知働化研究会

第一回に参加しての感想及び考察

Verizon Business
竹洞 陽一郎
yoichiro。takehora@jp。verizonbusiness。com

ソフトウェアとは何か?

- ソフトウェアとは
  - 仕事(業務、作業など)の遂行内容を論理的に記述して、コンピュータに遂行させるためのもの

コードとは何か

Michael:

あなたが Erlang のプログラムを書く中でやってしまった、過去最悪の失敗は何ですか?

Joe:

文書の無いプログラムだな。
僕は “コードを読め” というのが嫌いでね。

コードは問題の “解法” だ。

コードを読むときは、問題が何だったかを “推測” しないといけない。これは間違いやすい。問題は教えてもらえる方が嬉しい。推測するんじゃなくて。

だから最悪の失敗はこうだ。私は複数ページにまたがるコメントを書いたことがある。プログラムの特別トリッキーな部分を説明するものだった。でも後のバージョンで、そのコメントは誰かに丸ごと消されたよ。そんなもんだよな。

“Ten Questions with Joe Armstrong about Parallel Programming and Erlang” (結城浩さん訳)

機能の集合はソフトウェアではない

- 業務や作業には、フローがあり、一貫性がある
  - 機能毎に論理記述に整合性と一貫性があっても、それらを組み合わせた集合体が、より粒度の高い業務「全体」の仕事の論理記述として適合するとは思えない
- 機能は相互作用し合う
  - 複雑系の考え方が、機能の集合体にマッチするのかもしれない
  - 機能が増える毎に、機能同士は相互作用し合い、全体として異なる様相を呈してくる

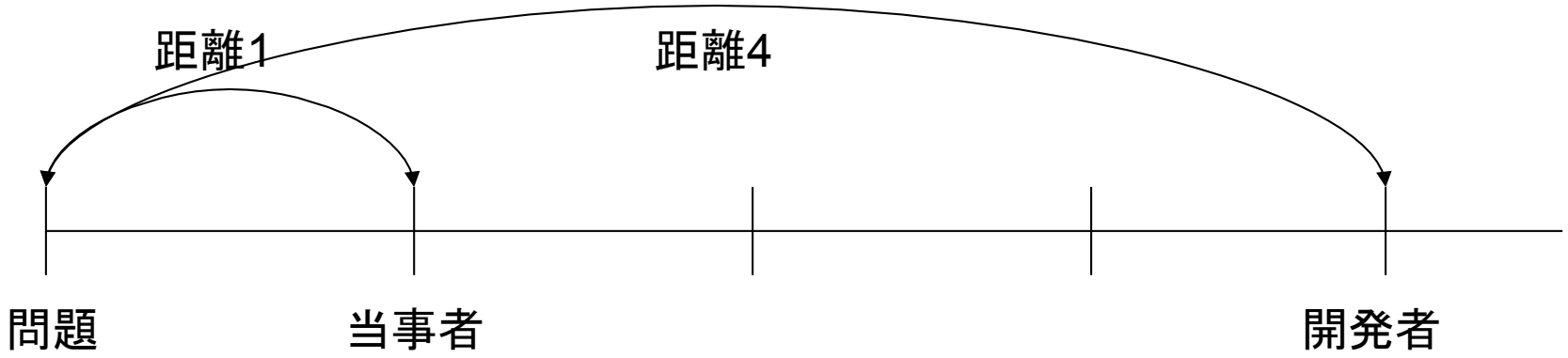
論理記述としてのソフトウェア開発

- 現在の日本におけるソフトウェア開発は、受託開発が多い
  - 受託開発とは、仕事の論理記述の代筆業なのかもしれない

理想的なコンピューティング利用

- 問題を提起もしくは発見し、それを解決しようとする人が、コーディングするのが理想
  - ツールとしてのコンピューティング利用
- しかし、殆どの人は、そこまでのITリテラシを持ち合わせていない
  - コーディングの専門知識を持ち合わせたソフトウェアエンジニアが必要とされる理由

ソフトウェア開発による問題解決の度合いと満足度



「問題」の解決のためにソフトウェアを作る際、その問題の様相について
どれだけ詳しく理解しているかが、問題との距離を決める

フィードバックシステム

- 人間は、フィードバックによって学習し、修正し、補完する
- 人間は論理一貫性については、不完全
- 試してみて、フィードバックを得て、修正して、また試すというサイクルを繰り返すことで、足りないものを補完し、間違いを修正し、求めている結果（Ideal）にたどり着く

