SEAMAIL

Newsletter from Software Engineers Association

Vol. 14, Number 1 April, 2004

目　次

ソフトウェア技術者協会
Software Engineers Asociation

ソフトウェア技術者協会 (SEA) は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985年 12月に設立されました．

その主な活動は，機関誌 SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約 200人にすぎなかった会員数もその後増加し，現在，北は北海道から南は沖縄まで，450 余名を越えるメンバーを擁するにいたりました．法人賛助会員も 20社を数えます．支部は，東京以外に，関西，横浜，名古屋，九州，広島，東北の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます．SEA は，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

ＳＥＡＭＡＩＬ Ｖｏｌ．１４， Ｎｏ．１ ２００４年４月１日発行 編集人 岸田 孝一
発行人 ソフトウェア技術者協会（ＳＥＡ）
〒１６０-０００４ 東京都新宿区四谷３－１２ 丸正ビル５Ｆ
T: 03-3356-1077 F: 03-3356-1072 E-mail: sea@sea.or.jp URL: http://www.sea.jp
印刷所 有限会社 錦正社 〒１３０-００１３ 東京都墨田区錦糸町４－３－１４
定価 ５００円 （禁無断転載）

自己組織化するプロセスを作る
―SPO への招待―

伊藤 昌夫
株式会社ニルソフトウェア

目次

1. はじめに

...
机, 書物, 窓...
各々の物には反駁できない.
そうだ,
現実はリアルだ.
そしてそれは浮かぶ.
—巨大で, 堅固で, 触知できる—
このうつろな瞬間に.
現実は
常に墓穴の縁にある.

オクタビオ・パス (真辺訳), 「予行練習」 続オクタビオ・パス詩集, 土曜美術社

神話

「プログラマはプログラムが好きで, 管理が嫌い.」
「プログラマは怠け者で, 品質に対する意識に乏しい.」

よみびと知らず

テレビを見ている嬰児
何かを見つけ 叫ぶ

激しい動きの中で 全身を緊張させて

フロイドの糸車 彼をはこぶ
そおっと
そおっと
そして気付かぬよう あなたは やはらかな胸に 銀の刃物を突き刺す

宇宙を超えた身体は 小さな人間のサイズに

胸に大きな穴を開けた 灰色の大人たち
さむい
さむい

胸の暗黒の空洞を みな モノで埋める
あせる
あせる
(おわかりでしょう) いつも モノはこぼれおちる

大人の胸の空洞に いま
赤い血を滴らせた 穴を持つ
こどもが わらって いる

1.1. SPO vs. SPI

SPO というのは筆者の造語になります．ソフトウェアプロセス最適化(Software Process Optimization)の略になります．対概念として，ソフトウェアの世界でよく知られた言葉に SPI があります．先ず，SPI の説明から始めます．

SPIという言葉は，ソフトウェア開発の現場において知る人ぞ知るというものから，もう少し拡がりを持ってきたように思います[1]．ちなみに，SPIは，Software Process Improvement の略語になります．一般にはソフトウェアプロセス改善(活動)と訳されることが多いようです．

もともと，この SPI の議論が面白いのは，ソフトウェアが置かれている特殊な状況があって，それになんとか対応しようとしていることだと思います．その特殊性というのは次のことになります．一つには，ソフトウェアは，所謂ハードウェアのような製造工程を持たない．次にソフトウェアは単品である，即ち一回限りでしか作られないこと．最後に，周辺技術の進化が激しいこと(プログラム言語，基本ソフトウェア，手法…)が挙げられると思います．

しかし，ハードウェアも大変化に対する対応が必要である，という傾向を持つようにも思います．その意味でソフトウェア開発の問題というのは，極めて現代的な状況を，先取りしているのかもしれません．

その中で我々ソフトウェアに関わる人間ははどうしているのか？ということをご説明したいと思います．更には，これまでに SPI を通して，分かってきたこと，解決できてないことも合わせてご紹介できればと考えています．

1.2. 何が新しいのか

さて，ソフトウェア開発に対して，他の分野の人はどのように考えるでしょうか．ある人は，進んでいるように思うでしょうし，別の人はとんでもない仕事の仕方をしていると思っていることでしょう．殆どの人は後者でしょうか．多分に専門化が進んでいないという意味で，そういう批判を受ける余地は多々あるでしょう．それには，ソフトウェア産業が主たる部分でゼネコンと同様なピラミッド構造をしているとか，教育を受けることなく参入する人が多いといったことを挙げることができます(多くの場合，人月でコスト計算がされるという事実がそれを証明しています)．

しかし，ここではそのように暗い話は脇において，ソフトウェア開発においてSPIが切り開いた新たな地平の話をしたいと思います．それはものつくりに係る様々な分野において，有効に機能すると信じています．ちょっと遠回りの説明をします．しかし，ここで迂遠な道を通らない限り，単なる他分野の視点でしかSPIを見ることができなくなります[2]．そうだとすると，少なくとも他分野の人にとっては，何のメリットもない議論になりますから．

1.3. ソフトウェア開発におけるカイゼンとは何か？ そしてSPO

SPI は先にも書いたように，ソフトウェアプロセス改善と訳されることが多いようです．しかし，私は余りこの訳語が好きではありません．プロセスはモノではないので，改善(カイゼン)しようがない．このようにいうと，そんなことはないだろう．君は製造現場の QC 活動を知らないのか？といわれそうです．ただ，ソフトウェアの場合は違うと思っています．ハードウェアの製造過程のように同じものを生産する場合には，歩数を数えたり，ストップウォッチを持つことの意味はあります．しかし，ソフトウェアでは同一のものが生産されることは決して！ありません．また，同様にソフトウェアを取り巻く技術的環境は急速に変化します．例えば，数年前にだれがここまで Web ベースのシステムが広まることを予想しえたでしょう．

このようなソフトウェアの世界において，カイゼンは古いパラダイムにおける成功体験でしかありません．カイゼンというよりは，環境へ適応可能な柔軟で健全な組織を作ること，これが現代の新しいパラダイムでのキー

[1] 例えば，経済産業省の次の新聞発表参照「ソフトウエアプロセスの改善に向けて～SPIへの今後の取組み～」
「http://www.meti.go.jp/kohosys/press/0003639/index.html

[2] なんと多くのSPIの議論が，製造メタファのもとになされてきたことか！

ワードになるだろうと思います．即ち，個々の一回限りのプロセスはもとより，自己組織化を可能とする(メタな)プロセスを作ることが重要になります．

そしてそのことが，カイゼン概念を引きずるSPIという言葉よりも，ソフトウェアプロセス最適化(SPO)という言葉を使用する理由になります．

1.4. SPOによって健全な組織とソフトウェアを作る

この健全なというのは，SPINのメーリングリストで色々話をしていた中で，新谷さんがおっしゃった言葉です．この健全というのは，もとの趣旨とは違っているのですが，とても気に入りました．SPO(ソフトウェアプロセス最適化活動)とは健全な組織とソフトウェアを作る行為だとも云えるのではと思います．このことについては後でまた述べることにしたいと思います．

SPO はもちろんたった一人でも始めることができます．しかし，最終的には組織が変化しなくてはなりません．変化によって環境に適合する形を組織が得たときを良とします．ただ，注意が必要なのは，不断に環境は変化するわけですから，常によい状態にいる必要があり，そのことをここでは「健全」と呼んでいます．

2. モノとコト

SPOという時に注意しなくてはいけないことがあります．あくまでプロセスをみるのだということです．お客様にお渡しするのは，プロダクトなのですが，あえてプロセスを見る．或いは，プロセスを見るしかない．ここに最初の難しさがあります．

最初に，プロセスを見るということの意味を考えます．次に，SPO が含意するようにプロセスを変更するということは，プロダクトにどのような影響を与えるかということです．

迂遠ですが，先ずはその点から考えてみたいと思います．

2.1. コト中心の世界観[3]

物事という言葉があります．辞書によれば，「一切の有形・無形の事柄」とあります．或いは「事物」と云う言葉もあります．この時は，物事と同じ意味で使われる場合もありますし，ある特定の事象における，事件（起こったこと）と関連する物を指します．

このように事物ないしは物事からは分解を通して，「こと」と「もの」を得ることができます．廣松氏の言葉を借りると[2]次のようになります．先ず暫定的にモノは，「"名詞類"で表される与件」であり，コトは「"文章態"で表わされる事態」である．「ウシが逃げる」というとき，ウシはモノで，「ウシは逃げる」というのがコトに相当します．なんとなく，分かったような気になります．

廣松氏は，精緻に分析をするのですが，それはさておき，ウシは絶対的にウシではないのだということを理解する必要があるのではと思います（多くの場合，ウシの代わりに cattle でも良い）．良い例として，「コレは逃げる」というときに，コレは何であるかということは，ある意味どうでも良いわけです．少なくともコレというモノはないのですから．

それは指示子の話である．便宜上コレといっているだけで，それはウシをポイントしている．しかし，これもそう簡単ではないだろうと思います．クリプキ流の指示に関する議論を待たなくとも，この発話で言われるコレは，逃げるものとして措定されているからです．指をさしてコレといっているかもしれませんが，指さなくとも逃げるという言葉でコレは（走っている）物体だということが分かる．

モノ的に見れば，ウシというものが厳然としてあり，それが走っているということになります．しかし，コト的に見れば，走っているというコトがあって，その事態の一部としてコレやウシがあるということになります．

このことは我々の認識が先ずはコト的であろうということに起因しています．反省的にコレは（このスピードでこの大きさならば）ウシだというのはあっても，「走っている」ということが先ずあるのだろうと思います．

先にお話したように，お客様にお渡しするのはプロダクト（モノ）であって，プロセス（コト）ではありません．しかし，認識的にはプロセスが常に優先します．このことは，認識論だけに限りません．ソフトウェア自体が，そもそも（岸田さんがよくおっしゃるように）プロセスを内在しています．

給与計算をするプログラムは，給与を作っているわけではなくて，ある給与計算のプロセスを（人間に代わり）実行しています．ソフトウェアというプロダクトもユーザのプロセス（それは新たに作られる場合もありますが）をなぞっているということになります．

開発プロセスと製品が内在するプロセスはもちろん別です．しかし，ソフトウェアがお客さんのプロセスを観察し，場合によっては新プロセスに変更することを企画し作るものである．そのことは我々自身の開発プロセスにおいても同様であるということは，最低限いえると思っています．

[3] この文は昔オブジェクト指向について書いたところを殆どそのまま使っています．オブジェクトというとモノだと考えるし，多くの入門書の類もそのような説明をします．しかし，それでは確実に行き詰ります．理由は簡単で世界はコト的にみるべきだからです．あなたの周りを見渡して，だまったまま居る机やイス，白いだけの部屋，こんなものを計算機の中にモデルとしてもちこんでもしかたがないでしょう．

2.2. プロダクトの良し悪し

プロセス改善に関して，以前，次のような議論がありました．「プロセスを良くしたら本当にプロダクトは良くなるのか」．

これは主として品質に関する問いだろうと思います．もし，プロセスを良くしたところで，プロダクトが良くなるという保証ができないのならば，何のためにするのだ，というのは，云わばプロセスに関わる人間にとって，常に気にかかる頭の上の重石のような存在であったわけです．

これに関して幾つかの説明があります．それは，私がその証拠を持っている，ということではありません．プロセスについて考えるということは，一見ただプロセスを見るということのように思いますけれど，プロセス思考とでもいうべき意識が必要になります．

2.2.1. プロセスはモノではない

先にも書いたようにプロセスはモノではありません．ですから，良いプロセスという云い方はそもそもが錯覚になります．ソフトウェア開発には多くの人が関わります．従って，このような錯視が生じやすくなります．典型的には，あなたのプロセスはと聞かれて，キャビネットを占領している社標準を示すようなものです．

プロセスは，そこに関わる人々の関係によって定まります．例えば複数の人が集まれば，何か暗黙のルールができますが，そのルールに従って人は行動するようになります．活動理論における我々の活動の説明はこのことを良く表現しています．

