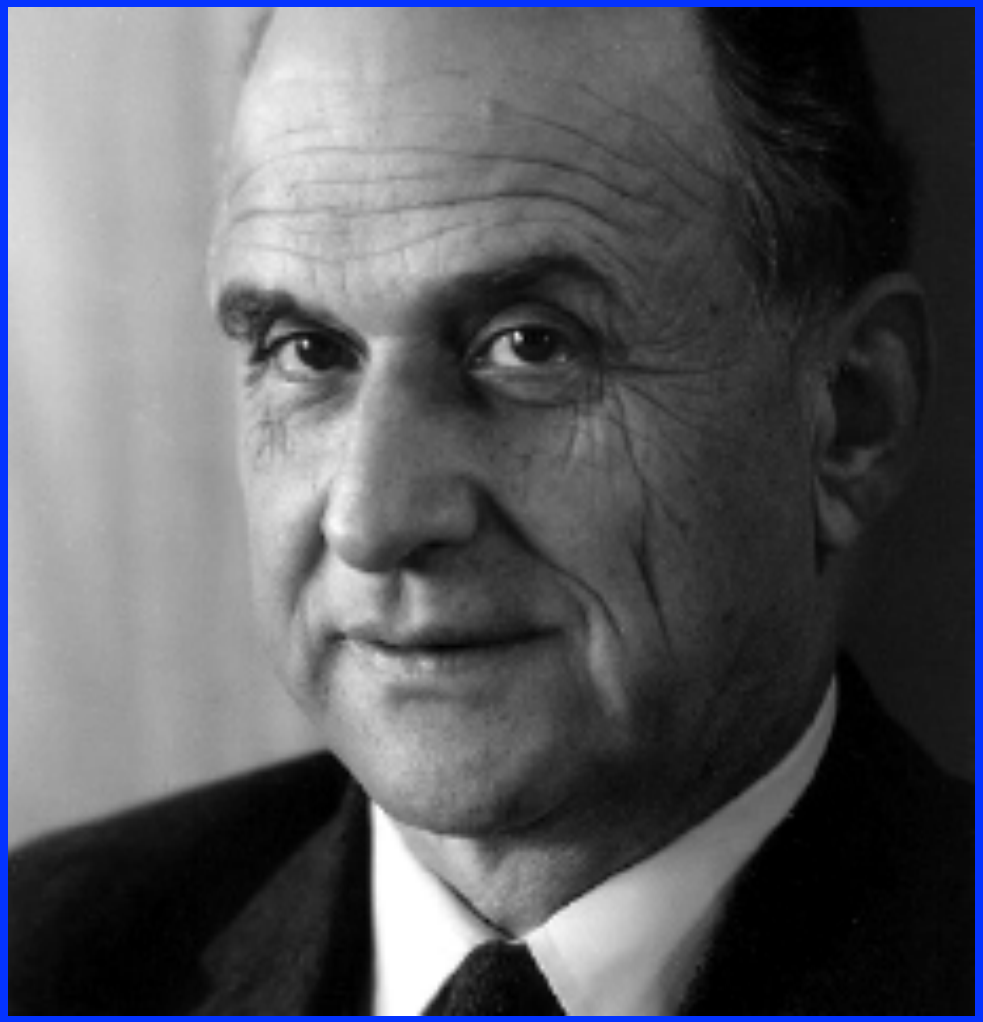
Michael Jackson

何から作るの？

人間が世界を作り出すのは，「無」からではない．
すべての「世界」（あるいは　Version ）は，すでに記号化された形で存在する世界を変形したり，それに何か付け加えることによって，いわば新しい Version として創造される．

ネルソン・グッドマン
「世界制作の方法」（ちくま学芸文庫）

Nelson Goodman



世界制作の実態

- 記述される以前の「世界」などは存在しない.
- われわれがおこなう「抽象モデル化」とは，すでに存在する世界のあるVersionから“仕様”と呼ばれる別のVersionを作り出す作業でしかない.
- ジャクソンのいう「世界」とは，すでに存在する所与の「機械」なのであって，ソフトウェアはそれを変形した別の「機械」であるにすぎない.

