SEAMAIL

Newsletter from Software Engineers Association

Vol. 14, Number 9-10 May, 2006

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，350 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 17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．９－１０ ２００６年５月３１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 市田印刷株式会社 〒１１４-００１４ 東京都北区田端２－３－２５
定価 １，０００円

ソフトウェアエンジニアリングの発展シナリオ

玉 井 哲 雄
(東京大学)

たまに SeaMail に投稿するのが，他の目的で書いた原稿の再利用ばかりで申し訳ない．柄にもなく表題のような未来予測的な文章を書いたのは，文部科学省科学技術政策研究所というところから，「注目科学技術領域の発展的シナリオ調査」に協力し，ソフトウェアエンジニアリング領域の今後 10 年から 30 年の発展シナリオを自由に書いてほしい，という依頼があったからである．1 年先のことも分からないのに，10 年先などとんでもないし，まして 30 年先の予想など冗談でしかないと思ったが，それだけに勝手なことが書けるかとつい引き受けた．それが 2004 年 9 月のことだから，いまやかなり古い話ではあるが，なにしろ 30 年先だから，1-2 年の経過はあまり関係ないともいえる．きわめて短期間で書いたので，正直なところ，内容は思いつくまま書き散らした，という感がある．

この報告書は 2005 年 5 月に刊行されていて，その概要は

http://www.nistep.go.jp/achiev/ftx/jpn/rep096j/idx096j.html

で見ることができる．この執筆に際し，文章を他で利用してよいか尋ねたところ，構わないとのことだったのでここに投稿する次第である．

(1) 現状分析

[ソフトウェアの需要]

ソフトウェアの需要は増大し続けている．経済産業省の特定サービス実態調査では，情報サービス産業の内のソフトウェア開発/プログラム作成の売上高は 2003 年度で 6 兆 7 千億円である．しかも，これはソフトウェア専業者の統計で，コンピュータ・メーカやコンピュータ利用企業で生産・消費されるソフトウェアは含まれていない．たとえば組込みソフトウェアはほとんど含まれていないが，その年間開発規模は別の統計から，約 2 兆円といわれている．

また，現在のソフトウェア市場の顕著な動向は，開発サイクルの短期化である．新しい機能が次々と求められ，それらの開発には数週間から数ヶ月という短い期間が割り当てられる．

[社会生活に及ぼす影響力の増大]

現代の生活の周囲にはソフトウェアが満ち溢れている．携帯電話だけでなく家庭内にあるあらゆる電気製品には膨大なソフトウェアが組み込まれている．またクルマには数 10 個のマイコンが積まれ，そこに載せられているソフトウェアは合計すると数百万行の規模に達する．Web サービスに代表されるインターネットの職場や家庭内での利用の増大も，大量のソフトウェアの開発に依存している．

このことは，ソフトウェアの信頼性や生産性の欠陥が一般社会生活に及ぼす影響力がきわめて大きなものであることを意味している．

[ソフトウェアの生産性]

ソフトウェア技術は進歩したが，生産性の伸びはハードウェアと比べれば低い．この 30 年で，ハードウェアの伸びは 50,000 倍であったのに対し，ソフトウェアの伸びは 3~10 倍程度であったといわれる．ソフトウェア開発に多くの手法とツールが使われるようになり，プロジェクト管理技術も発達したが，一方でソフトウェア開発が労働集約的であるという実態は，相変わらず続いている．とくに携帯電話市場など，激しい競争条件下にある分野では，開発量の増大はソフトウェア技術者の長時間労働で対処されている．

[日本のソフトウェア産業]

日本のソフトウェア産業は，上からと下からの技術の空洞化という問題点を抱えている．基本ソフトウェアはマイクロソフトに代表される米国企業の製品に押さえられ，その追随に終始せざるをえない状況がある．一方で，定常的に生産されるソフトウェアは，中国，インドなどへの外注（アウトソーシング）への依存度が高まっている．

日本のソフトウェアは輸入が輸出を上回る入超が常態化しているが，その超過量は圧倒的である．従来，日本語の壁で国内市場をある程度守ってきた経緯があるが，それは裏返して言えば，日本のソフトウェアの輸出を妨げ，輸出意欲を発達させない要因となっている．

[ソフトウェアエンジニアリグ技術]

ソフトウェアエンジニアリングという概念は 60 年代末に提唱され，70 年代は構造化プログラミング，構造化分析・設計を主導概念として発展した．80 年代はプロジェクト管理，品質管理，構成管理などの管理技術にシフトする一方，プロトタイピング，レヴューなどの実践的技術が普及した．90 年代はオブジェクト指向技術が脚光を浴びるとともに，並行してソフトウェアプロセスのモデル化や評価が注目された．2000 年代はプロセスからアーキテクチャへの関心の回帰が見られるとともに，コンポーネント技術などオブジェクト指向を発展させた新たな技術が展開してきている．

[現状のソフトウェアの性質と課題]

現在のソフトウェアの特徴は，遍在性（どこにでもある），発展性（常に変化する），多様性（多様な版とユーザが存在する）にある．ソフトウェアエンジニアリングは伝統的に大規模複雑化への対処ということを謳い文句にしてきたが，現在はこのような遍在性，発展性，多様性にいかに対応しつつ，信頼性を高め生産性を上げるかが課題となっている．

(2)　今後 10～30 年程度の発展シナリオ

[ハードウェア技術の予測]

ソフトウェアが載せられるハードウェアの情況をまず見ると，現状はギガ(10 の 9 乗)の世界である．すなわち，CPU はギガヘルツ，メモリはギガバイト，通信速度はギガビット/秒(bps)というオーダーの性能を持つ．これが，2015 年にはテラ（10 の 12 乗），2030 年にはペタ(10 の 15 乗)の世界になるだろう．

[ソフトウェアの需要]

過去のコンピュータの歴史では，ハードウェアの能力・容量が飛躍的に増大するたびに，それに見合う需要があるのか，との疑問が常に出されてきた．しかし，いつの場合もそれを上回る需要が生み出される結果となった．したがって，今後10年から30年の間についても，需要が拡大し続けることはほぼ自明である．

[ソフトウェアエンジニアリングの技術]

(a) 大量なコンポーネントのもたらす複雑性への対処

すでに相当量のソフトウェア・コンポーネントが作られ，一部流通しているが，2015 年までには膨大な量のコンポーネントが投入され使用されることになろう．そのような大量のコンポーネントから構築されるシステムは，複雑性のボトルネックに直面することになる．

コンポーネントは，さまざまな技術者，組織により，長い期間にわたり，高い頻度で追加され変更されていく．そのようなコンポーネント群から，適切なものを検索し選択する技術，それをカスタム化し組み立てる技術が必須となる．

その検索，選択，変更，組立てをすべて人手で行うことは不可能となるため，なんらかの知能化，自動化が工夫されなければならない．具体的には

1. コンポーネントに知識を付加し，エージェント化
2. ノウハウとして設計パターン，要求パターンなどの部品化と知能化
3. コンポーネントの擦りあわせの自動化
4. 人とエージェントの共同

が行われていくだろう．そして2015年前後にはかなりのレベルで実現していると予想される．

(b) ソフトウェア開発の自動化

ソフトウェア開発の自動化は古くから追求されてきたテーマである．上に述べたコンポーネント技術の知能化，自動化をベースに，2015 年には，モデルからのソフトウェアの自動生成が実用的なレベルで実現しているだろう．ただし，生成のもととなるモデルは，システムの構造や振舞いについて，ある程度の詳細を記述したものである必要がある．

さらに，2020 年ごろには，ソフトウェアに対する要求，ポリシーからソフトウェアが自動生成されることになろう．さらに 2025 年ごろには，問題領域で普通に使われる言葉からの自動生成が可能となろう．

(c) 自律性をもち環境に適応して自己組織化を行うソフトウェアの実現

ソフトウェアは常に環境に適応することを求められる．環境にはビジネス環境やシステム環境があり，それらが変化すればソフトウェアも変わらざるをえない．また，ソフトウェアが環境間を移動すれば，それに適応してソフトウェア自身が変化しなければならない．さらに，新しく開発されるソフトウェアも，独立に存在するものでなく，すでにある環境に投入される．

これらの適応変化は，これまでは基本的に人間の手でなされてきた．しかし，ソフトウェア自身が自律的に適応し，また自己組織化する技術が今後 30 年の間には進展するだろう．

その発展には2つの段階が予想される．

1. 動かしながら変える (動的な変更)　2010-2015 年ごろ

一部は自動化されても，変化を起こす主体はあくまでも人間であるような適応技術は，今後5年から10年の内に充分に実用化されよう．しかし，これは稼動しているシステムを停止せずに行うという意味で，見かけほど簡単ではない．従来，コンパイル時やシステム生成時に行われてきた，システムの変更，追加，部分削除に関する技術が，すべて実行時に移行していく．この範囲では，とくに事故，故障に際してのソフトウェアの自己診断とそれにもとづく自己修復の機能の実現が達成されよう．

2. 動きながら変わる (動的な変化)　2030 ごろ

ソフトウェア自身が，自己と外界を知りそれに応じて自己を変化させる能力を備える．また，利用者のフィードバックが直接ソフトウェアを変えていくようになる．この段階で初めて，ソフトウェアの自己再生産，自己組織化ができるようになったと見なせよう．

(d) 検証技術

ソフトウェアの信頼性，安全性，セキュリティを保証するための検証技術は，数段のレベルの向上が要求される．とくに上記の自己適応技術と関連し，自律的に変化したソフトウェアが正しく動作することも自動的に検証できなければならない．適応技術の発展段階に応じた検証技術の発達が予想される．

[ソフトウェアエンジニアリングの理論]

上に述べた技術を実現するためには，その基盤の理論的強化が必要である．まず，すでに理論としてはかなりの成果が挙げられているが，その実用化は今後に期待されるものがいくつかある．たとえば，関数型プログラミング，論理型プログラミングのような明快な計算モデルに基づくプログラミング技術は，5 年後か 10 年後に何かのきっかけで急速に普及することがありうる．また，ソフトウェアの仕様技術の一つとして，形式仕様記述言語を用いることは，分野を特定しながら徐々に普及していくだろう．さらに，定理証明技術による自動検証は，形式仕様技術と関連しながら，実用化されていこう．

技術の項で挙げた大量なコンポーネントからなるシステムの複雑性に対処していくには，エンジニアリグとして個々のコンポーネントと全体のアーキテクチャを設計し管理するだけでは不十分であることが予想される．そこで，人工物であるソフトウェア・システムを，あたかも自然物のように観測，分析する手法が求められよう．そこでは，統計力学に相当するようなマクロな分析手法と，その分析結果をソフトウェアのアーキテクチャやコンポーネントのミクロレベルにフィードバックする方法論が求められる．

そのような「科学的」アプローチとして，より大きな理論的枠組みを想定するとすれば，以下のようなものが考えられよう．

1. 連続時間の導入(2015 年ごろ)

計算は離散的な量を対象とするため，本質的に連続量である時間の扱いがこれまで困難であった．もちろん，応答型システム，実時間システムなどは時間を意識したシステムであるが，時間を理論的に扱うにはやはり離散化して対応するのが常であったと言える．しかし，ニュートン力学は連続時間を扱い，微分積分の概念を導入することで美しい理論をうち立てることができた．ニュートンの方程式に相当する計算理論の構築が求められるゆえんである．

2. 連続空間の導入(2030 年ごろ)

地球規模の互いにリンクされたおびただしい量の計算資源は，空間連続体として捉えることが可能である．これを理論的に扱うにはマックスウェルの電磁方程式に相当する時空間の計算理論が求められよう．

[他分野に学ぶ]

今後のソフトウェアは，計算機科学のみならず他の学問分野から学び，技術として発達させていく必要があろう．

1. 生物学

すでに挙げた環境適応，自己組織化という概念からして生物学から導入されたものである．さらにたとえば，免疫系（とくにウィルス攻撃への対応などのセキュリティ対策），反射系と中枢神経系と脳との役割分担(とくにロボットソフトウェア)，などが有用であろう．また，すでに述べた大量なコンポーネントがネットワーク上に生息し，生成，変化，分化，消滅を続けている状況の分析には，生態学的アプローチも有効であろう．

2. 文献学，考古学

ソフトウェアも一種の文書であり，それを歴史的なパースペクティブで文献学的に研究することは，新たなソフトウェアエンジニアリグ技術を発見するためにも有効であろう．

3. 社会学，文化人類学

ソフトウェアの開発組織はきわめて人間的な組織である．その分析，改善，研究には社会学や文化人類学の手法が有効であろう．

(3) 作成した発展シナリオを踏まえた日本のとるべきアクション

[教育，人材育成]

アクションの第一は教育，人材育成である．とくに次のような世代の特徴を活用することが有効であろう．

1. 生れたときからデジタル環境に育った世代のセンスの活用

これまでのユーザインタフェースの研究と実践は，機械を使い慣れないユーザにいかに使いやすいインタフェースを提供するか，ということであった．しかし，これから就業年齢に達していく世代は，生れたときからデジタルな機器が身の回りにふんだんにある環境に育っている．そのような世代が使用し，また開発するインタフェースはこれまでとは異なるセンスのものになるだろう．それを積極的に育てていくべきである．

また，不法アクセスなどの問題を起こす「ハッカー」は，当然若い世代が多いが，彼らの多くは高い技術と関心をもっている．その力を創造的な方向へ向かわせることが望ましい．それには，情報倫理教育も「しちゃだめ」でなく創造の芽を育てるようなものとしなければならない．

2. 熟年世代の経験と時間の活用

団塊世代は情報科学/工学教育を受けた最初の世代であり，またその後多くが，業務での本格的なソフトウェア開発経験を豊富に持つことになった．今後，その世代以降が次々と定年に達するが，その経験と技術を生かす方策が有用であろう．その方向は，楽しみの開発と楽しみながらの開発ということになるのではなかろうか．

3. ソフトウェア技術者の地位向上

現状では，計算機科学分野を志望する学生の数が米国でもヨーロッパでも減少傾向にあり，問題となっている．日本ではむしろかなり以前からその傾向が見られた．将来のソフトウェアの需要から考えると，この事態は深刻である．これに対処するには，ソフトウェア技術者の成功物語，ヒーローの誕生が必要である．より卑近な策としては，ソフトウェア技術者の作業環境を改善し，処遇を高めることが望ましい．

4. デザイナの役割増大

WWW ページに代表されるように，コンテンツ創造とソフトウェア開発の一体化しつつある．そこで広い意味でのデザイナの役割が増す．そのためにソフトウェア技術に高い素養があるデザイナの育成が肝要である．

5. 他分野の専門家であると同時にソフトウェア技術者であるような人材の育成

生物，物理，化学，アート，経営学などの分野でエキスパートでありながら，片手間でなくきちんとしたソフトウェアエンジニアリング教育を受けた人材がますます必要となる．

[研究開発投資]

研究開発投資にはさまざまなテーマがありうるが，以下いくつかに絞って記述する．

1. 知的財産処理の仕組み

知的財産権というととかく権利保護が全面に出がちがだが，オープンソースのような自由な開発形態を阻害しない工夫も必要である．一方で，知的財産権で保護されているコンポーネントやソフトウェアを正当な対価を払って簡単に使える制度，システムを構築すれば，権利者，利用者ともに便益がある．

2. 基盤研究の重視

UNIX，関係データベース，オブジェクト指向技術などに見られるように，ソフトウェアの研究成果が広く普及するには10年から20年以上の年月を必要とすることが多い．そこでやはり基盤的な研究を助成することは欠かせない．研究評価は重要であるが，その評価も長い期間にわたり追跡できる体制が必要である．

3. 国際交流

IT 技術はもっとも国際的な分野である．途上国でもパソコンを 1 台買えば，最先端の技術にアクセスが可能となる．そして，今日開発された技術は，明日には全世界に拡がる．したがって，欧米とだけでなく，アジア，南米など多くの地域との人的交流が，今後ともますます重要となろう．そのためには，他の文化の人々と闊達にコミュニケーションができる人材が求められる．このことは，人材育

成のあり方にも密接に関連してくることである．また，交流を促進する資金，制度が求められる．

[産学によるソフトウェアの大規模な実験の場の構築]

ソフトウェアは元来，実験が難しい．それは，ソフトウェア開発に人的な要素が大きく，実験をコントロールするのが難しいためである．企業は競争的な開発に追われ，実験をする余裕がない．一方，大学にはアイディアをスケールアップして試す場とリソースがない．たとえばセキュリティ関連の技術は実験が有効であるが，計画，管理が難しい分野である．そこで産学が共同して大規模な実験を行えるような場を作り，そこに資金を投入して他では見られないような実験を実施すれば，新たなソフトウェアエンジニアリング手法のきちんとした評価がなされ，大きなインパクトを与えることができるだろう．

SEA 名古屋ミニセミナーレポート

石川雅彦
SRA

はじめに

2006 年 3 月 1 日、SEA 名古屋主催ミニセミナーを開催しました。このレポートでは、ミニセミナーの内容を中心に、セミナー前後の様子も合わせて報告します。

セミナーテーマ

今回のテーマは、「企業にとってのブログ」でした。ご存知のように、ブログは情報発信ツールとしてまたコミュニケーションツールとして社会の動向にインパクトを与えるものになっています。更に、個人だけでなく企業もブログによって、カスタマーとのコミュニケーションを試みようとしています。しかし、企業ブログのケースでは失敗例も報告されるようになってきました。その中で、名古屋市に本社を置く企業、ブラザー工業の社員が企業ブログを始めて 1 年たち、様々なメディアに紹介されていることを、SEA 名古屋企画運営委員(kiui)の一人が目をつけました。そこで、メーリングリスト上で 2005 年末に提案を行い、忘年会、新年会の場で打合せを行い、講師や会場の調整を kiui メンバーで分担して行い実現となりました。

開催前

講師調整に際し、提案者から講師に対して、以下のような質問事項を送りました。

（1）なぜブログを始めようと思ったか？
（2）リスクもあるので社内の説得が結構大変だったのではないかと思うがどのようにクリアしていったのか？
（3）運用体制（記事を誰が書くか、内容チェックなど）
（4）はじめてからの反響
（5）ブログ前後で何が変わったか
　予測していたことと現実との違い（狙い通りだったところ、予想外だったところ）
（6）時間経過によって何が変わってきたか。
（7）今後の取り組み
　続ける場合：続けるパワーもいると思うし、やめる場合：やめるタイミングもあると思う。
（8）同様なことをはじめたいと思っている企業（または担当者）に対してのアドバイス。
（9）システム的な面の質問

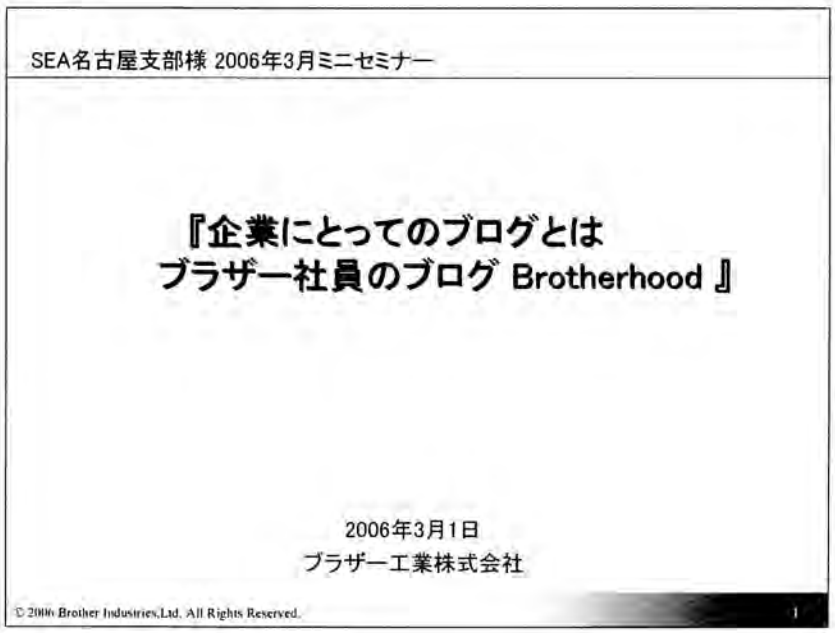
・外部ブログを利用した理由（自社で立てなかった理由）
・多くのブログの中から今のブログを選択した理由
・使用感についてよかった点、よくなかった点（不足する機能など）

セミナーはこの質問への回答を盛り込みつつ進められることになりました。

内容

講師としてグループ・マネージャーの小久保雅俊さんにお話いただきました。また、実際にブログ作成に携わっている古橋雅彦さん、松原淳さんも同席されました。（古橋さん、松原さんは講演後の質疑応答の時に回答側としてお答えいただきました。）

以下、古橋さんから提供いただいたスライドを掲載しながら、一参加者としての私がメモした記録を基に再構成した内容をご紹介します。



(1) ブラザーブログ「Brotherhood」の原点

innovative な output を求められた。

→At your side の精神でインターネットを利用し、顔の見える形で製品に投射できない思いや悩みを双方向コミュニケーションで伝える。さらに、

・自分たちだけではなくお客様に喜んでもらえるキャンペーン

・目標は具体化せず、継続だけを目標とした

(2) サイトの体裁：

コンセプトしては、人間味・手作り感を目指した　会社のコーポレートカラーは青であるのに対して、ブログサイトはオレンジを基調にしており、対比的である。

・ブログヘッダには新製品の情報を置くようにした。サイト内には、ブログ記述者やブログの登場人物の似顔絵を掲載している。この似顔絵は、製品のバンドルソフトで作成した。ブログ内には様々な説明を挿し絵と共に行っているが、これは、文章・技術用語での説明に限界を感じた結果であり、さし絵の有効性に気づいたことが大きい。

(3)はてな選定理由：

ブログサイトははてな内に置いている。理由としては、コミュニティとしての信頼感があったから。また、イントラに検索モジュールの提供を受けていたため、という理由もある。

(4)ブログチームの構成と立ち上げまでの経緯。

・運営チームとシンプルな運営フロー

・承認を得るまでに紆余曲折があった。

ブログチーム立ち上げは2004年夏が原点だった。まず、2004年8月、マネージメントクラスに対して社内プレゼンを行ない、インターネットで起こっている事業の理解と共有を行なった。このプレゼンを行った結果、インターネット利用に対して積極的推進論と慎重論があった。次いで、2004年11月、「ブラザー社員のブログ」企画の説明を第2回社内プレゼンとして行なった。そして、2005年1月に企画詰めと仲間作りを行った。

我々は、ブログ運用が主業務ではない。従って、実務クラスのキーマンの理解協力を取り付けることは重要だった。また、企画に賛同、協力してくれる仲間を見出すことも重要。『必要なのは役者とやる気』だと思った。

2005年2月初め　経営層への報告を経て、2月15日　ブログを開始した。

（5）数字で見るブラザー社員のブログ

2005 年 2 月 15 日から活動を始めて約 1 年の期間の間に、記事総数は約 155 記事。カテゴリ別に見ると、最も多い話題は開発で 25%、次いで商品 12%、営業 19%、動画 3%（動画は最近始めた）、想い(6%)、研究(3%)と続く。

（6）プレゼントキャンペーン

ブログの中でプレゼントキャンペーンを展開した。最初は「HL-2040 を 5 台プレゼント」という企画。応募が一週間で 800 名ほどあった。

次のキャンペーンは「『パソコンとプリンタのある風景』の写真を記載してください企画 」を実施し、コンパクトなモノクロレーザープリンタ複合機が本領発揮する所を教えて下さい。と読者に呼びかけた。この企画の中で、「複合機の上に猫が乗ってボタンを押してしまう。」といった写真が寄せられ、話題を呼んだ。

第 3 回プレゼントキャンペーンは、2005 年 6 月 16 日から 7 月 6 日まで 3 週間の期間で行った。「勉強させてください~ブラザー社員街角レポート編」企画として、コンパクトなレーザー複合機 DCP-7010 プレゼントキャンペーンを行った。
応募者は約 1700 名に上り、企画への提案数　210 件と大幅に人数が増えた。人数が増えた理由は不明。

第 4 回プレゼントキャンペーンは 2006 年 1 月 26 日から 2 月 8 日の期間で実施。
企画テーマ「勉強させてください~ブラザー社員実験編」で、MFC-9420CN を希望者にプレゼント。応募は約 1170 名、企画への提案数 60 件に上った。
実験編の内容は、レーザー複合機の良さがわかる実験方法のアイデアで、提案を参考に動画コンテンツを作成した。
（ここで作成した動画コンテンツをデモ）。

第 3 回から第 4 回まで間があいたがこれは、新しい企画にチャレンジするために時間が必要だったため。

（7）話題性

多くのメディアに取り上げていただいた。毎日新聞、NHK ニュース 10、日経 PC21、AERA　他。最近では、2006.1　Hotwired Japan の特集”Web 2.0 信頼の構築が重要”で 4 社紹介の最後にとりあげてもらった。

（8）トラックバックの中でのわれわれの気づき

多種多様な視点からの意見に触れることができた。　従来のコミュニケーションチャネ

ルからは得られないものだった。商品レビュー、より具体的なニーズ、スペックを調べてのコメントより深い知識や体験からの辛口コメントや提案。　自社全般に関する思い出やイメージ、ブログに対する印象など。

(9)社内の反響

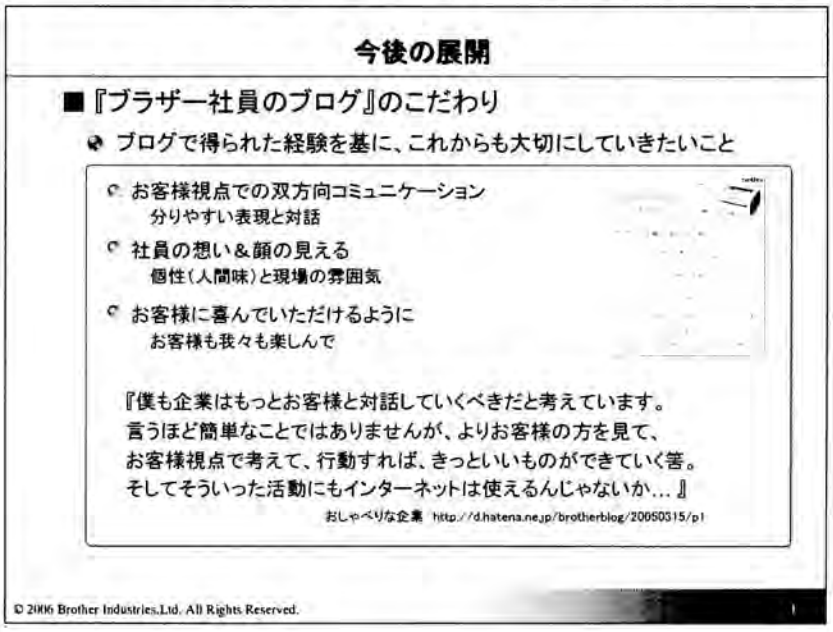
以外に社内読者が多いことがわかった。「何をするつもりだろう」といった、疑心暗鬼もあったが、おおむね、以下のような良好な反響をもらった。

・信頼性の高い製品を出せというプレッシャーを感じる
・よい評価が多い。信頼性のある高い製品を出す動機づけになる
・もっと自社製品を勉強しなければと感じた。
・技術者へのインタビューに好感を持った。

(10) ブログならではの発見

Open な双方向の対話から得られた気づきがある。

・一般人の常識と業界人の常識とのギャップ
・顧客が企業に求めていたもの。
・商品に対してロイヤルティの高いユーザー層を発見できたこと。
・ブログを通して直接いただいた声を社内共有することによる活性化



(11) 社員のブログのこだわり

ブログで得られた経験を基に、これからも大切にしていきたいことがある。

・At your side　お客様視点での双方向コミュニケーション

・社員の思い
・お客様に喜んでいただけるように
・企業はお客様と対話していくべき。
・また、通常はものづくりの意図（設計者の意図）は見えないが、ブログを通して見えない意図を伝えていくことができるのではないかと考えている。

以上が講演の話です。この話の最後にブログメンバーの自己紹介がありました。
セミナーはこの後、質疑応答に進みます。
以下は一参加者としての私がメモした記録を基に再構成した内容をご紹介します。

質疑応答

1．コメント

大学教員からのコメント。大学のケーススタディに使えそうだという感想を持った。講義の中で印象に残った点が２点ある。まず一点目。はてなの機能を有効に使っていると思った。例えば、キーワード機能、トラックバック機能など。
この一点目のコメントに対して講師の回答があった。外部のトラックバックも受け付けている。はてなのキーワード検索の範囲でしか機能しない。本当はもっと広い範囲でやりたいがうまい仕組みがない。個人情報を集めない範囲で他があれば使ってみたい、など。

次いで同じ人から二点目のコメントとして、ブラザー工業にまつわるリクルートの思い出を披露。会場からの笑いを誘っていた。

2．質問

自社サイトの情報発信を行っている参加者からの質問として、継続のこつに関する質問があった。講師回答は以下の通り。

・組織論よりも企画への興味を社員にもってもらえることがこつ。
・職権では動かないと思う。
・　社内でよく読んでくれる人が協力してくれる。

3．質問

ブログによる商業的な成功を判断しているか？

・していない。
・社内の関心事ではない（には笑いがおこった）
・成果についてトップから問われたことはない。

4．質問。双方向コミュニケーションについて

回答。コメントはつけていない。始める時に議論があったが、双方向コミュニケーションにはトラックバックだけで十分だろうという結論になった。

5．質問

40 年前にブラザータイプライターを使っていた、という人から以下の質問があった。

質問１）海外に商品展開されているので英語のブログは？

回答１）考えてない。単に英語に訳せればよいというものではないだろうと思うので。

質問２）ある企業で、次期商品を考えている人が多いが、社内の横の情報交換がないというケースに遭遇したことがある。

回答２）そのようなケースもあるだろう。(中略)一般的に、組織内コミュニケーションについてはどこでも改善の余地は見出せるだろう。

6．質問

殻に閉じこもりがちな技術者が多い中、顧客指向であることに感心した。顧客指向を感じたきっかけは？

回答）（会場の関係者から回答）私は技術者ではないので（会場から笑い）研究テーマがインターネットだったが...