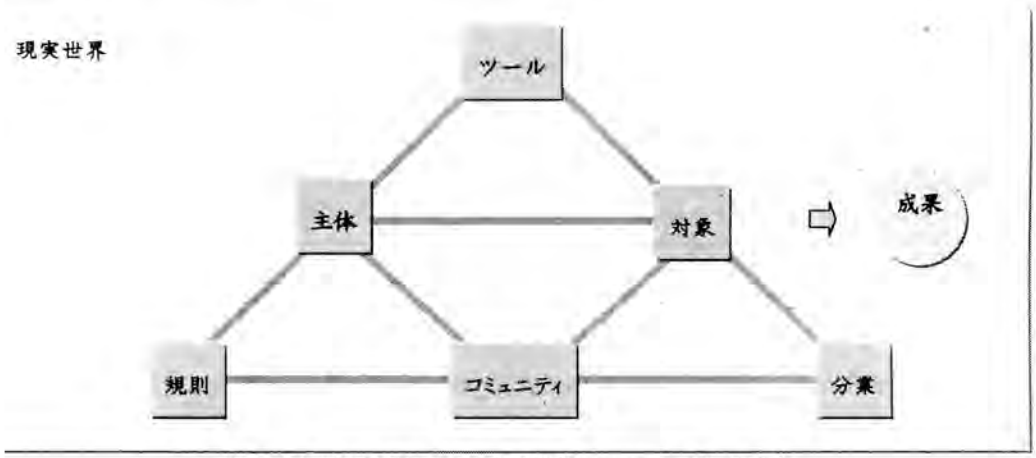


図 1．人間の活動システムの構造（Engeström, Y. 1987）

現実世界の中で，私はツールを用いて対象に働きかけることによって何かを作ります（もちろん，ここでの何かは私にとってはソフトウェアです）．私は一人でソフトウェアを作ることはなく，何らかのコミュニティに属しています．そこでは，分業と規則があります．

ですから，良いプロセスは何かというのは，多くの場合問題の立て方が間違っています．ある与件の中で，結果がどうだったかしかいえません．A 組織のプロセスは成功したから，私もそうしようといって，仮にそのマネができたとします（それすらも何をしてマネというかがありますが）．多くの場合望む結果を得ることはできません．そこには別のコミュニティがあり，人間が異なりますから，当然違った規則を持ちます．また働きかけるべき対象も異なりますから分業（作業分担）も異なります．他でのプロセス（活動）がそのまま移入できないということは，当然の結果です．このことは良く理解しておく必要があります．

例えば，ある変更をプロセスに加えようとする時，通常は抵抗に出会います．抵抗があるというのは，上記で説明したようにごく自然な反応です．そこをトップダウンでというやり方もあるでしょうが，一番良いのは，その変更が直ぐに効果を持つことを説明できる理解が得られることであると思います．もちろん，プロセスの変更

は，非常に小幅な変更を繰り返すのでない限り，理解を得がたい．それによる効果が想像できないから．その場合でも，パイロット的に特定のプロジェクトで実施するなどを通して，最終的には理解を得るべきだろうと考えています．逆にそれができないのであれば，そこには新しい規則と作業分担を持ったコミュニティは生まれないのではと思います．

2.2.2. プロダクトの良し悪しもプロセスによってしか分からない

一方で，プロダクトというのは，何かしらそれだけで直ぐに良いか悪いかが判定できるように思います．ソフトウェアはそれができないから難しいという云い方も良く聞きます．

しかし，よく考えてみると，プロダクトの良し悪しの判定というのは，今では殆どプロセスを見るしかないわけです．このことはハードウェアでもソフトウェアでも同様です．

例えば，ソフトウェアが仕様に対してある妥当性を持っているかということは，多くの場合テストによって判定することになります．そこでの不具合の数によって，出荷判定をすることになります．もし，テストを実施なければ不具合もゼロなわけです[4]．

テストというプロセスがあるからこそ，プロダクトの良し悪しが分かります．このことは，もちろん，多くのハードウェアの検査においても，それが検査という「プロセス」である以上，同様です．

2.2.3. 物象化による錯視

物象化という言葉によって示される代表的なものに，貨幣があります．紙幣は紙に過ぎないのに，大事に持ち歩きます．それは，その紙が大事なわけではなくて，それが何かと交換可能であるという点において，大事なわけです．逆に云えば安心して何かと交換可能である場合にのみ，大事なモノとなります（スーパインフレにならなければ通常は）．またそこには軽重があります．殆どの人にとって，一万円札は，千円札よりも立派に見えるでしょう．もちろん，原価は多少違うのでしょうけれど，それが立派なのは，より多くのものと交換可能であるという点においてのみそのような上下関係がつきます．

コトだけで生活するのはつらいので，必ずモノで代表させます．我々の認識も多くの場合そのモノに頼ります．それはそうすることが通常は楽だからであって，組織のプロセスを変革しようとする場合に，そのモノに頼ることは，錯視を正とするという意味において（つまり幻を扱おうとすることになるので），多くの場合間違えることになります．その良い例を上記に２つ挙げました．

[4] もちろん，ある一定以上の不具合が出てこないと，テストプロセスが不十分であるという判定を下す方法もあります．実際にそのような組織もあると思います．しかし，対象が常にあるレベルの品質を持たない限り，そのことに妥当性がないのは同様です．ここでも，ソフトウェアが一回しか作られないものであることが関係しています．

3. プロセス最適化の一つのある例

こんな声が聞こえそうです．抽象的な話ばかりで分かりにくい．それで，かなり簡略化しますが，一つの物語を作ることにします．

3.1. ある物語

受託の開発をしているA社の若い人は，いらだっています．毎日のように残業が続きます．それでもプロジェクトの終わりが見えてきません．本当は良いものを作りたいのだけれど，仕事に追われて十分なテストをすることもできません．なんとかしないと燃えつきそうです．

一方で，プロジェクトマネージャや会社の上層部もいらだちを隠せません．次から次へとプロジェクトはオーバーランするし，コストは超過，リリース後の不具合で，お客さんからは叱られっぱなしです．

みんな，それぞれの立場でなんとかしたいと思っています．しかし，品質を重視しようとすれば，納期はもっと遅れますし，コストもかかります．コストを絞ろうとすれば，倒れる人もでてくるでしょうし，品質が悪くなるのは目に見えています．結局のところ，QCD はトレードオフで，それぞれを解決する良い策が思いつきません．

行き詰っているところで，社内向けに開発ツールを作っているＢさんはふと思って，遅れに着目することにしました．いつも遅れているのに何故お客さんは嫌にならないのだろう．確かに機嫌よくはない，いつも叱られるけれど，終わればまた次の仕事を発注してくれています．それはコストが安いから？他ではない技術を自分たちが持っているから？もともと納期を気にしていないから？どうもそんなことはないようです．

では，見積りがいい加減だからでしょうか．確かにソフトウェアの見積りを完全にする方法などありません．しかし，本当に曖昧ならば，プロジェクトの半分は納期を超えたとしても，残りの半分は納期以内に収まるはずです．自然界はそのようになっているはずです[5]．では，なぜ，いつも遅れるのか．

何かお客さんが満足している点があるに違いありません．よくよくプロジェクトの人から話を聞いてみると，最初は曖昧な要求で，それは開発が進むにつれて決まったかと思うと，また変わるということがおきているようです．それはお客さんのビジネス要件で，そのことを否定できない．お客さんに向かって，仕様を凍結していただかないと作れませんとはいえない．何れはそういうのだけれど，それだとそのソフトウェアの存在価値がなくなると云われれば，仕方がない．その変更を最後まで飲むことによって，お客さんには満足があったわけです．

ここでもまたトレードオフの問題になりましたが，これを与えられた条件として，解決することをＢさんは考えたいと思います．そうすると問題のある細部が見えてきました．次々来る要求に対して，情報がきちんとみんなに伝わっていない．ある人は要求が変わったということで，他の部署とのインターフェイスを変更します．しかし，その部署では変えられたことを意識していません．もともとその変更は自分たちに関係ないと思っていましたから．テストをしている人たちは人たちで，混乱しています．なにせ，次から次へと似たようなものが開発側から吐き出されてくるのです．何をどうテストすれば良いのかが，分かりません．今テスト中のものを一旦終了し終えてから，変更箇所だけを新しいモジュールでテストするのか？　それとも古いものはテスト途中でも捨ててしまうのか？それだとテストがいつ終わるのか分からない．

このことが，問題として大きいということが分かりました．QCD の劣化という現象に対して，すこし様子が分かってきたことになります．

どうもお客さんの頻繁な変更要求に対応できていないようです．そこで次のように規則を決めることにしました．先ずはある段階のものを基準とします．これにバージョンゼロと名前をつけます．お客さんから変更して欲しいという要求があったところで，はい分かりましたと返事するのではなくて，みんなで(各サブシステムの担当者やマネージャが全て出席する場でという意味です)，その情報に関して議論することにします．ここでお客さんの希望を聞くか，どうかを決めます．それによる影響範囲はどこまで及ぶかといった調査や，どの時

[5] もちろん，我々は人為的な世界に住んでいるので，必ず営業的な圧力がかかるため，平均はオフセットします．ここでは圧力がかかるまでとしています．

点から反映するかを決めます.

変更指示は個々に差分として管理します. 変更がまとまったある時点で, リセットをして, バージョン1と名づける一連の要求やモジュールを作ることにします.

これによって, 自分たちが何を対象に仕事をしているか(例えば, どの仕様書に従って設計したり, 実装しているか)が分かるようになりました. また, お客さんにとって小さな変更でも, 影響範囲が広いので, といった形で要求自身も整理され, お客さんに説明しやすいものとなりました.

相変わらずの忙しさには変わらないものの, より納期に対しては確実になりましたし, テストが整然と行われるようになり, 重大障害も総じて少なくなっていきました. B さんは, よりこういった要求の整理が楽になるようにツールを提供しようと思い始めました.

3.2. 物語と真実

先の物語は物語として, 勝手に作ったものです. 所謂, ソフトウェア変更要求や技術指示書に基づく変更管理の採用により, 何が変わるかを示したものです.

この物語の中では, 実に簡単に要点が分かったように書いていますけれど, たった一つのプロジェクトでも規模が大きくなると, 数ヶ月の調査が必要になる場合があります. なぜならば, 複数の要因が絡み合っているからです. 定性的に, XXX をすればよいというのは, 簡単ですけれど, それでは損失の方が大きくなるかもしれない. 上記の場合ですと, みんなで変更要求を議論する場(これは教科書的にはソフトウェア構成管理委員会:SCCB と呼ばれるものです)を作って議論したりする時間や, 流通させる変更・構成情報の管理にかかる手間になります. お客さんからみても, 何かレスポンスが悪くなったように感じるかもしれません. 全てはトレードオフなのです. 利点が大きくなければ, 単にコストをかけることはムダです.

様々な現象から, 原因を見つけ, 対処の損得を考えた上で, どうするかを決めることが, 実際のプロセス最適化では必要になります.

4. 組織のプロセス最適化

4.1. ボトムアップによる新たな組織化

先の例は，あるプロジェクトに関してでした．個々のプロジェクトを成功させるというのはもちろん重要な問題です．しかし，ここで得た知見をモトに，ボトムアップ的に組織そのものを変えていくことができます．

Basiliさんの経験工場[3]の概念を借りて，パッケージングを行います．パッケージングとは，あるプロジェクトで得られた経験（そこには様々な条件と実行されたプロセス）をできるだけ数値化し整理することを示します[6]．これによって，あるところでの経験が（そのまま使えるというわけではないものの），他のプロジェクトで再利用できる可能性が高まります．

もし，そこに傾向があれば，それに応じた形に組織を変えたほうがよいことになります．この変えるということの意味は，場合によっては単純な組織の変更かもしれませんが，通常は対顧客との関係や開発者の教育といったことも含みます．

何らかの傾向は存在する可能性が高いといえます．ある組織を考えたときに，常に全く異なる製品をいつも作っているわけではありません．全く同じではなくても，類似の製品を作っている．その時には，必ず何らかの傾向を見つけることができます．

ここまでが，ボトムアップでのアプローチになります．もう一つは，トップダウンの方法です．全て決めうちで変革してしまう．ある特定のプロセスモデルに合うように，既存のプロセスを一律変更する．よいやり方のようには思いませんが，一般的にはこちらの方が多いように思います．導入までの時間が少ない，いきなり全社レベルで変更できるといったことが利点といえば，利点でしょうか．

どちらがよいかは勿論その企業おかれている立場によります．ただ，最適化ということを中心に考えるならば，ボトムアップしかないように思います．

次に示す図は，Agile 系の開発手法の一つである適応型ソフトウェア開発 [4] にある図を援用します．

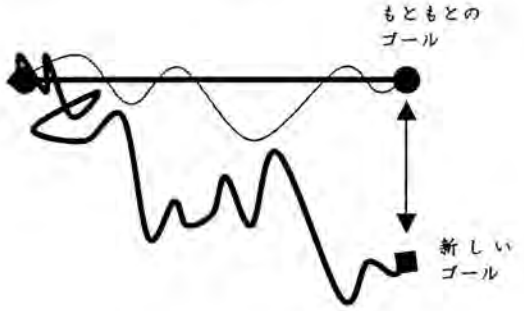


図 2. 変化するゴール

組織のゴールが今関心を持っている時間フレームに対して，固定と考えることができる場合．ある定まったプロセスモデルを導入すれば，何とかゴールにたどり着けるでしょう．しかし，環境の変化によってゴールも変化するのだとすると，固定することは阻害にこそなれ，役には立たないことになります．

最初に健全と書いたのは，SPO においてバイアスをかけないという意味で上記のことをお伝えしたかったの

[6] 単純なプロジェクトの振り返りを示す幾つもの言葉があります．PIR(Post Implementation Review: 実装後のレビュー)，Postmortem(事後分析)，Post-audit(事後監査)，loopback-analysis(折り返し分析)．しかし，ここでは単なる分析だけではなくて，それを次につなげるための整理も含んでいます．

です.