世界制作のプロセス
- グッドマン -

1. 対象の要素への分解と構成
2. 要素の重みづけ
3. 要素の優先順位づけ
4. 要素の削除と補充
5. 要素の変形

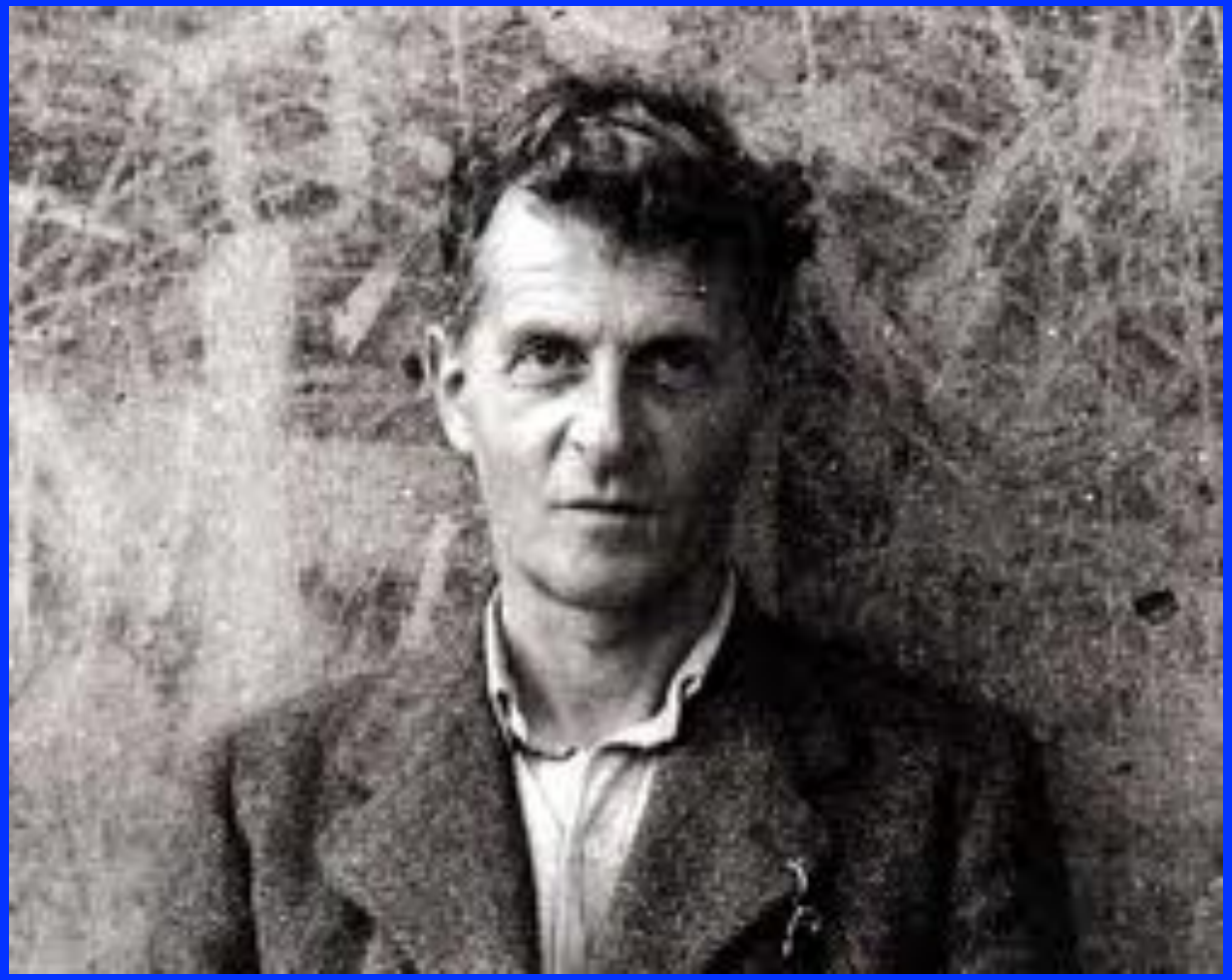
ただし，上記は論理ステップの区分であって時系列を示すものではない

ところで，世界とは？

- 世界は *もの* の集合ではない．
  成立していることがら の総体である．

L.ヴィトゲンシュタイン：「論理哲学論考」

Ludwig Witgenstein



仮説マネジメントの問題

いま「現実に成立していること」と
将来「成立の可能性があること」とは違う

その区別をどうするか？

仕様作成仮説の必要性

- 現実世界は，ほとんど無限に近いパラメータを含んでいる.
- しかし，仕様書は，いくら詳細に記述したとしても有限のページ数しかない.
- したがって，アナリストは，何らかの仮説に基づいて，そのシステムに無関係と思われるパラメータを削除する.