講師から補足回答）「At your side」というポリシーと、彼の研究テーマを組み合わせるとブログという手段が浮かび、そのブログという手段と伝え手の幸せな出会いがあり、とてもラッキーが重なっていたように思う。

セミナー終了後

セミナーは結局、参加者　25 名を数えました。　会場のキャパシティからすれば盛況と考えてよいでしょう。また、参加者からの質問も多くあり、参加者の関心の高いセミナーでした。参加者は名古屋地域だけでなく、関西方面からの方もいらっしゃいました。また、関東圏在住の方から「平日でなければ参加できた」という意見があったと伝え聴きました。このように関心が高かった理由はこの後に掲載するアンケートにも表れていますが、

- 身近にある企業の話だったこと
- インターネット上の身近なツールがテーマだったこと。
- 関係者から直接話しが聞ける機会だったこと。

などがあります。

また、セミナーの翌日、Brotherhood にこのセミナーの様子が載りました。
URL をアンケートの後に掲載します。

アンケート

回答者	良かった点	悪かった点	今後の希望テーマ	その他
1	司会の方がとても緊張していたように見えたが、懇親会で講師の方が、"司会の方がとても緊張していたので、かえって私の緊張感が溶けた"と言っていた。	・セミナーの締めが遅かった ・講師の話からclosingまでにprojectorの映写に時間がかかった(間が悪かった) ・closingが長かった。 ・講師の飲み物を提供するのが遅れた ・懇親会の移行までに時間がかかった。撤収の要領を考えたほうがよいのではないか。または、終了時刻を早めた方がよいのではないか?	・ソフトウェア開発に関する実用的な話。(TIPS, FAQではなく) ・形式仕様記述のテスト工程への応用の話 ・組込システムへのソフトウェア工学的手法の応用の話 ・Empirical Software工学の話 ・オブジェクト指向手法の理論と実践のギャップの認識とすりあわせの話 ・インターネットの未来に関する話 ・インターネット技術のソフトウェア開発への応用の話 ・インターネット基盤に支えられた、協調支援プロジェクト管理を含む)の話	
2	生の声が聞けたのが本当に良かったです。苦労した点やうれしかった点を聞けると、自分たちも頑張ろうって気になります。 今回のセミナーを聞いて、我々も改めて頑張らなければと思いました。	特にはありません。	うーん。今回みたいな、実際の運営者の方の声を聞けるとうれしいです。	
3	場所	宴会会場のサービスの遅さ	・IPv6の今後(デジタル家電の裏側) ・技術者の健康維持の方法 ・2007年問題へのとりくみ(ディスカッション)	こんどは、日付変わる前にかえります(おい
4	内容:小久保さんの人柄・熱意が、身近で感じられてよかった。 会場:伏見駅から近く、便利だった。飲み会会場も近かった。	自分は前の方で聞いていたが、後ろの方でもパワーポイント・声などがうまく伝わったか気になる。	今特に思いつきません・・すみません。	とても活発に質問が出て良かったと思います。
5	わかりやすい内容及び説明	席がせまい		
6	大企業ブログの運営について、直接話を伺うことができてよかった。	とくになし。	今回のような実務の運用系というか裏舞台のような内容。	
7	・内容は分かりやすかった。 ・それなりの人数が集まると、講師にとっても励みになる。	・会場の都合で、昔みたいに、珈琲やお茶菓子を出せないので、一般の1500円は高い? ・事前にアンケートを準備すること。内容に、満足度の点数を入れた方が良いかも。	・スキル標準(IT,組込み)の現状と活用事例 ・ラーニングオーガニゼーションの導入例、品質・見積精度向上との関係 ・海外(中国、インド、東欧など)でのソフト開発拠点の成功例、失敗例	・会場前30分で準備するのは非常にタイト。 ・定量的なものを、そのままお渡しするのは、ちょっとと言うことだったので、レジュメは敢えて無しとしました。
8		定時後に気軽にリフレッシュを兼ねたセミナーということで少し遅めに開始にしたほうがよいかなと思います。 19:00−20:30(1時間の講演+30分のQ&A) また、1時間の講演だと、比較的依頼しやすいかなと思います。1.5HR話すのは結構大変。		

参考 URL

http://sea.forums.gr.jp/nagoya/

SEA 名古屋支部 WWW ページ： セミナー案内を掲載。

http://pub.brother.com/pub/com/blog/Brotherhood.pdf

ブラザー社員のブログ「Brotherhood」

このブログの 2006 年 3 月 2 日（セミナー翌日）のブログにセミナーの様子が掲載されています。

ヨーロッパ出張日記

いくつかの会議をきっかけに考えたこと

岸 田 孝 一
SRA-KTL

1. 発端

ことの始まりは去年の秋にブダペストで開かれた ICSM(ソフトウェア・メインテナンスに関する国際会議)だった. Software Evolution Dynamics の提唱者 M.M.Lehman 先生に Special Award をさしあげるイベントが会議の晩餐会の席であり, たまたま故郷のブダペストに帰っておられた Les Belady さんが表彰のことば(昔話をまじえた長めの演説)をされた. Belady さんにお会いするのも久しぶり(前回は数年前にかれがボストンの三菱電機研究所 MERL を退職しオースティンに戻って地元のソフトウェア協会のボランティアを始められたとき)だったので, 日本に帰る日の朝, ホテル近くのカフェで久しぶりにいろいろな話をした.「11月の IWFST 上海に来ませんか?」とお誘いしたら即座に「OK」という返事. こうして空白だった IWFST2005 の基調講演が決まった.

さて, 新しい年が明けてすぐ(1月3日)にその Belady さんから E-mail. 年末に送った会社のカレンダーのお礼.「何年か前からクリスマスカードを出すのは止めたので mail でハッピー・ニュー・イヤーとあって, そのあとリクエストが1つ」:

> John von Neuman Association (ハンガリー・コンピュータ協会)で定期的に世界中からいろいろな人を招いてフォーラムを開いている. 次回は3月の予定だが, 東アジアから誰か呼びたいと思うので, 適当な候補者を推薦してほしい.

とのこと. 日本・中国・韓国とりまぜて何人かの名前をリストアップし, 何回か mail をやりとりしたのだが, 結局「面倒なのでお前自身が来い」をいうことになってしまった.

予定された期日は3月 22 日. ちょうどその翌週の土曜日から 1 週間, ウィーンで ETAPS (The European Joint Conference of Theory and Practice of Software) という会議が予定されていた.「理論と実践」とタイトルではうたっているが, Formal Method を中心として 18 もの併設ワークショップが集まった会議である.

SEA 幹事の野中さんから「併設の SPIN ワークショップに参加するので岸田さんもどうですか? 玉井先生も ETAPS 本会議の PC メンバーだし, セッションの座長もされるみたいですよ」という誘惑. 1週間も Fomal 漬けにされるのはいやなので, 最初の 2 日間の MBT (Model Based Testing) のワークショップだけおつきあいすることにした.

2. ブダペスト

出発は3月19日. 前回 ICSM 参加のさい, リムジンバスが高速上でパンクし, あやうく飛行機に乗り遅れそうになったので, 今回は早めに出かけたのだが. 春休みのせいか, 成田空港はものすごい混雑だった.

時差調整のためウィーン西駅前のホテルに 1 泊. 値段が安い割りに部屋は広かったが LAN の設

備はなし. チェックインのあと, すぐに駅へ行って, ブダペスト行きの切符を買う. 「払い戻し不可だけど安い切符があるよ」と駅員さんは親切. 駅のスーパーで飲み物と夜食用のスナック類を買って, とりあえず一眠り. 深夜に目がさめたのでそのままブダペストで使う発表スライドを作っていると, WBC 準決勝の日本チームの勝利の短いニュースが CNN で流れた.

翌20日午後一番のブダペスト行き EuroCity 特急に乗る. 2等車だが JRのグリーンより快適. 乗車率40%程度. すべての座席にPC用の電源コンセントがついているのは便利だ. 国境通過の前にオーストリア, 通過後にハンガリーの国境警察官が回ってきてパスポート検査. そのあとハンガリーの車掌さんの検札. 窓外の風景は延々と畑が続く. オーストリア側の畑には, ところどころ発電用の風車が林立していたのが印象的だった.

ブダペストのホテルは去年 ICSM の時に泊まったのと同じ K+K Opera. 今回は最上階の部屋なので天井が半分屋根裏風でおもしろい. ここは無料の LAN でインターネット接続ができる. ホスト役のブダペスト工科大学の先生から mail が入っていたので電話. 明後日の朝迎えに来てくれる時間を打ち合わせる.

翌21日の朝食後, 前日買い忘れたウィーン行きの切符を買いに地下鉄で駅へ行き, そのあとホテルへ戻る途中, 広場で青空市場を散策. 屋台のホットワインとソーセージで昼食を済ませ, ホテルへ帰ってスライドを仕上げる. 途中 SEA Forum の参加申込み mail が何通か届いたので返信. まさか受け取ったほうはブダペストからとは思わないだろう. 時差がまだ取れず眠くなったので寝る. 目覚めたのは深夜. スライドの見直しと修正をしているうちに朝が来た.

約束の 8:30 に Balint Domoliki 先生がタクシーで迎えにきてくれる, 途中で Belady さんを拾ってブダペスト工科大学へ. フォーラム司会役の Gymothy Tibor 先生, もう 1 人のスピーカ Lisziewitz Zsolt さん, ハンガリーにおける SPI 関連活動の中心的な役割をしている Miklos Biros 先生, その他の人たちに紹介される. 耳慣れない名前ばかりなのでなかなか覚えられない. .

わたしのプレゼンのタイトルは "Software Situation in Japan and Far East Asia". 先方からのリクエストは, 東アジア(中国・韓国・日本)のソフトウェア事情をなるべく幅広くという難しい注文だったが, あまり一般的な話をしても面白くないので, 自分自身のこれまでの経験にもとづいて, プロセス改善, アウトソーシング, 組み込みシステム開発, オープンソース, 産学連携, などの話題について話した. そして最後に, ソフトウェア工学と儒教哲学との思考フレームワークの類似性について, 荻生徂徠や富永仲基を引用して説明したのだが, これはかなり受けた.

一昨年イタリアのピサで開かれたフォーマル・メソッド会議 (FM2003) での Invited Talk で似たような話をしてけっこう面白がられたので, 柳の下の2匹目の泥鰌を狙ったのだが. まずは成功したようだ. 講演が終わった後 Biro 先生がそばへきて「わたしもソフトウェアにおける Cultural Factor の問題には大いに関心を持っている. 明日もう一度お会いするときにわたしが書いた Paper のコピーをさしあげる」といってくれた.

わたしのあとに話をしたのは L & Mark という小さなソフト会社をやっている Zsolt さん. いきなり漢字で書かれた「孫子の兵法」のスライドを出して, 中国でのビジネスについて滔滔と語りはじめた. どうやら GIS 関係のソフトを上海市政府に売り込んでうまくビジネスを展開しているみたいだ. 日本のソフト会社も人件費の格差を利用したアウトソーシングにばかり熱中していないで, 中国マーケットの将来性を睨んだ本格的なビジネス展開をしてもらいたいものだと感じた.

このフォーラムの詳細は, 2002年9月の第 1 回から今年5月の第15回まで(わたしがしゃべったのは第14回)の記録がすべて, 次の Web Page:

http://www.inf.u-szeged.hu/stf/rendezveny-en.php

に載っているので, 興味のある方は覗いてみられるとよい.

大学のカフェテリアでみなさん一緒にランチを食べて散会. Biro 先生の車に Balady さんも同

乗して大学キャンパスの隣にある情報産業パークを見に行く．新しい建物が立ち並び，エリクソンや IBM などの会社の産学協同研究所の看板が出ている．インドの TATA がここにも進出してきているのには驚いた．「今朝ベルリンでの国際ミーティングから帰ったばかりで疲れたから家に帰って寝る」という Biro 先生に，ドナウ川をはさんで向こう側のブダペスト経済大学（いまは農業大学を吸収合併して CorvinUS 大学と改名，かれはそこの情報学部の教授）まで送ってもらい，翌日の再会を約束して別れる．大学のすぐ裏手にある有名な市場を一回りぶらついてからホテルへ帰って昼寝をする．夜は Berady さんと食事をしながら四方山話．

翌朝 Biro 先生が迎えに来てくれて Corvinus 大学へ．学部長さんと儀礼的なお話をしたあと，かれの研究室へ行って約束の論文コピーを貰う．

(1) Cultural Environment Protection in the Information Society
(2) The Impact of National Cultural Factors on the Effectiveness of Process Improvement Methods

(1)は，有名なホフステッドの経営人類学モデルを下敷きにして，いくつかのケーススタディの紹介をまとめたもの．(2) は CMMI/SPICE の導入と民族的価値観との関係について，アイルランドおよびデンマークの研究者と協同で行ったフィールド調査のレポート（それほど大掛かりなものではない）．両方ともなかなか興味深い．日本でも海外の開発途上国へのアウトソーシングは盛んに行われているが，その背後でこうした研究がなされているという話はあまり聞かないが，どうなっているのだろうか．

「ところで，こんな会議があるのだがこんど来てみないか？」とかれが見せてくれたのは Euro SPI Conference のパンフレットだった．ホテルへ帰ったあと，ネットで調べてみると1944年から毎年開かれている会議だった．昨年は11月にブダペストで開催されて Biro 先生はローカル・アレンジメントを担当されたとのこと．今年の会議は10月中旬にフィンランドのヨエンスウ市での開催が予定されている．詳しい案内は次の Web に：http://2006.eurospi.net/

フィンランドには何年か前の夏にオウル市で開かれた PSOCOS 会議に行った経験しかない．あの時は蚊の大群に悩まされた．会議場の大学からホテルへ帰る途中の道に何本も蚊柱が立ち並び，そこを通り抜けるために Jacky 高橋さんと 2 人でタバコを 2 本ずつふかし，ふだんはタバコキライな鳥居宏次先生がその煙幕に隠れるようにして早足で駆け抜けたというおかしな思い出がある．ヨエンスウというところ（大学町らしい）には行ったことがないし，いささか心惹かれるが，10月のフィンランドはもうかなり寒いだろう．

ネットサーフをすると，今年ヨーロッパで行われるソフトウェア・プロセス関連の国際会議はほかに，SPICE 2006 (5/3-5, ルクセンブルグ) と European SEPG 2006 (6/12-15, アムステルダム)の2つが予定されているのだが，両方ともタイトルが示すように SPICE や CMMI 一辺倒の集まりなのであまり魅力的ではないように感じられる．

それにくらべると Euor SPI のほうは Web に載っている過去の会議の Proceedings を見ても，かなり幅広い話題をカバーしているようだ．Biro 先生の話では，近々ヨーロッパでは アメリカの CMU-SEI の動きに対抗して ISO/IEC15504 (SPICE) の Assessor Certification を本格的に始めるようだ．

学部長先生を交えてしばらくお話をしたあと，学生たちや大学のスタッフで混雑している近くのイタリアン・カフェで軽い昼食を取り，大学本館の建物に案内してもらう．正面玄関を入ったところに大きなホールがあり，左側の壁の前に，読書しているカール・マルクスの彫像が置かれている．共産主義政権が崩壊した後，関係する彫像はすべて取り払われたのだが，「ここは経済大学なのだ

からマルクスをはずすわけには行かなかった」とのこと．イギリスのチャールズ皇太子がここを訪問しスピーチをしたとき，ふつうは右側の壁に演壇を設けるのだが，「自分と同じ名前の有名人と並んで話したい」(ドイツ語のカールは英語のチャールズ)という強い要望で，そのときだけマルクスの隣に演壇を作ったのだそうだ．

「どこか観光案内をしてあげよう」という Biro 先生の申し出に甘えて，近くにある民族博物館へ行き，10世紀以降現代までのハンガリーの歴史展示を，先生の解説つきで見てまわった．トルコ，オーストリア，ロシアという大国の狭間でもみくちゃにされた小国ハンガリーの苦難の歴史を，メーカや大ユーザそして政府機関にはさまれて苦労してきた独立ソフトハウスのこれまでと引き比べて，いささか感慨深いものがあった．博物館の最後の1室は，フォン・ノイマンをはじめとするハンガリー生まれの著名な科学者たち16人の写真と業績の展示．そのうち8人がノーベル賞受賞．「え，この人もハンガリー人だったの！」という驚きの連続．

ホテルへ帰って一休みしたあと，夕食は今日・明日の2日間で行われているドイツ・ハンガリー協同研究プロジェクト・ミーティングの晩餐会に招待される．迎えに来てくれた Belady さんとタクシーで市のブダ側(この街はブダとペストの2つに分かれている．わたしのホテルはペスト側)にドナウ川を渡り，王宮の下にあるレストランへ．ドイツ・サイドの研究代表者は，5月の ICSE2006 in 上海のプログラム委員長である Dieter Rombach 先生．お互いに「どうしてあなたがここにいるの？」と挨拶を交わす．かれらのプロジェクトは，やはり Belady さんの「外交的努力」によってまとめられたものらしい．「金の鹿」という店の名前にちなんで鹿肉のステーキを食べ，上海での再会を約束して別れる．

3. ウィーン

3月24日，金曜日．遅い朝食を食べたあと，タクシーで駅へ．今度の車両はコンパートメント式だったので，Smoking の席がある．6人用のコンパートメントに1人で座っていたら，あとから.ハンガリーの若者(たぶん学生さん)がひとり入ってきて，タバコを1本せびられた．

ウィーン到着は予定通り．ホテルはミュージアム・クオーター裏の K+K Maria Theresia．ここは，9.11事件に遭遇した FSE&ESEC2001 のとき，そして2年前の CHI2004 のときにも泊まった馴染みの宿．西駅から地下鉄でフォルクス劇場前まで行き，通いなれた坂道を登ってチェックイン．今度もまた最上階の半分屋根裏のような部屋．それはいいのだが，部屋に湯沸しがないことを発見．たしか以前に泊まったときはあったはずだがと思いつつ，近くの大規模家庭用品ストアへ行って，TEFAR の湯沸しポットを見つけて買った，東京の家で使っているものより新しいモデル．値段は3,000円だからかなり割安．しかし電圧が違うので帰国したら変圧器を手に入れなければならない．

さっそく沸かしたコーヒーとスナックで昼食をすませ、インターネットに接続してたまっていた mail の処理．香港のスポーツ TV の録画をネット上に見つけて，WBC 決勝戦での川崎選手のスライディング・ホームインのシーンを見る．．

夕方，野中さん到着．かれはウィーンは初めてだというので，さっそく市内散歩に出る．オイゲン皇太子の厩を改造したミュージアム・クオーターを通り抜け，マリア・テレジア女帝の彫像に敬意を表し，王宮からステファン教会，ザッハ・ホテル，オペラハウスあたりを歩く．夕食はどうしようかとしばらく迷ったあげく，ベルベデーレ宮殿正門脇の有名なビアホール Salm Brau まで歩くことにした．暗い夜道でちょっと迷った末にたどりつく．満員だったがなんとか相席で座ることができた．ご当地初めての野中さんは，当然ウィーナー・シュニッツェル，わたしは白ソーセージと白ビール．

翌 25 日(土曜日)はワークショップの初日．会場のウィーン工科大学へはこの前 FSF に通った

のと同じ道をぶらぶら歩いてゆく．ところが，ここが会場だろうとわたしが思い込んでいた建物には，まったく会議などやっている雰囲気がない．土曜日なので学生さんたちの姿も見当たらず，しばらくうろうろした後，受付のおじさんに訊くと，会場はどうやら少し離れた別の建物らしいことがわかり，小雨の中をそちらへ向かう．

会場の情報科学の建物は2つに分かれていて，参加登録受付は新館の1階．受付は専門の会社に委託してあって，きわめてスムースである．ワークショップ1つしか申し込んでいないのに，フル参加の人たちと同じ立派なデイバッグをくれる．われわれが参加する BMT Workshop は旧館2階の階段教室なので，雨の中を移動．これがまた旧世紀の立派な建物でエレベータはなし．幅広い階段を上って会場へ．いつも参加する会議とは違うコミュニティなので知った顔がいない．1 人だけ中国の若い研究者に会っただけ．こういうミーティングは気楽でよい．

オープニング・セッションの講演者は Harry Robinson さん．Bell Labs から Microsoft，そして現在は Google という経歴の持ち主．風貌はちょっとふとった陽気なカリフォルニア人という感じ．講演のタイトルは"Model-Based Testing for Masses（一般大衆のための MBT）"．そのプレゼンテーション・スライドは次の Page: http://react.cs.uni-sb.de/mbt2006/ で見ることができるが，以下にリストアップした見出しからも想像されるように，きわめて実際的な内容で興味深かった：

Is the Industry Ready for Model-Based Testing?
The State of Testing Today
But Formality Intimidates Testers
And Big Model Intimidates Testers
And … "Modeling is hard".
A Great Industrial Technology should be …
Cheap, Easy to apply, and Measurable.
Factors in Favor of MBT: Time and Machines
What's Happening Now?
Where Could Models Take the Industry?
MBT is Entering the Mainstream
What Needs to Happen?
How Can You Help?

Robinson さんは，テスティング一般および MBT について，次の2つの Web Page も運営されている．興味のある方はぜひ覗かれるとよい：

http://www.geocities.com/model_based_testing/
http://www.geocities.com/harry_robinson_testing/

コーヒー・ブレイクの場所は新館1階のロビーなので，休憩時間にはまた階段を下りて戻らなければならない．インターネット接続用の PC は，そのロビーと旧館3階の部屋に設置されていた．日本語のニュースは見られるが，入力のローマ字・カナ漢字変換はできない．午前中の次のセッションの発表を2篇聴いたが，内容はあまり感心しなかった．1つはドイツ，もう1つはオランダの，いずれも組込みシステム開発の話だった．

昼食は，大学の裏手にあるナッシュ市場のファースト・シーフッドのチェーン・レストラン Nord See（北海）．そのあと,すぐ近くにあるセセッションの地下室でクリムトの壁画「ベートーベン・フリーズ」を

観る.