4.2. SPO は誰が実行するのか.

4.2.1. SPI 視点:プロセスグループ

ソフトウェアのプロセスの変更は, 最終的にはプロジェクトに閉じることなく, 個々のプロジェクトを横断し, 組織に広がる必要があります. この時に, プロセスに関心を持って組織を見る人々のことを, 一般にプロセスグループ或いは SEPG (Software Engineering Process Group)と呼びます. この他には, ソフトウェア開発側で構成管理を行う人や, 品新保証(管理)をする人が必要ですが, プロセスに関しては先ずはプロセスグループに属する人たちが対応するとします. [1]に従えば, 彼らのアクティビティは次になります.

- 総てのレベルの管理者の支持を得て, 維持する.
- 現状のプロセスの様子を分かりやすくする.
- あるやり方を変更した時に, その意味を伝えるなど, 現場の手助けを行う.
- 開発者と協力して, 新しいやり方や技術を入手したり導入するのに協力する.
- プロセスの向上のためにトレーニングや教育を手配する.
- 特定のプロセス向上が行われているか否かを検査し, その状況を報告する.
- 管理者や開発者と協力してプロセス定義を作り, 維持する.
- プロセスデータベース[7]を維持する.
- プロジェクトや管理を進めるにあたり, プロセスに係るところで, 相談にのる.

[1]では開発人員に対して, 1~3%がこの役割を担うとしています.

4.2.2. バランスする QCD

先に述べたとおり, 実際のプロジェクトでQCDはバランスしている. 今, それぞれをあるレベルに維持するためにかけることのできる費用をINV_Q, INV_C, INV_Dとすると, 与えられたプロジェクトに関して以下が成立します.

$$INV_Q + INV_C + INV_D = \text{一定}$$

例えば, 納期を短くしようとすれば, INV_D は増加しますから, 品質を犠牲にするか, 見合うだけ製造コストを変更しなくてはなりません. 例えば, 調達の仕方を変えることによって実現するということになります. ところが, このことは決して容易ではありません. 部品の調達ではなくて人間が対象ですから, 開発者を変える, 長時間労働を強いるということは, 当然の帰結として, 新たな教育が必要という別のコストを生んだり, 品質の劣化を招く可能性があります. 後者の場合, 更に品質維持のためにコストをかけようとすると, ますますINV_Cが増加することになりますから, 更に別の手段をとらないといけないということで, 破綻に向かうことになります.

先の例でプロセスグループに対して 1~3%とはいえ, コストが増大するとすると, それはハイリスクの投資であると考えない限り, 個々のプロジェクトは, 余計に苦しむことになります. また, 開発のやり方の変更は, それが上記の費用のいずれかに効果があるということが明らかでない場合, 受け入れがたいものとなります.

4.2.3. SPO 視点:実行主体は明らか

プロセスグループがなくても, 実行の主体は明らかです. 開発プロセスにおいての実行主体は開発者です. しかし, 幾つかの注意が必要です.

SPI が持っている製造業的カイゼンの隠喩は, その帰結が明らかな場合においてのみ, 実行主体にとってプロセス変更が納得のいくものとなります. ラインで, A というボルトの入っている良く使うバケットを, B という余り使用しないバケットの前に持ってくる. ボルトを手にするまでの所要時間が一回あたり 0.5 秒縮まった. これによりラインスピードのアップに貢献できたとすれば, 目に見える結果ですし, 因果関係も納得できます.

[7] プロセスデータベースとは, プロセスに係る情報を集めたものです. 具体的には, 個々のプロジェクトに対してプロセス及びプロダクトメトリクスとなるものを集約します. 例えば, レビューの実施回数や時間, プログラムの行数等が相当します.

かつ，これに要するコストはバケットの位置を一回変更するだけに過ぎません．

一方で，ソフトウェア開発の場合は，原因－結果がそれほど容易ではありません．単純に，キーボードのストロークを短くすることで，生産性が上がることは考えられません．或いは，もっと大胆にデータベースの設計方法を変えたからといってそれが有効か否かは俄かには判定できません．

もともとがかように難しい問題であり，故に答えが常に見つかるわけではありません．これはプロセスグループが居たとしても同様です．

そこで，自身が開発者であるとして何がプロセスに対してできるかを考えてみることにします．

手段は大きく分けて次の 2 つになります（ここでは，やさしいお客さんが機能を落としてくれたとか，できる新人がふと入ってきて彼は研修期間中でコストに繰り入れなくて良いという僥倖はないものとします）．

［対処策-1］ プロセスの変更

プロセスの変更は次の三つに限られます．(a)分離するか，(b)併合するか，(c)省略するか，(d)追加するの何れかです．分離の場合は，主として並列的に動作する場合に有効です．単一のアクティビティやタスク[8]では当然並行動作ができませんから．併合するのは，不要なオーバヘッドをなくすために取られます．あるプロジェクトで，異なるチームが同様のことを行っている．この時に併合することで，全体の効率を上げるという場合です．省略するというのは，もちろん何かの犠牲を伴います．しかし，その犠牲が省略によるメリットよりごく小さいと見なせる時に省略することが可能になります．例えば，十分なコードレビューがなされていて，かつ動作に関してはモジュール組み合わせで確認でき，カバレッジもこの時に取得できるという場合に，ホワイトボックスによる単体試験を省略するという場合が相当します．最後の追加するは，一般にはかなり高度なプロセスの変更になります．直接的に，原因－結果が見えないという意味においてそうなります．例えば，総ての設計アクティビティに対して，ウォークスルーというタスクを義務付ける．ここではコストが掛かるけれど，それは品質の向上を伴い，結果的に不具合修正に要する時間を大幅に削減できるというのが，このケースに相当します．しかし，本当にそうかというのは，昔あった月着陸船のゲームと同じで，効果が遅れて現れてくる分，判断がしづらいということになります．

［対処策-2］ ツールの導入或いは製作

活動理論の図（図 1）にあるように，我々が活動するとき必ず道具を使用します．ソフトウェア開発者は特殊な立場にいます．それは，ツールもまた多くの場合ソフトウェアであり，自分達で作れるということです．機械を製造している人たちが，治具を作ることができるのに似ています．それほど，大規模なものでなくても，マクロを書いてみてといったエンドユーザで加工できる環境もありますから，ツール作りにかけるコストはそれほど大きくないかもしれません．

ツールは自動化しますから，その製作コストを下回る自動化効果があれば，十分に QCD に対して貢献するということになります．

一方で，注意がないわけではありません．プロセスに適合しないツールの利用によって，余分なコストがかかる．或いは，個別のツールが増え，組織として一貫したデータ取得が出来ない（誰でも自分の作ったツールがかわいいですから）．

どちらにしろ，実行の主体は，開発者であったり，品質保証（QA）に携わったりしている人であって，プロセスグループの人間ではありません．プロセスグループの人間は，単にそのサポートができたり，横どおしの視点を持てるということに過ぎません．決して，機械のように開発者のあなたを動かすわけではありません．

先ずは，当事者である開発者や QA の人がプロセスに対して関心を持つということが重要だろうと思います．

[8] 習慣的に，プロセスはアクティビティに分解され，アクティビティはタスクに分割できるとします．この時，プロセスは開発プロセス，保守プロセスといった大きなくくりであり，タスクは，それ以上分解できない最小単位であるとします．アクティビティはその中間になります．

5. まとめ

今回は，ソフトウェア開発におけるプロセス改善の基本的な考え方についてご説明しました．何度も書いてますようにかなり迂遠な説明ですので，お分かりにくかったかもしれません．

しかし，10 年以上，ソフトウェア開発プロセスに関わってきた人間として，昨今のプロセスをめぐる議論が，レベルの取得とか，何かの認証を取得するといった議論に終始していることをさびしく思っています．もし，プロセスの議論がそこにとどまるのであれば，他の分野の方にも何かしら役に立つという拡がりや力は持ち得ないと思っています．

少し根本的なところからプロセスを考える始めること．次にそのことを踏まえた様々な側面について考察すること．そうすることで，最終的にみなさんの仕事のお役に立つだろうことを，私自身は確信しております．

【参考文献】

[1] Fowler, P., Rifkin, S., "Software Engineering Process Group Guide", CMU/SEI-90-TR-24, SEI, Sep., 1990
[2] 廣松渉, 世界の共同主観的存在構造, 廣松渉著作集, 第一巻, 岩波書店
[3] Basili, Victor R., Caldiera, Gianluigi and Rombach, Dieter H. The Experience Factory, Encyclopedia of Software Engineering - 2 Volume Set, pp 469-476, Copyright by John Wiley & Sons, Inc., 1994.
[4] Highsmith, J. ADAPTIVE SOFTWARE DEVELOPMENT: A Collaborative Approach to Managing Complex Systems, Dorset House, 2000.

A. サイドバー 「SPA:ソフトウェアプロセス評価」

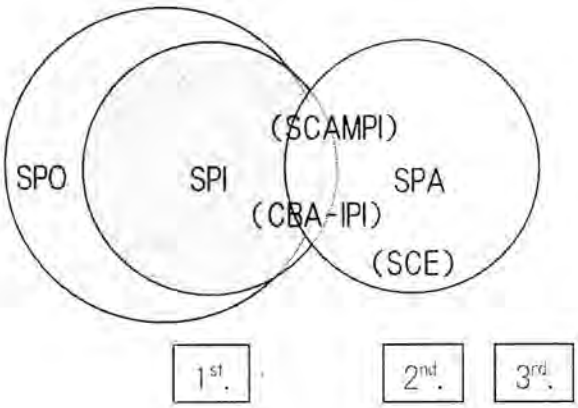
SPIという時に，注意しないとSPAの話をして済ますことがある．SPAとはSoftware Process Assessment の略で，良く知られたものに，CMM(Capability Maturity Model – 能力成熟度モデルと訳されることが多い)がある．メンテナンスをしているのは米国のSEI(ソフトウェア工学研究所)である．初出は1986年である．

この他に欧州を中心とした幾つかのSPAがある．CMMと同時期から活動しているものとしては，SPICE (Software Process Improvement and Capability dEtermination)という活動がある．英国がISOに提出した15504という番号のものに対して，試行の意味を兼ねて作られたグループの名前である．かつての欧州共同体のEspritプロジェクトの一つとして実施されたものである．

公式には，ISO/IEC15504が標準であるから，それに従うということになるのだが，これはSPAに対する規格であることを忘れるべきではない．決して，SPIではない．

評価をすれば，自分の位置が分かる．従ってよい点を取れるように努力すれば，いつかは良いソフトウェア開発ができるようになる．というのも一つの考え方である．しかし，それでも注意が必要である．SEIはDoD (米国国防総省)から資金を得ることによってできた組織であり，SPICEに関わる活動も基を辿れば，やはり軍需関係が主体となっている．本来は，彼らがベンダを選別するという意図を持って作られたことを覚えておくべきである．

ちなみに，良く似た概念として，SCE(Software Capability Evaluation:ソフトウェア能力評価)というのがある．



図．SPIとSPA，そしてSPO
[図の説明： 1st, 2nd, 3rdはそれぞれ，プロセスの評価を行うのが，第一者，第二者，第三者であることをあらわしている．SPOは先ず自律的な活動であり，深い評価を必要とする．]

これは，SPIとSPAの関係を表したものである．CMMでは，誰の評価かによって区別をつけている．SCEでは第二者ないしは第三者による評定(Evaluation)を指している．第一者(当事者)も関与する評価は，CBA-IPI(CMM based Appraisal for Internal Process Improvement: 内部プロセス改善のためのCMMに基づく査定)であり，CMMIの枠組みでは，SCAMPI(Standard CMMI Assessment Method for Process Improvement: プロセス改善のための標準的なCMMIの評価手法)と呼ぶ．

B. サイドバー「プロセスの成熟」

下表に示すとおり[9], CMMは(その成熟度に関する)原型を辿るとCrosbyに行き着く.