ソフトウェア開発におけるフィードバックシステムの導入

- アジャイル開発

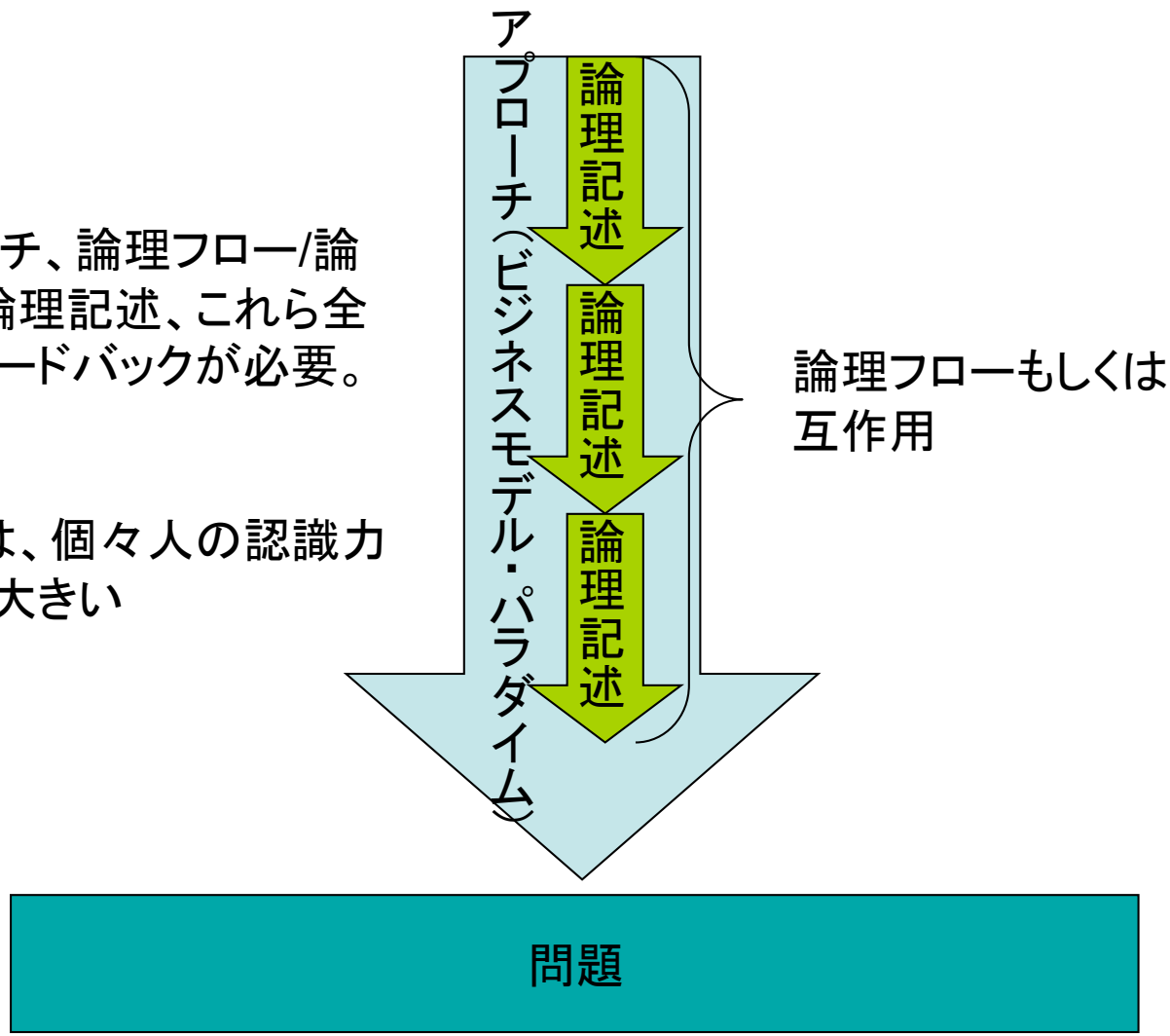
フィードバックシステムが必要なもの

- 問題の定義
  - 問題そのものを正しく把握しているかどうかを、様々な人とやり取りすることで、フィードバックを得て正しく理解し、問題を正しく把握・定義する
- 仕事の論理記述
  - 仕事の論理記述をプログラミング言語で行うことで、論理矛盾や論理の欠如、論理の欠陥などを発見し、論理をIdeallに近づけていく
- ビジネスモデルの具現化としてのシステム
  - 問題解決のために考えて具現化したシステムを使ってみることで、現実世界に適用したことによるフィードバックを得て、問題の定義が正しかったか、問題解決のためのパラダイム、アプローチが正しかったかを検証し、Ideallに近づけていく