そのあとワークショップに戻るという野中さんと別れ，わたしは建築美術アカデミーのギャラリーへ．ここでは CHI 会議のとき，ディジタル・バウハウスの構築を目指した EU 協同プロジェクト「アトリエ」の成果発表レセプションに招かれた建物である．あのときは夜だったのでギャラリーは閉まっていた．以前から念願だったヒエロニムス・ボッシュの「最後の審判」の実物をようやく目にする．思ったほど大きな画面ではない．宗教的な寓意はともかくとして，化け物絵としてのできばえは，葛飾北斎の作品群に匹敵すると感じた．

夕方まで少し時間があったのでどこかでウィンナコーヒーでも飲もうかと思ったが，土曜日のせいか，中心街は大混雑．ホテルへ戻って mail の処理．夕食は，あまり遠出したくなかったので，ホテルの裏手にある上品なカフェ LUX へ．やたら背の高いウェイトレスのサービスで，白そして赤のワイン．野中さんはウサギの腰肉．わたしは血のソーセージ．夜中にサマータイムがはじまるというので，寝る前に時計の時刻を 1 時間進める．

26 日，日曜日．ワークショップの2日目．昨日とは違う道を歩き，マリアヒルファー通りの教会で日曜礼拝の光景をちょっと覗いてから大学へ向かう．途中の商店や食堂は，市場も含めてほとんど閉まっている．最初の招待講演は IBM イスラエル研究所の Alan Hartman さん．"Ten Years of MBT –A Sober Evaluation."というタイトルで，要するに失敗続きだったという経験談．ほとんど得るところがなかった．

ランチタイム．大学近くの食堂はほとんどお休みなので，地下鉄駅近くの公園に向かう．食事の前に，そばのカール教会に入ってみると，内部の修復工事用のエレベータで円天井のすぐ下まで上がれるようになっている．それに乗って天井のすぐ下まで行き，フレスコ画を間近に鑑賞して，さらに危なっかしい階段でドームの上まで行くことができる．高所恐怖症でもう足がすくんでいる野中さんをそこに残して，天辺まで上って街を一望した．

教会を出て，周辺で唯一開いていた公園のカフェで昼食．午後のプログラムはあまり面白くなさそうだったので，ホテルへ帰ってまた mail の処理．4月3日の SEA Forum の参加申し込みが何通か．そして ICSE2006 in 上海のポスター・セッションについての問い合わせ．まだ2ヶ月先の話なのだからそんなにあせらなくてもいいと思うのだが，やはりプレゼンターの方々は気になるのだろう．しかし，上海サイドの準備がもたもたしているので仕方がない．展示担当の委員にとりあえず質問を転送しておく．NTT データの山本修一郎さん宛てに mail で，5月末に予定している SEA Forum (システム・トラブル)にスピーカーをお願いする．

夕方，野中さんの帰りが遅かったのでどうしたのかと思ったら，市電でリングを一周してきたとのこと．「今日の最後の発表が岸田さん好みの哲学的な話でしたよ」という．それは，わたしもちょっと気になっていた論文で，中国での国際会議でいつもお会いしている UNU/IIST（マカオにある国連大学ソフトウェア研究所）の Chris Geroge さんが共著者に名前を出していた．かれが会場にきていないようだったので発表を聴かずに失礼したのだが，残念なことをした．あとで論文をよく読んでみようということで，とりあえず街に出る．

映画ファンの野中さんの「ぜひに」という希望に応えて，地下鉄でプラッター公園の観覧車に乗りに行く．映画「第三の男」でオーソン・ウェルズとジョセフ・コットンとの有名な対話シーンに使われた旧世紀の観覧車である．時代物だけに，現代の優雅なデザインとは違って，小ぶりの鉄道客車のような箱をぶらさげた不恰好なもの．スピードもときどき止まったりして，のんびりしている．

第2次大戦後のイギリス映画といえば，キャロル・リードとデイヴィッド・リーンの2人が代表的な監督だった．デイヴィッド・リーンは，カラー映画になってから「旅情」や「戦場に架ける橋」，そして「アラビアのロレンス」など，いくつもの話題作を撮っているが，わたしが熱心に洋画を観ていた1950

年代前半から半ばにかけては，圧倒的にキャロル・リードのほうが油がのっていたように思う．日本での公開の順序でいうと，「邪魔者は殺せ」(1951)，「第三の男」(1952)，「落ちた偶像」(1953)，「文化果つるところ」(1953) といった具合．わたしはその中では，ジョセフ・コンラッドの小説 "Outcast of the Islands"（島々の除け者）を題材にした 4 番目の映画が一番気に入っていた．原作をわざわざ丸善で買って読んだりした．デイヴィッド・リーンの当時の作品としては，後の「旅情」の原型ともいうべき「逢引き」が有名だが，日本公開は1948年．まだ子どもだったわたしには少し早すぎた．飛行機マニアのはしくれとして1953年公開の「超音ジェット機」は封切りで観たが，老優ラルフ・リチャードソンの名演技以外，ほとんど記憶に残っていない．

「第三の男」は，キャロル・リードの他のいくつかの映画と同じく，グレアム・グリーンの小説を下敷きにしている．いい映画だったとは思うが，第2次大戦直後の社会状況の表現としては，ヴォルフガング・ボルヒェルト（戸口の外で）やハインリッヒ・ベル（アダムよお前はどこにいた），あるいはハンス・エリッヒ・ノサック（死神とのインタビュー）などの小説（アメリカ兵捕虜としてドレスデンの収容所にいたカート・ヴォネガットの小説「母なる夜」を加えてもよい）のほうが，空襲で廃墟と化した東京の風景とオーバーラップして，はるかに強烈な印象をわたしの心に与えた．しかし一般には，こうした戦後ドイツ文学はほとんど関心を持たれなかったようだ．ウィーンの夜景を眼下に眺めながら，あらためて，戦後 60 年の時間の経過を感じる．

帰りは地上にしようと，ちょうどやってきた5番の市街電車に飛び乗ると，電車は中心街の外を大回りして，結局，西駅まで戻ってしまった．駅のコンビニで飲み物とおつまみを買い，地下鉄で一旦ホテルへ．夕食はホテル前の市電通りを少し歩いて見つけた「七つ星」という名前の地ビール・レストラン．看板通り7種類のビールがある．今夜もまたソーセージをつまみにビールを飲み，部屋に戻ったのは10時過ぎ．玉井先生がすでにチェックインされていたので，ブダペストで買ってきた Tokai ワインを開けて乾杯．

翌 27 日（月曜日）から ETAPS の本会議が始まったのだが，わたしは無関係．野中さんも次の SPIN Workshop まで暇だというので，2 人とも休息日．上海の杉田義明さんから ICSE に参加する人たちのための割安ホテルの交渉がまとまったという mail が届いたので，日本からの参加予定者の人たちに案内を発信し，2人でシェーンブルン宮殿へ出かける．急に天候が回復して，散策にはもってこいの陽気になった．宮殿内部の部屋を一回りして庭園へ．後に「ルードウィヒ/神々の黄昏」や「暗殺者のメロディ」で演技派女優に成長したロミー・シュナイダーがまだカワイコちゃんだったころに主演した映画「プリンセス・シシー」を思い出したのだが，野中さんは観ていないらしい（ひとつの世代間ギャップ）．ゲオルグ・トラークルやシュテファン・ゲオルゲがこの庭園を舞台にした美しい抒情詩を何篇か書いているのだが，これもまた，ドイツ文学愛好家以外にはほとんど縁のない話である．

気温は急上昇して暑いくらいなのだが，庭園のはずれの池にはまだ氷が張っている．その先の丘に登り，カフェで休息．帰りはまた適当な市電に飛び乗り，途中で地下鉄に乗り換えて中心街へ．もうドイツ風の料理には飽きたのでイタリアン・カフェの路上のテーブルでスパゲティと白ワインのランチ．月曜日なのでほとんどの美術館や博物館は閉館だが，カフェ近くのユダヤ博物館が開いていたので入り，これまで名前を知らなかった2人のユダヤ人音楽家についての展示を見て感心する．そのあと，マリア・テレジアほか歴代のオーストリア帝国皇帝たちの棺が並べられた地下室へ，こういう場所を観光施設にする神経はどうもよくわからないが，大げさな彫刻群で飾られた巨大な鋼鉄の棺の列が歴史の重みを感じさせることはたしかである．

まだ夕方まで時間があったので，有名なチョコレート・ショップ Demer へ．店の奥のテーブルに座って，ガラス越しに菓子職人さんたちの仕事振りを眺めながらコーヒーを飲んで，1日の散策終

了．夕食は，会議を終えて帰ってこられた玉井先生，そして東京農工大の吉澤康文先生と一緒に夕食．ホテル近くのレストランをいくつか見てまわったが，静かに話ができるところがよいだろうということで，また一昨日行ったLUX へ．ウェイトレスは同じ背高お姐さん．吉澤先生が日立時代に手がけておられたコンパイラのテストの話などで楽しい時間を過ごす．

28 日，火曜日．ワークショップの中休みを利用してブダペストへ1泊旅行にでかけるという野中さんを見送ったあと，ゆっくりホテルをチェックアウトして，地下鉄で中央駅へ．そこで荷物をチェックインし，空港行きの快速電車に乗る．春休みを利用したツアー客が多いためか，フライトはほぼ満席．機内食のサービスが終わったあと，MBT Workshop の論文集を開いて，一昨日の野中さんの報告で宿題になっていた次の Paper を読むことにする：

Bernhard K., Aichernig and Chris George:
"When Specification-Based Testing Fails,"

第 1 著者の Aichernig さんは，オーストリアのグラーツ工科大学の先生で，いまは UNU/IIST に Visiting Researcher として赴任している人らしい．「仕様にもとづくテスティングが失敗するとき」というタイトルは別に何の変哲もない．思わず，ローレンス・ブロックの傑作ミステリ小説 "When the Sacred Ginmill Closes"（邦訳書名は「聖なる酒場の挽歌」，二見文庫）を思い出してしまった．しかし，Abstract には，次に訳したように，かなり過激な文章が書いてある：

> Armando Haeberer は，科学哲学の成果をソフトウェア・テスティングに翻訳する上で大きな貢献をなしとげた．かれは，理論や仮説から導かれる観測結果にもとづいて理論の正しさを検証する標準的な方法論の限界がソフトウェア・テスティングの場合にも存在するということを示したのである．こうしてかれは，テスト・ケースの生成に関する Gaudel の古典的アプローチを強く批判し，それに代わるものとして，Glymor のブートストラップ・アプローチにもとづく方法論を提示した．しかしながら，その後，この提案をめぐって当然沸き起こるものと予想された議論は起こらなかった．Model-Based Testing のアプローチが，かれの批判した方法にしたがっていることからすれば，これは，まったく奇妙な現象だといわざるを得ない．この論文は，Haeberer の発見を新しい視点から見直すことによって,議論を再開させることを目指す．

これでは，発表のあとの Q&A が侃々諤々の状況になったことは想像に難くない．故 Haeberer 先生の名前は，何年か前に Zhou Caochen 先生に続いて UNU/IIST の 3 代目の所長に就任され，しかしすぐに亡くなられたということだけしか知らなかった．Reference を見ると，ここでいわれている論文は 2001 年の ICSE in Toronto で発表されたものだ．わたしが ACM SIGSOFT から Distinguished Service Award を貰ったときの会議である．日本に帰ったらすぐ Proceedings を調べなければと思う．

また，Haeberer 先生に批判された古典的テスト・ケース生成アプローチというのは，ICSE のプログラム委員会で何度か顔をあわせたことのある Marie-Claude Gaudel 女史の有名な論文 "Testing Can Be Formal, Too" を指しているようだ．これも，Lee Osterweil さんの "Software Process Is Software, Too" をもじったタイトルだという程度の興味しか持っていなかったので，探し出して内容をよく読んでみなければいけない．

さて，さらに本文の Introduction に読み進むと Rudolf Carnap や Carl Popper の名前が登場してくる．理論を現実に照らしてその検証が可能かどうかという問題を論じるのだから，当然といえば当然

だが，これはちょっと困ったことになった．大学時代に詩の同人雑誌を一緒にやっていた友人が科学史・科学哲学の専攻で，故・大森荘蔵先生に師事していた．その関係で十数年前に先生に会社で講演をお願いし，その夜新宿で飲み明かしたとき（大森先生は知る人ぞ知る大酒豪），弟子入りを希望したのだが，「きみがもう少し若ければ OK と引き受けてもよいけれど，もう世俗の塵に染まってしまったからダメだね」と，あっさり断られた記憶がある．

わたしはもともと論理的にものごとを考えるのが苦手なのだ．自分のそうした欠点をなおすためにプログラミングという職業を選んではみたものの，本格的に科学哲学の世界に足を踏み入れることをこれまで意図的に避けてきた．しかし，こうなったら仕方がない．帰国したらいろいろな文献を集めてもう一度勉強しなければいけないと思いながら，ひとまずまた論文をバッグにしまって眠りについた．

4. Model-Based ?

翌朝 29 日（火曜日）の 8 時半過ぎに成田着．とりあえず家に荷物を置いて，その夜は SEA 幹事会．翌日，会社の図書室で ICSE の Proceedings や Springer LNCS シリーズから次の2つの論文を探し出してコピーし，翌週月曜日（4/3）の SEA Forum の発表スライドを準備しながら拾い読みすることにした．

AM Haeberer and TSE Maibaum: "Scientific Rigour, an Answer to a Pragmatic Question: a Linguistic Framework for Software Engineering"
(Proceedings of ICSE-2001 in Toronto)

Marie-Claude GAUDEL: "Testing Can Be Formal, Too"
(Proceedings of TAPSOFT'95, Springer LNCS 915)

この論文探し作業の思わぬ副産物は，TAPSOFT'95 の Proceedings の中に，以前からわたしが気にかけていた Christiane Floyd 女史の講演原稿が入っていたことである．

Floyd さんの名前を初めて知ったのは，1988 年春，ACM/SIGSOFT のニュースレターに掲載された論文を読んだときだった．ちょうど産学協同の国際プロジェクト SDA で，ソフトウェア設計プロセスの支援環境をどうすればよいかについて，連続ワークショップ形式の討論を行っていたところだった．「ソフトウェア工学の世界でプロダクト指向からプロセス指向へというパラダイム・シフトが起こっている」というかの女の鋭い指摘は，わたしにとってひとつの衝撃だった，

TAPSOFT という会議は，もともと 1985 年に，当時ベルリン工科大学にいたかの女が，同僚の Hartmut Ehrig 先生と一緒に，ソフトウェア工学における実践派とフォーマル派との間の交流のための場を作ることを意図して始めたものらしい．したがって，その10周年を記念する会議に，かの女が基調講演者の 1 人として招かれたようだ．この講演で，かの女は次のように述べている：

> 形式手法派の人たちはもっぱら Computation についての議論に熱中し，一方で実践派は Computing とくにその人間的・社会的側面を重視している．そして仲間同士だけでかたまり，お互い相手のグループを軽蔑または無視しているようだ．これではいけない．ソフトウェア工学の進歩にとっては，2つのグループの間の継続的な対話や討論がどうしても必要なのだ．TAPSOFT その他の技術的な会議が，そういう意味でまだ十分に機能していないのは残念なことである．

たしかにその通りだと，(フォーマル手法にも興味を持っている)実践派の 1 人として，わたしも考える．もうひとつこの講演原稿で面白いと思ったのは：

> 北欧のソフトウェア技術者・研究者たちの多くは，オブジェクト指向プログラミングの元祖といわれる SIMULA63 を用いて，社会システムのシミュレーション・プログラムを書くことから勉強を始めているので，ソフトウェア工学の社会的側面 (Computing Process に関することがら)に強い．

という指摘であった．そういえば，わたしが最初に目にした SIGSOFT ニュースレターでのかの女の論文は，もともと「コンピュータと民主主義：スカンジナビアからの挑戦」という本から転載されたものだった．

さて，問題の Haeberer-Maibaum の論文「数学的厳密性，実践的質問へのひとつの回答：ソフトウェア工学のための言語論的フレームワーク」だが，その Abstract は次の通り：

> ソフトウェア工学における数学の役割についての議論は，これまで20年ほどの間，日常的に続けられてきたが，ほとんど何の成果もあげていない．現在では，ソフトウェア構築という仕事が持つ「直観的」な性質について多くのことが論じられ，グラフィック・デザインや(伝統的な)建築などとの(誤った)類推が語られている．そこでいわれている結論はといえば，数学は必要以上に贅沢であって，他の諸分野におけると同様に，日常の実践には不要だというものである．われわれは20世紀の「科学哲学」の分野で開発されてきたさまざまなアイデアを見直すことによって，そうした考え方に異議を唱えたいと思う．われわれは，これらのアイデアが活用できる理由を示し，ソフトウェア工学上の多くの中心的なアイデアを形式化し組織化するためのフレームワーク (哲学者カルナップがいう意味でのフレームワーク)を提示して，単純な「直観」の思想を反駁したいと考える．科学哲学によってもたらされる諸概念はソフトウェア工学の理論や実践を正しく評価するために必要な科学的・工学的なフレームワークをわれわれに提供してくれる．

まるでフォーマル派一刀流の大太刀を真向上段に振りかぶったような筆致で，このままでは「恐れいりました」と膝を折るしかないように感じられるが，そこをぐっと我慢して，駆け足で本文に目を通して行く．

第1章は，ソフトウェア工学において数学が果たす役割について，さまざまなエピソードを交えたイントロダクション．そして第2章はカルナップほかの哲学者によって発展してきた科学哲学的意味論のソフトウェア工学問題への適用例．そのなかで，仕様からテストケースを自動生成するという Gaudel 女史のフォーマル・アプローチが批判されている．ただし，詳細な議論は別の論文[*]を参照ということで，簡単にしか述べられていない．そして，第3章では一転して，ソフトウェア工学のための認識論的基盤の構築を詳しく論じている．

[*] この論文は Haeberer 先生がミュンヘン大学で1999年に書かれたもの：
M.V.Cengarle and A.Haeberer:
"Towards an Epistemology-Based Methodology for Verification and Validation Testing".

http://www.pst.informatik.uni-muenchen.de/personen/cengarle/proceedings.ps
ダウンロードしてみたら70ページもあるので，読むのはちょっと保留．

この第3章に載せられたダイアグラムには，いささか興味を惹かれるものがある，「ソフトウェア開発の認識論的フレームワーク」と名づけられたその図は：

(1) Reality (仮定されたユーザ・ニーズ)
(2) Modified Reality (ソフトウェア・プロダクト)
(3) Requirement Specification
(4) Design Specification
(5) Refined Design Specification

という5つの頂点を持つ5角形の形をしている．そして，それらの5つの頂点のうちの3つを結ぶ10個の3角形がソフトウェア開発プロセスを構成する10のサブプロセスを表すようになっている．ソフトウェア・プロセスの分析に関して，この認識論図式は，これまでになかった新しい発見をもたらすのではないかという予感がする．

第2章で否定した Gaudel の手法に代わるものとして，Glymor のブートストラップ・アプローチにもとづく手法というのが紹介されているのだが，これは初めて聞く名前である．ネットサーフして調べてみると，CMU の科学哲学の先生だということがわかった．Reference に出ていた論文 "Hypothetico-deductivism is Hopeless" (なんとなくセンセーショナルなタイトル)をダウンロードしてみたが，面倒な記号論理式が並んでいるので，これも読むのは保留．

MBT ワークショップの Aichernig-George の論文の序論には次のように書かれている：

> 形式的仕様(モデル)にもとづくテスティングは，Gaudel が "Testing can be formal too" とその論文で宣言して以来，アクティブな研究領域になった．それは仕様がブラックボックステスティングを可能にするからであった．仕様は，テストの結果を予測する上での福音であり，その構造を分析することによってテスト・ケースの生成が可能になる．現在の仕様記述言語のほとんどにテストケース生成機能がついているのは，そうした研究の影響である(たとえば VDM, Z, B, CSP, Lotus, あるいは RAISE など)．しかし，そうした研究は，認識論に根ざした深い基礎を考慮することを忘れていた．
> Armando Haeberer は，20世紀科学哲学の成果がソフトウェア・テスティングと密接に関連していることを発見した．特に，有限な観測結果によって科学的仮説を検証することについての議論が，形式的モデルからテスト・ケースを生成する問題にそのまま当てはまることに，かれは気づいたのである．Carnap は，Statement View と呼ばれる論理的フレームワークを考案した．考えるべき問題の1つは，仮説を記述するための理論的言語が無限の要素を含んでおり，一方，観測の世界は有限であることである．有限の観測によって無限の理論をいかにして検証できるか．この問題をめぐっての Carnap 対 Popper の論争はよく知られている．そして，「テスト・ケースはバグの存在を証明することはできる．しかし不在証明にはなりえない」という Dijkstra の有名なことばは，基本的に Popper の側に立つ発言であった．しかしながら，Haeberer その他の人びとは，コンピュータ科学のコミュニティに対して，もう1つの重要な研究成果を報告した．Clarke Glymor が，仮説およびその背後にある理論から期待される観測結果を導き出すという古典的なアプローチには基本的な問題

点があり，この仮説演繹的(Hypothtico-deductive)アプローチの欠陥を補おうとするすべての試みが失敗に終わるということを，証明してみせたのである．
これは，われわれにとってきわめて重大な警告であろう．なぜなら，現在行われているほとんどの Model-Based Testing のアプローチは，Gaudel が代数的仕様記述について行った仕事も含めて，本質的に仮説演繹的だからである．

仕様は有限な記述であり，対象システムは無限の要素を含む現実世界に属しているということは，ソフトウェア開発に携わる者ならだれでも知っている(しかし，時に忘れてしまうこともある)事実であろう．無限の要素から成る現実世界をわれわれが認識するための手段としての言語は，可能性としては無限の文を作り出す機能を持っている．しかし，現実に書かれる仕様は，可能な文の無限集合のうちの有限な部分集合でしかない．無限に変化しうる可能性を秘めた対象システムを有限な記述でとらえるために，われわれはシステム分析にさいして，ある前提条件を仮定し，いくつかの要素を切り捨てるしかない．

昨年の ICSM に併設して開かれた Software Evolution ワークショップの基調講演で，M.M.Lehman 先生はシステム進化におけるこの問題を指摘している．当初のシステム設計が行われる時点でとりあえず切り捨てても問題のなかった要素が，時間の経過にともなってシステムにとって重要な影響をもたらすものに変わってくることがあり，それがいくつかの大規模システムのトラブルの原因になる．このシステム・メインテナンスにおける Assumption Management の問題は，テスティングとは別の次元のことがらであるが，その背後に認識論(Epistemology)あるいは存在論(Ontology)の影がちらついているという点では同じカテゴリに属している．

さて，Haeberer は，眼前の問題点を克服するために，Glymor のブートストラップ・テスティングのアプローチを提案したのだという．ブートストラップとは何か．要するに，仕様から生成されたテストケースを用いてプログラムの正しさを証明することは本質的に無理だから，他のやりかたを考えようというのだろうが，その代替手法とはどんなものなのだろうか．Aichernig-George の論文に出ている簡単な解説は次の通り:

ブートストラップ技法は(仕様->テストとは)逆向きに働く．それはまずテスト結果から出発して，そこから演繹的に理論上の仮説を導き出す．たとえば，ソーティングのプログラムが入力データをきちんと並べ替えたものを出力したとすれば，その事実からわれわれは，その出力が入力の並べ替えであるという仮説を確認できる．こうしたアプローチは自然科学において日常的に行われている．たとえば，惑星の軌道の観測から，まず，それらの軌道が太陽を焦点とする楕円であるというケプラーの仮説が導き出され，さらに後年，同じ観測結果からニュートンの万有引力の仮説が導き出されている．

なるほど，そんなものかも知れないが，まだ何となくピンとこない．やはり，邦訳されているカルナップの論文をベースに，Haeberer 先生の論文をじっくり読んでみる必要がありそうだ．しかし，これに続く次のパラグラフが興味深い:

Haeberer が指摘したように， 観測された事実が仮説と矛盾した場合，通常は仮説が捨てられるのである．それには長い時間がかかるが，上に上げた例の場合でいえば，われわれは 最終的に観測された事実を受け入れ，真円軌道のエレガントさや単純さに対するわれわれの好みにもかかわらず，プトレマイオスの太陽系理論を捨てることにしたであ

った. しかし, ソフトウェア・テスティングの場合には, そうした矛盾が生じると, われわれは, 前提となった理論すなわち要求仕様を確認することができなかったプログラムのほうを捨ててしまう. そこで Haeberer は, ある種の逆ブートストラップ・アプローチともいうべき方法を提案した:われわれは理論を検証したいのではなく, プログラムを検証したいのである.

うーむ, なにやら面白そうではあるのだが, いまはいろいろ雑用で忙しいので, これ以上この問題に首を突っ込んで記号論理式と格闘している暇がないのは残念. ICSE2006 で上海のホテルに1週間滞在している間にはなんとか時間が取れるかもしれない. フォーマル・メソッドが専門の先生方も何人かご一緒するので, 論文のわからない個所も質問できるだろう.

ところで, ここで Floyd 女史の指摘にしたがって Computation の視点から Computing の世界に足場を移して, ユーザ・プロセスという視点からこのテスティングを眺めると, 様子が少し違ってくるように思われる.

4月3日に行われた SEA Forum で「ソフトウェアが, 仮説すなわち要求仕様(テスト仕様)通りには動かないという現実の結果が得られたとき, どうして仮説(仕様)のほうを捨てないでソフトウェアを直そうとするのか?」という素朴な質問を, なかば悪戯心でわたしが投げたとき, フロアの何人かの方から「われわれはそういう世界に住んでいるのだから仕方がない」という応答をいただいた. それは, ソフトウェア開発者の立場からは当然の答えだったと思う.

しかし, 視点をちょっと変えて, 開発プロセスとは切り離されたユーザの立場で, 自分の仕事のプロセスのツールとしてのソフトウェアを考えると, それが(自分の望んでいた)仕様通りには動かないという仮説を否定するたしかなテスト結果が得られたとき(つまりあらかじめ考えていた仮設が否定されたとき), その仮説(すなわち仕様)を捨てて, テスト結果から逆に再構成するかたちで(現実の)仕様を作り直し, それを前提として, そのソフトウェア・ツールを自分の作業プロセスのなかにどう組み入れるかを考えるほうが, 実際的なのではないだろうか.

ユーザ・サイドの Computing Process すなわちソフトウェアの開発/委託/購入そして運用までを含めたプロセスにおける受け入れテストの意味合いは, そんなものではないかと思われる. もちろん, テスト結果が示す現実の証拠を開発者に突きつけて, ソフトウェアがあらかじめ与えられた仕様を満足するよう改訂要求を出すことは当然であるが, 現実のビジネス・プロセスは, その改良が終わるのを待ってはいられないことが多い.

実際われわれが市販の COTS を使って仕事をするさいの Computing Process はそうなっている(いろいろ文句をいいながら Windows を使っているプロセスがその典型). 欧米では, 数年前から COTS-Based Software Engineering と題された会議が毎年開かれているようだが, そうした問題も論議されているのだろうかと, ふと考えた.

COTS といえば, CMU/SEI の Web Site に, 毛沢東語録のスタイルを模した1篇のドキュメントが載っている.

> Quotations from Chairman David
> — A Little Red Book Truths to Enlighten and Guide on the Long March Towards COTS Revolution
> http://www.sei.cmu.edu/publications/documents/99.reports/lrb/little-red-book.html

筆者の David Carney さんがどんな人なのかは知らないが, なかなか洒落たセンスの読み物である. このスタイルを真似ていろいろな分野の入門書が書けるような気がする.

SEA Forum April

Waterfall Model 再考(最高？)

主催: ソフトウェア技術者協会(SEA)

W.Royce が, いわゆる"Waterfall Model"論文を 1970 年に発表してから, 早いものでもう 35 年が過ぎました. その後いくつものソフトウェア開発プロセスモデルが発表されていますが, このモデルは, 現在でも, 特に大規模プロジェクトにおいては, 基本となる開発モデルとして使われているようです.

しかしながら, 原論文に提示されているフィードバック付の逐次型開発プロセスモデルは, その後の環境変化に応じてさまざまな解釈がなされ, 多様なプロセス実装が行われています.

今回は, この"Waterfall Model"の概念的フレームワークを, もう一度原論文に立ち戻って理解するとともに, その後の開発プロセスモデルの変遷を, 最近のアジャイルプロセスまで含めて展望し, ソフトウェア開発の現状と将来につい討論するフォーラムを企画しました.

『W.Royce がその論文で提起した大規模ソフトウェア開発管理の問題に対する提案を, その後の技術発展やコンセプトの進化をふまえて検討し, ソフトウェア工学 40 年の歴史で整理された見解を踏まえて, 今そして明日何をなすべきかについて考え行動するかを, 参加者のみなさんと一緒に考えてみたいと思います. (落水)』

春一番の楽しい討論が期待されます. 多数の方々の参加をお待ちしています.