表 品質向上活動におけるプロセス成熟度の起源

時期	内容	提唱者
1930 年代	統計的品質管理	Shewhart, W.
1956 年	Shewhart 原則の展開	Deming, E., Juran, J.
1979 年	品質管理成熟度格子	Crosby,P.
1986 年	成熟度レベル	Humphrey,W.
1987 年-	成熟度フレームワーク, 成熟度質問表	SEI

Crosby の品質管理成熟度格子の一部を以下に示す.

表 品質管理成熟度格子

測定カテゴリ	1.Uncertainty	2.Awakening	3.Enligtenment	4.Wisdom	5.Cerainty
	不確実である	目覚めている	みんなが品質向上についての正しい知識を持っている	継続的に品質向上に努めている	確実である
売上に対する品質のコスト	計上: 未定 実際: 20%	計上: 3% 実際: 18%	計上: 8% 実際: 12%	計上: 6.5% 実際: 8%	計上: 2.5% 実際: 2.5%
会社の態度	どうして品質に問題があるのかが分からない	品質に関して常に問題を抱えるていなくてはならないのか.	管理者の関与や品質向上活動によって問題を特定し, 解決できる	欠陥防止が, 我々の業務の一部となっている.	どうして品質に問題がないかを分かっている.

[表の説明: 初期の段階でのコストは, 主として手戻りや不具合対策に大して思いもかけない費用がかかることを示している. 品質を維持することを皆が意識することによって予期しない品質維持のための支出はなくなる. これを象徴的に表しているのが書名「品質確保にコストはかからない(Quality is Free)」である. もちろん, 品質維持に費用はかかるのである. それは, 問題がなくなれば今まで支出していた分をキャンセルできる, 或いは非常に小額で済むという意味で, コストはかからないとしている]

1,5 に大きな意味がある. 1 は品質に関する状態が分かっていない. 謎ばかりである. 5 は逆に問題がない理由を分かっている. 2 から 4 はその過程できわめて自然といえる. たまたま品質に問題がないことが分かったりはしないからである.

また, 確かに品質は成熟する. 完全なる品質!とは, 不具合がないソフトウェアと定義できれば, 完全なるソフトウェアに向かって, 成熟するのである(それは漸近線でしかないだろうが). しかし, CMM が置き換えたように, プロセスは品質の代替にはならない. 開発プロセスは成熟しないからである. そもそも良いプロセスがあったり悪いプロセスがあったりするわけではないのだから.

この品質管理成熟度格子に基づいた CMM におけるレベルが良く話題になる. これもプロセスがモノでない以上, 或いはプロセスとはそこに関与する要素が複雑であるがゆえに, レベルないしは段階そのものに余り意味がない. そもそも, Crosby も次のように述べている(Quality is Free, pp.111).

管理者達が自分達は Wisdom の段階にあると考えている. しかし, あなた自身はそんなことはない, Awakening の段階に過ぎないと思ったとしよう. しかし, そんなことで議論すべきではない. 部屋をでて, 品質向上のために何かをしなさい.

[9] Zahran, S., SOFTWARE PROCESS IMPROVEMENT Practical Guidelines for Business Success, Addison Wesley, 1998.

c. サイドバー 「SPI でないもの」

すこしこういうものは SPI ではないというのを挙げておく(或いは SPO でないということでも本論では良いかもしれないが). 以下は先の経済省の協議会レポートから. ちょっと長が, 引用します.

> 問題例1) A社では、最近必要充分な品質が確保されたソフトウエアが提供できない。
> 対応例) A社では、ソフトウエア開発に関連するプロセスを明確にするために、プロセスを定義し、品質が確保できない原因を特定できるようにした。
>
> 問題例2) B社では、作業のやり直し(手戻り)が頻繁に発生し、予定よりも多額のコストが発生している。
> 対応例) B社では、プロジェクト初期における要件定義の方法がこの問題の主な原因であることを調査により明らかにした。そこで、顧客対応をする営業部門の社員も含めて、要件定義の技術を教育し、また、高いスキルをもつ社員がプロジェクト初期段階で定義された顧客の要件を厳しくチェックすることにより、その後の作業のやり直し(手戻り)を減らし、コスト低減を図った。
>
> 問題例3) C社では、慢性的にプロジェクト遅延が発生し、予定の納期に間に合わない。
> 対応例) C社では、見積り技術及びプロジェクトマネジメントの方法が主な原因であることを調査により明らかにした。そこで、信頼性の高い見積り支援ツールを導入し、プロジェクト実施期間をより適切に算出するとともに、プロジェクトマネジメントの高いスキルを持った人材を配置し、プロジェクトの進捗状況を適切に把握・管理することにより、納期遵守の確保を図った。

これは, SPI のすばらしさを伝えるという例とのこと. 明らかに旧世代のパラダイムになる. なんとなれば, P という問題がある. P に対して手段 M を行った. 上手くいった. という良くできた, ただ単純な物語である. それも手段というのが, 無条件にプロセスを定義したとか, 教育をしたとかのレベルである. よしんば, これは例だからということにしよう. それでもやはり SPI の本質とはかけ離れているように思う. 即ち, コトではなくてモノにしか着目しない. 或いは, QCD はトレードオフであり, その止揚にあたってはプロセスを精緻に観察することが必要ということを示していない. 問題例 3 のツールにいたっては, 一昔前の CASE ブームを想い起こさせるような楽天的云い方になっている. そもそも信頼性の高い見積りツールなど, この世にたった一つも存在していない!

SPI に関わる最大の注意事項は以下に尽きる.

<u>自らのプロセスを詳細に観察することなく, あるモデル(手順・ツール・教育・手法)を導入してはいけない.</u>

Yet Another View of SPI Based upon ActivityTheory

岸田 孝一
Kiss cedar, Call witch!
SEA-SPIN Meeting Mar 11, 2004

1

What is Activity Theory?
活動理論とは？

何らかの文化的・歴史的背景を持つ人間の集団的活動を研究するための概念的枠組みを提供する複合的理論.
- 教育(発達)心理学
- >> 社会学・経営学
- >> CSCW, HCI

2

What is Activity Theory?
（続き・その1）

分析の基本単位は人間の **Activity**.
Activity は 何らかの目的によって動機づけられた **Subject** によって行われ, **Object** を **Outcome** に変換する.
Object は目的とする **Outcome** を目指して一緒に活動する **Subject (Actor)** の **Community**によって共有される.

3

What is Activity Theory?
（続き・その2）

Tool, Rule, および **Division of Labor** が **Subject, Community,** および **Object** の相互関係を **mediate (調停)** する.
Activity は一連の **Action** によって行われ, **Action** はまた一連の **Operation** によって具体化される.

4

What is Activity Theory?
（続き・その3）

Subject-Object-Subject という相互関係には次の3つのレベルがある:
- **Coordination:**
  個々の **Actor** にフォーカス
- **Cooperation:**
  Shared Object にフォーカス
- **Co-Configuration:**
  Work Practice の **Re-elaboration** にフォーカス

5

Activity Theory の発展

- **Philosophy**: **Marx, Engels**
- **Psychology**: **Vygotsky, Leontiv, and Luria**
- **Socio-Computing**:
  Yrjo Engeström
  http://www.edu.helsinki.fi/activity/people/engestro/
  Bonnie Nardi
  http://www.darrouzet-nardi.net/bonnie/

6

参考資料

L.Vigotsky, *思考と言語(新訳版)*. 新読書社. 2002.

J. Engestrom, *拡張による学習:活動理論からのアプローチ*, 新曜社, 1999.

B.Naadi (ed), *Context and Conciousness: Activity Theory and Human Computer Interaction*, MIT Press, 1996.

B.Nardi & D.Redmiles (ed), *Special Issue on Activity Theory and the Practice of Design*, (CSCW誌, Vol 11, No.1-2, 2002.

Web Page: (Colorado 大学 Denver 校教育学部)
http://carbon.cudenver.edu/~mryder/itc_data/activity.html

7

5 Principles of AT

- **Hierarchical Structure of Activity**
  Activity, Action, Operation
- **Object-orientedness**
  OOP, OOSD ではない!
- **Internalization/Externalization**
  内的活動と外的活動を同時に観察することが必要
- **Mediation**
  人間の活動を調停(Mediate)するものとしてのツール
- **Development**
  活動は時間とともに発展(進化)する

8

Technical Term の比較

Activity	SE: プロセス中の1ステップ AT: 分析の単位
Object	SE: ソフトウェアが扱う抽象データ AT: 活動の動機となる対象あるいは問題空間
Artifact	SE: ドキュメントその他の成果物 AT: Subject, Object, コミュニティの間の関係を調停するもの

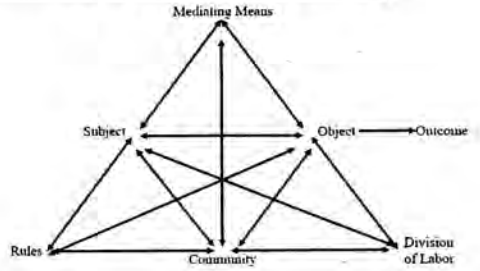
9

Technical Term の比較(続き)

Tool	SE: 人間に役立つアプリケーション AT: 主体-対象間の調停を行う物理的または心理的な仕掛け
Agent	SE: 活動を行う人間またはアプリケーション AT: ある Object を共有するコミュニティの要素
Community	SE: 開発チーム AT: ある Object を共有する Subject の集まり

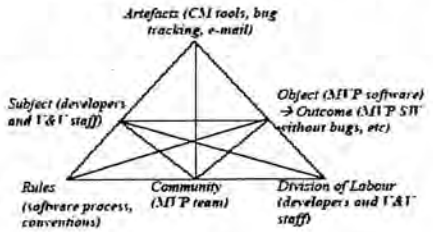
10

エンゲストロームのモデル



11

ソフトウェア・メインテナンスの例

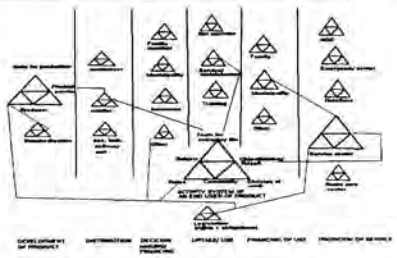


12

カスタマ・サポートの例

13

Alarm Telephone の例



14

分析のツールとしての エンゲストローム図

- 図中のそれぞれのノードに内包される矛盾や対立の分析
- 各ノード間の関係に存在する矛盾や対立の分析
- 複数の三角形のあいだの矛盾や対立の分析
- How to ではなく, Why を考える道具

15

AT Model for SPI
- Community -

きちんと整理された構造を持つ
開発チーム
vs
わがままで自由な
プログラマ集団

16

AT Model for SPI
- Rule -

組織としての
標準開発マニュアル
vs
個人的美意識にもとづく
プログラミング・スタイル

17

AT Model for SPI
- Division of Labor -

設計・制作・検査 etc の
機能別組織
vs
Ad Hoc で柔軟な
チーム編成

18

AT Model for SPI
- Subject -

それぞれの役割ごとの
職能別エンジニア
vs
万能の
スーパー・プログラマ

19

AT Model for SPI
- Object -

予定通りの
開発スケジュール
vs
行きあたりばったりの
仕事の進行

20

AT Model for SPI
- Mediating Artifact -

CMMあるいはISOにもとづく
プロセス・モデル
vs
快適なプログラミング
支援環境

21

Collaborative Activity としての ソフトウェア開発 - *Collaboration の 3つのレベル*

- **Coordination**
  - プロセスモデリング言語
  - 組織のモデル化
  - ユーザ・インタラクション
    - パラダイム / 強制 / ツール統合
- **Cooperation**
  - コミュニケーション
  - アーティファクトの共有
    - 同時性制御
- **Co-Construction**