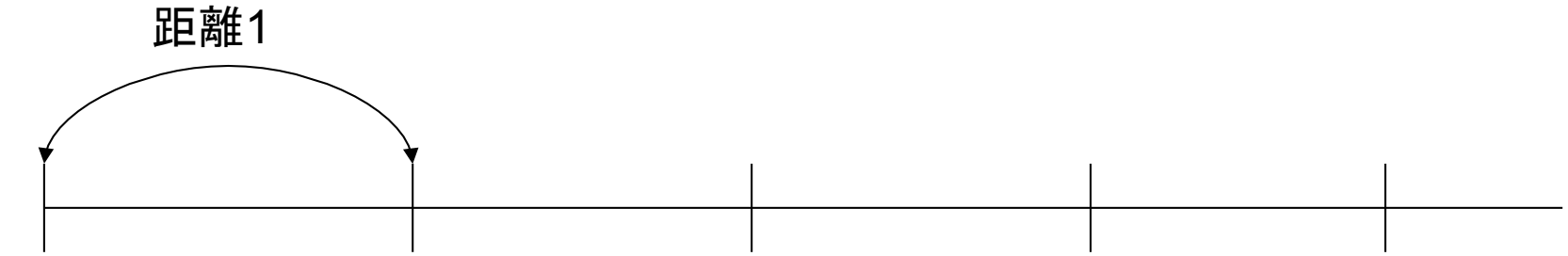
フィードバックの粒度

問題、アプローチ、論理フロー/論理相互作用、論理記述、これら全てについてフィードバックが必要。

フィードバックは、個々人の認識力によるところが大きい

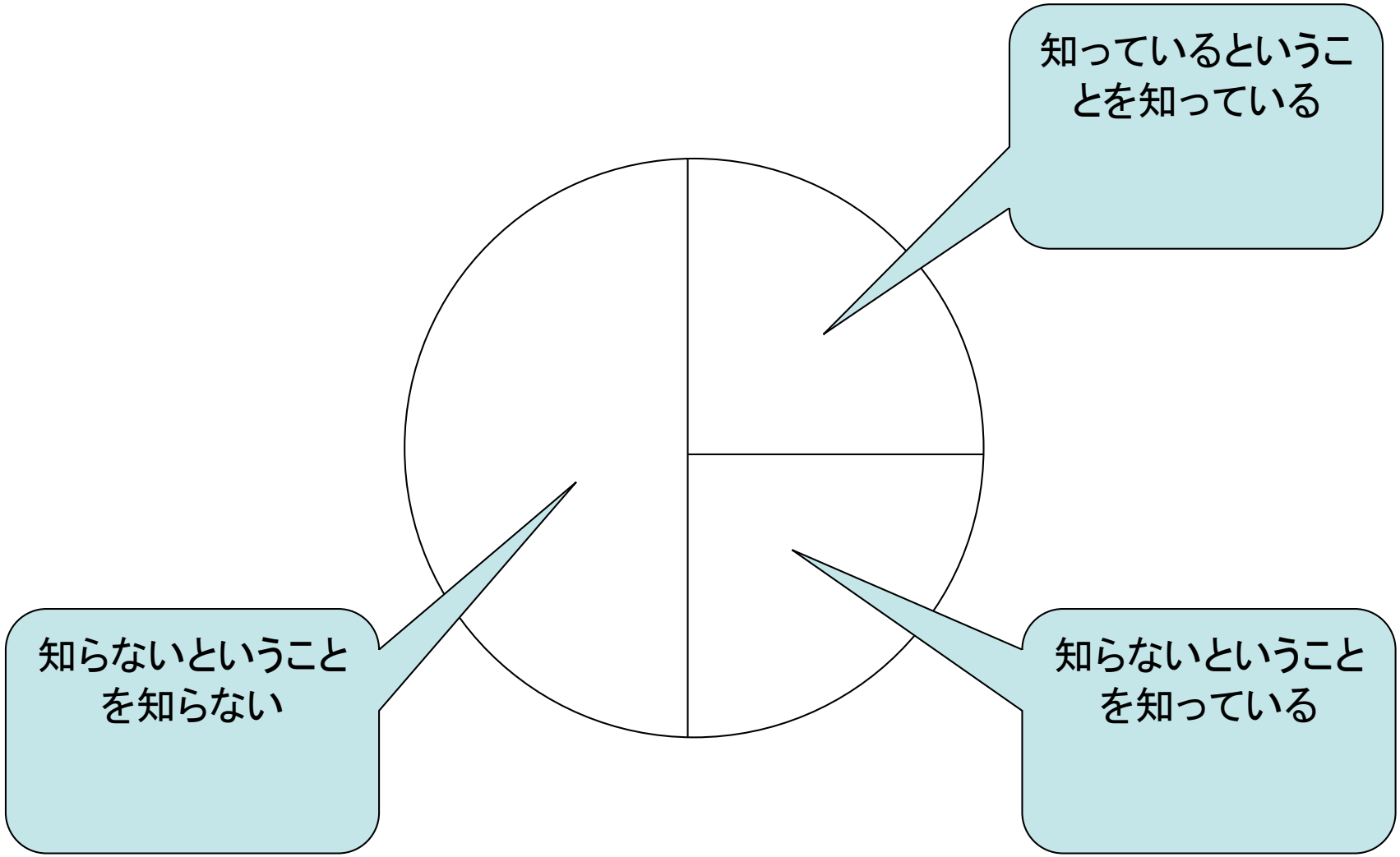


オープンソースにおける問題解決の度合いと満足度



オープンソース開発においては、問題の定義者とその解決者たる開発者は同一人物であることが多く、問題解決の潤滑度は高く、満足度も高い

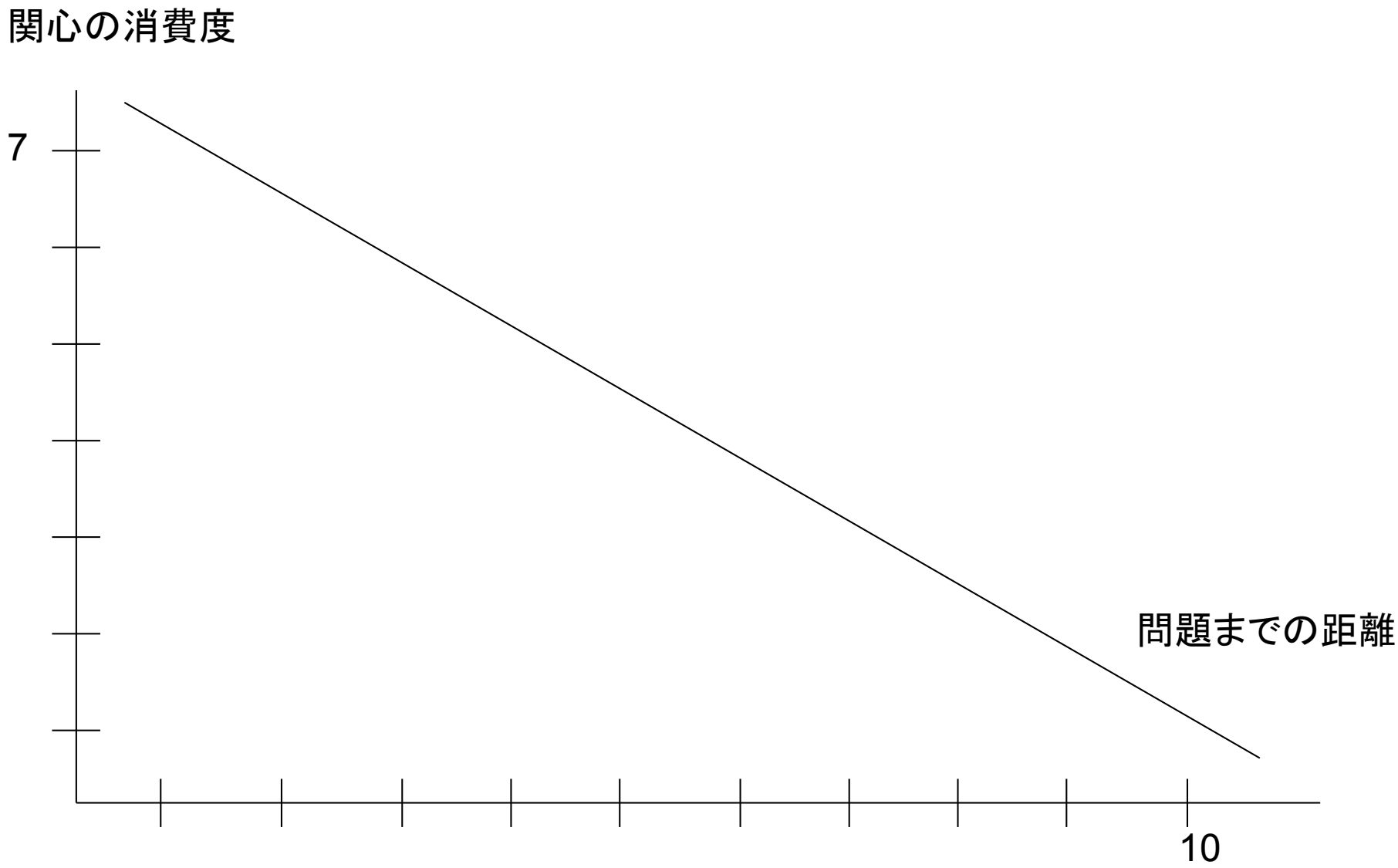
認識力



有限の関心

- 人間の関心は有限である
  - 10個以上の関心は保持し続けることは難しい
- 関心の有限性が、問題、アプローチ、論理フロー/論理相互作用、論理記述をIdealから引き離す
  - エンドユーザは、日々の業務遂行に関心を消費する
  - 開発依頼者は、様々なシステム関連業務に関心を消費する
