スライド集

〔知働化研究会誌第２号〕
知働化研究会
Executable Knowledge and Texture Laboratory

知働化の最前線

山田正樹
masaki@metabolics.co.jp
2015.03.05

- ソフトウェアは消えていく

- ソフトウェアは見えなくなる

- 社会に溶けていく
  - 溶解 dissolving
  - 浸潤 infiltration

- かたちなきかたち
  - anarchitecture アナーキテクチャ

- big data
- deep learning
- internet of things
- “social”

- そこに欠けているソフトウェア「らしさ」
  - self descriptive 自己記述性
  - dialogical 対話性
  - reflective 反映性 (メタ)
  - executable 実行可能性

- 本当にそれでいいのか

- ソフトウェアとは何か

- 世界制作の方法

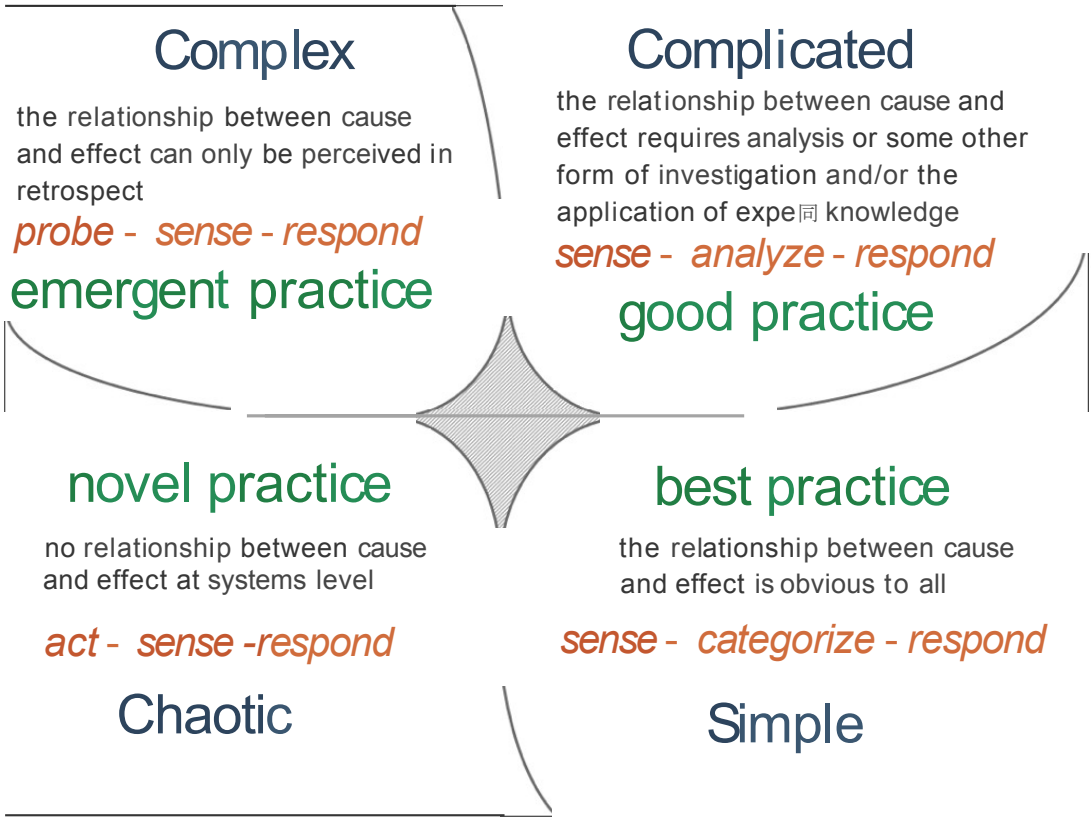
- 実行可能な/実験可能な/実現可能な哲学
  - Software as a Philosophy
  - Philosophy as a Software

- 世界の知働化のさまざまな潮流
  - Programmer Anarchy (Fred George) ソフトウェア開発
  - の現象学 (Charles Tolman)
  - リアリティ・コンストラクションとしてのソフトウェア開発 (Christiane Floyd)
  - 有機体論 コトバ・ソ
  - フトウェア