参考:W.Royce の原論文(スキャンイメージです)
http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

********** 開 催 要 領 **********

1. 日時: 4月3日(月) 16:00～20:30
2. プログラム
   - 15:30 - 16:00 受付
   - 16:00 - 17:30 講演:"Waterfall Model"再考
     落水 浩一郎 (北陸先端科学技術大学院大学)
   - 17:30 - 17:45 とりあえずの質疑応答
   - 17:45 - 18:15 休憩
   - 18:15 - 20:30 討論: ソフトウェア開発の現状と将来を考える
     コーディネータ:田中 一夫 (SEA 代表幹事)
     パネリスト: 平鍋 健児 (チェンジビジョン)
     伊藤 昌夫 (ニルソフトウェア)
     落水 浩一郎 (北陸先端科学技術大学院大学)
     岸田 孝一 (SRA 先端技術研究所)
3. 会場: 全国情報サービス産業厚生年金基金(JJK)会館 2F B会議室
4. 定員: ５０名

WATERFALL Model 再考

ソフトウェア開発管理とソフトウェア開発方法論の融合について

北陸先端科学技術大学院大学　情報科学研究科
落水　浩一郎

本フォーラムの目的は、W.Royceが論文[1]で提起した大規模ソフトウェア開発管理に関する問題を、ソフトウェア工学40年の歴史（その後の技術発展やコンセプトの進化）をふまえつつ、再度検討することにより、今、明日、何をすべきかについて考えることであると理解する。まず、W.Royceの論文内容を著者なりに要約する。つぎに、1970年のW.Royceの論文を基準点として考え、彼が提起した問題に対して、その後どのような解決が試みられたかを整理する。最後に著者なりの問題提起を行い、フォーラムにおける議論のきっかけの一つにしたいと思う。

1．W. Royce による"Managing the Development of Large Software Systems(1970)"の紹介

彼の論文の主旨は、大規模ソフトウェアの開発管理をどのようにして実現するかという提案と、そのような開発手段を採用したときに起こる問題をどのように解決するかという2点である。まず、図１に彼が論文で導いた結論を示す（図１）。

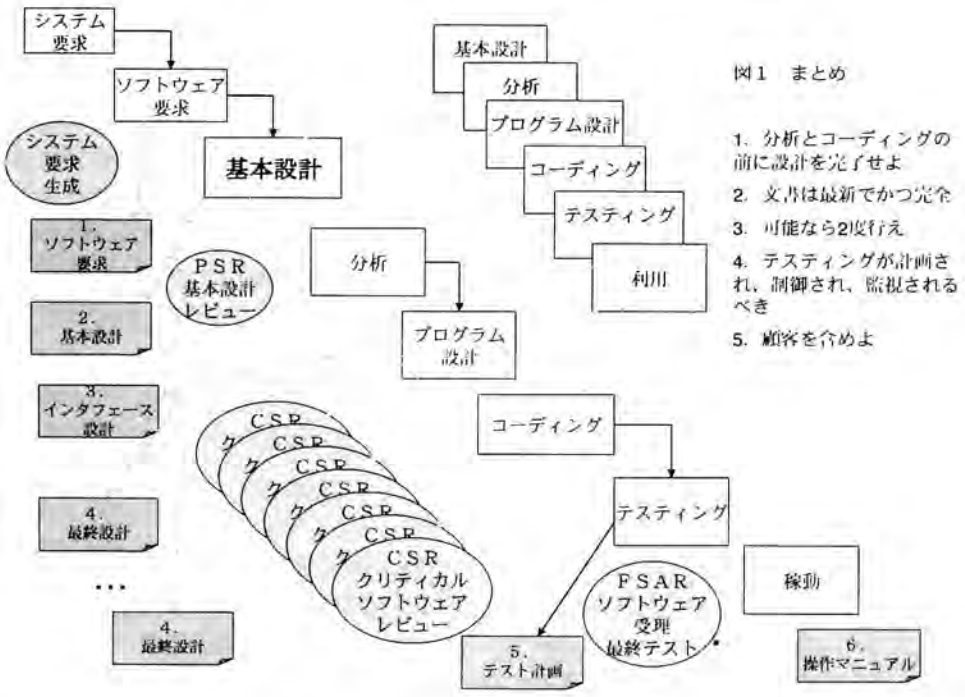


図１にいたる、彼の思考の展開を論文の流れに沿って追ってみよう。

(1) プログラムの規模や複雑さにかかわらず、どのようなプログラムをつくるべきかとい

う「分析」と、それをどのように「コーディング」するかというステップは、プログラマにとって本質的なものである。また、最終プロダクトに直接貢献する創造的なステップとして、顧客はよろこんで報酬を出す。しかし、この簡単なモデルは、労力が十分に少なく、最終プロダクトが開発者によって稼動される場合のみ（すなわち内部利用の場合のみ）有効である（図２）。

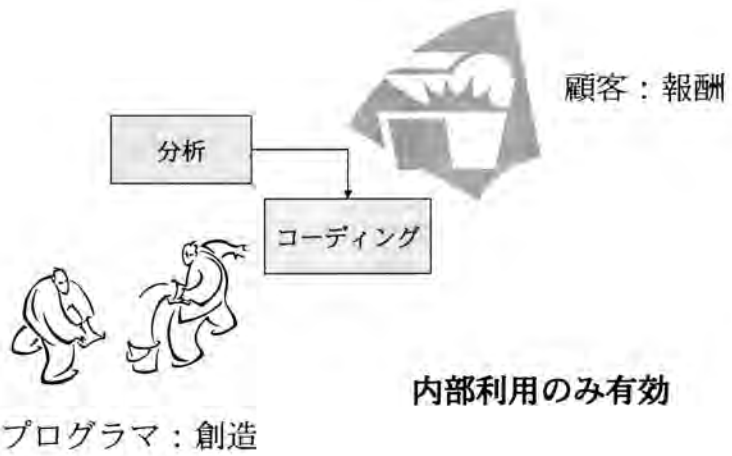


図２　内部利用のための小規模なプログラムを引き渡すための実現ステップ

(2) より規模の大きいシステムでは、この二つのステップのみではプロジェクトは失敗に終わるので、最終プロダクトに直接貢献しない多くの付加的な開発ステップが必要になる（訳注：最終プロダクトに直接に貢献しない付加的な開発ステップという表現の真意は不明である）。それらのステップはコスト増をまねく。また、顧客はあまり報酬を支払いたがらないし、開発担当者はそれらのステップを実行したがらない。管理の第一義の機能は、これらのコンセプトの必要性を双方のグループに納得させ、また、開発担当者には従うことを強制することである。それらのステップは、順に「システム要求定義」、「ソフトウェア要求定義」、「分析」「プログラム設計」、「コーディング」、「テスト」となる。「システム要求定義」、「ソフトウェア要求定義」、「プログラム設計」、「テスティング」の付加的ステップは、実行される手段が「分析」や「コーディング」とは異なるので、分離して処理される（図３　実行順序を示す矢印を除く部分）。

(3) ７つのステップに、「詳細化のために、順次実行する」、「一つ前のフェーズに後戻りする」という実行順序に関する情報を付加する（図３）。すなわち、「各ステップの進行につれて、設計内容はより詳細化される」、「一つ前のステップへの後戻りは起こるが、数ステップ前への後戻りはめったに起こらない」という仮定を置く。この考え方のよい

点は、設計が進行するにつれ、変更が管理可能な範囲に限定されることである。すなわち、予期せぬ困難が発生した時、どこに戻るかを指示している、安定した、移動するベースラインが近距離にある。

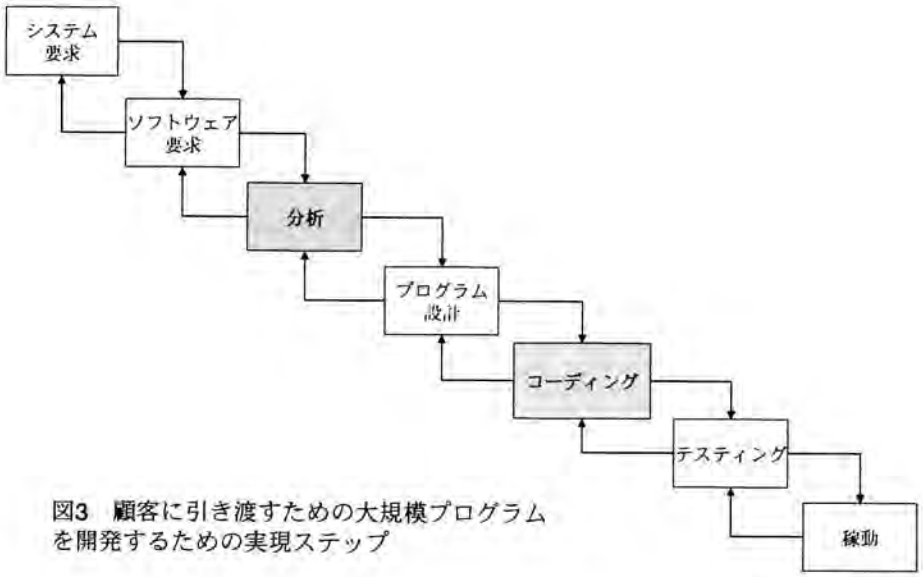


図3　顧客に引き渡すための大規模プログラムを開発するための実現ステップ

(4) 上記仮定をそのまま実現するのは危険であり、失敗をまねく（訳注：順次的なステップの実行、すなわち、ベースラインの移行が現実的な仮定ではないことを彼自身は認識していたことになる）。

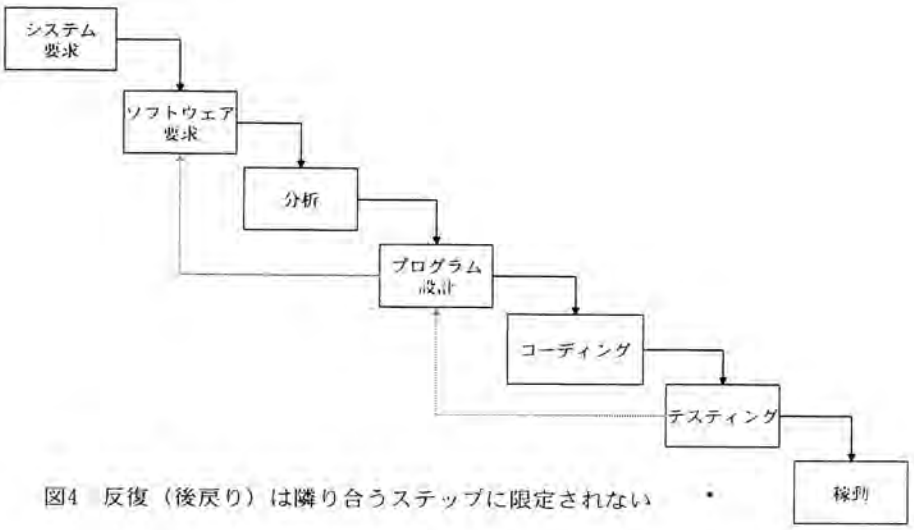


図4　反復（後戻り）は隣り合うステップに限定されない

何故なら、開発サイクルの最後にくるテスティングでは、タイミング、記憶容量、入出力転送などに関して、分析段階とは異なる現象が認知される。これらの現象を厳密に解析することはできない。さらに、これらの現象が様々な外部制約を充たすことに失敗した場合は、再設計が必要となる。必要とされる設計変更は破壊的なものであり、これはさらに、設計に根拠を与えている要求に違反することになる。この結果、「テスティング」、「プログラム設計」、「ソフトウェア要求」という大幅な手戻りを生じ、要求が変更される場合も、本質的な設計変更がある場合も、100％のオーバーランをコストやスケジュールに引き起こしてしまう（図４）。

(5) それにもかかわらずこれらのアプローチは基本的には健全なものであり、これらの開発上のリスクを除去するような以下の５つの特徴をつけ加えることで対応できる。

(5－1)「基本設計」フェーズの導入

「ソフトウェア要求定義」と「分析」の間に基本設計を挿入する（図５）。

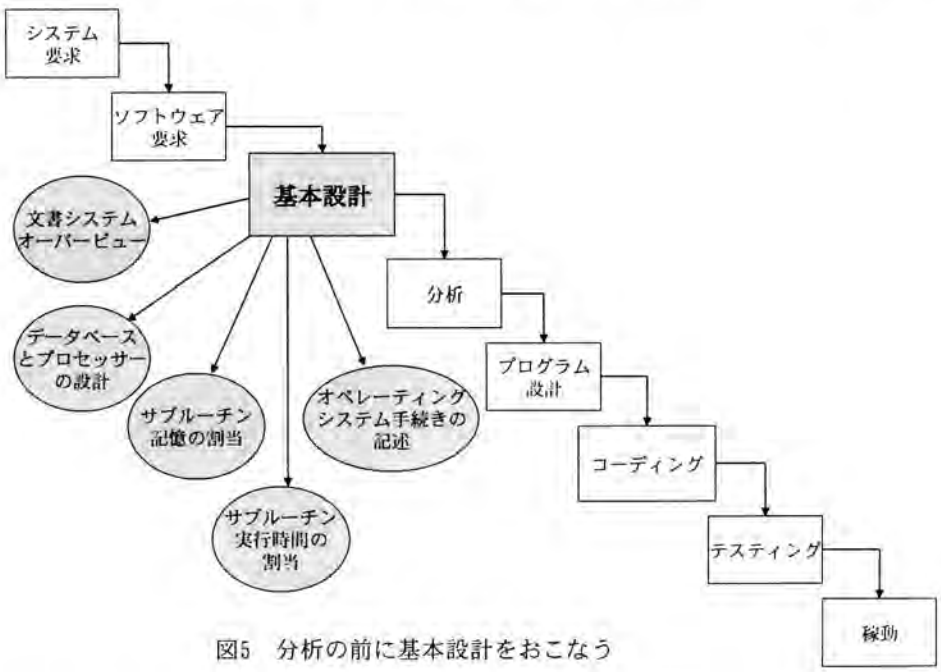


図5　分析の前に基本設計をおこなう

これは、「要求は存在するが分析結果が存在しないという相対的真空状態のなかで設計することをプログラム設計者に強制する」という批判を生み出しうる。この批判はあたっているが一つポイントをはずしている。基本設計を行うことにより、プログラム設計者が記憶容量、タイミング、データの不安定さによっては失敗しないことを保証できる。なぜなら、次の分析フェーズで、プログラム設計者は記憶容量、タイミング、稼動に関する制約を解析の課題に加えることができるからである。すなわち、タイミング、記憶容量、稼働に関する制約を考慮した「分析」を可能にする（訳注：これが納得のいくレベルで実施できる技術はまだ開発されていないと思う）。基本設計フェーズは以下の３点からなる。

・ 基本設計プロセスを、分析者やプログラマとではなく、プログラム設計者と実施する
・ 状況が悪いというリスク下でもデータ処理モードを設計し、定義し、割り当てる： 処理と機能を割当、データベースを設計し、データベース処理を定義し、実行時間を割当、オペレーティングシステムとのインタフェースと処理モードを定義し、入出力処理を記述し、基本的な稼動手順を定義する。
・ 理解容易であり、情報に富み、かつ最新のオーバービュードキュメントを書く：すべての作業者はシステムについての基本的な理解を持つべきである。少なくとも一人は、オーバービュー文書を書き、システムについての深い理解を持つべきである。

(5－2) 設計結果を文書化する

ここでドキュメントをどの程度書くべきかという問題を提起する。「かなり書く必要がある」というのが著者の考えである。「かなり」とは、プログラマ、分析者、プログラム設計者が自身の工夫で書こうとするものより多いことを意味する。ソフトウェア開発を管理するための最初のルールは文書化要求を冷酷に課すことである。文書は受理できる水準にする必要がある。高水準の文書化なしにはソフトウェア開発管理は不可能である。5百万ドルのハードウェア装置には30ページの仕様書、5百万ドルのソフトウェアには1500ページの仕様書が必要である。何故そのように大量の文書が必要なのであろうか？

- 各設計者は、インタフェースをとるため、他の設計者とコミュニケーションする必要がある。また管理者、場合によっては顧客とコミュニケーションする必要がある。話し合いの記録だけでは決定事項を管理するのには不十分である。受理でき得る記述は設計者に明白な立場をとり、形ある完了の証拠を与えることを強制する。これにより90％シンドロームを回避できる。
- ソフトウェア開発の初期のフェーズでは、文書は仕様であり設計である。コーディングが始まるまでこれらの３つの言葉（文書、仕様、設計）は単一のものを示す。文書が悪ければ、設計も悪い。もし文書がまだ存在しなければ、設計もまだ存在しない。
- 良い文書の値打ちは、テスティングから稼動、再設計にいたる開発工程の下流で認識される。
  （ア）テスティングフェーズでは、良い文書は、管理者が担当者にプログラム中の誤りに集中することを可能にする。良い文書がない場合は、そのプログラムを理解している唯一の人間である誤りを犯した人によって、誤りは解析されることになる。
  （イ）稼動フェーズでは、良い文書はオペレータを稼動に専念させうる。良い文書がない場合には、それを書いた人が稼動させることになる。一般にプログラムを書いた人はそのオペレーションに関心がないので仕事を効果的に実施できない。
  （ウ）最初の稼動に引き続いて、システム改良の順番になったとき、良い文書は効果的な再設計、更新、改造などを可能にする。

図６に、各ステップのキーとなる文書化プランを示す。６つの文書が生成され、引渡し

時期に文書１、３、４、５、６が更新され最新化されることに注意して欲しい。

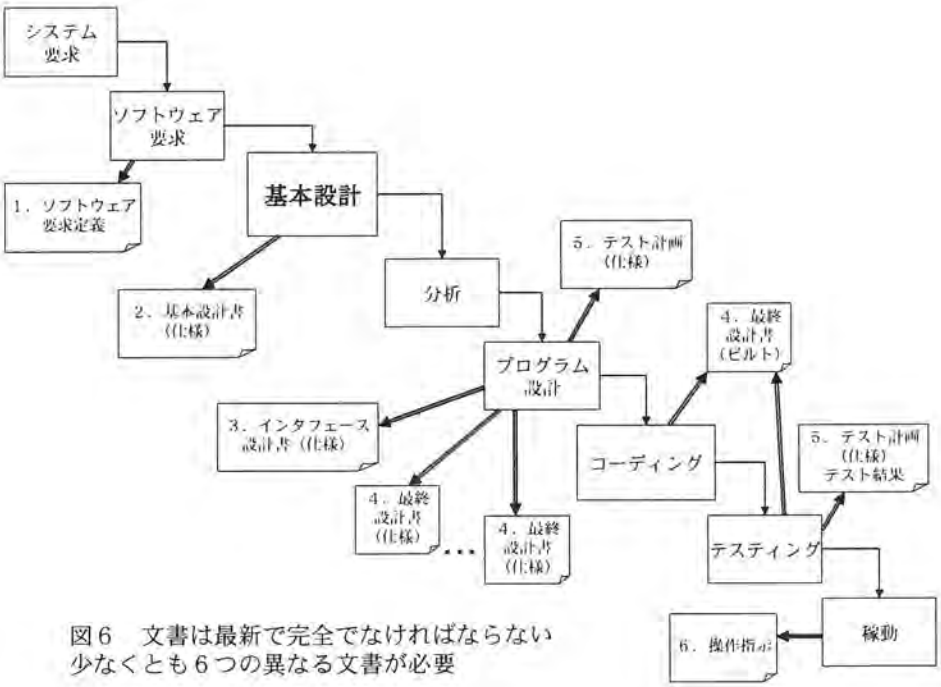


図６　文書は最新で完全でなければならない
少なくとも６つの異なる文書が必要

(5－3) 基本設計以下を２度行え

はじめてのシステムの場合、運用のために顧客に最終的に引き渡される版は第2版であるようにする。開発労力が30ヶ月の場合は10ヶ月、12ヶ月の場合は3ヶ月を開発のシミュレーションにあてる。設計における問題点を検知しモデル化せよ、代替案をモデル化せよ、不要な詳細を無視せよ、最終的には虫のないプログラムを達成せよ。この結果、タイミング、記憶容量その他の判断事項はある程度正確に理解できる。このように一度目は問題発見に力を注ぎ、2度目に解決策を組み込む（図7）。

(5－4) テストの計画と制御と監視

（マンパワーであれ、ＣＰＵ時間であれ、管理者の判断であれ）プロジェクト資源の最大の消費者はテストフェーズである。テストは、コストやスケジュールに関して、最大のリスクを持つフェーズである。それはスケジュールの最終点でバックアップする代替物がほとんど利用できないときに起こる。5－3までに述べた３つの推奨：分析やコーディングのまえに基本設計を行う；文書化を完全に行う；パイロットモデルを構築する；はすべて、テストフェーズに入る前に問題を発掘し解決することに狙いがあった。しかしながらこれらのことを行っていてもテストフェーズが残っており、そこでなされるべき重要なことがある。図８にテスティングに関する視点を示す。

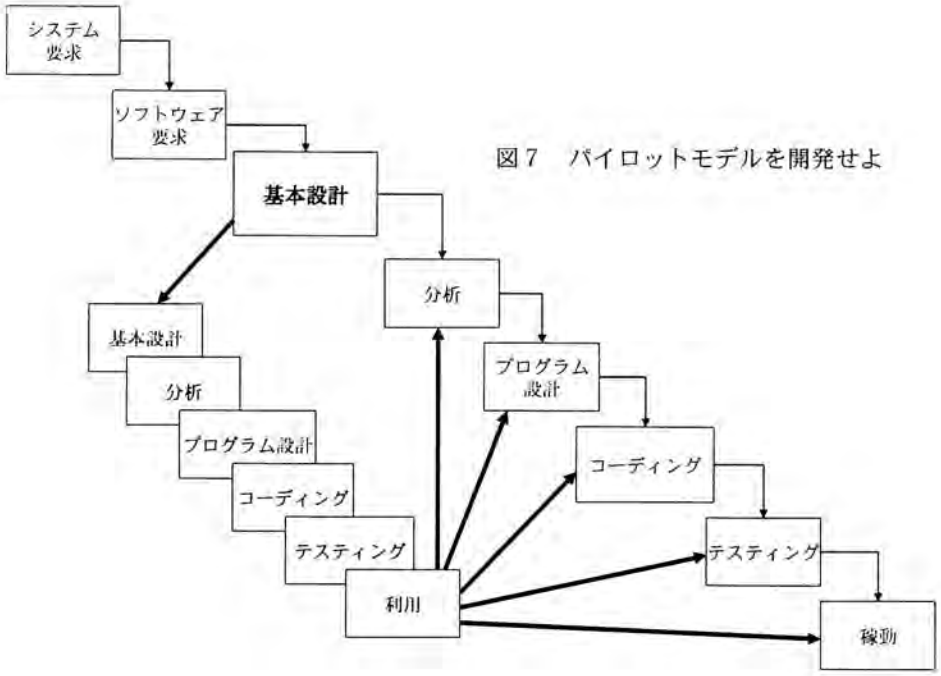


図7　パイロットモデルを開発せよ

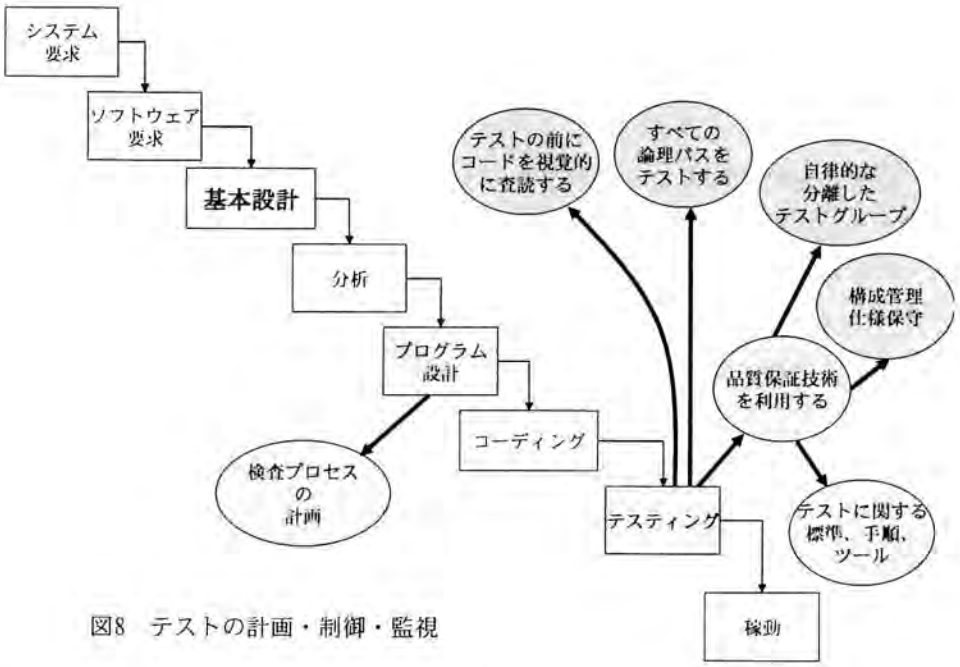


図8　テストの計画・制御・監視

テスト計画を立てるとき、以下の点を考慮することを奨める。

- テストプロセスの多くの部分は、テストの専門家によって最もうまく処理される。テスト専門家は必ずしもオリジナルな設計に寄与していなくてもよい。もし、設計者しかテストをできないことが問題になる場合は、文書化が不十分な兆候である。

- ほとんどの誤りは、視覚的な検査により容易に特定できる。すべての分析結果やソースコードは第3者の視覚的検査を受けるべきである。コストがかかりすぎるので、この目的で計算機を利用しないほうがよい。
- プログラム中のすべての論理パスをテストせよ。巨大で複雑なプログラムの場合、これは大変困難であり、ときには不可能な場合もある。しかし、カバレッジ100％の必要性を主張しておきたい。
- すべての単純な誤りを除去したのち（これがほとんどであるが）、点検のため、ソフトウェアをテストにまわせ。

(5－5) 顧客を含めよ

最終引渡しの前、早い時点で顧客に責任を持たせるため、顧客を開発に公式にふくめることが重要である。「要求定義」と「稼動」の間で契約者に自由な裁量を与えることはトラブルを招く。要求定義の後にくる、顧客を含めるべき３箇所を図９に示す。

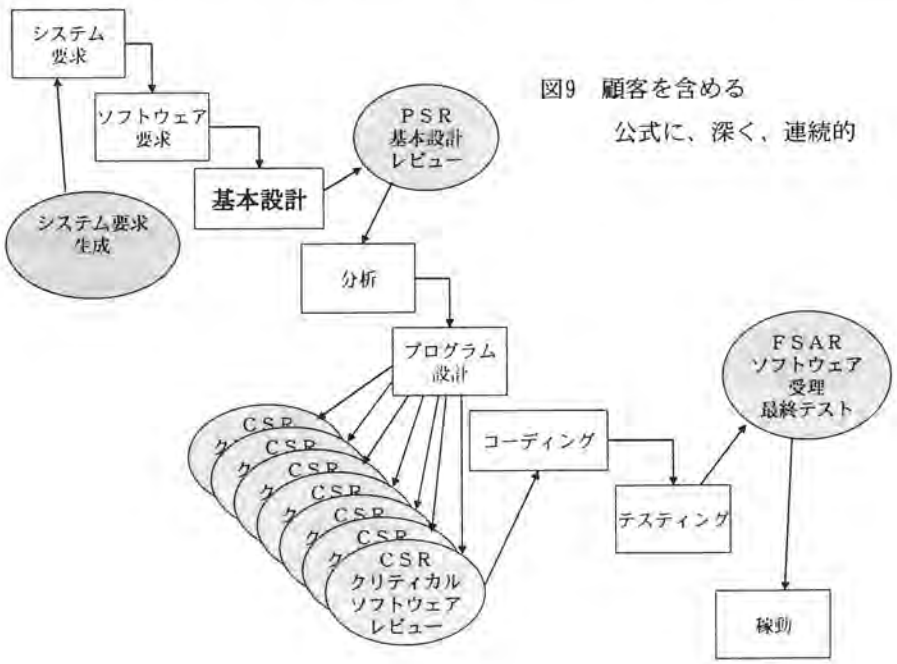
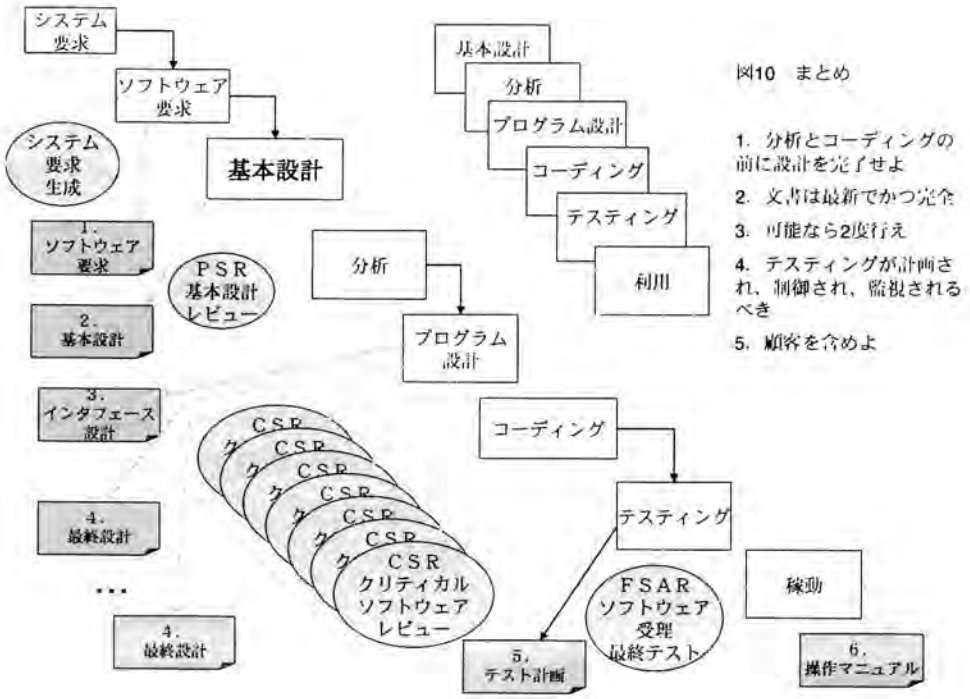


図9　顧客を含める
公式に、深く、連続的

（6）まとめ

大規模ソフトウェア開発の経験に基づいた以上述べてきた考察の結果として、冒頭（図１）に示したウオーターフォール・モデルを提案する（図10に再掲）。図10に、リスキーな開発プロセスを所望の製品を提供する開発プロセスに変換するのに必要な５つのステップをまとめ、それらを統合したものを示す。それぞれが幾分かのコスト増をまねくことを強調しておく。より簡単なプロセスを採用すれば余分なコストはかからないが、大規模なソフトウェア開発ではうまくいかない。



つまり、彼は1970年の時点で、あとで問題になるかなりのことを意識的（あるいは無意識に）予想し、それなりの対応策を提案していたことになる。彼の提案にはウォータフォール・モデルに限定されない、ソフトウェア開発が内蔵する様々な問題点が含まれている。彼が論文中で提案した解の是非も含めて、その後のソフトウェア工学の発展（試行錯誤？）の歴史をたどってみよう。