22

Coordinated Level

グループまたは組織によって行われる日常的・反復的な作業に対応する．

そうした作業は，あらかじめ計画された順序に、それぞれの役割を担うエージェントによって実行される．

それらの役割が何かは，なんらかの規則の形で記述されるか，あるいは暗黙の了解事項として認められている．

各エージェントの行動は，あたかも見えない神の手が働いているかのように，共通の目標達成を目指して，自然に統合される．

23

現在の開発支援環境

ソフトウェア・プロセスすなわち coordinated level に焦点を当てている．

ソフトウェア工学の Production-oriented な思想にうまくマッチしている．

現実の作業におけるパターンや定型性を最大限に活用しうる．

しかし，逆に，Collaboration の別の側面に関して「盲目」になっているきらいがある．

24

Cooperative Level

エージェントのグループの間のインタラクションを含むレベル．このレベルでは，仕事はもはやそれだけで独立したものではない．

それぞれのエージェントの行動は他のエージェントの行動に影響を与え，ある種のシナジー効果をもたらす．

エージェントの関心は，みんなが共有していることがらにフォーカスされ，相互に受け入れることが可能な概念的枠組みや問題解決手段を捜し求める．

25

Cooperative Level における障害

ソフトウェア工学の背後にある **Production-Oriented** な思想が第1の障害である．

それは「設計とは **Collaborative** な学習・問題解決活動である」という一般の認識と矛盾する．

第2の障害は技術的な問題である．たとえば，分散構成管理の技術はまだ確立されていない．

26

Co-constructive Level

現実作業の再構築に対応するレベル．

このレベルでは，作業それ自体が検討・考察の対象である．

新しい，そしてよりよい仕事のやり方を考えることが主題である．

Co-construction の結果は，組織および共有オブジェクトとのインタラクションに関する概念的枠組みの再構築をもたらす．

27

Co-Construction への突破口

Information Interaction Design の意味するものは？

- **Beyond GUI Design ?**
- **Beyond XP ?**
- **Beyond Requirement Engineering ?**

28

現在のソフトウェア工学の弱点

Production-oriented Process という考え方に毒されている．

- Cooperation Support が弱いのは，技術的 理由もあるが，それ以前の原因も大きい．

- Co-Construction Support が欠けているこ とは CSCW や Interaction Design を考え ると致命的．

29

発達的学習活動としての SPI
（エンゲストローム）

- 発達は，習得の達成にとどまるのではなく，与えられたものを部分的に破壊してゆく一種の「拒絶」である．
- 発達は，学習する個人の転換にとどまるのではなく，その個人が所属する組織の集団的転換である．
- 発達は，レベルを垂直的に超えて行くことにとどまるのではなく，社会組織の境界を水平的に横切って行くことでもある．

30

モデルの歴史的タイプ
（エンゲストローム）

1. 自然発生的な典型
2. 分類的・唯名論的
3. 手続き的
4. システム的
5. 胚細胞型

さて，CMM はどのタイプか？
たぶん2と3のあいだ？

31

Resources on the web（追加）

伊藤昌夫さんのコラム：
http://www.cqpub.co.jp/dwm/column/ito/dwm0034itom13.htm
エンゲストローム先生の Video Interview:
http://csalt.lancs.ac.uk/alt/engestrom/
CSCW Magazine の Web Page:
http://www.kluweronline.com/issn/0925-9724
XP (Kent Beck) 対 Interaction Design (Alan Cooper) のディベート:
http://www.fawcette.com/interviews/beck_cooper/
日本活動理論学会の Web Page:
http://www.jarat.org/

32

自然言語処理技術を用いて、特許請求項の可読性を向上させる手法に関する研究について

新森昭宏
（インテック・ウェブ・アンド・ゲノム・インフォマティクス）

1. まえがき

1983 年に就職して以来、1990 年代末に至るまで、ソフトウエア開発環境、ヒューマンコンピュータインタラクション、グループウエア/電子メール、ソフトウエア障害対応に関する研究や開発の仕事に従事してきた。2000 年秋からは、東京工業大学の社会人博士課程に所属し、会社の仕事を行いながら、自然言語処理の研究、特に特許文書を対象としてその可読性向上に関する研究を行っている。

本稿は、2000 年秋以降にこの研究を行ってきた動機、現在までの到達点、これからの方向性についての考察を報告するものである。この研究は、SEA 所属会員の関心の中心であると思われるソフトウエア工学とは分野を異にするものであるが、私自身は、

1) 今後、自然言語処理は、ソフトウエア技術者が活躍すべき・活躍できる有望な分野の一つである
2) 好むと好まざるとに関わらず、ソフトウエア技術者は特許というものを意識せざるを得ない状況になっている
3) しかし、特許文書、特に特許請求項の部分は、極めて読みにくく敷居の高いものになっており、自然言語処理によってその敷居を下げることができれば、ソフトウエア技術者にとっても朗報となるだろう

と考えており、本稿として報告するものである。

2. なぜ、自然言語処理か

1993 年頃からパソコン向けのパッケージソフトの開発・販売支援・トラブル対応支援に従事し、Microsoft、Novell、Lotus（当時）などのプラットフォームベンダーが提供する「事例ベース」を活用しながら仕事を行っていた。事例ベースの中から状況に適合した事例をうまく発見することができれば仕事をうまく進められること、しかし自然言語で記述された事例が膨大に蓄積された事例ベースの検索は容易でないこと等を実感していた[1]。そこで、Bayesian Network の形式で不確実な知識を蓄積し、事例ベース検索に利用することを考えた[2]。しかし、自然言語で記述された膨大な事例だけがまず存在する中で、不確実なままでもよいとは言え、どうやって知識を形式化・蓄積するかの課題は残ったままであった。結局のところ、誰かが（人間またはコンピュータが）自然言語で記述された膨大な事例を読んで理解し、コンピュータ利用可能な形に変換しなければならないのである。

ちょうどその頃ぐらいから、「ナレッジマネジメント」ということが言われはじめ、組織内の知識をどのように収集・蓄積し、有効活用するかが議論されるようになった。私自身も、Java House メーリングリストなど、ソフトウエア開発者向けのメーリングリストに参加し、そこから有用な情報を得ていたこともあって、メーリングリストを対象としてそこから自動的または半自動的に情報抽出・知識獲得ができないかということを考えはじめた。

ちょうどその頃に出会ったのが、北陸先端科学技術大学院大学（当時）の奥村学先生による、テキスト自動要約に関する一連のサーベイ論文[3]と解説記事[4]であった。これらを読んで、自然言語処理の研究には長い歴史があること、そこには難しい問題が残されていること、しかし着実な進歩もあることなどを改めて認識した次第である。そして、ちょうど 2000 年から東京工業大学に移籍された奥村学先生のもとで研究指導を受けることになった。

3. なぜ、特許文書か

かつて特許は、機械・製薬・化学など特定分野の企業において、知的財産権担当者や研究者が主に関わるものであったと思う。しかし、ビジネスやサービスの方法を権利の対象とする「ビジネスモデル特許」の出現や、コンピュータプログラムを対象とした「ソフトウエア特許」の認知により、広い範囲の企業関係者が特許に関わらざるを得ない状況が生まれている。大学においても、研究成果や技術移転のコアとしての特許の重要性が認識されるようになっているようである。

こうした状況の中で、好むと好まざるとに関わらず、ソフトウエア技術者も特許というものを意識せざるを得ない状況になっていると思う。実際、情報サービス産業協会からもソフトウエア技術者を対象とした啓蒙的なブックレットが出版されている[5][6]。ソフトウエア特許に関しては、その弊害を指摘する声も多いが、多くのソフトウエア特許が既に成立しており、また日々多くのソフトウエア特許が出願されているという現実がある。

特許文書は一種の法的文書であり、以下のような構造を持っている。

- 明細書
  - 発明の名称
  - 特許請求の範囲
  - 発明の詳細な説明
  - 図面の簡単な説明
- 図面
- 要約

特許明細書中で最も重要な箇所である「特許請求の範囲」には、「特許請求項」(クレーム)が列挙される。特許請求項の記述は、１文で発明内容を記述するという制約と、記述の長さ、そして独特の記述スタイルにより、専門家以外の人にとっては極めて読みにくいものになっているのが通例である。図―１に実際の特許請求項の例を示すが、これを一読してその意味するところを理解できる人はいるだろうか？

> 各種情報を蓄積し掲示用情報として出力するホストコンピュータと、このホストコンピュータと大学構内に付設されたデータ回線網を介して接続し、前記掲示用情報を入力あるいは受信して表示し、前記掲示用情報に含まれる各種サービスの要求と任意の情報の入力および出力とを行う複数の端末とからなる大学構内掲示板サービスシステムにおいて、前記端末として各種掲示用情報を入力し利用者の要求を受付ける事務端末および図書端末と、前記利用者が使用し前記掲示用情報の取得と前記掲示用情報に含まれる各種サービスの要求と登録と予約とこれらの入力に対応する回答とを表示し出力する利用者端末とを備えることを特徴とする大学構内掲示板サービスシステム。

図―１　特許請求項の例（特開平 10-11007 の第一請求項より引用）（304 文字）

過去に出願された特許明細書の電子データは膨大に蓄積されている。さらに、年間 40 万件以上と言われる新規出願に伴い、データ量は日々増加している。企業関係者は本来、これらの膨大な特許明細書群を監視し、自分の業務に関連するものがないかどうかをチェックする必要があるが、量の膨大性という面と、記述の特殊性という面の両面において、これは現実には困難である。

自然言語処理技術によって、特許請求項の可読性を向上させることができれば、上記のような監視業務も少しは楽になるはずである。また、特許明細書を学会論文と同様の技術文書であると考えるならば、これを統合的に解析することで技術動向の把握ができるようになるはずである。

自然言語処理に関する研究を行うにあたり、特許文書を対象とすることにしたのは上記のような理由からである。

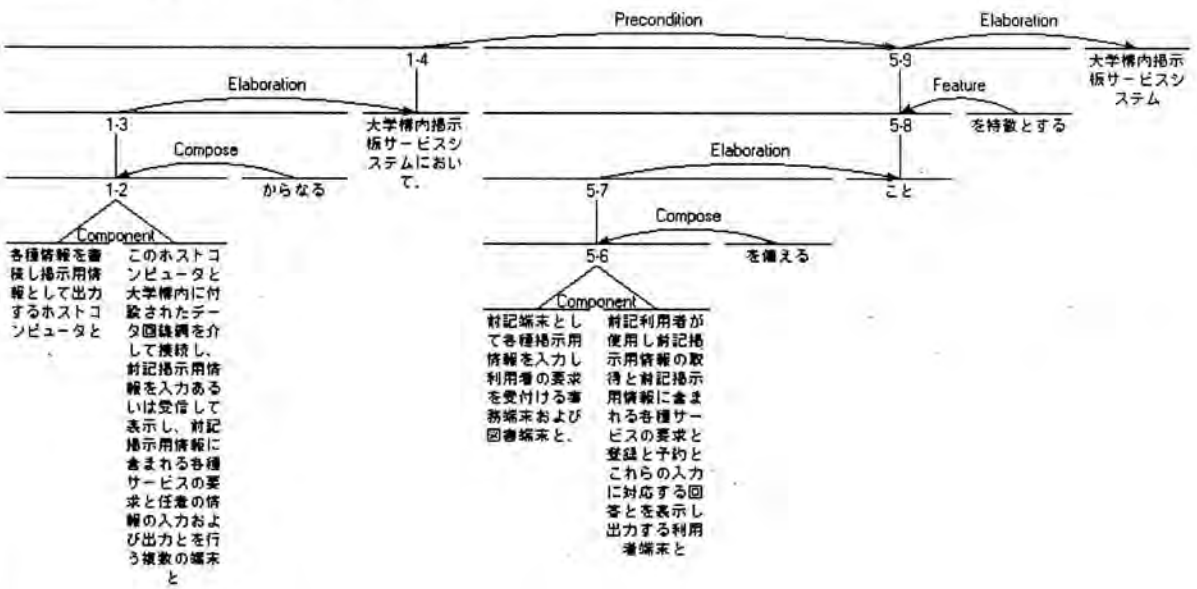
4. 現在までの到達点

弁理士によって書かれた文献[7][8]や、実際の特許明細書データ（国立情報学研究所が主催して開催された評価型ワークショップ NTCIR3[9]の参加者に配布された「特許データコレクション」）を調査分析し、特許請求項の記述スタイルを表—1のように類型化した。

表—1 特許請求項の記述スタイルの類型

名称	説明
順次列挙形式	「～し、～し、～した、～」のように、処理を順序的に記述する形式
構成要素列挙形式	「～と、～と、～とからなる、～」のように、構成要素を列挙する形で記述する形式
ジェプソン(Jepson)的形式	「～において、～を特徴とする～」、「～であって、～を特徴とする～」のように、最初に公知部分(既に知られている内容)または前提条件を述べたうえで、新規部分（この発明の特徴となる部分）または本論部分を記述する形式