仮説マネジメントの問題

いま「現実に成立していること」と
将来「成立の可能性があること」とは違う

その区別をどうするか？

仮説はいつか破綻する

- 卑近な例： 西暦2000年問題
- 多額な費用損失の例： ヨーロッパの宇宙衛星ロケットの打ち上げ失敗
- 人命が失われた例： フォークランド紛争時のイギリス海軍新鋭艦の撃沈事件

仮説の管理と検証

- これから大規模かつ複雑な社会システムの増加とともに，ますます重要性を増す．
- だれが，どうのように行うのか？

- それがこれからのテスターの役割！
- どのような技法やツールが使えるのだろうか？

プロセス問題再考

- ICSSP2013 (5/18-19) の CfP から
  - Processes for Emerging Software Platforms and Ecosystems
  - The latest trends towards creating software ecosystems on emerging software platforms, such as clouds, cyber-physical systems, or social media are bringing up new approaches to software and systems development. This also leads to new opportunities and challenges regarding development processes, such as process interoperability, scalability, or the coordination of multiple platforms and lifecycles.

プロセスのモデル化と改善

- あらゆるプロセス・モデルにおける中心的な概念は，同様なステップの「繰り返し」すなわち「反復」である.
- SPI (プロセス改善)の運動は，この「反復」にさいしての改善(次はもっとうまくやろう)を目指している.

成功例に学ぶ

- そこで，世の中でプロジェクトの遂行がうまくいった例を「ベスト・プラクティス」として収集し、それに学ぼうという動きがあちこちで見られる.

オスカー・ワイルドの警句

- Experience is the name everyone gives to their mistakes.

  Oscar Wilde

- 「経験」とは、みんながそれぞれの失敗につける名前のことだ.

  – EuroSPI2013 (6月末 in アイルランド)

Oscar Wilde

失敗に学ぶ

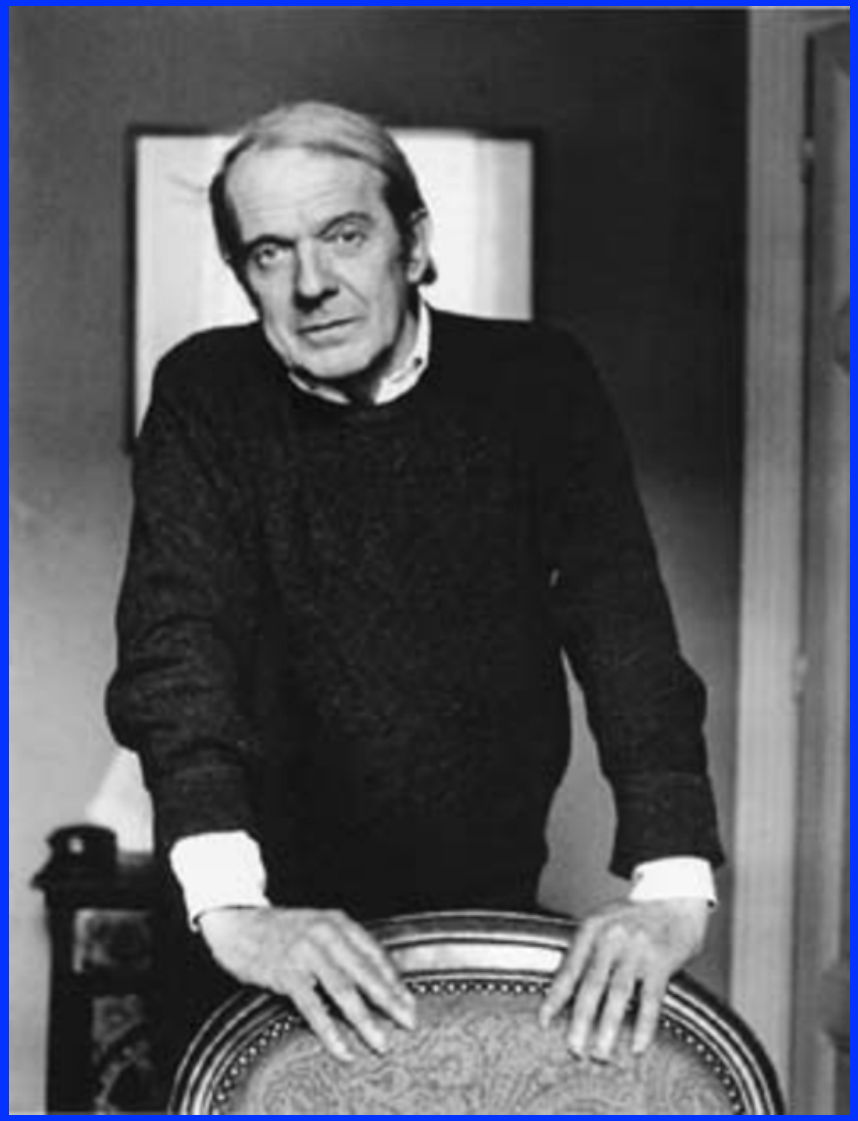
- ベスト･プラクティス環境をその通りに再現することはきわめて難しい.
- しかし，ワースト･プラクティス環境が再現されないように注意することはできる.
- それが「経験から学ぶこと」.