2．それ以降の技術発展

ソフトウェア開発方法論とプロジェクト管理は、ソフトウェア開発プロジェクトの成功にとって、車の両輪の関係にある。それらをまとめるものがソフトウェア開発パラダイムである。ソフトウェア開発方法論の研究では、ソフトウェアの望ましい「構造」を模索しつつ、それらの構造を作りこむ手段を検討する。追求されてきた「構造」の属性として、正しさの確認容易性、変更波及の局所化、作業分担の容易性、再利用容易性、進化容易性などがあげられる。一方、プロジェクト管理は、方法論を遂行するプロジェクトチームのコスト、スケジュール、品質などに関して、それらを計画する、見積もる、資源を割当てる、プロジェクト状況を監視し制御するための手法を追求する。

2.1 プログラミング方法論（1970年代前半）

構造化プログラミング（1972）、情報隠蔽原理（1972）、ＪＳＰ法（1975）により、全体の作業を、独立性を保証する作業単位へ分割する手段、作成中のプログラムが所望の動

作を行うことを確認する手段の提供などに力点を置いたプログラミング方法論が発展した。

2.2 設計方法論（1970年代後半）

複合設計（1978）、構造化設計（1979）などにより、変更の波及を局所化できるようなソフトウェア構造とその作りこみの手段が検討された。

2.3 要求定義（1970年代後半）

「問題を定義する手段なしに、解を構築する手段を追求しても意味がない」という反省のもとに、70年代の後半（1977）に、ウォータフォール・モデルの出発点となる要求定義フェーズの形式化が課題となり、機能要求や性能要求を記述し解析するための要求定義法に関する研究が始まった。要求定義法のみによっては、要求仕様を確定できないことがすぐに明らかになった。

2.4 定量的プロジェクト管理（1970年代後半から1980年代前半）

コスト見積り（ＣＯＣＯＭＯ，1981）、危険要因の検出（MaCaveのＶ、1976，HalsteadのＥ、1977）、テスト時期の打ち切り（信頼性曲線、1982）などの定量的プロジェクト管理に関する試みが開始された。この段階で、ソフトウェア開発パラダイムとプロジェクト管理の関係、ソフトウェア開発パラダイムとソフトウェア開発方法論の関係が検討されはじめたが、ソフトウェア開発方法論とプロジェクト管理の関係が検討されなかったのは問題であると考える。

2.5 システム工学の導入（1980年代中期から後半）

この時期にウォータフォール・モデルの様々な改良が試みられた。文献[２]の内容を引用することにより以下に説明する。

１．ウォーター・フォールモデルの派生型はそのほとんどがシステム工学の原則から生み出されたものである。システム工学は複雑なシステムのコンポーネント設計、仕様化、実装の検証を主な目的としたものであり、元々はスペース・シャトルのような航空宇宙システムやNATOの指揮制御システムといった、大規模システム構築時に発生する問題を解決するために開発されたものである。上記のようなシステムは、あまりにも規模が大きいため、開発そのものを複数の企業に分散させねばならない。まず、ある企業が元請業者となり、システム全体の設計と統合を担当し、そして、元請業者のシステム・エンジニアが集まってシステム要求を分析し、その後、複数のサブシステムへと分割してシステム設計を行なう。

２．システム工学的アプローチは：システム要求を定義する；システム要求をサブシステムに配分する；詳細なコンポーネントを定義する；コンポーネントを検証する、サブシステムを検証する；といったＶ字型のプロセスになる。

3．システム工学アプローチは一見すると、大規模なソフトウェア・システムを開発するアプローチとして適切であるように思える。しかし、システム工学的アプローチが基としている前提は、ソフトウェア開発ではうまく適用できないものである。その前提とは、要求が安定していること、実装に先立ってインタフェースを適切に仕様化できることである。

4．ウォータフォール・アプローチによって、要求管理、アーキテクチャと設計、実装、テスト、運用、保守といった、プロジェクトの基本的なアクティビティが洗い出された。しかし、要求ドキュメントや設計ドキュメントを凍結できないというソフトウェア開発特有の性質のため、近代的なソフトウェア・プロセスにおける詳細なアクティビティや成果物は、システム工学プロジェクトのそれとは大きく異なっている。

2.6 反復型ウォータ・フォールモデル（1980年代前半）

これについても文献[2]の内容を以下に引用する。

W.Royce や B.Boehm といったソフトウェア開発管理の創始者達は、かなり初期から古典的なウォーターフォール・プロセスの問題に気付いていた。1980 年代に彼らは、ミニウォータフォール、あるいは Boehm スパイラルモデル(1988)といった、顧客とのやり取りを改善できる、ウォーター・フォールの改良版を考案している。

1．改良版では、開発サイクルを短い期間に分割し、それぞれの期間でウォーターフォール・プロセスを実行し、要求のレビュー、ドキュメントの更新、コードの開発を行う。ある反復での成果物は次の反復の入力になる。顧客レビューと仕様の更新といったプロセスを、連続しておこなうことによって、プロジェクトのリスク管理を可能にする。

2．古典的なウォータ・フォール管理アプローチにおけるいくつかのコンセプトは、適切に変更すれば、ソフトウェア・マネージャにとって有益なものとなる。中でも、プロジェクト計画と進捗管理の原則は常に重要である。特に、作業分割の構造、コストとスケジュールの変化、出来高といったものを理解することは有効である。アクティビティに基づいた計画や厳格なタスク分割は、プロジェクトの本当の性質を反映したものでないために、各アクティビティにかかった時間を評価してもプロジェクトの実状すら把握することはできない（著者注：この言明を是とするならば、付録 B「PMBOK の内容」における「計画プロセス群」、「プロジェクトタイム管理」における様々の技法はあまり有効でないという結論になる)。

3．問題を早期に洞察すればするほど、それを修正できる可能性がましていくというのが大規模システム構築における基本原則である。修正コストというものは、プロジェクトの進捗に従って劇的に増加していく。統合とテストを最終段階で行うウォータフォール・アプローチは、統合段階に到達しなければ実際の作業状況を認識できないという問題がある。

2.7 プロトタイピング（1980年代前半）

Royceの論文ではライフサイクルでの「内なる繰り返し」が推奨されているが、1980年代初頭（1985）におこったプロトタイピングは、利用者世界を含めてサイクルを回す必要があるとの認識に基づいている。この時代のプロトタイピングはどちらかというとイテレーティブイン（反復）開発であり、プロジェクト状況の学習にともなうプロセスの改善を目指すインクリメンタル開発ではなかったように思える（文献[３]の５．３節を引用して、両者の定義を付録Aで説明する）。後者の場合、固定的な標準化は学習の効果を無視することになり望ましくないように思える。しかし、プロジェクト進行中に頻繁にその進め方を変更することはある種の混乱を招く。プロセス改訂の頻度をタイミングが問題である。

2.8 実行可能仕様（1980年代中期）

リアルタイムデータフロー(Ward 1986)、JSD(M. Jackson, 1983), PAISLey(P. Zave, 1986), PROTnet(Bruno, 1986), UI(Wasserman, 1986)など上流工程をフォーマルに書く技術が発展した。これらは、後に上流工程の各種プロダクトを表現する言語の発展につながった。しかし、コーディングを労働集約型として蔑視する動きを助長した問題点もあるように思える。

2.9 フォーマルメソッド（1980年代中期）

上流の文書化の質が下流のプロダクトの質に影響を与えるというW.Royceの見解に対して、上流の文書を数学的記法で厳密に記述すれば、人間が解析困難であるさまざまなソフトウェア性質の検証、証明、自動化が可能になり、検査フェーズのコスト減や最終プロダクトの品質につながるという認識が、1980年代の中頃におこり、モデル検査につながっている

2.10 プロセスプログラミング（1980年代後半）

1987年、L.OsterWeilはソフトウェア開発プロセスを記述するプロセスプログラミング言語を開発することにより、プロセス改善の土台にできると考えたがあまり成功しなかった。

2.11 プロセス改善（1990年代初頭）

CMM(1989). 1990年代にはいっておこり、PMBOX[４]その他の体系化として実を結びつつある、ソフトウェアプロセスの改善に関するフレームワークの確立とベストプラクティスの整備は開発管理手法の発展をうながした。付録Bに、文献[４]における表３－４５に、その他の部分の記述内容を、主に、ツールと技法に注目して埋めこんだ表を示す。ところで私見であるが、CMMはレベル３以上に実質上の意味があると考える。すなわち、コスト、スケジュール、信頼性達成などの状況を、ファンクションポイント（FP）にもとづく様々なメジャー[５]（生産性に関する指標：FPあたりの費用、IT部門の生産性、開発速度、ユーザが利用する機能と開発された機能；　品質に関する指標：機能要求の規模、完備性、変更率、欠陥除去率、欠陥密度、テストケースの網羅率、文書量；　経済性に関する指標：

FP当りの費用、補修費率、ポートフォリオ資産価値； 保守性に関する指標：保守性、信頼性、要員配置、成長率、ポートフォリオ規模、バックファイア値、安定度）や、Caper Jonesが定義した意味でのベースラインやソフトウェアベンチマーク[6]などで（定性的に問題を把握し）、定量的に証拠だてることによりテクニカルプロジェクトマネージメントが可能になる。

2.12 CASEツール（1990年代初頭）

「上流工程で、早期に、コストのかかる誤りを除去する」などの目的のもとに、StP（１９８８）、PCTE（１９９３）などのCASEツールや統合環境が開発された。これについては興味深い意見がある[7]。文献[7]の内容は非常に興味深い、付録Cに文献[7]の要約引用を示すが、ぜひ[7]原本を一読されることを薦める。

真実５． 開発ツールの大げさな宣伝はたちの悪い伝染病みたいなものだ。ツールや技法の大部分は、生産性や信頼性を５－３５パーセント程度、改善するにすぎない。

真実６． 新しいツールや技法を学習すると、最初は生産性や品質が下がる。実際に効果がでるのは、ツールや技法が完全に身についてからだ（習熟曲線）。オブジェクト指向は表面的には３ヶ月、自由に使いこなすには３年かかる。ツールや技法の導入を効果的にするには、（１）効果が現実的である（２）気長に効果を待てる場合に限る。

真実７． ソフトウェア技術者は、ツールの話が大好きである。いくつものツールを買い、評価もしているが、開発で実際に使った人はほとんどいない。コンパイラ、デバッガ、エディタ、リンクローダ（無意識）、構成管理ツールで十分である。

2.13 アーキテクチャ中心開発

W.Royceの「基本設計フェーズ」の導入の意図は、1990年代中頃におこったアーキテクチャパターンにより加速されアーキテクチャを評価し、検証する技術が進んでいる。しかし、必ずしも十分な成果が得られているとはいえない。

2.14 オブジェクト指向

再利用のメリットについての議論はともかく、プログラムの構成単位（オブジェクト）に再利用性や変更容易性に関する機構を内蔵させたこと、要求仕様、設計仕様、プログラム、計算機内部の動作間の対応関係をシンプルにし、反復を容易にしたことは見逃せない。

2.15 UML

上流、中流における各種文書を記述するための言語を標準化しようとする努力は無視できない。

2.16 アジャイル

Royceの論文にある、「テストの計画と制御と監視」、「顧客を含めよ」などの問題提起に形を与えた。また、このアプローチはウォータフォールに対して正反対のアプローチである。[役に立たない要求文書や設計文書のことなど、すべて忘れてしまおう。こういったものが正しくなることはないのだ。ただひたすらにコードのビルドを行おう]。このようなラピッドプロトタイピングのアプローチには利点と欠点がある[2]。

- 利点：チームの生産性が向上する。顧客とのやりとりが建設的なものになる。
- 欠点：プロジェクトを収束させることが難しくなる。スケジュールや予算が見積もりにくくなる。大規模ソフトウェア開発には適さない。成果物がプロトタイプでしかない危険性がある。

3．まとめと課題

いわずもがなであるが、W.Royceの話がすべての原点であるなどというつもりはない。[2]によればウォータフォールモデルの欠点は以下の通りである。

- ウォーターフォール・プロセスが機能しない理由を理解しよう。ウォーターフォール・アプローチには、すべてのプロジェクトに適用可能な、重要な原則が含まれている。
  ① 開発におけるアクティビティを洗い出す。
  ② アクティビティ毎に工数計画を立案する。
  ③ マイルストーンを用いてアクティビティの進捗を管理する。
- このような原則は、マネージメントがしっかりしているプロジェクトであれば、どのようなプロジェクトにも存在する要素である。このため、クリティカル・パス分析や細密マイルストーン(inchstone)管理といった、成熟した古典的プロジェクト管理テクニックが採用できると考えるのは自然なことである。
- このアプローチの問題として、**まず、ソフトウェア開発プロジェクトは建築プロジェクトほど簡単に計画、評価できない**と言う点をあげることができる。橋の塗装が半分完了したというのは、誰の目にも明らかだが、コードの半分がいつ完了したのかを知ることは難しい。また、橋の各部分を塗装する時間は簡単に見積もることができるが、コーディングやデバッグにかかる時間など、最終的なコードの規模が判らなければ誰にも判らない。
- 精密な進捗見積りと完了時間を要求することで、こういった不確実性に取り組もうとするマネージャもいる。しかし、これはマネージャに対して嘘をつくことを開発者に強要することになる。
- **2つ目の問題は、あるアクティビティが完了しないと次のアクティビティが始まらないという、アクティビィティの直列化によってひきおこされる。**
- アクティブティが完了するということは、成果物が完全なものとなったことを意

味する。橋の塗装の場合は簡単にわかるだろうが、要求文書が完成したかどうかなど、確信を持って答えられるはずもない。

- ソフトウェア開発のアクティビティを直列化しようとした場合、たいていのプロジェクトは遅かれ早かれ失敗することだろう。興味深いことに規律が厳しいプロジェクトほど早い段階で失敗する傾向にある。要求定義の完了や設計仕様文書の完成といった、初期のマイルストーンに到達する前に失敗してしまう。ドキュメントがレビューされる度に新たな問題が持ち上がり、疑念が生じ、不整合が発見され、誰にも答えられない質問がでてくる。それに対して、マネージメントは、品質を力説し、規格に固執することで、規律を守ろうとする。彼等の要求が満足されることはなく、最後には多くの資金を投じた役に立たない成果物だけがのこることになる。
- 早い段階での失敗よりも巷でよく見かけるのが、遅い段階での失敗である。この場合、プロジェクトは終了間際までうまく進んでいるように見受けられる。ドキュメントは完成し、レビューも済み、前半のマイルストーンも完了したかのように見える。しかし、現実的には、日程だけが消化されており、内容は基準に満たないものであることにみな感づいているはずである。

物事の起源の一つを振り返り、それを基準点にして、これまでの研究、技術開発の世界を体系的に把握し、それをふまえて今やるべきことを考えることは大事な作業である。いくつかの課題を整理してみる。

1. 方法論については、変更や検査・検証、再利用が容易な構造をどのようにしてソフトウェア構造につくりこんでいくかと理論と技術の開発に主眼がおかれてきたと思う。
2. プロジェクト管理については、コスト見積り、スケジュール管理、品質保証などの各種メトリックスとそれを活用するためのベストプラクティスが整備されてきた。
3. 双方の研究が最近では乖離しているように思える。
4. 温故知新。W.Royceの初期の考察から学べる最大のポイントは，ソフトウェア開発方法論とプロジェクト管理技術を融合してとらえるという姿勢であろう。
5. 融合の手段についてはいくつかの視点がある。
   (ア)いくつかの問題を早期に洞察すればするほど、それを修正できる可能性がましていくというのが大規模システム構築における基本原則である(マレー・カンター)。
   (イ)プロジェクト開始時には、向かっている目標がわからない。インクリメンタルな開発は、一度開発サイクル全体を経験させ、新しく発見した知識をすぐ使えるようにする。すべてを知っていると思っていても、プロジェクトには突発事項が待ち構えている。インクリメントによって、そのような突発事項に早く気付くことができる。突発事項は何かということに早く気付くと、作業方法を変更する機会

が得られる。各インクリメント境界は、行っていることすべてを改善する機会である（アリスター・コーバーン）。

(ウ)アセスメントとは、ソフトウェアプロジェクトが設置され維持される方法を検討するための、公式的かつ構造化された手段である。定性的データに基づくことが多い。ベンチマークやベースラインと相補的に利用する。定性的データによって原因を理解し、定量的データでそれを正当化する（ケーパージョーンズ）

これらの指摘が示唆していることは以下の点であると思う。

（1） ソフトウェア開発プロジェクトの特徴は不安定である

（2） しかもその不安定さは、プロジェクト依存である

（3） プロジェクト状況を把握にするにはプロジェクトに関する「学習」が必須である

ソフトウェア開発方法論とプロジェクト管理をどのように融合できるのであろうか。その一つの視点は「開発管理とはプロジェクト状況の学習である」というとらえ方ではないだろうか。ソフトウェア開発方法論は、変更容易性、検査容易性、再利用容易性などの、ソフトウェアが持つべき望ましい性質をつくりこむ技術である。ところで、ドメインの特性、規模、新規開発か経験済みか、それに割り当てることのできる資源はどのくらいか、担当者の技術レベルはなど、ソフトウェア開発活動を特徴づけるパラメータがある。ソフトウェア開発が進行するにつれて、これらの内、見積もりや計画段階で読みきれなかったことがらが顕在化してくる（状況）、プロジェクト管理の一つの重要な活動として、これらの状況を知る（学習）ことがあげられる（プロジェクト監視）。 問題を定性的に把握しそれを定量的に証拠だてる。 開発手段とそれを実行するための諸元は、プロジェクト毎にことなるので 、開発手段とプロジェクト管理の技術をうまく結合するには、特定の開発手段（例えば、インクリメンタル開発）やその実行環境に依存する議論が必要なのではなかろうか。その接点として、プロジェクトの状況を学習する手段の検討が挙げられる。学習に必要なデータとその利用法（フィードバック法）はひとつの接着剤候補であるように思える。

フィードバックをかけるタイミングはいつかという問題がある。次期プロジェクトで学習の成果をフィードバックするというのがごく自然な発想であるが、私見では、状況はプロジェクト毎に異なり、効果がないことになる。インクリメンタル開発では、「一回のインクリメント毎にプロセスを改善せよ」と指針がアリスター・コーバーンによって与えられている。確かに同じプロジェクト内でのフィードバックは学習の効果を期待できる。しかし、物事の進め方（標準）を頻繁に変更することは混乱をまねきやすい。さじ加減を検討する必要がある。

4．おわりに

ソフトウェア工学の成果が少しでも世の中の状況を改善しうるためには、問題点を産学

協同で真正面から取り組む必要がある。できることのみをやっていたのでは、我々の問題はいつまでたっても解決しない。ソフトウェア開発管理に関する要素技術はかなり成熟してきているように思える。それらを実践し、さらなる改良を積み重ねる努力が必要である。

文献

[1] W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques", Proc. of IEEE WESCON, pp.1-9, 1970 (Proc. of 9th ICSE, pp328-338, 1987 に再掲)

[2] マレー・カンター著、村上雅章訳、「ソフトウェア開発管理の要」、ピアソン・エデュケーション、2002.

[3] アリスター・コーバーン著、長瀬嘉秀、今野睦監訳、「アジャイルプロジェクト管理」、ピアソン・エデゥケーション、2002.

[4]「プロジェクトマネージメント知識体系ガイド第３版（PMBOK ガイド）」、Project Management Institute, Inc. 2004.

[5] デービッド・ガーマス、デービッド・ヘロン著、小泉、中村、向井訳、児玉監修、「ファンクションポイントの計測と分析」、ピアソン・エデュケーション、2002.

[6] Caper Jones, "Software Assessments, Benchmarks, and Best Practices", Addison-Wesley Information Technology Series, 2000.

[7] ロバート・L・グラス著、山浦恒央訳、「ソフトウェア開発５５の真実と１０のウソ」、日経 BP 社、2004.

付録 A　インクリメントとイテレーション（文献[3]より引用）

インクリメントは開発プロセスを修正または改善できるようにする。インクリメンタルな開発のメリットは以下の通りである。

1. 教育：プロジェクト開始時には、向かっている目標がわからない。インクリメンタルな開発は、一度開発サイクル全体を経験させ、新しく発見した知識をすぐ使えるようにする。
2. 無知の無は何かを発見する：すべてを知っていると思っていても、プロジェクトには突発事項が待ち構えている。インクリメントによって、そのような突発事項に早く気付くことができる。
3. 作業方法を修正する：突発事項は何かということに早く気付くと、作業方法を変更する機会が得られる。各インクリメント境界は、行っていることすべてを改善する機会である。

イテレーションは、システムの品質を修正または改善できるようにする。イテレーティブな開発は、システムの各部分を改訂し、改善する時間をどのように取るかという、やり直しのスケジュールを立てる戦略である。イテレーティブな開発はインクリメンタルな開

発を前提にしてはいないが、同時に行うととても効果的である。「イテレーティブな開発」という言葉自体は、納品物を改訂する量とタイミングをしめしていない。要求を改訂するか、設計だけを改訂するか、改訂を別のインクリメントとして扱うか、インクリメント内に組み込むかは、選択の余地が大きい。以下はイテレーティブな開発のメリットである。

1. リスク削減
2. 品質向上
3. コミュニケーション改善
4. ユーザが使用できるものを納品していることの保証

V－Wステージングは、検証V（要求・設計・テスト・出荷）－学習－検証V（要求・設計・テスト）（全体としてW）からなり、インクリメンタルな開発とイテレーティブな、スパイラル、プロトタイプ戦略を含む一般的な戦略である。V－Wステージングは、プロジェクトマネージャに以下の作業を行う方法を提供する。

- スパイラル開発の螺旋をまっすぐにし、アクティビティがプロジェクトスケジュールに対して一直線になるようにする。
- 恒常的進捗という考えを犠牲にせずに、プロトタイプとイテレーションを用意する。
- 「フェーズ納品物」ではなく、納品された機能に基づいて、効果的な進捗表示を導き出す。
- フェーズ納品物ではなく、納品されたインクリメントで下請を管理する。
- 禅を学習せずに「ゲシュタルトラウンドトリップ（システム全体を経験すると、プロジェクトが進む方法に関して「そうか」という感情が得られる）」の価値を得る。

プロジェクトインクリメント

- 一回目のインクリメントの目的
  - ソフトウェアのアーキテクチャを確立する
  - プロジェクトスポンサーとユーザにフィードバックを届ける
  - プログラミングと設計の良いパターンを開発する
  - おばけを発見し、無知の知を学習する
- 二回目のインクリメント
  - 一回目のインクリメントで発見された主な間違いを修正する
  - ソフトウェアアーキテクチャを確立、または調整する
  - 役立つプロセスとチームを作成する
  - よい開発習慣を決める
  - ユーザに多くの機能を納品する
- N回目のインクリメントまでに、成功の習慣とパターンを得ている。注意すべきリスクが二つある。眠ってしまうこと、および拡張しすぎてしまうこと。

付録 B　PMBOX　第 3 版の内容

プロジェクト管理プロセス群 / 知識エリアのプロセス	立上げプロセス群:新しいプロジェクトやプロジェクトフェーズを開始するための公式な認可を支援する	計画プロセス群：目標を定め、それを洗練し、プロジェクトが取り組むべき目標とスコープを達成するために必要な一連の活動を計画する	実行プロセス群：プロジェクト管理計画書を実行するために、人やその他の資源を統合する	監視コントロールプロセス群：潜在的な問題をタイムリーに識別できるように、プロジェクトの実行を監視。必要に応じて是正処置を講じられるように、プロジェクトの実行を制御	終結プロセス群：プロダクト、サービス、所産を正式に受け入れ、プロジェクト（またはフェーズ）を公式に終了する
プロジェクト管理統合：プロジェクト管理の様々な要素を統合するプロセス	**プロジェクト憲章作成**(プロジェクト選定手法。プロジェクト管理（PM）方法論。PM 情報システム（IS）、専門家の判断) **プロジェクトスコープ記述書（暫定）**高レベルの記述	**プロジェクト管理計画書作成**（PM 方法論。PMIS（構成管理、変更管理）。専門家の判断）プロジェクト管理計画書は、プロジェクトの実行、監視コントロールおよび終結の方法を規定する。その内容は、プロジェクトの適用分野と複雑さによって異なる。それは、統合変更管理プロセスを通じて更新され、改訂が行われる。PM 計画書作成プロセスは、すべての補助の計画書の定義、統合、調整等を行い、プロジェクト管理計画書とするために必要な活動からなる。	**プロジェクト実行の指揮・管理**（PM 方法論。PMIS）プロジェクトスコープ記述書に規定された作業を達成するために、計画したプロジェクトアクティビティの実行を指揮し、プロジェクト内に存在するさまざまな技術上と組織上のインタフェースの管理をおこなう	**プロジェクト作業の監視コントロール**（PM 方法論。PMIS。プロジェクトのパーフォマンスの測定と予測を行うアーンド・バリュー法。専門家の判断）。**統合変更管理**　承認・却下、承認済み変更のベースラインへの組込み	**プロジェクト終結**　事務終了手順。契約終了手順
プロジェクトスコープ管理：プロジェクトの成功に必要な作業を過不足なく含めることを確実にするために必要なプロセス		**スコープ計画**（専門家の判断、テンプレート・書式・標準）**スコープ定義**（プロダクト分析（システム分析/工学、価値工学価値分析、機能分析）。代替案識別(ブレーンストーミング、水平思考)。専門家の判断。ステークホルダー分析）**WBS 作成**（テンプレート。要素分解）		**スコープ検証**（検査）**スコープコントロール**（変更管理システム。差異分析。再計画。構成管理システム）	
プロジェクトタイム管理：プロジェクトを所定の時期に完了させるために必要なプロセス		**アクティビティ定義**（要素分解。テンプレート。直近の作業は下位レベルまで詳細に、遠い先は上位レベルだけ WBS を定めるローリング・ウェーブ計画法;専門家の判断;計画構成要素）。**アクティビティ順序設定**（プレシデンスダイアグラム法、アロー・ダイアグラム法、スケジュールネットワークテンプレート、依存関係の決定（強制、任意、外部依存関係）　**アクティビティ資源見積り**（専門家の判断。代替案分析。公開見積りのデータ。プロジェクト管理ソフトウェア。ボトムアップ見積り）　**アクティビティ所要期間見積り**（専門家の判断、類推見積り、係数見積り、三点見積もり、予備設定分析）、**スケジュール作成**（スケジュールネットワーク分析、クリティカルパス法、スケジュール短縮、what-if シナリオ分析、資源平準化、クリティカルチェーン法、プロジェクト管理ソフトウェア、カレンダーの適用、リードとラグの調整、スケジュールモデル）		**スケジュールコントロール**（進捗報告、プロジェクトスケジュールを変更する際の手続きを規定したスケジュール変更管理システム。スケジュールの差異に対して是正処置が必要か否かを決定するパーフォーマンス測定（スケジュール差異，スケジュール効率指数)。プロジェクト管理ソフトウェア。目標スケジュール日付を予測開始日や実開始日及び実終了日と比較する差異分析。バー・チャートによるスケジュール対比）	
プロジェクトコスト管理：プロジェクトを承認された予算内で完了させるために必要なプロセス		**コスト見積り**（類推見積り、資源単価、ボトムアップ見積り、係数見積り、プロジェクト管理ソフトウェア，ベンダー入札の分析、予備設定分析，品質コスト)、**コストの予算化**（コスト集約、予備設定分析，係数見積り、限度金による資金調達）		**コストコントロール**（コスト変更管理システム。アーンドバリュー法によるパフォーマンス測定分析，予測。プロジェクトのパフォーマン	