Programmer Anarchy

- post-agile

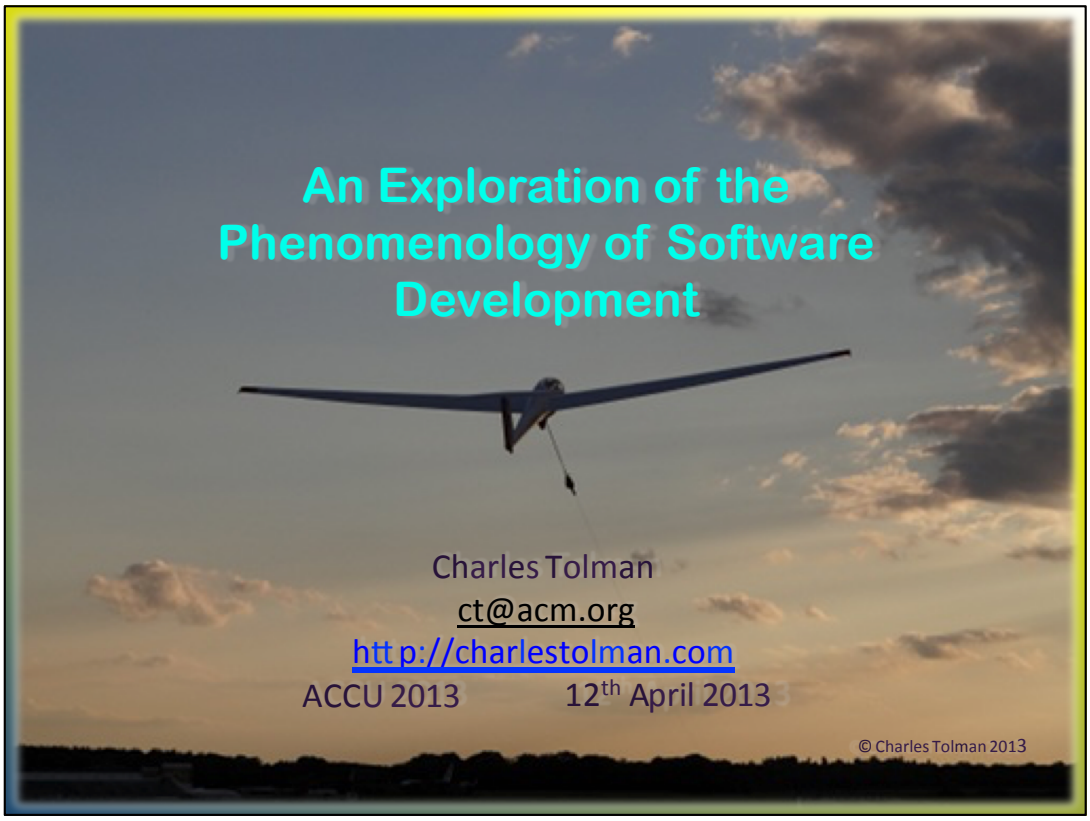
- 開発者は, 毎日のスタンダップ・ミーティングで, 自分が今日やることを決める
- PM, BA, QAはいない, 顧客と開発者のみ 一切
- のルールはない, 創造性と生産性がすべて 異
- 議と失敗が重要
- 恐れを取り除く
- 信頼
- Agile Manifesto, XP Valuesは有効だけど, アジャイルのプラクティスは使わない
- ビジネス価値はすべて開発者が負う
- Cynefin Framework


Complex
the relationship between cause and effect can only be perceived in retrospect
probe - sense - respond
emergent practice
Complicated
the relationship between cause and effect requires analysis or some other form of investigation and/or the application of expe同 knowledge
sense - analyze - respond
good practice
novel practice
no relationship between cause and effect at systems level
act - sense -respond
Chaotic
best practice
the relationship between cause and effect is obvious to all
sense - categorize - respond
Simple


μService Architecture:
A Personal Journey of Discovery
Fred George
fredgeorge@acm.org
@fgeorge52
Copyright © 2012 by Fred George. All rights reserved.
1

- Programmer Anarchyと整合性のある技術
  - loosely coupling 500行
  - のサービスを200個
  - EntityからEventへ
  - REAL objects
  - 要件無しで作る
  - ビジネス価値に基づいてテスト

ソフトウェア開発の現象学



- ソフトウェアを作るとは,
  - 考え続けることであり メンタル・モデルを作
  - ることであり それをソフトウェアにトランス
  - コードすること
- 経済的な側面が強調されすぎているけど, 本来それは遊戯なのだ
- よいプログラマはそれに気付いている コンピュータとは「思考
- の鏡」なのだ

- 理論構築としてのソフトウェア開発
  - Peter Naur
  - 1985
  - http://www.dc.uba.ar/materias/plp/cursos/material/programmingAsTheoryBuilding
- 世界のある事象がプログラムによって, どうやって取り扱われるべきか
- 世界のある事象の見方が変わったときに, プログラムをどのように構成的に変えていけばよいか
- という理論

- Peter Naurの「理論構築としてのプログラミング」という考え方を哲学的に再評価する
  - Boris Wyssusek
  - 2007
  - http://is2.lse.ac.uk/asp/aspecis/20070190.pdf
  - プログラミングの本質はコード, ドキュメント, 仕様などの成果物ではなく, 構築される理論にある