反復とは？

- プロセス・モデルその他，これまでのソフトウェア工学の規範は反復パラダイムにもとづいていた.
- しかし，「同一のもの（こと）が反復されることは決してない．差異だけが反復される」
  - ジル・ドゥルーズ「差異と反復」

- そうした差異の中にこそ革新の芽がある

Gilles Deleuze



視点のちがい

- 同一性の反復に注目するのは「傍観者」あるいは「管理者」の視点.
  - Should-be Process
  - あらかじめ意図された最適化
- 「実行者」には現在しか見えない.
  - As-is Process
  - 走りながらの最適化

同一性と差異

- 同一性の反復を経由することなく，直接に差異だけに注目しよう.

（ジル・ドゥルーズ）

- Bottom-up の現場型思考によるプロセス改善.
- しかし，ほんとうの改革は既存のプロセス構造を破壊することだと思う.

創造性について
- ヨーゼフ･ボイスの発言 -

- 人間はすべて芸術家である.
- ただし，それは誰もがピカソやわたしのようになれるという意味ではない.

Josef Beuys

ボイス発言の真意

- すべての人間の日常的な行動の中には，かならず，なんらかの創造性を必要とするようなものが含まれている.

- 自分がそれに気づくこと，あるいは他人にそれを気づかせることが重要なのだ.

考えるためのヒント

- ソフトウェアの品質や価値には3つの相異なる次元がある
  - 客観的(物質的)次元
  - 社会的次元:
  - 主観的次元

客観的(物質的)次元

- プロダクトとしてのソフトウェアの機能や品質を考える.
- 開発技法やテスト技法を汎用性のあるプロダクトとして扱う.

- つまり, material labor の視点

社会的次元

- ソフトウェアを，ユーザとのコミュニケーションや相互学習のツールまやはメディアとしてとらえ，
- そのコミュニケーション/相互学習プロセスの進化に注目し，
- そのプロセスを通じてのソフトウェアの進化を考える.

主観的次元

- ユーザ・コミュニティとソフトウェアとの感覚的あるいは情緒的な関わり
- ユーザ・インタフェイスではなく，ユーザ・インタラクション・プロセスの評価
- ソフトウェアのファッション化

- たとえば，ゲーム・ソフトウェアやSNSのテスティング問題

生産と消費

- Immaterial Product としてのソフトウェアは，消費によって破壊されることはない.
- むしろ，ユーザ・コミュニティとの密接な相互作用を通じて進化してゆく.
- その意味で，ソフトウェアの使用(消費)は.将来の進化プロセスのための新しい要件を生み出す源泉である.

これからの技術革新

- 新しいプロセス・モデルは？
- ひとつの有力候補：
  - ジル・ドゥルーズとフェリックス・ガタリが提案した「リゾーム(根茎)」のメタファ
  - 「千のプラトー」(河出文庫)の序章

メタファとしてのリゾームの魅力

- 連結性・異種混合性
  - 始まりはなく，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」

しかし，それはまた別の話

- 参考：SS2011 でのわたしの発表論文

「革新的ソフトウェア・プロジェクトを目指して」

  - http://sea.jp/blog/wp-content/uploads/2011/03/k21.pdf

最後に PR

ソフトウェア・シンポジウム 2013

7月7日 プレイベント

8日～10日 論文発表 & WG討論

岐阜市・長良川国際会議場

現在，論文 & WG提案募集中

http://sea.jp/ss2013/

Thank you!