				スレビュー。プロジェクト管理ソフトウェア。差異管理）	
プロジェクト品質管理：プロジェクトが所定の目標を満たすことを確実にするためのプロセス		**品質計画**（費用便宜分析によるトレードオフスタディ。ベンチマークによる比較改善。実験計画法による開発中のプロダクトやプロセスの特定の変数に影響を与える要因の識別。品質コスト。品質計画ツール）	**品質保証**（品質計画のツールと技。品質監査。プロセス分析。品質管理のツールと技法）	**品質管理**（特性要因図、管理図、フローチャート化、ヒストグラム、パレート図、ラン・チャート、散布図、統計的サンプリング、検査、欠陥修正レビュー）	
プロジェクト人的資源管理プロジェクトチームを組織化し管理するためのプロセス		**人的資源計画**（チームメンバーの役割と責任を階層型、マトリックス型、テキスト型のいずれかの書式で記述する組織図と職位記述図。積極的相互交信、昼食会議、非公式の会議、取引上の会議などの人的資源のネットワーキング。要員、チーム、単位組織などの行動様式に関する情報を提供する組織論）	**プロジェクトチーム編成**（先行任命、交渉、調達、バーチャル・チーム）**プロジェクトチーム育成**（一般的な管理スキル、トレーニング、チーム形成活動、行動規範、コロケーション、表彰と報酬）	**プロジェクトチーム管理**（観察と会話、プロジェクトのパフォーマンスの評価、コンフリクト管理、課題ログ）	
プロジェクトコミュニケーション管理プロジェクト情報の生成・収集・配布・保管・廃棄をタイムリーかつ適切に行うために必要なプロセス		**コミュニケーション計画**（コミュニケーションに対する要求事項の分析によってプロジェクトステークホルダーの情報に関するニーズを収集する。情報の緊急度、技術の可用性、予想されるプロジェクト要員配置、プロジェクト期間、プロジェクト環境などのコミュニケーション技術）	**情報配布**（コミュニケーションスキル、情報収集システム、情報配布手法、教訓プロセス）	**実績報告**（情報提示ツール、パフォーマンス情報の収集・編集、状況レビュー会議、タイム報告システム、コスト報告システム）	
プロジェクトリスク管理プロジェクトのリスク管理を行うプロセス		**リスク管理計画**（計画会議と分析）、**リスク識別**（文書レビュー。情報収集技法（ブレーンストーミング、デルファイ法、インタビュー、根本原因の識別、強み、弱み、好機。脅威の視点から分析するSWOT分析）。チェックリスト分析。前提条件の妥当性分析。図解の技法（特性要因図、システムやプロセスのフローチャート、インフルエンス・ダイアグラム））　**定性的リスク分析**（リスク発生確率・影響度査定、発生確率・影響度マトリクス、リスクデータ品質査定、リスク区分、リスク緊急度査定）、**定量的リスク分析**（データ収集・表現技法。定量的リスク分析とモデル化の技法（感度分析、期待金額価値分析、デシジョン・ツリー分析））　**リスク対応計画**（マイナスのリスク（脅威）に対する戦略（回避、転嫁、軽減）。プラスのリスク（好機）に対する戦略（活用、共有、強化）。脅威・好機両面戦略。発生時対応戦略）		**リスクの監視コントロール**（定期的に実施するリスク再査定、識別したリスクとその根本原因に対するリスク対応策の有効性やリスク管理プロセスの有効性を調査し文書かするリスク監査。アーンドバリュー分析による差異・傾向分析。技術的実績の測定。プロジェクトの任意の時点で残存している予備が十分であるかどうかを、コンティンジェンシー予備の残存量を残存リスク量と比較する予備設定分析、状況確認会議）	
プロジェクト調達管理：プロダクト、サービス、所産の購入または取得のプロセス		**購入・取得計画**（内外製分析。専門家の判断。契約タイプ（定額契約または一括請負契約、実費償還契約（コストプラスフィーまたはコストプラスパーセンテージ、コストプラス固定フィー、コストプラスインセンティブフィー）、タイム・アンド・マテリアル契約））、**契約計画**（標準書式、専門家の判断）	**納入者回答依頼**（入札説明会、入札公告、適格納入者リストの作成）、**納入者選定**（重み付け法、独自見積り、スクリーニングシステム、契約交渉、納入者点数評価システム、専門家の判断、プロポーザル評価法）	**契約管理**（契約変更管理システム、購入者主催のパフォーマンスレビュー、検査および監査、実績報告、支払いシステム、クレーム管理、記録管理システム、情報技術）	**契約終結**（調達監査、記録管理システム）

マトリックスの格子点には、関連するプロセスと、そのプロセスを実行する際に利用できるツールと技法を記す。

付録 C　ロバート・L・グラス著、山浦恒央訳、「ソフトウェア開発５５の真実と１０のウソ」より要約引用

プロジェクト管理

人員

真実１．ソフトウェアの開発で最も重要なことは、プログラマが使うツールや技法でなく、プログラマ自身の質である。SEI-PCMM を調べよ。

真実２．プログラマ個人を分析した研究によると、最も優秀なプログラマは最悪に比べ、２８倍優れている。給与が能力を反映していないとすると、優秀なプログラマは、最高の掘り出し物と言える。

真実３．遅れているプロジェクトに人を追加すると、教育に時間をとられるため、もっと遅れる。

真実４．作業する環境は、プログラムの生産性や品質にきわめて大きく影響する。狭い場所に詰め込むな。

ツールと技法

真実５．開発ツールの大げさな宣伝はたちの悪い伝染病みたいなものだ。ツールや技法の大部分は、生産性や信頼性を５－３５パーセント程度、改善するにすぎない。

真実６．新しいツールや技法を学習すると、最初は生産性や品質が下がる。実際に効果がでるのは、ツールや技法が完全に身についてからだ（習熟曲線）。オブジェクト指向は表面的には３ヶ月、自由に使いこなすには３年かかる。ツールや技法の導入を効果的なにするには、（１）効果が現実的である（２）気長に効果を待てる場合に限る。

真実７．ソフトウェア技術者は、ツールの話が大好きである。いくつものツールを買い、評価もしているが、開発で実際に使った人はほとんどいない。コンパイラ、デバッガ、エディタ、リンクローダ（無意識）、構成管理ツールで十分である。

見積り

真実８．プロジェクトが大失敗する原因は二つある。一つは「見積りミス」だ。もう一つは「仕様未凍結（真実２３）」だ。

真実９．ソフトウェア開発の見積りは、プロジェクト開始時に実施する場合が非常に多い。これだと要求定義が固まる前に見積ることになり、どんな問題がどこにあるかを理解する以前に予測するので意味がない。従って見積り時期として適切でない。

真実 10．見積りは、上層部かマーケティング部門が実施する場合がほとんどだ。結局、適切な人が見積もっていない（政治的見積りと本質的見積り）。

真実 11．プロジェクトが進むに従って、見積りを調整することはまずない。従って、不適切な時期に不適切な人が実施した見積りが修正されることはまずない。

真実 12．見積り精度がいい加減だと、実際のプロジェクトが見積もり通りに進まなくてもまったく気にならないはずだが、現実にはみな気にする。

真実 13. 管理者とプログラマの間には、大きな断絶がある。ある研究によると、重大な見積りミスを起こし、管理者は大失敗プロジェクトと考えているのに（コスト 419％増、スケジュール 193％増、規模 130％増、ファームウェア 800％増）、開発担当の技術者はこれまで従事した中で、最も成功したプロジェクト（仕様を充たし、リリース後のバグが０）と考えているものがあったらしい。

真実 14. 実現可能性の分析の結果はいつも YES である（ICSE87 ワインバーグの講演）

再利用

真実 15. 小規模な再利用（例えば、ライブラリやサブルーチン）は、50 年前に始まり、すでに解決済みの問題である。

真実 16. 大規模な再利用（コンポーネント単位）は、誰もがその重要性を認識し、なくてはならないと感じているが、今なお未解決である。多種類にわたる特定目的のシステムで使えるように、コンポーネントの機能を一般化するのはほぼ不可能。

真実 17. 大規模な再利用は、類似システム間でうまくいく可能性が高い。応用分野の類似性に依存するため、大規模流用の適用範囲は狭くなる。

真実 18. 再利用には二つの「３の法則」がある。（１）再利用可能なコンポーネントを作るのは、単一のプログラムで使うモジュールを開発する場合に比べて 3 倍難しい。（２）再利用可能なコンポーネントは、ライブラリに取り込む前に、３つの異なるプログラムでテストする必要がある。

真実 19. プログラムを再利用する場合、流用母体の変更は、バグの原因になる。２０－２５％も変更する必要があるなら、最初から作ったほうが、効率が上がるし、品質も良い。ベンダーが作ったパッケージのソフトウェアシステムを改造するのはまず、うまくいかない。

真実 20. デザインパターン方式の再利用（設計方式の再利用）は、ソースコードの再利用における問題を解決する一手段である。

複雑性

真実 21. 対象となる問題の複雑度が２５％増加するたびに、ソフトウェアによる解法の複雑性は１００％上昇する。改善しなければならない数字ではない。こうなるのが普通である。

真実 22. ソフトウェア開発の開発は、８０％が知的な作業である。かなりの部分が創造的な仕事であり、事務作業はほとんどない。

ライフサイクル

要求仕様

真実 23. プロジェクトが途中打ち切りになる二つの原因のうち、一つは仕様を確定できないことだ。

真実 24. 仕様不良の修正コストは製品出荷後は最も高い（100 倍　by Boehm and Basilli）が、開発の初期であれば最も安い。

真実 25. 仕様の抜けは、仕様関係の不良の中で修正が最も難しい。最もしつこいバグ、すなわち、テストをかいくぐり製品に潜り込むのはロジック抜けの不良だ。仕様の漏れがロジック抜けにつながる。

設計

真実 26. 仕様定義フェーズから設計フェーズに移るとき、膨大な数の「派生仕様(derived requirements：仕様を具体的な設計方式にブレイクダウンする場合、設計方式に対する要求仕様)」が生じる。これは問題解決プロセスが複雑なために発生するもので、この設計仕様の量は、元の仕様の５０倍になることもある。

真実 27. ソフトウェア開発において、ベストの解法が一つしかないことはまずありえない。

真実 28. ソフトウェアの設計は、複雑で反復が必要なプロセスである。従って、最初に考えついた設計方式が間違っている可能性は高く、最適解ではない。

コーディング

真実 29. 問題を「基本モジュール」レベルにまで詳細化できた時点で、設計フェーズからコーディングフェーズへ移る。コーディングをする人と設計者が同一人物でない場合、「基本モジュール」の認識にズレが生じトラブルのもととなる。設計とコーディングを分けるのは良くない。

真実 30. COBOL は非常に悪い言語だ。しかし、他のビジネスデータ処理用言語はもっとひどい。

不良除去

真実 31. ソフトウェア開発のライフサイクルで、不良除去に最も時間がかかる。

テスト

真実 32. 十分テストをしたとプログラマが自信を持つソフトウェアでも、全パスの５５－６０％程度しか網羅していない。パス・カバレッジ・アナライザのような自動化ツールを使うと、網羅率が８５－９０％に上がる。しかし、１００％のパスを網羅するのは不可能である。

真実 33. １００％のテスト網羅が可能でも、完全テストとはいえない。バグの約３５％は、パスの抜けが原因であり、４０％はパスの特定の組み合わせを実行したときに起きる。このバグは、パスを１００％カバーしても検出できない。

真実 34. ツールを利用しないと不良除去はうまくいかない。デバッガはみんな使うが、カバレッジアナライザはほとんど使わない。組込みソフトウェアのテストは、環境シミュレータがないと不可能である。

真実 35. テストの自動化は非常に難しい。テストプロセスの中には、自動化できるし、自動化しなければならないものもあるが、大部分は自動化が不可能な作業である。

真実 36. プログラマがソースコードの中に組み込むデバッグ用の命令語（条件コンパイルできることが望ましい）は、テストツールを補ってあまりある。

レビューとインスペクション

真実 37. インスペクションを厳しく実施すると、プログラム実行前に９０％ものバグをたたきだせる。

真実 38. 厳密なインスペクションは大きな効果を上げるが、テストのかわりにはならない。

真実 39. 出荷後レビュー（遡及的レビュー[retrospectives]と呼ぶ場合もある）は、顧客満足度の測定およびプロセス改善の両方の観点から重要だが、実施している企業はほとんどない。

真実 40. ピアレビューには、技術的問題と社会学的問題がある。レビュー相手のことを考慮しないと悲惨な結果になる。集中力こそ、レビューに不可欠な厳密性を生む。

保守

真実 41. 保守には、ソフトウェアコストの４０－８０％（平均60%）がかかる。従ってソフトウェアライフサイクルの最重要フェーズである。

真実 42. 保守の６０％は機能拡張であり、バグ修正は１７％にすぎない。このため、ソフトウェアの保守は不良修正ではなく、古いソフトウェアに新しい機能を追加する作業である。

真実 43. 出荷後レビュー（遡及的レビュー[retrospectives]と呼ぶ場合もある）は、顧客満足度の測定およびプロセス改善の両方の観点から重要だが、実施している企業はほとんどない。

真実 44. ソフトウェアの開発作業と保守作業は大部分は共通している。例外は、保守の場合、「既存プログラムの理解」が新たに加わることであり、保守作業の約３０％が必要となる。したがって保守は開発より難しい。

真実 45. 優れたソフトウェアエンジニアリングに沿ってプログラムを開発すると、構造がよくなるため、保守は減らずかえって増える。

品質

品質

真実 46. 品質とは属性（移植性、信頼性、効率、利用容易性、検証性、理解容易性）の集合である。

真実 47. ソフトウェアの品質とは、ユーザを満足させることではない。仕様を満足させることでもなければ、コストとスケジュールを満足させることでも、信頼性でもない。ユーザの満足度＝仕様の実現度+スケジュール通りの出荷+適正なコスト+高品質の製品。最初の３つは品質に関係ない。

信頼性

真実 48. 誰もが共通に作りこむ種類のバグが存在する。

真実 49. バグは固まって存在する。

真実 50. 不良除去に唯一無二のベストな方法はない。

真実 51. プログラマ中の残存バグは簡単には摘出できない。従って不良除去では、重大なバグを最小に抑えたり、なくすことを目標とすべきである。

効率

真実 52. プログラムの処理効率には、良いコーディングよりも、良い設計が大きく影響する。

真実 53. 高級言語は、適切な最適化コンパイラがあれば、アセンブリー言語で記述した場合の９０％の効率で処理できる。最新の複雑なハードウェアを使えば、アセンブリープログラムより速くなる。

真実 54. プログラムの大きさと処理時間には、トレードオフがある。片方をよくすると、他方が悪くなる。

研究

真実 55. 大部分の研究者は、技法を「分析」するのではなく、「擁護」する。その結果、(1) 研究対象の技法は、実際には、自分が信じるほどの効果はなく、(2) 技法の本当の価値を検証する評価研究が足りない。

プロジェクト管理

管理

ウソ１．計測できないものは管理できない。

ウソ２．ソフトウェア製品の品質は管理できる。

人員

ウソ３．プログラミングからエゴを取ることは可能だし、そうでなければならない。

ツールと技法

ウソ４．ツールや技法はどんな状況でも適用できる。

ウソ５．ソフトウェアには、もっと開発方法論が必要である。

見積り

ウソ６．コストやスケジュールを予測する場合、まずソースコードの行数を見積もる。

ライフサイクル

テスト

ウソ７．ランダムテストにより、テストを最適化できる。

レビュー

ウソ８．多くの目にさらせばバグはとれる。

保守

ウソ９．将来の保守に要するコストや、いつ現在のソフトウェアを入れ替えるのかの決定は、過去のコストのデータを見ればわかる。

教育

ウソ 10. プログラムをどう書くかを見せれば、プログラムの方法を教えられる。(実際は書く前に読む)

Waterfall Model 再考

ソフトウェア開発方法論とソフトウェア開発管理の融合について

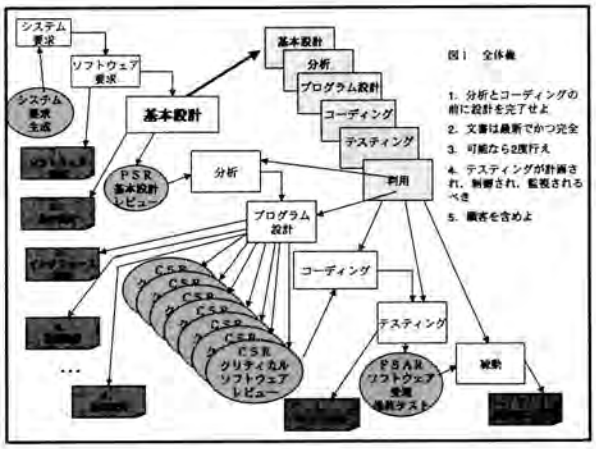
北陸先端科学技術大学院大学
情報科学研究科
落水　浩一郎

内容

1. W. Royce による論文「大規模ソフトウェアシステムの開発を管理することについて」（１９７０)の紹介
2. ソフトウェア開発方法論、ソフトウェア開発管理（プロジェクト管理）の発展の歴史のサーベイ
3. いくつかの問題提起
   1. ソフトウェア開発という生き物の把握
   2. 開発方法論と開発管理技術の融合
   3. 標準化
   4. 教育

W. Royce が提案したことは

1. Waterfall Modelの提案
   大規模ソフトウェアシステムの開発管理をどのように実現するか
2. Waterfall アプローチを採用したときに起こる問題の指摘とその解決策の提案



結論に至る論理の展開

- モデルの構築
  - （プログラム）「分析」と「コーディング」から始める
  - 大規模ソフトウェア開発を制御するために必要なフェーズ群を付加する　（システム要求定義、ソフトウェア要求定義、プログラム設計、テスティング、稼動）
  - フェーズの実行順序に関する情報を付加し、ベースラインの移行法に関する仮定を導入する
- 問題点の指摘と対応策の提案
  - 「テスト」→「設計」→「要求」となる、大幅な手戻りに関する問題の指摘と対応策の提案
    - 基本設計フェーズの導入
    - 設計結果の文書化
    - パイロットモデルの開発（基本設計以下を2度行う）
    - テストの計画、制御、監視
    - 顧客を含める

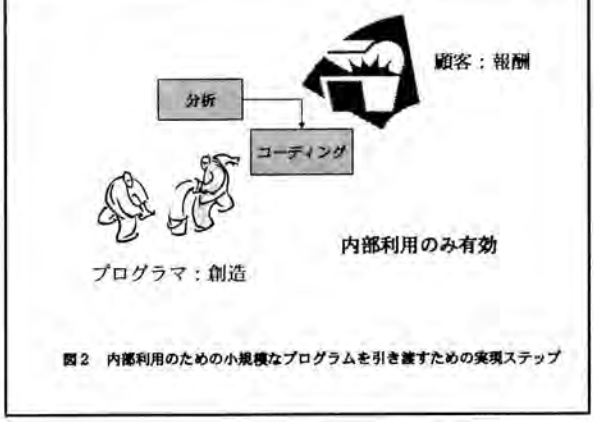


図2　内部利用のための小規模なプログラムを引き渡すための実現ステップ

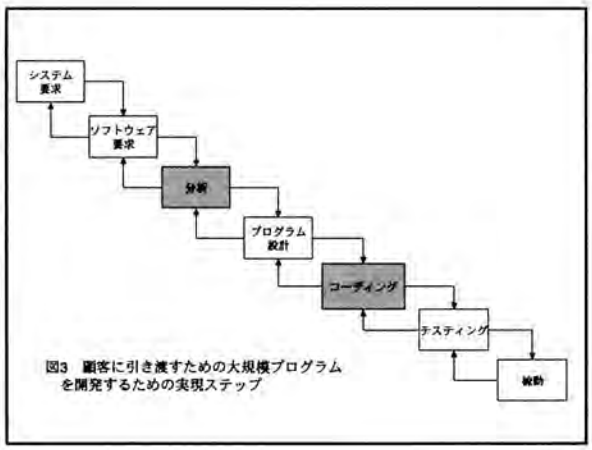


図3　顧客に引き渡すための大規模プログラムを開発するための実現ステップ

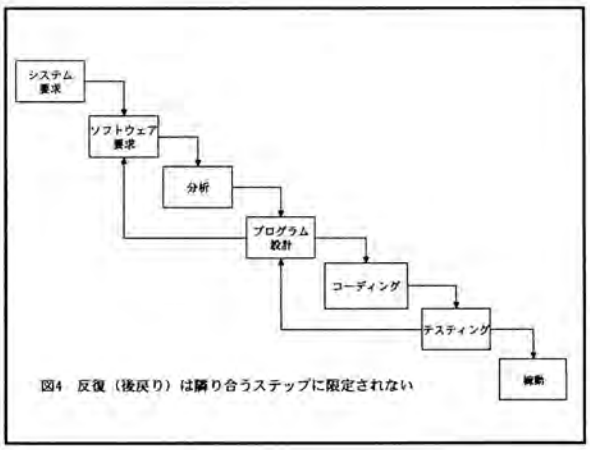


図4　反復（後戻り）は隣り合うステップに限定されない

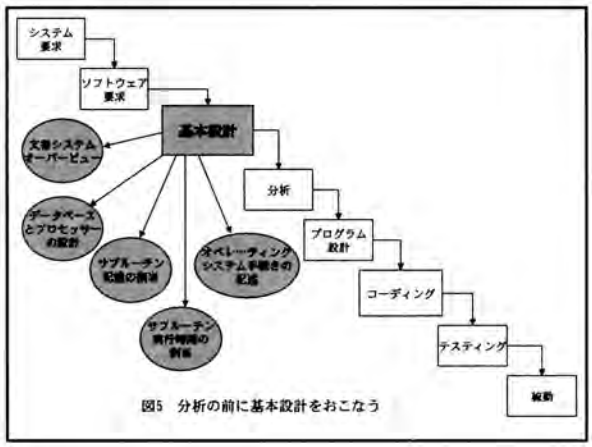


図5　分析の前に基本設計をおこなう

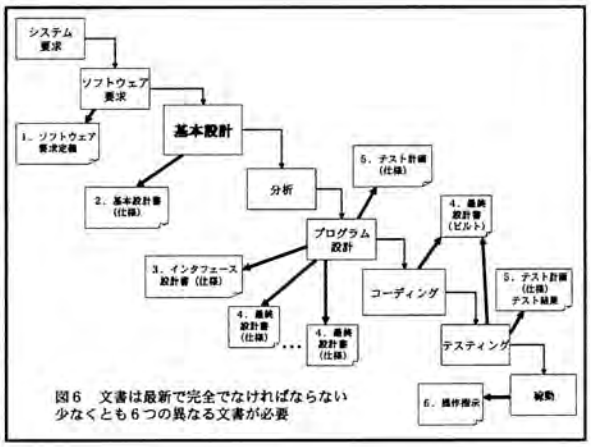


図6　文書は最新で完全でなければならない
少なくとも6つの異なる文書が必要

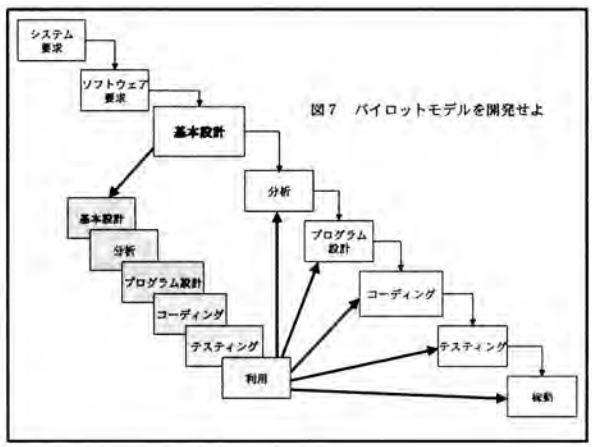


図7　パイロットモデルを開発せよ

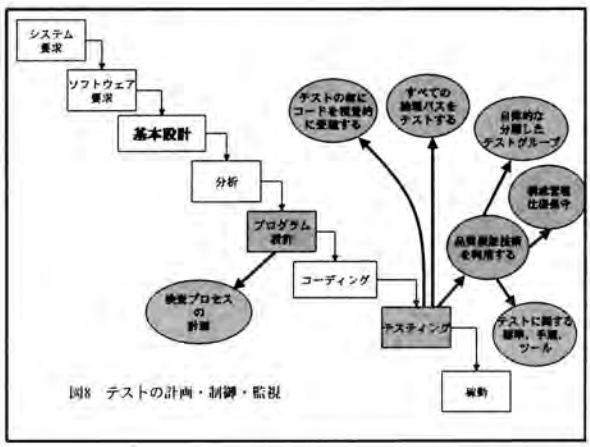


図8　テストの計画・制御・監視

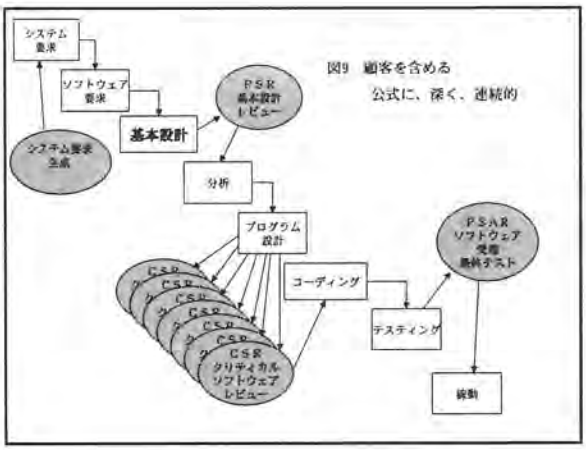


図9 顧客を含める
公式に、深く、連続的

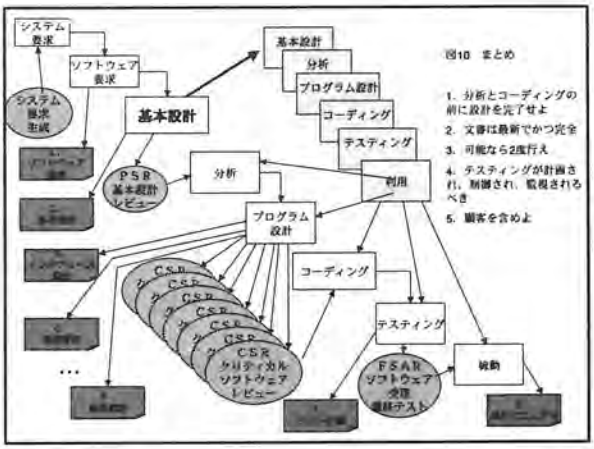


図10 まとめ

1. 分析とコーディングの前に設計を完了せよ
2. 文書は最新でかつ完全
3. 可能なら2度行え
4. テスティングが計画され、制御され、監視されるべき
5. 顧客を含めよ

ソフトウェア開発方法論、プロジェクト管理技術の発展の歴史（サーベイ）

- プログラミング方法論の提案(1970年代前半)
- 設計方法論の開発(1970年代後半)
- 要求定義技術の開発(1970年代後半)
- 定量的プロジェクト管理のはしり(1970年代後半から1980年代前半)
- システム工学の導入によるWaterfall Modelの改善(1980年代中期から後半)
- 反復型Waterfall Modelの提案(1980年代前半)
- プロトタイピングの出現(1980年代前半)
- 実行可能仕様とフォーマルメソッド(1980年代中期)
- プロセスプログラミング(1980年代後半)
- プロセス改善技術(1990年代初頭)
- CASEツール(1990年代初頭)
- アーキテクチャ中心開発(1990年代中期)
- オブジェクト指向技術（1980年代以降）
- UML(1990年代後半)
- アジャイル

Waterfall Modelの問題点

- ソフトウェアプロジェクトは簡単に計画・評価できない
- アクティビティの直列化
- どうすればよいのかはパネルの主題？

開発方法論と開発管理技術の融合

- 開発方法論
  - 変更や検査・検証、再利用が容易な構造を定義し、それらを作りこんでいく手段を与える
- プロジェクト管理
  - コスト、スケジュール、品質を管理するためのメトリクスの定義とベストプラクティス
- 乖離？
- 融合：視点「開発管理とはプロジェクト特有の状況の学習である」