そして、これらの類型にそれぞれ特徴的に出現する記述パターン（手がかり句）を収集し、それを用いて、特許請求項の構造を解析する手法を考案した。その際、特許請求項は複数の要素や処理の説明を1 文の中に詰め込んだ形で記述されていること[8]を考慮し、複数の文・節から構成された談話の構造を解析するための理論である修辞構造理論（RST: Rhetorical Structure Theory）を応用することを考えた。特許請求項の構造を表現するために、6 つの関係を定義し、字句解析と構文解析から構成されるコンパイラの手法をベースとした構造解析アルゴリズムを設計し、プログラムを実装した[10][11]。図—2に、図—1の特許請求項の構造を解析し、RST 解析のために開発された RSTTool というツールを利用して視覚表示した結果を示す。



図—2 特許請求項の構造解析結果（図—1の特許請求項の構造を解析し、視覚表示した結果）

また、こうした構造解析の結果を利用して、簡単な操作を行うことで、自動言い換え処理を行うプログラムも実装した。図―3に、図―1の特許請求項を自動言い換え処理した結果を示す。

以下からなっている大学構内掲示板サービスシステム:
　各種情報を蓄積し掲示用情報として出力するホストコンピュータ
　複数の端末
　　このホストコンピュータと大学構内に付設されたデータ回線網を介して接続する。
　　掲示用情報を入力あるいは受信して表示する。
　　掲示用情報に含まれる各種サービスの要求と任意の情報の入力および出力とを行う。

以下を備えていることを特徴とする大学構内掲示板サービスシステム:
　端末として各種掲示用情報を入力し利用者の要求を受付ける事務端末および図書端末
　利用者が使用し掲示用情報の取得と掲示用情報に含まれる各種サービスの要求と登録と予約とこれらの入力に対応する回答とを表示し出力する利用者端末

図―3 特許請求項の自動言い換えの例
(図―1の特許請求項に対して自動言い換えを行った結果)

5. これからの方向性

特許請求項の低可読性の問題については、前章で述べたような構造解析とその結果の視覚的表示により、かなり緩和されたと考えている。しかし、特許明細書が読みにくいという問題を解決するには、更にいくつかの山を越える必要がある。

その一つには、用語の問題がある。特許請求項中に使われる用語は、発明の範囲をできるだけ拡張するために、できるだけ上位概念の用語が使われる傾向がある（例；「プリンタ」ではなく、「印字装置」等)。また、当該分野に固有の専門用語が何の説明も無く使用されることも多い。これらの問題に対しては、「発明の詳細な説明」中の記述を利用してその注釈付けを行うことを考えている[12]。

また、1つの特許明細書中に複数の特許請求項が記述される「多項制」という形態がとられる中で、特許請求項間の関係の把握が困難なものも多い。また、1つの特許明細書中の特許請求項数が異常に多いものもある。たとえば、私が遭遇した特許明細書の中には、185 個の特許請求項を含むものがあった！ コンピュータによる支援無しに、いったい誰が 185 個の特許請求項を解読して理解できるというのであろうか？

もちろん、私が自然言語処理の研究に着手することになったもともとの動機、すなわち、ソフトウエアに関する様々な文書やメールデータを自動解析するという仕事にもいずれは着手したいと考えている。今後、自然言語処理は、ソフトウエア技術者が活躍すべき・活躍できる有望な分野の一つであるというのが私の現在の考えである。

謝辞

本研究では、NTCIR3 ワークショップで配布された「特許データコレクション」を使用しました。

参考文献

[1] 新森昭宏, 柴垣ゆかり, 「パソコン環境におけるソフトウエア障害対応プロセスの分析とそれに基づく業務改善アプローチについて」, ソフトウエアシンポジウム'97, ソフトウエア技術者協会, pp.123-130, 1997.

[2] 新森昭宏, 「Bayesian Network によるソフトウエア障害対応知識のモデル化と事例ベース検索への応用」, ソフトウエアシンポジウム'98, ソフトウエア技術者協会, pp.127-134, 1998.

[3] 奥村学、難波英嗣、「テキスト自動要約に関する研究動向」、自然言語処理、Vol. 6, No. 6, pp.1-26, 1999.

[4] 奥村学、望月源、「連載：テキストを自動的に要約する技術、第一回 テキスト中の重要な文を抜き出す」、bit、Vol. 32, No. 2、2000.

[5] JISA ブックレッツ-1、「ソフトウエア特許入門」、(社)情報サービス産業協会編、1998.

[6] JISA ブックレッツ-5、「SE のための特許入門」、(社)情報サービス産業協会編、2003.

[7] 葛西泰二、「特許明細書のクレーム作成マニュアル」、工業調査会、1999.

[8] 糟谷洋治、「『請求の範囲』の文体と作文技法の考察」、パテント、 1999.

[9] 岩山真、藤井敦、高野明彦、神門典子、「特許コーパスを用いた検索タスクの提案」、情報処理学会研究報告, FI-63-007, 2001.

[10] 新森昭宏, 奥村学, 丸川雄三, 岩山真, 「手がかり句を用いた特許請求項の修辞構造解析」, 情報処理学会第 149 回自然言語処理研究会, 2002-NL-149, 千葉大学, pp.65-72, 2002 年 5 月.

[11] Akihiro SHINMORI, Manabu OKUMURA, Yuzo MARUKAWA, Makoto IWAYAMA, "Rhetorical Structure Analysis of Japanese Patent Claims using Cue Phrases", Proceedings of the Third NTCIR Workshop on research in information Retrieval, Automatic Text Summarization and Question Answering, National Institute of Informatics, 2003.

[12] Akihiro SHINMORI, Manabu OKUMURA, Yuzo MARUKAWA, Makoto IWAYAMA, "Patent Claim Processing for Readability - Structure Analysis and Term Explanation -", ACL-2003 Workshop on Patent Corpus Processing, Association for Computational Linguistics, 2003.

華東師範大学 軟件学院 訪問

荒木 啓二郎
(九州大学)

1. 1. 激変の街，上海

昨年(2003年)12月に1年振りで中国の上海を訪問しました．SEA が毎年中国で開催しております ISFST (International Symposium on Future Software Technologies) というソフトウェアに関する国際会議に参加するために，ここ十年以上の間，ほぼ毎年中国に行っておりましたが，昨年は，新型肺炎 SARS のためにこのISFSTの開催延期が決定されましたので，この1年間は中国に行っておりませんでした．

私は福岡に住んでおりますので，ISFST等で中国に行く時は，上海経由にします．福岡／上海間は，飛行機で1時間15分程度ですので，東京に出張するのと変わらないか，却って早いくらいです．これまで，上海に行く度に，半年振りであっても，その変化の速さに驚いておりましたが，今回は1年振りでしたので，その変わりように改めて驚きました．

上海に着いたとたんに，まずは，自動車の数の多さと車種の豊富さに驚きました．十二年前のことでしたでしょうか，その時上海では，小型の中国製の(たいがい赤く塗装されて排気ガスをまきちらして走っていた)車や日本製のボロボロの中古車に交じってフォルクスワーゲンのサンタナが走っていました．その直前にやっとの思いでフォルクスワーゲンのジェッタを3年ローンで買った私としては，自分の車より高級な車種であるサンタナが中国でタクシーとして使われているのを見て驚いたりがっかりしたのですが，上海にフォルクスワーゲンの工場ができたと聞いて，(当時はまだ「へ～，へ～，へ～，...」と右手は動かなかったものの:-) なるほどと思いました．今回は，赤い小型の中国車は全く見ませんでした．フォルクスワーゲンもサンタナだけではなくて，ポロやパッソートさらにはサンタナのステーションワゴンも走っているのでした．ゼネラルモータースも上海に工場を造ったそうで，ビュイックも沢山走っていました．ホンダ，日産などなど日本の車も走っていました．おかげで環状の自動車専用道路も，従来の一般道も大渋滞です．

上海のことは，また最後に少し触れるとして，表題に挙げています華東師範大学軟件学院(ソフトウェア学部)訪問の話をしましょう．

2. 華東師範大学軟件学院

He JiFeng という人が，マカオにあります国連大学国際ソフトウェア技術研究所 (UNU/IIST) で主任研究員をしています．彼は，英国オックスフォード大の C.A.R. Hoare のもとで長年研究をしてきており，形式手法(formal methods) の分野では知る人ぞ知る研究者です．一昨年，彼が福岡に来るから会いたいと言うので，会って，中洲に席を移してから，いろいろと話を聞くと，上海の華東師範大学 (East China Normal University) にソフトウェアだけの学部を作った，自分が学部長をやっている，日本と連携したい，と言うのです．

その時は，それで終って，特に具体的な話も出なかったのですが，昨年の9月に，私がプログラム委員長を務めた Formal Methods 2003 という形式手法の国際会議がイタリアのピサで開催されて，それに参加しましたら，JiFeng も参加していて，件の軟件学院の話になりました．

学部の学生が各学年200名程度在籍していて，修士課程にも各学年100名以上いるというのです．そんなに沢山卒業生を出して，仕事はあるのか，と訊くと，即座に，あると，自信たっぷりに答えるのです．どこにあるんだ，と訊くと，日本だというのです．

以下に，JiFeng の話や，実際に上海で見聞したことを基に，上海のソフトウェア学部やソフトウェア会社の状況の一端をお話しいたします．

現在，中国で35校の大学に軟件学院というかソフトウェア学部が新設されたそうです．上海では，上海交通大学，復旦大学，同済大学と華東師範大学の4校だそうです．師範大学としては，この華東

師範大学が中国で1校だけ指定されたと，誇らしげに言っていました．

いずれも華東師範大学と同じ規模の学部だとすると，毎年1万人ものソフトウェア技術者が社会に輩出されることになります．日本でも情報関連の学部学科や大学院から，毎年それくらいの卒業生は出ているかと思いますが，日本の大学の場合は，情報何々という名称でも学科や学部の設立の経緯から，必ずしも情報専門となっていないことが多いので，ソフトウェアだけで各学年200名の専門学部というのは，日本から観ると脅威的な人数といえます．ちなみに，九州大学工学部電気情報工学科では，各学年160名の定員のうち，ソフトウェアに関係する研究室に所属するのは20名くらいでしょうか．

そもそも，このように中国にソフトウェアの学部が一時に沢山できたのは，中国の御偉方がインドを視察して，これはいける，中国でもやれという鶴の一声があったからだというような話をききました．

華東師範大学軟件学院について，もう少し詳しく述べましょう．ホームページによると，設立は2002年1月だそうです．学部長は，前述のHe JiFengです．彼は，2002年度の国家自然科学賞を受賞したとかで，応接室には，大勢の受賞者たちに交じってJiFengが江澤民のわりと近くで写った写真と賞状とが飾ってありました．

学生数は，学部と修士課程併せて現在1000名程度ですが，じきに1500名に増やすそうです．学部が各学年150から220名で，4学年合計で約700名です．2002年1月設立のわりには，2000年度の学生が155名いるという説明が気になりましたが，まあ中国のことですので，そんなことは気にしてはいけないのでしょう．この軟件学院に直接入学した学生以外にも，他学部からの転入学生がけっこう沢山いるそうです．大学院の修士課程には，2002年度に72名，2003年度春に127名，2003年度秋に95名入ったそうですが，これも他学部からの転入を含んでいるそうです．ちなみに，学部卒業生の大学院進学率は5%程度だそうです．修士課程には，社会人学生も沢山いるそうです．

上述の学部長の話によると，仕事は日本から取って来るという訳ですから，日本語教育にも力を入れていて，現在は，日本語検定を目標に，日本語は必須科目となっています．しかし，今はまだ，一般的な日本語教育だけなので，次の段階では，専門科目を日本語で教えたいということです．

このようなことを私の目をじっと見つめて言うものですから，気の弱い私は，う～ん，わかった，私にできることは協力しましょうなどと言ってしまった訳です:-)

もっとも，前述のように大量のソフトウェア技術者が輩出されて日本のソフトウェア開発に乗り込んで来るのですから，その大きな流れは防ぐべくもないと感じて，それなら日本の言葉は勿論のこと，歴史や文化や社会的な背景も理解したうえで，日本向けのソフトウェアを開発して貰うのが顧客となる我々日本人にとってせめてもの仕合わせになるのではないか．そのための中国人ソフトウェア技術者教育に少しでも協力しようなどとも考えた次第です．もっとも，それは，私には荷が重すぎることですが，SEAで岸田さんをはじめ皆様方から教えて頂いたことを基にできることをやろうと考えています．

さて，華東師範大学を訪問して，対応してくれた若い教員と具体的な打ち合せをしましたが，実際に私に求めることは何なのかが解るまでに随分時間がかかりました．日本では情報処理学会が1997年にJ-97という理工系の情報専門学科に対するカリキュラムの案を提示しています．一応，それを彼らに提示し，日本の大学の一般的な学期や授業のやりかたを説明したところ，一応，ソフトウェア工学関連の科目を私が担当しようということになりました．ちなみに，このJ-97のソフトウェア工学の項目は荒木が担当しています．