  - 開発者は、掛け持ちしているプロジェクトや、最新技術情報に関心を消費する

関心の振り分け度合いと問題への距離の相関性



Idealなソフトウェア開発

- 問題解決の主体者が、開発者である
- 問題解決の主体者がプログラミング言語に詳しくない場合は、開発者が問題解決の主体者の知的パートナーとして関心を多く振り分けて関与する
  - プロジェクトは掛け持ちすべきではない～関心が消費されるため
  - 問題解決のために、様相について認識力が高い人であるべき
    - 論理記述のみならず、全体をIdealに近づけるために
  - 問題解決のためには、問題そのものを検討すべき
    - IT技術は、業務の効率化に17%しか貢献していない～マッキンゼー調査
  - 開発者は、仕事の論理記述のベストプラクティスの経験伝承者であるべき
    - 論理記述そのものではなく、仕事を論理記述して具現化した、そのものについてのベストプラクティスを、経験と口承？から自分の頭の中に溜め込む

問題を更に生み出すソリューション

- 要素が増えると、他の既存要素と相互作用し合い、様々な様相を更に生み出す
  - シンプルなものは、美しく、強固で、正しい
  - 新しくシステムを開発するのであれば、既存のなにかを消滅させるべき。そうでなければ、複雑系の理論から、更なる様相を生じさせ、新たな問題を生み出す

知働化とは

- 日々の仕事、作業、知的活動は、ほぼ論理的ではない
- これらの人間の活動を、論理的に記述し、洗練し、Idealにすることで、知は具現化し、知が働き始める
- 論理的に具現化することは、いくつかの粒度がある。それぞれの粒度において、フィードバックが必要。なぜなら、それらを考えている人間自体が不完全で、頭脳の動きがフィードバックシステムをベースにしているため
- ソフトウェアとは、論理的記述のひとつの方法であり、知の具現化には便利な方法ではあるが、一方法でしかないことを認識すべき。
- 知働化には、人の認識力が大きく作用する。
- 人の認識力は、人の関心の有限性が大きく関与する。