論点

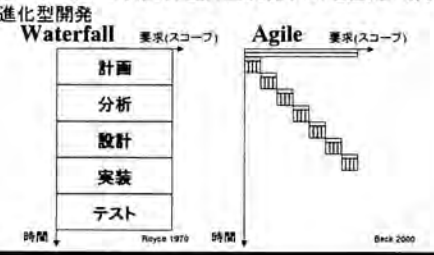
- ソフトウェア開発プロジェクトという生き物をどうとらえるのか？
- 標準化とはなにか？（効果とデメリット）
  - 枠組み
    - ソフトウェア開発方法論
    - CMM, PMBOX
    - その統合法（落水の当面の関心）
  - 社内標準
  - プロジェクトに特化したカスタマイズ
- どのように鍛錬・訓練するのか？
  - 大学教育とOJTの分担、協調

Waterfall と Agile

平鍋 健児
チェンジヴィジョン

Waterfall と Agile

- 短いサイクルで、分析、設計、実装、テストを並列に行う
- 進化型開発



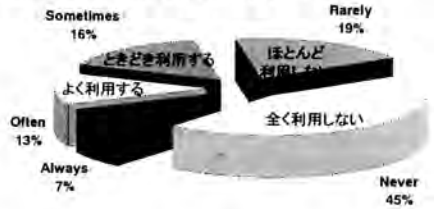
なぜ？

1. 要求は変化する

- ビジネス変化が速い
- 人間は見てからでないと分からない

要求は変化する

- 使われない機能（ムダ）を作ってしまう



決定を遅らせることで、余分な機能のムダを排除できる。

2. 開発は発見プロセス

- 発見には学習が必要
- 人間は経験からしか学べない

「開発」は発見のプロセス

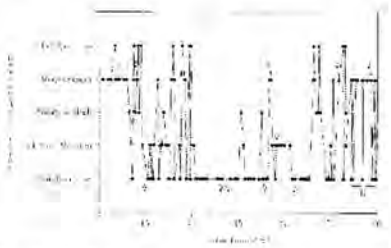
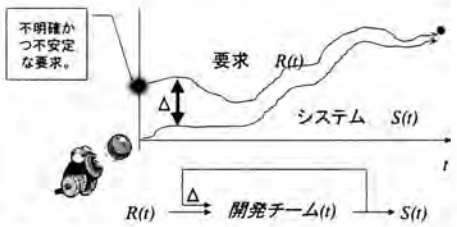


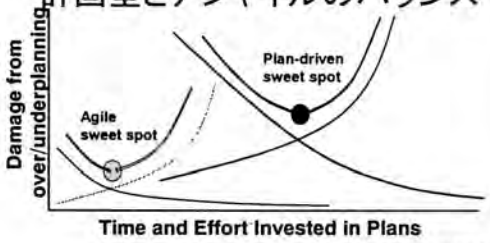
Figure 2.1 Design activity[5]

「開発」は問題解決型。最初から正しく計画、設計できるわけではない。

Moving Target

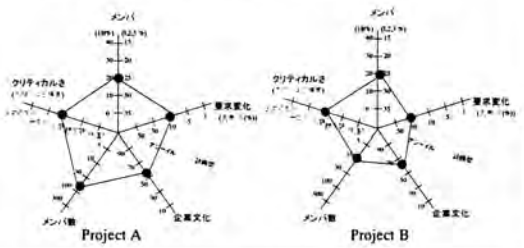


計画型とアジャイルのバランス

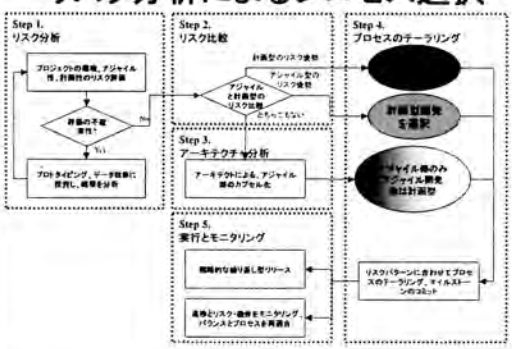


"Get Ready for Agile Methods – With Care"
(Barry Boehm, IEEE *Computer*, January 2001)

プロジェクトのアジャイル適正



リスク分析によるプロセス選択



Waterfallモデルについて考える

伊藤 昌夫
ニルソフトウェア
ヴィキャドソリューションズ

ウォータフォールライフサイクルモデルのイメージ

経験

- 20 年程前
  - MIL-Std-1679ベースの開発
    - 防衛関係(航空機)
    - 1679 は, 1978年に初版が発行された.
- 10年程前
  - IPSE(Integrated Process Support Environment)
    - ミクロ(タスクレベル)プロセスの環境への実装
- 今
  - SPO(Software Process Optimization)
    - 如何にプロセス変更を行うか.

ライフサイクル(プロセス参照モデル)

Waterfallライフサイクルモデルは, 基本形を忠実に表現したもの. 派生系には, それぞれ理由がある

基本形	R	S	D	C	T						
逐次的	R	S	D	C	T	D	C	T	D	C	T
増分的	R	S	D	C	T	R	S	D	C	T	R	S	D	C	T
XP	P	S D TC	P	S D TC	P	S D TC	P	S D TC	P	S D TC	

R:要求, S:仕様化, D:設計, C:実装, T:テスト, P:プラン

議論

- ライフサイクルモデルの基本形
  - waterfallといって指しているものの向こうにあるもの
- 現在におけるマッピング(Royce論文の各ステップとの対比から)
  - 予備設計
    - アーキテクチャ
  - 文書化
    - 抽象 昇華でのモデル表現
  - 繰り返し
    - 適切なゴールの設定
  - テストの計画／統制／モニタリング
    - 検証とその信頼性確保のための仕組みとは
  - 顧客による早期の確認
    - システムシミュレーション, プロトタイプ

Forum での討論を終わって

伊 藤 昌 夫
Nil Software

Royce の論文は，決して Waterfall モデルと呼ばれるものの解説論文ではない．Waterfall に関して参照すべきものがなく，たまたま問題点と改良点を指摘した Royce 論文があったに過ぎなかったからと考えている．その点で，Waterfall というライフサイクルプロセスは Royce の発明ではなく，ソフトウエア工学の有史以前に既に人口に膾炙していたと考えている．

当時の大規模プロジェクトの多くは，軍関係であったと想像でき，そこでは国と特定軍需企業が結びついて，契約が結ばれる．開発期間も一般に長期に渡る．そこでは，何らかの契約・管理・支払いを明確化するためのモデルが必要で，Waterfall は，MIL-std-1679A(Navy),DOD-std-2167A として長く必要とされてきた．その点においてのみ都合の良い仮想的なライフサイクルモデルであると（経験上）感じている．

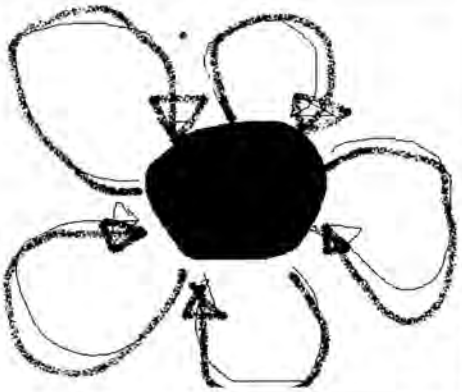
故に，*実*プロジェクトをどう運営するかは全く別のことと考える．SLCP（ISO/IEC-12207）が，結局のところ，各プロセス要素の順序を定義せずに，共通語としてのプロセス要素（プロセス / アクティビティ / タスク）の説明に終始していることからも分かる．

今日的な問題として， Waterfall を考えるときに，我々自身の思考停止に気付くべきである．あらゆる，ライフサイクルモデルや様々な手法が提示する開発のフローは，Waterfall と呼ばれるものが持っているのと同一の落とし穴を持っている．そこから単純に逃れることはできない．唯一，脱出する道は，"生きた"プロセスについて思考する事で，それが今回の Waterfall ライフサイクルモデル論議を通じて多少なりともできたことは，自分自身良かったと思っている．

フラワー・モデル再び
Flower Model Revisited

岸田孝一
K2@sra.co.jp

2006/4/3 @ SEA Forum April



Process という英単語の意味

- Something that sets a pattern to be followed
  - model for, as in model citizen
- Something that mirrors some other (real) entity
  - model of, as in models of aircraft

Process との私的かかわり

1970 Waterfall paper @ Wescon
1978 Evolution Dynamics by M.M.Lehman
1984 V-model, Spiral Model @ ISPW-1
1986 Process Programming, CMM @ ISPW-3
1988 Paradigm Shift paper by C.Floyd

2つの Perspectives by Christiane Floyd

- Process-oriented
  プロセスを，環境変化のなかで行われるさまざまな学習・作業・コミュニケーション活動
- Product-oriented
  プロセスを，確立されたモデルおよび関連ドキュメントから構成されるプロダクトとして扱う．
  との関連においてとらえる．
- これらの2つは相補的

"Software is Process, too!"

プログラムの実体はプロセスである
  わたしにとって構造化プログラミングとは何だったか？
  プログラムの正しさと構造（プロセス）の美しさ
プロセスの諸相
  プログラム実行プロセス，ソフトウェア開発プロセス，
  そして システム進化のプロセス
ネットアーティストからのメッセージ
  完成した作品がプロダクトではなく，それが創られて
  行くプロセスこそが作品なのだ

さて Flower Model

発想の発端
- 1980年代初め，旧ソフト協技術委員会ワークショップでの要求分析についての討論

開発アーカイブを中心とする活動のイメージ
- そこから何かを取り出し，変形し，また格納するという活動のくりかえしが開発である．さまざまな活動があり，しかしそれらの順序は規定されていない．

ネルソン・グッドマンからののメッセージ
- 「世界」は無からではなく，すでに存在する世界の「バージョン」を変形する形で作られる

最初の小競り合い

とりあえずの提案 @ ISPW-2 (1985)
- 「これはあまりに Context Free すぎるのでは？」という批判でおしまい．

しかしこの WS での討論の主流は別だった
- プロセス概念のピラミッド構造に対する B.Balzer さんの強烈なオブジェクションオブジェクションが場を支配した．
  - ツールが変わればプロセスは変わる！
  - プロセスを変えられないツールはツールじゃない！

その意味では
- 花芯にあたる知識ベースツールがまだ未整備だった

あらためて考え直し

ちょっと視点を変えてプロセスを視る
- トラベルプラン型モデル
  - @ ISPWs & SDA Meeting

プロセス・プログラミング
- いささか衝撃的．
- しかしあまりに古典力学風という印象．

古典哲学フレームワークの再発見
- Max Weber とオブジェクト指向
- 儒教哲学と CMM あるいはソフトウェア工学全般

気になっていたこと

形式化された手続きをどう埋め込むか？

- それぞれの状況に応じて形式モデルに依存し，そのモデルにしたがってものごとを進めるルールを自動化する
  - → Centralized Control
- 形式モデルを利用するコンテクストをオープンにし，さまざまな資源を自由に利用できるように提供する
  - → Situated Coordination

インフラストラクチャの発展

情報レポジトリ
- 開発環境におけるDBの変遷
- 知識ベースとオントロジー

開発環境の進化
- インターネットの普及
- Workstation から高性能 PC へ

知的共同作業環境の実現可能性
- たとえば DynC プロジェクト
- CHI 技術の進歩

初めて具体化された フラワー・モデル環境

- Vincent System
  - for Process Evolution Management
  - By Bill Riddle
  - @ Fraunhofer IESE

http://www.iese.fraunhofer.de/vincent/

Fraunhofer Institut Experimentelles Software Engineering

Vincent

rationalizing process management

Fraunhofer IESE offers a new and innovative approach to helping you cope with process changes by gaining and maintaining control over the evolution of your business and engineering processes.

Fig. 1. Process Evolution Dynamics Framework

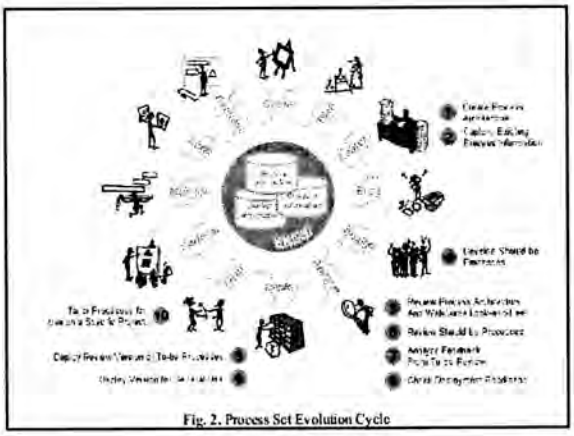


Fig. 2. Process Set Evolution Cycle

Vincent における プロセス・ステップス

- Gather
- Plan
- Capture
- Elicit
- Design
- Analyze
- Deploy
- Train
- Perform
- Monitor
- Audit
- Examine

考えられる次の一歩

- 開発/管理支援のフラワー・モデル環境
  - オントロジー・デザインの問題
    - DynC における知識ワークスペース
  - CHI 技術の応用
    - Awareness Notification
  - 活動ステップの新しいカテゴリ分け
    - Goodman の Ways of Worldmaking

Goodman による 世界制作ステップのカテゴリ分け

(1) Composition and Decomposition
(2) Weighting
(3) Ordering
(4) Deletion and Supplementation
(5) Deformation

プロセス・ステップ取り扱いの留意事項

- Operational Function による切り分けはどうしても物理的・論理的順序関係に引きずられてしまう.
- Goodman のカテゴリ分けはこの難点を打開する上で有効では？

References

- M.M.Lehman
  http://www.doc.ic.ac.uk/~mml/
- Lee Ostwerweil
  Software Processes are Software Too,
  Proceedings of the 9th ICSE, 1987.
- Christiane Floyd
  Theory and Practice of Software Development,
  Proceedings of TAPSOFT '95, LNCS 915 (Springer)
- Bill Riddle
  A Framework for Coping with Process Evolution
  Proceedings of SPW2005, LNCS 3840 (Springer)

References（続き）

- Nelson Goodman
  Ways of Worldmaking
  Hackett Publishing Company (June 1978)
  ISBN: 0915144514

  世界制作の方法
  菅野盾樹訳, みすず書房 (1987)
  ISBN: 4622006235

Forum を終わって考えたこと

岸 田 孝 一
SRA-KTL

Slide に書いたように，この花びら型プロセスモデルのアイデアは，1980 年代初めに伊豆高原で開かれた旧ソフト協技術委員会のワークショップでの討論の中から生まれたものである．当時はまだソフトウェア開発環境がきわめて貧弱だったので，モデルの花芯にあたる情報レポジトリは仮想的なものとしてしかイメージされていなかった．しかし，それが他のモデル (Waterfall を含めて)と異なっていたのは，単にプログラムや仕様書，テストデータなどのドキュメント類だけではなく，開発チーム・メンバーのアクションや相互のコミュニケーションの記録など，開発に関わるすべての情報を含むものとして考えられていた点である．今様のいい方をすれば，オントロジー設計が異なっていた．

このモデルのアイデアを最初に世の中に出そうと試みたのは 1985 年の第 2 回 ISPW だったが，あまりに Context Free 過ぎるという批判を浴びて引っ込めざるを得なかった．それから 20 年ほどの時間が過ぎたのち，老朋友 Bill Riddle さんの Vincent System にそのアイデアの一端が具体化されたのはうれしいことだった．この 3 年間，産学協同で進めてきた DynC プロジェクトの知的共同作業支援のためのワークスペースというアイデアも，Flower Model の一種だといえばいえないこともない．[*]

[*] “知識共創のためのダイナミックコミュニティ:理論・アーキテクチャ・応用,”
ソフトウェアシンポジウム 2004 論文集

プロセスの問題を考えるとき，わたしがいつも意識するのは Christiana Floyd さんのいう，プロダクト指向か，それともプロセス指向か，という視点のちがいである．プロダクト指向の視点に立てば，「ソフトウェア開発プロセスとは，形式的に確立されたモデルとそれを構成する活動およびドキュメントの集合だ」ということになる．明らかに Waterfall Model はこの視点で考えられたものである．一方，プロセス指向の視点では，「ソフトウェア開発プロセスとは，ビジネス環境の絶えざる変化の中で，ユーザと開発者とが行うコミュニケーションおよび学習および作業のつながりだ」と考えられる．2つの視点は相補的であり，どちらか一方だけあればよいというん¥ものではないが，コンピュータの社会への浸透が始まった1970 年代以降，プロダクト指向からプロセス指向へというパラダイムの変化が起こったという Floyd 女史の指摘は正しいと思う．そして，このパラダイム・シフトは，ネットワークの普及にともなってさらに進みつつあるように感じられる．

フラワー・モデルは，そうしたパラダイム・シフトをうまく吸収することができるプロセス・モデルではないかと，あらためて考えている．そのとき思い出されるのは，なぜか，ルードヴィッヒ・ヴィトゲンシュタインが「論理哲学論考」に書いた次のコトバだ:

— 世界はものの総体ではなく，成立していることがらの総体である．

世のオブジェクト指向派の人たちは個のコトバをどうとらえているのだろうか．

SEA Forum April

Waterfall Model 再考(最高？)に参加して

2006年4月3日 JJK 会館

三 輪 東
CSK システムズ

開演前の 16:00。40 人程の参加者でしょうか。なかなか活況のようです。田中さんの開会の挨拶でスタートしました。まずは、今後の活動予定について。どうやら、WEB2.0、をもじった Binary2.0 という企画も考えられているようです。Binary2.0 とはいったい何か？ さっそく Google で調べてみると、WEB2.0 を支える基盤技術のようなものらしい。これも面白そうですね。

1. 基調講演

16:05 挨拶も終わり、早速本題へ。JAIST の落水浩一郎先生による講演 「"Waterfall Model"再考(最高?)」の始まりです。

予想通り、図が出てきました。あれっ、よく見ると、いつも見慣れている図とは違うようです。いつも見る図は、フェイズが段階的に線で繋がっているのですが、このスライドに示されたのは、水が単純に落ちていくだけではなく、それ以外のものも記述されています。また、

1. 基本設計フェイズを入れよう。
2. 文書はたくさん書くべきだ。
3. 2度の開発をしなさい(プロトタイピングへの言及はありませんが、反復については述べている)。
4. テスティングは大事。
5. 顧客に責任を取ってもらうためにも、顧客を含めよう。

といったことが書いてあります。

知らなかったのですが、このモデルでは、基本設計に5つのプロセスを追加すべきだと提唱しているようです。追加されるプロセスは、分析、プログラム設計、コーディング、テスティング、利用。実際のコーディングの前に、設計の制約事項となるようなものは、洗い出してしまうのですねぇ。つまり、性能、制約に関する分析を事前にすませてしまうことを意味しているようです。

私の知っている Waterfall とは随分違います。

文書化せよ！ といっています。ソフトウェアを管理する＝文書化を強制することだ！！ 考えを形にして残す必要性を考えれば、文書化は役に立つ。はい、その通りです。反論の余地はありません。...この図には、具体的な成果物も書かれています。Waterfall って、フェイズと線だけではないようです。

2度の開発をしなさい！ 実際には、反復のプロセスが書かれています。Waterfall で反復って、噴水みたいなものでしょうかね。ここでいわれているのは、プロトタイピングではなく、パイロットモデルなのだそうです。ユーザーも巻き込んで検証するようなプロトタイピングではなく、現場に閉じたパイロットモデルということで、開発可能性の確認に主眼が置かれているようです。テストについても、分離したテストグループを作れとか、品質保持技術を利用しろとか、いろいろと述べられています。

最後に、ユーザーも参加させるべきだといわれています。その通りですね。実は、Waterfall って、それによって発生しうる問題も事前に予測して、その処方箋を用意することについても述べられていたんです。いやぁ、知りませんでした。

世の中で、Waterfall Model が採用されている割合ですが、IPA/SEC のデータ白書2005では97%だとあります。ただし、今回の説明をおききすると、正しく理解している人が全てではないのかもしれません（少なくとも私は理解していませんでした)。ですから、実際に Waterfall が使われている割合はもっと少ないのかもしれませんね。

「みなさん、Waterfall を正しく理解していますか？」という質問に、会場からは苦笑が起こっていましたし。Waterfall、なかなかよさそうじゃないという雰囲気になりかけた（私だけがそう感じた?）ところで、やはりいわれました。失敗したら一つ前に戻ろう、ということは現実的じゃないでしょう、と。ここは、明らかに、問題です、と。

さてさて、Waterfall の説明が一通り終わり、過去の開発方法論、プロジェクト管理技術の歴史をふりかえってみようということになりました。

70 年代には、まず、全体の作業をどのように単位活動へ分割していくかということから始まったようです。このことは、プログラム設計の分野ではかなりの貢献したという評価です。その後、変更に強くテストしやすいシステムはどのようにすればよいかがいろいろ議論がなされ、要求定義の技術が話題の中心になったようです。そして、定量的プロジェクト管理ということが叫ばれるようになり、現実に何がおきているかを把握するために、複雑性のメトリクス、バグ曲線などが出てきたとのだそうです。この頃から、プロジェクト管理とソフトウェア工学が分離さする傾向が生まれたことは、重大な問題だと指摘されました。

80 年代に入ると、まず反復型 Waterfall Model が出てきました。いわゆるV字モデルのことだそうですね。でも、うまくいかなかった。そこで、スパイラルモデルの登場です。小さな Waterfall Model を繰り返す。それでも、あまりうまくいかなかったので、次に、プロトタイピングが出てきました。そこで「内なる繰り返し」から「利用者世界も含めた反復」へ変わっていったようです。その次は、実行可能仕様とフォーマルメソッド。しかし、私には、両者の違いがよくわからない。仕様から厳密に書けば、ミスも無いし、自動生成も可能だってやつですよね。また、落水先生は、モデル検査はよいものだとおっしゃっていました。が、私には、それがどんなものなのかはわかりません。

90 年代になって、プロセス改善があちこちで騒がれるようになってきました。この時代は、まさに、「ソフトウェア工学 vs プロジェクト管理」という時代だったと苦笑されておられました。そして、CMMやCASEツールなどが騒がれ、アーキテクチャ中心開発、UML、アジャイルなどと進んで行ったようです。

この講演の中で強調されていたポイントは、プロジェクト管理とソフトウェア工学の分離という問題でした。時代が進むにつれ、この2つが別々に切り離されて議論されるようになり、また、それぞれの分野の技術者・研究者の仲も悪くなったようで、両方をバランスよく取り込んだ方法論が生み出されなかったという点を指摘されました。

結論。Waterfall Model は本当に失敗かということですが、落水先生は、失敗だったという評価です。理由は、

1. 要求を汲み取りきれない。
2. 評価しきれないよね。
3. アクティビティの直列化は現実的じゃないでしょ。

ということのようです。

Royce の Waterfall Paper には、さまざまなプロジェクトに適用しうる重要な原則が含まれているが、それは、マネージメントがきちんとできている組織では、たいした問題とはいえないのではないか。ただし、見習うべきことは、プロジェクト管理とソフトウェア工学の両方の観点を踏まえているという点だといわれました。やはり、この2つは車の両輪であって、両者が乖離している現状が問題だという指摘です。双方の分野の技術者・研究者たちが、お互いにもっと歩み寄ることが大切だ。わたしも、その通りだと思います。どちらも大事です。

さて Q&A タイムです。3 つの質問が出ました。

Q1: Waterfall Model というネーミングは、Royce の論文が発表された時には、既にいわれていたのでしょうか？ それとも、かれが名付けたのですか？

A1: 恐らく、本人が名付けたのでしょう。ただ、この論文には Waterfall という表現は出てきていませんね。

Q2: W.Royce って、RUP の人ですか？

A2: RUP の提唱者は息子の Walker Royce です。Waterfall Paper を書いたのはお父さんの

Winston Royce。RUP に父親の影響があったかどうかは不明です。

Q3: フェイズの時間配分等には言及されていますか？

A3: この論文では、具体的に述べられてはいませんが、それを意識した書き方にはなっています。基本設計、テスティングなどは、既に意識されており、プロジェクト成功のキーになると書かれています。ただ、時間配分などの詳細には、言及されていません。他の論文で、その点を書いたものがあったと思いますが、よく覚えていません。

最後に私自身の感想ですが、Waterfall は、プロセスの基本を考えるモデルとしては非常によいものだと思いました。これをベースに考えれば、よいも悪いも見えてくる。ある意味、バランスよく説明されているように思えます。Waterfall をそのまま現実のプロジェクトに適用するのは賢いやりかたではありませんが、ダメな部分を認識しながら参照するモデルとしては、使えるのではないでしょうか。いろいろと批判はありますが、イメージしやすいから叩かれやすいのだということのような気がします。モデルとしてわかりやすいというのは、重要なことです。そういう意味で、Waterfall は、なかなかの役者ではないかと思います。

17:30～17:45 は休憩です。サンドイッチとコーヒーで腹ごしらえ。

2. パネル討論

さて、17:45。基調講演の落水先生のほかに次の3人のパネリストが登場しての発表と討論です。

平鍋 健児さん (チェンジビジョン)
伊藤 昌夫さん (ニルソフトウェア)
岸田 孝一さん (SRA 先端技術研究所)

トップバッターは、平鍋さん。Waterfall と Agile というタイトルのお話です。Agile では、フェイズという考えではなく、視点を大事にしており、状況に応じて帽子をかぶり分けるというイメージだとのことです。つまり、時間の経過だけでものごとを判断するのではなく、繰り返し、さまざまな視点を取り入れてリスクヘッジしようということのようですね。

なぜでしょうか。大きな理由は、次の二つだといわれました:

1. 要求は変化する
2. 開発は発見のプロセスである

要求の変化はそう簡単に予想はできない。予想できない要求をそう簡単に確定できるわけがないというわけですね。また、人間は、分かったつもりでいても、最終的なものを見てみなければ、本当に理解したとはいえない。よいと思って作ってもらったものの、実際に使ってみると、使い勝手が悪いので変更してほしいといった要求が出てくることは、珍しい話ではない。ある調査によると、システムが持っている機能のうち、「まったく利用されない」ものが全体の45%を占め、「ほとんど利用しない」の19%を合わせると、実に6割以上の機能が有効活用されていないとうのが現実だそうです。要求は、時間をかけて変化し、固まっていくものであり、最初に要求を決めてしまうのは、あまりにリスクが大きい。そういった要求分析・確定に関する問題点を解決するには、Agile が最適だという主張です。

第2点の「開発は発見プロセス」については、発見には学習が必要であり、人間は経験からしか学べないということを強調されていました。結局のところ、人間は、現在のコンテキストから、ものごとを判断している。現在を理解するには、現在あるものでしか理解できない。このことを、平鍋さんが翻訳された本の中に出ている図を使って説明されました。その図では、上流工程から下流工程まで、思考が何十回か繰り返される様子が、グラフとして書かれています。グラフ上のあちこちに電球マークがあり、それが発見を意味しています。ポイントは、その発見が、上流から下流までいろいろなところに散らばっているとい

う現実です。つまり、発見は開発プロセスのさまざまな場所で起こる。下流工程での発見が、上流工程における新たな発見を促し、そうしたことを繰り返していくことによって問題解決を図るというのが自然な姿ではないか、と述べられました。

フィードバックを繰り返しながら、徐々に品質を上げていきましょうということですね。その際、情報をあちこちに分散しないように工夫をしよう。たとえば、要求が最後に確定するまで、ドキュメントはなるべく軽くした方がよいと指摘されました。

計画型とAgileとでは、プロジェクトの特性で損益分岐が異なるので、Plan-driven の Sweet spot を取るべきだとのことです。また、すべてのプロジェクトに Agile が適切とは限らないので、メンバや、要求変化の度合い、企業文化、プロジェクトのサイズ、クリティカルさなどから、計画型かそれともAgileかを選択するべきだと、「あたりまえのことですが」という但し書きつきでいわれました。Waterfall と Agile のバランスを常に意識しましょうということですね。

Agileでは、フェイズではなく、視点であるということ。帽子をかぶりわけることで、見えるところが見えてくる。これは、なるほどと思いました。繰り返すことの意味を、非常によく表現されていると思いました。「開発」は発見のプロセスだ」という説明に使われた図が、とても印象に残りました。

2番手は伊藤さん。以前、軍需関係の仕事に従事されておられたときの経験を語ってくださいました。20年程前には、Waterfall のように段階的な開発が一般的だったとのことでした。特に防衛関連のシステム開発では、それがあたりまえだったようです。