日本の大学では，1コマ90分の授業を週に1回，期末試験を入れて15週というのが，よくある授業形態です．ところが，華東師範大学では，欧米の大学でよく見られるように1コマ45-60分の講義を週に5コマ程度実施するそうです．今回は，初めての試行ということもあって，華東師範大学の通常の学期の中で1週間特別の時間割にして貰って，日本の細切れ授業の1科目相当分を集中講義で実施することにいたしました．

ところで，問題は，何をどのように教えるかです．これに関して私に対する要求が，今ひとつはっきりしないのです．専門の内容を日本語で教えるということに重点があるのならば，ソフトウェア工学概論のような内容で，日本語で一通りの内容を教えても良いのですが，この案を提示しても先方は乗ってきません．この提案自体の意味あいも理解して貰っていないようでした．何を教えてくれるのかと繰り返し質問します．日本語での専門教育をしなくてはいけないという先入観をもつ私は，また同じような提案をします．最初の2時間程度，概論めいたことを話して，それから特定の科目の具体的な内容を詳しく教えましょうか，などといっても，やはり腑におちない様子です．

しばらく膠着状態が続いたところで，華東師範大学軟件学院の授業内容を見せて貰うことにしました．科目一覧のコピーを見ると，ソフトウェア関連の科目がびっしり並んでいます．但し，科目のいくつかに印が付いています．これらは，まだ開講していない科目なのだそうです．後で数えてみると，約50の専門科目のうち，ほぼ半数が未開講です．

だんだん分かってきました．前述のように，中国で急にソフトウェア学部を沢山つくったため，どうやら教員不足になっているようです．日本語での講義云々の以前に，専門科目の講義が求められている訳です．私が日本語で授業をしようと英語で授業しようと，どうでも良いことで，未開講の科目を補ってくれれば彼らにとって有難い訳です． JiFeng の話を最初に聞いた時に感じた高邁な思いは，ガタガタと崩れて，なんだか，お人好しの日本人が葱を背負って自ら飛び込んで行ったという感じもしてきました．しかし，まあ，何でも良いけど，何か面白そうでもあるし，なかなか体験できないことでしょうから，最後まで付き合ってみることにします．

という次第で，形式手法の大御所である JiFeng が学部長をしているにも拘らず，形式手法の講義を荒木が自分の教科書を使って教えることにしました． 3月下旬に上海を1週間訪問して，集中講義をしてきます． JiFeng 自身は月に2回程度来るけど，授業は担当していないそうです．

華東師範大学軟件学院では，専門科目の8割9割は英語の教科書を使って中国語で授業をしているそうです．上海の他の大学では，当然のことながら英語の教育に力を入れているけれども，日本語教育を重視しているのは華東師範大学だけだと言っていました．九大の他の先生を上海に招いて講義をして貰う他に，九大へ学生を数ヵ月派遣して，授業への参加や研修なども実施したいと言っていました．ソフトウェア技術者の専門教育を日本の大学にアウトソーシングしたいということでしょうか．

ところで，中国のこれらのソフトウェア学部は，産学合作で設立運営するという制約が課せられているそうです．華東師範大学の場合は，復星 (FuXin) グループというのがパートナーとなって，上海師策教育投資管理有限公司 (Shanghai SEI Education Holding Ltd.) を作っています．軟件学院のロゴとこの会社のロゴとは，同じ形で配色が違うだけですので，混乱します．しかも，同じ形で別の色のロゴを持つ Shanghai Embedded Systems Institute (上海組み込みシステム研究所) という組織もあります．同じ人の名刺の裏表にこれらの組織が記載してあったりして，いよいよ混乱します．産学合作というよりは，産学合体といった印象を受けました．

3. ソフトウェア会社訪問

FuXin グループのソフトウェア会社も見学させて貰いました． 100-200名程度の従業員が働いていて，上海では有数のソフトウェア会社だということです．

日本の大手企業からの発注を請けていて，職場に入ってまず目についたホワイトボードに貼られているビラや書かれた進捗表が日本語だったりして，一瞬ここは日本かと思うくらいです．プログラマが使っているパソコンには日本語版 Windows が載っていて，部屋の隅に置かれたプリンタの廻りには日本語文書の反古紙が散乱しています．まるっきり日本のソフトウェア会社の光景です．日本語の文書では漢字を使っているので，読み書きは問題ない，でも，会話はできない，と言っていました．中国人のプログラマ達は，お昼休みに日本語の勉強をしているとのことです．

始まって間もないプロジェクトを担当しているマネジャーの話を聞くことができました．日本の企業から詳細設計書を貰って，それを見てプログラムを作っているそうです．形式仕様記述なんてなことをやっている関係上，詳細設計書見ても良く分らないでしょうと質問したら，意外にも，いえ，以前は分らなかったけれども，最近ようやく分るようになりましたとの返事でした．このマネジャー

は，日本留学経験があり，日本語に堪能で，彼が担当しているプロジェクトの上海側の窓口になっていて，日本側とのメイルのやりとりを常にしているようでした．

最初に日本から担当者が上海にやって来て，詳細設計書を渡して，1-2週間かけて説明をするそうです．以後は，この設計書を見てプログラム開発を進めて行くのだそうです．もちろん，詳細設計書だけで全てが分る訳ではないので，インターネットを活用して，電子メイルやオンラインチャットや場合によってはNetMeetingを使って，日本との間での技術的な確認を行ない，また，意思の疎通を図っているのだそうです．それらを前述のマネジャーがメイルできちんと記録に残すのだそうです．コミュニケーションが一番大事ですと，このマネジャーは言っていました．ちなみに，プログラミングの対象は，300-500行程度のモジュール単位になっているそうです．

上海のソフトウェア産業にとっては，日本からの下請けというのは，今の所，美味しい仕事なのだそうです．安定して仕事が来るし，設計後のコーディングが主体だから，自分達で新規にシステム開発を請け負うのに較べるとリスクが少ないからだそうです．

これらの話を聞いて，私は，残念な思いをし，また，不安な気持になりました．今は詳細設計書が分るようになりましたと言ったマネジャーの言葉から，日本企業の安易なアウトソーシングのやり方を想像したからです．外国の下請け企業にまで，行間を読ませ，特定の企業文化を押しつけて，せっかくの異文化交流の機会に文書や開発プロセスを見直す好機を逸したと思うからです．

今は，中国にコーディング中心の下請けをやらせていますが，その内に中国のソフトウェア技術者が力をつけて上流工程まで彼らがやって日本企業の出る幕がなくなったり，下請け依存体質の日本企業が中国の下請け企業からしっぺ返しをくらって，彼らのいいなりにならざるを得なかったり，というような状況が早晩おこりはしないでしょうか．

4. 若者の街，上海

夜になって，大学というか，会社というか，産学合作の活動を支えている若い人に食事に誘って貰いました．秘書をやっている女性と，ソフトウェアの教育に携わっている青年です．この青年は，医学部を卒業したけど，医者よりはソフトウェア技術者になる道を選んだそうです．

上海の浦東地区にある洒落た中華レストランに連れて行って貰いました．明るくモダンな内装で，とってもトレンディな雰囲気です．小綺麗な身なりの20歳台30歳台の人達で賑わっていました．西洋人も沢山来ていました．これまで岸田さんに連れて行って貰ったところとは大違いです:-)

食事の後で，川岸のStarbacks Coffeeに行こうということになりました．Starbacks Coffeeといっても侮ること勿れ．以前上海といえば必ず出て来た外灘のちょうど対岸が，若者たちの人気スポットなのだそうです．日本で言うと，東京の御台場のような感じでしょうか(博多もんには良くわかりませんが:-) 若者のグループやカップルで賑わっていました．私の相手をしてくれた若い人達も，休みの日にはよくこの辺に遊びに来るそうです． Starbacks Coffeeは，ガラス張りの洒落た建物で，外灘の眺めが良く，浦東の中でも一番人気で，当日も満員で入ることができませんでした．しかたなく，隣の類似のカフェに入りましたが，ここも間もなく客で一杯になってきました．アメリカのオールディーズの曲が流れて，皆，カクテルなんかを気取って飲んでいます．アメリカの古い曲といえば，林香さんや酒匂さんの得意とするところで私には食傷気味ですが(俺にもマイクを渡してくれ:-) 上海の若者には新鮮で好まれているそうです．

某ANAの機内誌に北京は東京のライバルとかいったような言葉が載っていました．日本企業が中国の巨大な市場への参入を目指すといったような新聞記事なども時々目にします．どうも，認識が甘いように思います．私を浦東に案内してくれた上海の二人の若者も，日本は「大鍋飯」だとか言って，日本社会の非効率さや理不尽さに言及していました．中国は，日本なんか既に相手にしていないような気がします．

いずれにしましても，3月下旬に1週間上海に行って，上述の集中講義をする予定ですので，上海で何が起きているのかとか，大学生のレベルや気質を見て参ります．面白い話があれば，また報告いたします．

October 2, 2003

キューバ危機とソフトウェア危機

佐原伸
日本フィッツ（株）

日本では「事実に基づかない常識」が横行している。キューバ危機とソフトウェア危機を例に、その実体を明らかにし、結論として形式手法を勧めつつ、朝鮮半島危機の解決策はすでに提示されていることを指摘するのが本稿の狙いである。

なぜキューバ危機とソフトウェア危機なのかといえば、キューバ危機は人類の歴史上もっとも解明された歴史的事象であり、ソフトウェア危機は我々ソフトウェア技術者がその渦中で苦闘している現在進行形の危機であり、ともに「解明された事実」があるにもかかわらず、その事実を無視した「常識」に従って危機を拡大しているからである。

キューバ危機

日本ではキューバ危機の真相はほとんど知られていないが、1964 年 10 月 27 日我々は核戦争 10 分前を体験した。ソ連フルシチョフ書記長のミサイル撤去受諾があと 10 分遅れていれば、米軍がキューバ上陸を開始し、米軍の予想していなかったキューバ駐留陸海空ソ連正規軍 4 万 2 千が、現地司令官の裁量で使用できる戦術核兵器をもって反撃したはすなのである。もちろん、その後は米軍上陸部隊の全滅、米軍の核による反撃でキューバが全滅、米ソの核戦争によって日本を含む主要国が壊滅的な打撃を受けることとなっていた。

問題は、米国・ソ連・キューバのいずれもが核戦争を望まず、危機の拡大も望んでいなかったにも関わらず、上記の事態が発生したことだ。3 カ国がお互いの意図を誤解し、危機を拡大してしまったのである。

この間の経緯は、米国ブラウン大学の主導で、キューバ危機 30 周年にあたる 1992 年に、当時の米国・ソ連・キューバの政府・軍の主脳（マクナマラ元米国防長官やシュレジンジャー元補佐官、カストロ・キューバ書記長、グリコフ元ワルシャワ条約機構軍参謀長など）や学者達（旧ソ連のフルシチョフ書記長とミコヤン第一副首相の息子達が、米国の大学の研究者として参加している

のも興味深い）がハバナに集まってキューバ危機を再検証する中で明らかになった[1]。キューバ危機から得られる教訓は、

- 危機は制御できない
- 核の傘は、核戦争を招く
- 軍事力は、戦争を招く

である。考えてみれば「核の傘」以外の２つの教訓は、すでに、旧日本軍による中国侵略でも証明されている。

しかし、最近の日本では「核の傘」と「軍事力」で「日本を守る」のが常識かのように「宣伝」されている。事実に基づかない考え方であり「宣伝」である。

ソフトウェア危機

ソフトウェア危機もキューバ危機と同じように、ユーザーもソフトウェア会社もソフトウェア危機を拡大させる意図はないにも関わらず、筆者がソフトウェア開発会社に就職した1975年以来、ソフトウェア危機は常に『今そこにある危機」であった。対処法は、常に

- テストを十分にせよ
- 開発プロセスを遵守せよ

であったが、危機は深まるばかりである。原因は、筆者から見ればはっきりしている。

- 30年前のソフトウェア工学に基づく開発プロセスに従って、
- 欠陥の多いソフトウェアをいくら十分にテストしても、

信頼性の高いソフトウェアは作成できないのである。

信頼性の高いソフトウェアを作るための形式手法については、以下のような「常識」が存在する[2]。

- 形式手法は役立たない
- 形式手法は難しい
- 形式手法は金がかかる

このあと本稿は上記「常識」に反論していく訳である。

1. James. G Blight, Bruce J. Allyn, David A. Welch: **Cuba on the Brink**, 2002, Rowman & Littlefield Publishing Inc.
2. 本当は、J.A. Hall: **Seven Myths of Formal Methods**, IEEE Software, Vol.7, No.5, pp.11-19, 1990. に示されているように、もっと存在するが、ここでは主要なものにとどめた。