- 方法論無しのソフトウェア開発
  - Duane Truex, et.al
  - 2000
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.2819&rep=rep1&type=pdf

- リアリティ・コンストラクションとしてのソフトウェア開発
  - Christiane Floyd
  - 1992
  - http://swt-www.informatik.uni-hamburg.de/uploads/media/SD_AS_RC.pdf

- 今までの知働化のキーコンセプトは「実行可能な知識」
  - でも「知識」という言葉は「いやらしい」
  - 知識とは実際は
    - 知るという働き (仏教用語としての識), 知るに
    - 際して有用な (有害な) 道具であるコトバ とし
    - て考えた方がよい

- ラジカル構成主義
  - 知覚は感覚やコミュニケーションを経由して受動的に受け取られるものではない
  - 知識とは認知主体によって能動的に構築される
  - 認知の機能は, 生物学的な意味で適応的なものであり, 適合や実行可能性への傾向性を有している
  - 認知は主体による経験世界の組織化の役目を果たすのであって, 客観的な存在論的実在を発見しているわけではない

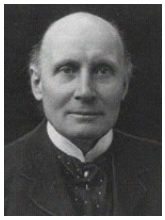
- 製造としてのソフトウェア開発には限界がある

- デザインとは, 以下のような異なる世界をリンクさせること
  - 問題となっている社会的な世界 (ビジネスとか)
  - 実装の手段としての技術的な世界 手法と概念
  - の形式的な世界

- デザインとはこんがらがった現実領域を解きほぐすこと
  - 製造としてのソフトウェア開発の線形的な方法では無理 要求を分
  - 析せずに, 我々が我々のパースペクティブから構成する
  - あらかじめ定義された方法論を使うのではなく, その場で状況に応じて創り出す
  - 固定された実装手法を参照するのではなく, 既にあるものの中からその場でテスト, 選択, 補完によって構成する

- デザインとは, 判断の網の目である

- デザインには対話がなければならない
  - ひととひとの間 ひとと
  - ソフトウェアの間 ひと
  - とセカイの間



A. N. Whitehead

- メレオロジ
- 有機体論
- 出来事

Joseph Goguen

コトバ・ソフトウェア

- ソフトウェアはコトバで出来ている

- ソフトウェアはコトバという材料 (だけ) を使って, 人工物を作る唯一のやり方 (エンジニアリング) である

- コトバとは, 何らかの対象を, その対象とは直接関係のない別の対象をもって, 表出 (記述や発話, コーディングなど) する試みである

- コトバには, コトバを表出するものの他に, コトバを解釈するものが必ず存在する (あるいはその存在が想定される. 少なくとも表出したものは潜在的に解釈するものでもある)

- 解釈とは, 解釈した結果, 解釈したものの性質が変化してしまったり, 解釈したものが外界に何らかの作用を及ぼしてしまったりすることである

- 蒸気機関が単にお湯を沸かすだけのものではなく，お湯を沸かすことによって「仕事」をすることで，エンジニアリングになり得たように，ソフトウェアはコトバが解釈を引き起こし，解釈が現実世界を変化させることでエンジニアリングとなり得る

- ソフトウェアはあるコトバを別のコトバに連続的に変換していく (エンジンがエネルギーを別のエネルギーに変換していくように). そして最終的には CPU と呼ばれるものが解釈可能なコトバに変換する.
CPU は, あるコトバを解釈することによって, 現実世界にある変化を引き起こす仕組みを持っている

- 認知言語学的なコトバ
- から 生成文法的なコ
- トバ
- へ

- thing, entity, relation, process
- イメージ・スキーマ
- カテゴリ化
- 喩え

- 今までのコトバ
  - コトバを固定化する コトバを明確に区別
  - する コトバの算出方法を制限する コトバ
  - の解釈方法を制限する コトバからコトバ
  - への変換方法を制限する
  - 「コミュニケーション」による意味伝達を意図する
  -
- コトバが死ぬ
- うまく行かない

- いきいきと生きるコトバ

- イメージ・スキーマに相当するコトバを作る
- コトバがコトバを生み出すような仕組みを作る
- コトバの体系 (言語) を作る コトバを媒介とす
- るカップリングを導く

- 過程

- ソフトウェアにおけるアスペルガー

まとめ

- object, modeling, agile (1994 ~) 実行可能な
- 知識としてのソフトウェア (2004 ~)
- 知働化研究会 (2009 ~)
- 識とコトバ (2014 ~)
  - real objects
  - M.E.T.A.
  - まく
  - dissolving architecture, anarchitecture