ソフトウェア危機

形式手法は役立つ

一緒に仕事をしている酒匂寛さんの新刊書（「課題、仕様、設計」、インプレス、2003 年 10 月か 11 月出版予定）の草稿を校正をしていて、VDM++ 仕様の部分を VMDTools の構文チェックと型チェックツールを使ってチェックした。

FIGURE 1. 草稿の「診察受付」仕様

```
class 「診察受付」is subclass of 「業務論理」
operations
public 適用（ある患者：「患者」，ある予約：「予約」）
 ある診察記録：「診察記録」== is not specified yet
pre
 (isofclass(「再診予約」，ある予約）=> ある予約 in set ある患者．予約 ) and
 not（ある予約．状態 = 「予約」`<受付済>)
post
 ある予約．状態 = 「予約」`<受付済> and
 not（ある診察記録 in set ある患者~．予約）and
 ある診察記録．患者 = ある患者 and
 ある診察記録．診療科 = ある予約．診療科 and
 ある診察記録．診察状態 = 「診察記録」`<受付済> and
 ある診察記録．初診再診区分 () = 患者区分
end 「診察受付」
```

上記の仕様には、２ケ所の間違いと、数カ所の用語の統一上の問題と、数カ所の文法違反があった。

間違いは以下の通りである。

- not（ある診察記録 in set ある患者 ~. 予約）
  で、「ある診察記録」ではなく、「ある予約」でなければならない
- 患者区分が未定義

用語統一上の問題は以下の通りである。

- 操作を通して値を得るはすが、属性を直接参照している
- 予約状態は「状態」として参照しているのに、診察状態は「診察状態」として参照している

文法違反も修正した仕様は、以下のようになる。

FIGURE 2. 診察受付の「修正した」仕様

```
class 「診察受付」is subclass of 業務論理
```

```
operations
public 適用(ある患者 : 「患者」, ある予約 : 「予約」)
  ある診察記録 : 「診察記録」== is not yet specified
pre
 (isofclass(「再診予約」, ある予約) => ある予約 in set ある患者.予約()) and
 not (ある予約.状態() = <受付済>)
post
 ある予約.状態() = <受付済> and
 not (ある予約 in set ある患者.予約 pre()) and
 ある診察記録.患者() = ある患者 and
 ある診察記録.診療科() = ある予約.診療科() and
 ある診察記録.状態() = <受付済> and
 let患者区分 = if isofclass(「新患予約」, ある予約) then <新患> else <再診>
 in
 ある診察記録.初診再診区分() = 患者区分
end 「診察受付」
--------------------------------
class 「患者」
instance variables
public 予約集合 : set of 「予約」:= {};
operations
public 予約 pre : () ==> set of 「予約」
予約 pre() == is not yet specified
post
 RESULT = 予約集合~;
end 「患者」
```

筆者の場合、上記のような間違いは酒匂さんのおよそ 10 倍の確率で発生させる可能性が強い。にもかかわらず、実際のプロジェクトの中で、高い信頼性のある仕様を書けた[1] のは、VDM++ の構文チェック・型チェック・インタープリタ・コードカバレージ機能・証明課題生成機能といったツール群を使用したからである。さもなければ、仕様段階で数千のエラーを包含したモデルを作ってしまっただろう。

実際に、私の経験した形式手法を使わない多くの開発現場のシステムでは、上流工程で埋め込まれた数千のエラーを、テスト工程で必死に除去しようとして、多大な努力を払い、それでもリリース後にトラブルを発生させることが多かった。

1.「共通機能の括り出しが不十分でしたよ」と同僚の佐藤圭さんが言っているが

以上から明らかなように、形式手法ツールは、開発の早い段階でエラーを見つけるのに役立つ。結果として、下流工程に引き継ぐエラーが少なくなり、作業の手戻りが少なくなり、「泥沼化したプロジェクト」になる可能性が低くなる。

ページ数の関係で、形式手法ツールだけでなく形式手法も役立つことの証明は省略するが、上記仕様中の post 以下で示される事後条件を記述するだけでも、開発現場の品質向上に大いに寄与することを指摘しておきたい。

また、形式手法は各種の大規模システムで適用され、役に立つことが報告されている。NASA の全宇宙システム[1] は SPIN[2] でモデルを記述しチェックされている。ロシアの人工衛星制御システムとオランダの花市場システムは VDM[3] で記述され、後者は VDM から C++ を生成し 1 人年で完成した。パリの地下鉄制御システムは B[4] で記述され、いずれも、エラーが 0 または極めて少ないことが報告されている。

もちろん、形式手法だけが役立つと言っている訳ではない。実際に、ソフトウェア・シンポジウム 2003 で報告したプロジェクトの例[5] では、形式手法以外に、ソフトウェア工学のうち「開発現場で容易に使える」あらゆる技術を動員した。コードカバレージによるテストや COCOMO81 による見積もりなど、20 年前にも使っていた技術もある。回帰テスト用のライブラリーは、オブジェクト指向技術の流用であり、形式仕様の記述スタイルは、今、流行りらしい XP に近いものがあった。cvs による構成管理と版管理がなければ、形式手法の威力も半減しただろう。

形式手法は難しくない

次の攻撃目標は「形式手法は難しい」という「常識」である。

形式手法は確かに難しい。しかし、形式手法を使わないソフトウェア開発は「難しくない」のだろうか？筆者の 28 年間にわたる「開発現場」経験では、「ソフトウェア開発は常に難しい」というのが実感である。そして、「手を抜けば抜くほど難しさが増す」というのが経験則である。

従って、「形式手法が難しい」という命題が成り立つためには、「現在の開発手法は、形式手法にくらべれば難しくない」ことを立証しなければならない。寡聞にして、そのような報告を聞いたことがない。

具体的に、システムが何をすべきかの仕様を書く場面に絞って考えてみよう。ここで、日本語で仕様を書く場合と、VDM や CafeOBJ[6] といった形式仕

1. FM2003 ディナーでの Gerard J. Holzmann 博士との会話
2. http://spinroot.com/spin/whatispin.html
3. http://shinsahara.com/www/vdm/index.html
4. http://www.afm.lsbu.ac.uk/b/
5. 佐原伸：大規模事務処理システムにおける形式手法の適用経験、ソフトウェア・シンポジウム 2003 事例報告
6. http://www.ldl.jaist.ac.jp/cafeobj/

様記述言語で仕様を書く場合とを比べてみよう。以下は、最初の VDM++ 仕様の一部を日本語で書いたものである。

FIGURE 3. **日本語による仕様例**

もし、予約が新患予約なら患者区分を「新患」とし、再診なら「再診」とする。

人によっては、構造化日本語を考え、以下のように書くかも知れない。

FIGURE 4. **構造化日本語による仕様例**

```
if 予約 = 新患予約 then
   患者区分 = " 新患 "
else
   患者区分 = " 再診 "
```

問題は、上記の２通りの書き方は、仕様を書く人が「その場の雰囲気」で決めることが多いことである。筆者の経験では、「この場面ではどう表記しようか」と考えている時間がかなりある。仕様の内容以外に時間を使うことは、考えてみれば無駄なことである。しかし、「日本語という曖昧な仕様記述言語」を使う限り、この時間の浪費は避けられない。しかも、書いた仕様に曖昧さが残り、構文チェックや型チェックもできない。

結局、文法が決まっていて、構文チェックや型チェックなどもできる形式仕様記述言語で仕様を書く方が簡単なのだ。VDM++ の場合、言語仕様の複雑さはせいぜい Java 程度である。UML2.0 を勉強するより易しい。

形式手法も、証明などで一部に難しい部分も残っているが、開発現場の泥沼状態のプロジェクトで苦労することを思えば、大したことではない。いや、本当に難しければ、その部分だけ使わなければ良いだけである。

形式手法は金がかからない

次は、「形式手法は金がかかる」という「常識」を攻撃する。

確かに、VDMTools の場合１本 100 万円近いコストがかかる。しかし、筆者の周りで常に見聞きする「開発現場のトラブル」は数千万円から数億円のコストを浪費している。顧客の信用の低下を考えれば、実際のコストはもっと大きいだろう。

形式手法を学習するコストもかかる。多分、ツールを買う値段以上にコストがかかる。しかし、大規模システムの泥沼状態の中で、１件のエラーを修正するコストは高くないのだろうか？筆者の経験では、少なくとも１件 10 万円以上のコストはかかり、平均で言えば１件 100 万円は下らず、中には億円単位のエラーもある。最近の携帯電話システムなどでは、１件が数百億円に値するようなエラーもあるようである。

つまり、形式手法などの新しい技術に仮に一人当り300万円かかったとしても、その技術を使うことで、高々数個のエラー発生を防ぐことができれば、十分元が取れるのである。ソフトウェア開発での最大のコスト削減策は、高品質ソフトウェアを作ることである。決して、コピーに裏紙を使うことなどではない。

ソフトウェア危機がキューバ危機から学ぶこと

ソフトウェア危機が形式手法によって幾分か緩和されるとしても、キューバ危機の教訓と比べて足りないことがある。キューバ危機では「何が起きたか」「誰が何をしたか」が場合によっては分単位で明らかにされている[1]。

ソフトウェア危機は、多くの「現場」が経験しているにも関わらず、その詳細は当事者達の「記憶」ないしは「酒飲み話し」でしかない。もちろん、世界が絶滅するか否かの瀬戸際の事件と、高々数百億円の損害の発生でしかないプロジェクトで、同じ密度で記録が存在する必要はないが、やはり「過去の経験に学ぶ」基礎となるデータは保存されるべきではないだろうか？大きな社会的トラブルを起こしたプロジェクトの実態や履歴は公表されるべきであろう。

さもなければ、数百年後のSEA MAIL編集長が、現SEA MAIL編集長のFM2003[2]におけるスピーチ「Back to the future」に類する話ができないことになる。

キューバ危機から日本人が学ぶべきこと

最近の日本人の多くは、朝鮮半島危機について「北朝鮮なんかやっつければよい」という気持ちがあるらしい。しかし、米軍の予測でさえ、戦争が起これば、死者は米軍5万人、韓国人100万人、北朝鮮の人々にいたっては何人が犠牲になるか分からない。このような悲劇を肯定するのは「狂気」としか言いようがない。戦争は、ソフトウェアのエラーとは比較にならないくらい大きなコストがかかる。「コスト削減」命の日本人が、勇ましいことを言うのは自己矛盾としか思えない。

キューバ危機の教訓からも「危機を起こしてはいけない」のである。方針ははっきりしている。韓国の言う「太陽政策」しかないのである。58年前に米国に対し一億玉砕を叫んでいた日本は、韓国に従うしかないではないか。

1. http://www.gwu.edu/~nsarchiv/nsa/cuba_mis_cri/chron.htm
2. http://fme03.isti.cnr.it/

編集後記

☆

あいかわらず原稿が集まりません.

☆☆

そうした状況のなか，佐原さん，新森さん，そして荒木先生の３人からお寄せいただいた原稿を材料に編集にとりかかったのですが，いささかページ数が足りません.

☆☆☆

そこで先日 (3/11) に開催された SEA-SPIN Meeting のスピーカだった伊藤昌夫さんにお願いして，当日の参考資料として配付された力作論文を転載することのお許しを得ました．ついでに埋め草として，わたし(岸田)が発表に使ったスライドのコピーも載せることにしました.

☆☆☆☆

次号は，２月に SOBA Framework をテーマに行われた SEA Forum の記録を渡邉雄一さんがまとめてくださっていますので，それを中心にまとめるつもりです.

また，４月初めに大阪で開催予定の SEA Forum on PLSE のために講師の K.C.Kang 先生が用意してくださったスライドが，かなり情報量の多い力作なので，SEAMAIL に掲載することの了承をお願いしてみようと考えています.

☆☆☆☆☆

それにしても，原稿が欲しいです．SEAMAIL は，ふつうの雑誌ではなく，会員相互の交流と情報交換のためにあるのですから，みなさんぜひ何か書いてください。

☆☆☆☆☆☆

岸田孝一 @ SEA Office

ソフトウェア技術者協会
〒160 東京都新宿区四谷3-12 丸正ビル5F
TEL.03-3356-1077　FAX.03-3356-1072