結局のところ、どのような分野のアプリケーションであれ、システムを作るという仕事には、共通項があります。要求がなければ仕様は書けない。仕様がなければプログラムは作れない。プログラムがなければ、テストはできない。論理的にはこの順序は必ず存在する。見かけは違っても、基本形は同じです。そういう意味では、Waterfall Model はプロセスの基本要素をよく表している。ただ、このモデルは万能ではない。ドメインにより、プロジェクトによって、やり方、重点ポイントを変えていくことが必要になる。そのことが、今回はあまり言及されませんでしたが、伊藤さんが提唱されている SPO(Software Process Optimization) の考え方につながっているのでしょう。

Royce 氏の論文が Waterfall の始まりではないというのが伊藤さんの主張でした。Waterfall モデルは、あの論文が書かれるそれ以前に存在していた。政府とのプロジェクト契約はそうしたモデルにもとづいて行われていた。実際に、軍事関係の大型プロジェクトはそうなっていたということでした。

最後に、議論のポイントということで、ライフサイクルモデルにおけるプロセスの基本要素の提示と、現実のプロジェクトにおけるそれらの「マッピング」とはまったく別の次元の問題であり、仮想的なプロセス・モデルにとらわれて思考停止状態に陥ることなく、現実の「生きた」プロセスを見つめて考えることが大事だと述べられました。

おそらく、Waterfall 論文で提唱されていたことがらは、基本をしっかりと残しながらも、形をかえて現在に至っており、それらを議論し、発展させていくことに意味があるといわれたいのだと、私は理解しました。

予備設計は、アーキテクチャに相当するもので、これは、非常に重要であり、早期にやるべきだ。文書化は、抽象度の異なるモデルで、適材適所、表現を変えていくのが大事でしょう。繰り返しは、ゴールの設定がすべてです。すべての部分を繰り返しやる必要はないので、その判断を適切に行う。安定度によって、必要なものだけやることが肝心である。テストの計画/統制/モニタリングについては、検証と妥当性確認のどちらを優先するかを考えなければいけない。テストファーストにするか、テスト計画ですませるかをきちんと選択しましょうということなのでしょう。

どのあたりでいわれたか忘れてしまったのですが、一つ印象的なお話がありました。それは、Waterfall Model が提唱された時期に、そんな風に段階的に開発を進めるというやり方に問題を感じていた人たちが既に存在しただろうという指摘です。そのことから、万能ではないとわかりながらも、W Waterfallを提唱したRoyce の論文の意義のようなものを感じました。Waterfallがダメだといわれながらも、プロセスの基本形はそこにあり、皆、それを議論のネタに使いながら、人々の考え方が進化してき

ていることは事実でしょう。

続いて、岸田さん。フラワー・モデルのお話です。まだ、ご自身でこのモデルを具体化してはいないが、それがどんなもので、どのような経緯で発想されたものかのお話です。最初になんだか花のような絵が出てきました。花の中心にあるのがプロジェクトの情報を全部入れたデータベース、そこから何かを取り出し、加工してまた元に戻す花びらの形の活動がプロセスの各アクティビティなのだそうです。Waterfall のようにそれぞれのフェイズの順序は決められておらず、いろいろな活動がランダムに発生するというイメージです。

まず、「モデル」という英単語の意味を再確認されました。モデルには、(1)標準的、模範的、(2)現実を何らかの形で映し出したもの、という2つの意味がありますが、プロセス・モデルについて議論するとき、この単語をどちらの意味で使っているのかをはっきりさせないと話が混乱すると指摘されました。そうかも知れません。

岸田さんがいつもいわれていることですが、ソフトウェア工学におけるプロダクト指向からプロセス指向へというパラダイム変化の紹介をされ、プロセスに興味を持てない人間はこの業界に向いていないのではないかといわれました。ただし、プロセスといってもいろいろある。例えば、プログラムの実体はプロセスであり、マシンの中でのその実行プロセスがある。そして、いまここで問題にしているソフトウェア開発プロセス、それらを含んで現実世界の中で進行するシステム進化のプロセスなど。

フラワー・モデルのアイデアは、1980年代の初めに、あるワークショップでの討論の中から生まれたのだそうです。要求定義についての議論の中で、要求仕様は与えられるのを待つのではなく、こちらから積極的にユーザの懐に飛び込んで作って行くものだ。そのさい、まったくゼロから始めるのではなく、これまでに経験した類似システムのイメージが最初にあり、それをユーザのあいまいなリクエストと照らし合わせて仕様の第1版ができる。そんな風に考えてみると、ソフトウェア開発のすべての活動は、過去の経験や知識を加工して新しい情報を作り出すということの繰り返しではないか、と考えたのが始まりなのだそうです。

そして、そのさいに行われる個々の活動は、決して Waterfall に示された順序には起こらない。たとえば、既存の類似システムに適当なデータを入力してユーザに見せ、それが要求とどの程度一致しているか(または食い違っているか)を確認するのは、要求分析の1ステップではあるが、実体はシステム・テストの変形である。そこで何か問題が生じてソフトのどこかを直してみようということになれば、当然プログラミングやデバッグという仕事が発生するが、しかし要求分析のフェイズはまだ終わっていない。

このように、あるシステムを開発する際に必要な情報(それはさまざまなドキュメントであったり、あるいは既存システムのプログラムであったりする)が、何らかの形で仮想的なデータベースを構成していて、そこから何かを取り出して加工し、また元に戻すということの繰り返しがソフトウェア開発なのではないかという考え方です。

岸田さんは、このアイデアを1980年代半ばに、ある国際ワークショップで発表したそうですが、「そのモデルはあまりに Context-Free 過ぎる」という批判を浴びてしまったとのこと。そこで、視点を少し変えて、「トラベル・プラン型モデル」というものを考えられたそうです。あらかじめスケジュールがきっちり決められたパッケージ・ツアーとはちがって、適当に何処と何処を見に行くという大雑把な旅程は作るが、歩いている途中に面白いところが見つかったらそこに立ち寄る(場合によってはもともとの予定を大幅に変更する)。このモデルのほうが、状況の変化に対応して変わってゆく開発プロセスの姿に近いのではないかという話で、なるほど、これは面白いなと思いました。

どうやら、フラワー・モデルという考え方のの核心は、あらかじめ定められた形式的な手続きの適用を集中的にコントロールするのではなく、開発に必要なリソースのひとつとして状況に応じて自由に利用できるようにしておこうというあたりにあるようです。

「世界」は無から作られるのではなく、すでに存在する既存の世界の「バージョン」を利用し、それを変形する形で作られるのだという、アメリカの哲学者ネルソン・グッドマンの思想を紹介され、ソフトウェアもまた、無からではなく、過去のシステムやそれを開発したときの知識や経験を再利用して作られる。そう

した開発知識を入れたデータベースのまわりで、一見ランダムにいろいろな活動が展開されるのがソフトウェア開発だという主張です。その類の知識ベースは、これまで仮想的なものでしかなかったが、そろそろ具体的に構築できるまでに技術が進歩してきたといわれました。

その1つの証拠として、岸田さんとも長いおつきあいのある Bill Riddle さんが、プロセス・マネジメントの仕事を対象として、フラワー・モデルのアイデアを具体化したヴィンセントという名前のシステムを、ドイツのフラウンホーファ研究所と協同で開発されたという事例を紹介されました。このシステムの名前は向日葵で有名な画家ヴィンセント・ヴァン・ゴッホの名前からとられたようです。

さて、以上 3 人のパネリストの発表が終わったところで、質疑応答の開始です。最初は会場とパネリストとの間の質疑応答でしたが、次第に会場の参加者同士のやりとりになり、パネリストがときどき間に入るという状況でした。時間と共に熱くなっていく感じが、SEA らしくてよかったと思います。まとめは、かなりわかりにくくなってしまいました。マインドマップ風に書ければよかったのですが、そのようなメモもとれなかったので、どうかご勘弁を・・・。

(1) Waterfall は現在も使われていますが、なぜでしょう？
  (1a) かたちはどうであれ、活動はある時間軸でしか起こらないのではないでしょうか？ 対象は論理的な時間軸である必要はないのですが。
  (1b) 結局のところ安心だからという要素が大きいのでは。管理者の立場からは、 安心しやすいのでしょう。
  (1c) 契約の問題が大きいと思います。
(2) 基本設計フェイズでは、何を意識して考えるべきでしょうか？
  (2a) 思考はいろいろなレベルで落ちてきます。人によってさまざまです。恐らく、知識・経験・性格によって考え方が異なるからでしょう。どうしても、やってみないとわからないという部分があります。ですから、安心できるところから始めるというのがポイントなのではないでしょうか？
(3) 現実の開発には複数の異なった会社が関係します。親会社、子会社、孫会社などの複雑な関係。そこに Waterfall 的な分業という構図が成り立つのは事実でしょう。
(4) フラワー・モデルで、開発のスターティング・ポイントは？
  (4a) 仕様分析とか、基本設計からは始まらない。似たようなシステムを見せて、ユーザが何を求めているのかを確かめてみる。具体的には、オペレーションと受け入れテストみたいな作業から入るのが自然でしょうね。
(5) たしかに、最近のシステム開発では、まず何をするかというと、類似システムの検索から入ることが多いのではないでしょうか？ いわば、知的財産の流用から始まる(笑)。時代も変われば、開発プロセスの入口も変わってくるように思います。
(6) 現実のアプリケーション開発はなかなかうまくいかない例が多い。ドキュメントがめちゃくちゃだったりして、問題が山積みになっている。結局のところ、マネージャがよいスタッフを集められるか、システムを見る目があるか、といったことがキー・ポイントで、プロセスは関係ないのではと思われる。
  (6a) エンジニアはよいシステムを作りたいと思っている。管理者を安心させることが大事か、開発者としての良心を大事にするかで、結果は変わってくるのでは？ 技術者としての良心を第一義に考えようという教育が大事なのではないでしょうか？

(7) Waterfall がなぜ使われるか。それは、失敗しないケースもあるからでしょう。たとえば、サムソンが日本の地方自治体の注文で作ったシステムは非常によいできばえだったそうです。全工程が15ヶ月のところ、最初のプロトタイプを5ヶ月で作ったらしい。これは修正 Waterfall のプロセスですね。
(8) 人間は言葉で100%分かり合えません。 だから、言葉の論理性を重要視しなければなりません。コミュニケーションが全てでしょう。計算機は馬鹿なので、それを相手にするには、正確なコミュニケーションが絶対に必要です。

(9) 先ほどから聞いていると、ソフトウェア・プロパーの話が多すぎるように感じます。たしかに何もないところからシステムは作れない。業務が出発点かもしれないし、ツールかもしれない、何か頼りになるものがあって、話が始まるのでしょう。そうした事情は、プロジェクトごとにまったく異なる。一番確実な部分、足元を見て、固まっているところから始めれていくのが現実的だと思うし、そのようにすればよいだけではないでしょうか。

(10) システムが何とか動いてしまえば、それが当初想定していた仕様と少し違っていても、それを正しいものとして受け入れるという考え方も、場合によっては「あり」なのでは？ ちょっと乱暴かな？

(11) 生産現場の能力に応じたプロセスがあるのではないでしょうか？ 設計が最初でもよいし、テストが最初でもかまわない。コミュニティに応じたプロセスを選択すればよいのでは？

(11a) しかし、結局、何故プロセスがうまくいかないかといえば、要求仕様を正しく文書化できる人間がいないことが問題なのではないですか？

(12) Waterfall モデルはすでに崩壊していると思おます。開発納期が短くなったために、現状では、もうそのまま適用することはできないのではないでしょうか？

(13) 組込みシステムの現場ではプロセスはどんどん変わっていく。しかし、事務系のアプリケーションの場合はあまり変わらないように思われます。たとえば、自動車の組込みシステムでは、メーカー自身が実際にアセンブラで開発し、それを部品メーカーが文書化したりしているのが現状ではないですか？ Waterfall とはちがうアプローチがとられています。

(14) わたしは、本来の Waterfall モデルは見たことがありません。「Waterfall とはいっているけど、ほんとう？」と思うものが多い。ドメインによって少しずつ違っています。

(15) 組込みソフトを手がけているわが社の場合も Waterfall といっています。しかし、実際には、そうはなっていません。組込みの世界には、まず HW の開発サイクルがあり、そのサイクルに従って、期日までにソフトをフィックスし、リリースしなければならない。これが非常に重要なので、その納期に実現が間に合わない機能は次期リリースにまわすというのが普通です。

(16) Waterfall がその通りにうまくいくとは思っていない。でも、それなりにやれば、うまくいくはずでしょう?

(17) CMM を導入することについてですが、現実が標準と乖離していたって、別に悪くはない。標準を逸脱するのはまずいと考えて、適当にお茶を濁すのが問題なのでは？ プロセスについても、決められたルールを変な風に誤魔化すのが悪いのでしょう。

(18) Waterfall はあくまで管理の視点あるいは契約の視点から見たモデルですね。エンジニアの作業という視点からみれば、あくまでもバーチャルなもの。管理者とエンジニアとでは視点が異なるということを意識すべきではないでしょうか？ Waterfall を本格的に適用するのは、プロジェクトの期間が少なくとも1年以上は必要です。期間が短いプロジェクトは、必然的に Agile になる。

(19) スパイラル・プロセスでは仕様の漏れは発生しないのでしょうか？

(19a) 要求が不明の作業では、とりあえず動くものを提供し、そこで次のゴールを設定していかないと、話が始まらないでしょう。プロセスはケースバイケースで選択すべきだと思います。

(20) V字モデルが会社の標準だと教えられたとき、それぞれのプロセスやアクティビティがなぜそうなっているのかは説明されませんでした。CMM、ISOでは、レベルが上がれば組織は成熟するという考えです。しかし、個人の成長はどうなのか？ プロセスはこうあるべきと人に教えるとき、その意義をどう教えたらよいのかが問題ですね。

(21) 仕事の中で、ものを考えないエンジニアが多い。そのプロセスがなぜ必要なのかを意識せずに漫然と仕事をしている人間が多いのが現実です。

(22) ISOにおけるプロセスの定義は、「入力を出力に変換するもの」、それだけです。Waterfall モデルが議論された時代は、まだ、開発の規模も比較的小さく、内容も現在のように複雑ではなかった。いまは、環境も大幅に変わってきた。環境に応じてプロセスの定義を再考すればよいのではないでしょうか？

(23) ある組込み系の現場での話。プロジェクト・リーダにヒアリングしてみると、「われわれは Waterfall

を実践しています」という。しかし、エンジニアに訊くと、「スパイラルで仕事をしています」と力説する。リーダは管理の視点、エンジニアは開発の視点でプロジェクトを見ているので、同じ現場であっても違って見えるのでしょうね。

(24) スパイラルも、伸ばせば直線になる(笑)。

(25) 世の中でCMMの是非についていろいろ議論がありますが、それはあまり意味がないと思う。モデルのパターンはさまざまです。開発が最終成果にいたるまでには、いろいろな道程がある。しかし、どんなやり方をとるにせよ、何らかの評価は必要です。CMMでいわれている成熟度レベルはその1つの参考例。何もあの通りにしなければいけないというわけではありません。

(26) プロジェクトにおいて、何に価値を置くのか？　組込みプロジェクトの場合は、システムとして意味があるかを第一義に考えているように思います。ソフトウェアだけを考えるのではなく、システムとしてのビジネスを重視している。プロセスを考えるときに、そうした視点がどこにあるかを意識することも重要なのではないでしょうか？

(27) ソフトウェア・エンジニアにとって、実は、でき上がったプロダクトは仕事の目的ではないのだと思います。プロダクトはあくまでユーザのためのもの。SE にとっては、それは仕事をする上での例題にすぎません。目的は、そうして次々に与えられる課題としてのプロダクトをどのようなプロセスで作ったらよいかを考えること、つまりプロセスの開発こそが仕事のほんとうの目的なのではないでしょうか？

(28) 大規模な組込みシステム開発の場合には、数年間のスパンを考えて活動します。通常の製品よりもテスト期間が長いので、開発手法は固定していません。ベースはXPで、Waterfall 的なプロセスを1週間程度のサイクルでひたすら繰り返します。プロジェクトに関わっている時間が長いために、よい製品とは何かについて、また、次回のサイクルでのプロセスをどうするかについて、といったことをつねに考えることができる。このように製品やプロセスについて考え続けることで、それらを自分たちのものだと思うことができるわけです。何か間違いを犯したときも、それを認識して改善することができる。それが大事だと思います。

(29) 新規開発の場合に、Agile 方式でものが作られるという状況が想像できないのですが、ほんとうにできるのですか？

(29a) できます。テスト検証、リファクタリングのフェイズがあるので、可能です。

(30) プロセスの幹と太い枝は、Waterfall 型のモデルで作り、枝葉はアジャイルでよいのではないでしょうか。

(31) Waterfall では問題が表面化することが遅いので、現場の課題解決にマッチしない。そういう意味で、Agile に興味があります。トヨタ生産方式を実践すると Agile になっていくと聞いたことがありますが、そうなのでしょうか？

SEA Forum April "WaterFall 再考 "

参加者のアンケート分析

田中 一夫
SEA 代表幹事

4月3日に開催された、SEA ForumApril「WaterFall 再考」に参加された方々に、メールでアンケートを発送し、回答を頂いた。今回の参加者は46人で、内訳は SEA 正会員21、賛助会員3、一般22であった。

1.回答率:68%。44人に発送し、30人からの回答をいただいた(正会員15、賛助会員2、一般13)。

2.今回の開催をどのルートで知ったか?

SEA からの案内	11 (37%)
SEA の Web ページ	0
SEA からのメールの転送	3 (10%)
IPA/SEC からの案内	6 (20%)
知人からの紹介	7 (23%)
その他	3 (10%)

開催案内を知る手段として、メールが中心になっていることがわかる。「SEA からのメールによる案内の転送」が 3 人いるが、いずれも正会員なので、SEA からの案内と同等だと思われる。今回の Forum から、IPA・SEC に案内を出していただき、6 人の方が、このメールで参加された。

3.WaterFall Model を正しく知っていたか?

以前から知っていた	7 (24%)
今回再認識した	18 (62%)
その他	4 (14%)
(合計 29)	

「以前から知っていた」あるいは「その他」の中には、今回の Forum の案内で、原文の所在を知り、読んだ人もいるので、論文とは違った Model をほとんどの人が認識していたことになる。

4.討論について

有意義であった	19 (73%)
無意味であった	0
その他	7 (27%)
(合計 26)	

「その他」の意見には、「落水先生が指摘した問題点について討論すべきだった」とか、コーディネータの進め方に問題」という指摘がいくつかありました。

5.今後、開催して欲しい Forum は？

みなさんから寄せられた希望は以下の通り(順不同)：

1) アジャイル再考！
2) 開発プロセスの現場的(研究的ではなくて)な比較をあつかったものが 良いと思います。
3) グローバルに通用する Package ソフト
4) ソフトウェア輸入大国から輸出大国を目指して
5) エンタプライズと組込みの開発・管理手法の相違
6) 組込みがソフトウェア開発の手法を変える
7) 要求工学って有意義なの？
8) SOX 法が与えるインパクト
9) SOA ってどのように適用するの？
10) モデルベース改善がよいのか、課題ベース改善がよいのか？
11) プロセス改善(PJ 管理系)とプロダクト改善(設計方法論系)の融合方法
12) ソフトウェア開発プロジェクトを効率よく行うための契約のあり方（具体的には、定額請負方式でもなく、コスト償還型でもない インセンティブ付きの新しい契約として、どのようなものが考えられるか)
13) XPのプラクティスをレビューする（実際にXPを実施している技術者に、各プラクティスの持つ意味と その実効性について評価をしていただき、議論する)
14) ソフトウェアアーキテクチャとソフトウェアプロセスは関係あるのかないのか？
15) オープンソース
16) SOA の実際
17) 組み込みシステム開発のプロセス改善
18) 学術的ではなく、実践的な取り組み
19) 「How To？」ではなく，「Why？ What？」を議論する
20) 落水先生が提起されていた「方法論とプロジェクトマネジメントの融合」について ぜひもう一度テーマとして取り上げていただき、もう少し突っ込んだ議論を深めたい。
21)下記の視点を絞ったフォーラム：
視点，論点を明確に絞って，各視点毎の掘り下げた議論に入っていただけるとありがたい．
例えば，
a.開発手順の視点であれば，
- XP の場合のアクティビティがSLCPのアクティビティとどう違うのか？
(かなり違うと思うので)それは，どのような条件では有利で，どのような条件では困るのか？
・開発プロセスと品質保証プロセスに分け，開発プロセスだけXpを使うのか？
・ユニファイドプロセスとの関係，即ち方向付け，推敲，作成，以降のフェーズの概念とウォーターフォールとの違い，XP との違い．
・組込みのように毎年機能強化を繰り返すプロセスはマクロにはインクリメンタルと考えているのか？ それとも，毎回ウォーターフォール開発と考えているのか？
・要求定義をうまくやるために，テスト用のプロとタイプを作って確かめるという場合，ウォーターフォールではそれを要求定義のアクティビティと考えるのか？ 要求定義と実装のアクティビティと考えるのか？ XP ではどう考えるのか？
などを予め質問リストとして作っておき，パネラーに答えてもらい， その上で各手順のメリットデメリットや，今後の課題の議論に入っていただけると大変ありがたいです．
b.開発技術の視点であれば，
- アーキテクチャ中心の開発と，XP のリファクタリングを用いる開発との違い。そして，必要な

スキルの違い

・使い捨てプロトタイプと製品に進化させるプロトタイプの違い。そして、ウォーターフォールとXP でのやり方の違い

・トレードオフ関係を調整する方法の違い。例えば，予算，期間，機能，性能のトレードオフをどのように決定，調整，修正するかなど

なお，文献がダウンロードできる場合は，その解説は端折って，議論の時間をふやしていただいたほうがよいです．

22) 形式的仕様記述
23) 品質(品質管理、品質保証)
24) プログラミングっぽいもの。青木さん、酒匂さんの6月の Forum に期待しています。
25) レガシーシステムの保守・メンテナンスについて
26) アーキテクチャ，コンポーネント技術と現状。
サービスオリエンテッドアーキテクチャの思想と技術。
27) どうしたらソフトウェア・エンジニアが新技術を試行するための余裕を持てる？
28) Agile や XP 等の手法を用いて実際に行われた開発プロジェクトの事例．
(日本の)ビジネスと OSS との関係についての事例
29) 高度なエンジニアの育成など教育関係のテーマ
30) 今回の議論でも、開発当初にすべての要件を洗い出すことは、ビジネスの変化が激しい現在，何らかの形で動くプロトタイプがないとフィードバックを得られないのでむずかしいということが，暗黙のうちに合意されていたと思います。
開発プロセスの整備やオブジェクト指向などの新しい手法の導入により、ソフトウェア開発の生産性はあがっているのかどうか？ もし，あがっているとしたら，それはどの程度で，その理由は何か？ 今後、画期的に生産性を向上させる方法があるのか？ コンポーネント技術、SOA、MDA などはどうか？ など。
31) 今回は、組込ソフトに携わる方が多かったようで、WaterFall Model はかなり悪役になっていましたが、開発プロジェクトを推進させるということに関してではよい面もあると思いますので、プロジェクトをどうやって成功させるかという観点で，各モデルの比較討論などはいかがでしょうか？
32) 伊藤さんをメインのスピーカにした Forum。SPO とか。
33) Wiki を使った情報共有
34) (ソフトウェアの)QC 活動事例
35) ソフトウェア・エンジニアリングに関するトレーニングや実践の事例

6.その他の面白い意見

ビル建築もアジャイルでやってみるといいかも！？

編集後記

☆

船便状態からなんとか脱却すべく努力中です．

☆☆

巻頭の玉井先生のエッセイはソフトウェア工学のこれからについての展望です．お読みになった感想や異論・反論など，お寄せいただければ幸いです．

☆☆☆

名古屋支部の石川さんからは支部で開いたForumのレポートをいただきました．それぞれの支部あるいは分科会ミーティングの報告をお待ちします．

☆☆☆☆☆

次の旅日記は，年度末の仕事をサボってヨーロッパにでかけた編集子の備忘録です．ここに出ているウィーンでのワークショップについては，真面目に参加された野中哲さんから，いずれきちんとした報告があるでしょう．

☆☆☆☆☆☆

最後は4月3日に開催されたSEA Forumの記録です．新年度の始まりの日だったにもかかわらず，大勢の参加者が集まって熱心な討論が行われました．

☆☆☆☆☆☆☆

このところのSEA Forumは，引続いて，なぜかいずれも満員の盛況です．次号では5月17日および31日のForumの報告，そしてICSE2006 in 上海の報告をお送りする予定です．

☆☆☆☆☆☆☆☆

SEA Forum June

オブジェクト指向の理想と現実

主催: ソフトウェア技術者協会 (SEA)
http://www.sea.jp/

オブジェクト指向は、多くのソフトウェア開発プロジェクトにおいて日常的に適用される開発手法となっています。多くの場合、適用の目的は生成したコンポーネントの再利用率を高めることにより、高生産性、高保守性を狙うことにあると考えられます。この傾向は、オブジェクト指向分析/設計/開発を効率的に実施する技法とツールの普及によって益々拍車がかかっています。

しかし普及と成熟の一方で、誤解や混乱が拡大しているとの指摘も挙がっています。例えば、開発フェーズにおいてオブジェクト指向言語を採用しているものの、一つのクラスしか存在しないプログラム、1メソッドが数百行のコードから成るプログラム、責務の混合したプログラム、相互に依存度の高いコンポーネント群などの存在を容易に許してしまいます。このような問題は、プロジェクト管理や品質管理体制の強化によっては解決することが困難な側面があります。つまり、分析設計開発の各フェーズにおいて支援ツールの採用実績を挙げ、テストを通過し、納期が守られているというポイントはクリアしているため、見かけ上はオブジェクト指向開発プロジェクトは成功したと評価されますが、実際は手続き型言語と変わらない設計、開発が行われているケースも観測されています。

オブジェクト指向の理想と現実の乖離に焦点をあてた今回のフォーラムでは、黎明期よりオブジェクト指向手法に親しみ、理論家であると同時に優れた実践家としても知られるお二人の講師をお迎えしました。この場で私達は、オブジェクト指向が正しく適用された世界がどのようなものであるかを再認識すると同時に、実際の開発現場においてその世界に到達するにはどうしたらよいかをディスカッションしたいと思います。

実際の開発プロジェクトにたずさわる、多くのソフトウェア技術者の参加をお待ちしています。

********** 開催要領 **********

1. 日時:2006年6月16日(金) 13:30受付, 14:00開始, 17:30 終了
2. プログラム
   13:30 - 14:00 受付
   14:00 - 16:00 講演 (1) オブジェクト指向において保存・変形・視座・多重・名前・位相を 青木淳(SRA 先端技術研究所)
   オブジェクト指向プログラミングを20年以上も続けてきた観点から、保存・変形・視座・多重・名前・位相について言い及びます。現在、日経ソフトウエアに連載されている「青木淳のプログラマ魂」のトピックスになります。それぞれについて、どのようにオブジクト指向で昇華されているのかを開示してゆきます。一介のプログラマが到達したひとつの世界観(理想と現実)の紹介になります。
   (2) オブジェクト指向の理想と現実(「使えない」モデルはいらない) 酒匂寛(Designers' Den)
   オブジェクト指向開発においては「モデル」が話題になることが多いのですが、現場におけるそのモデルの位置付けは曖昧です。多くの場合初期段階にスケッチのように書かれて、あとは保守もされないという場合が多いのではないでしょうか。しかし、こうなる理由は明らかで、書かれたモデルが「検証」できないことに起因しています。多くの開発プロジェクトの混乱が仕様(モデル)の混乱に起因することを考えると、これは大変残念なことです。今回は形式手法をプロジェクトに導入することによって、モデルを活性化する方法について、事例を交えながらお話します。
   16:00 - 16:10 Break
   16:10 - 17:30 討論:オブジェクト指向の理想と現実 コーディネータ: 桜井麻里(SEA 幹事)
   パネリスト:: 上記の講師のお2人
3. 会場: 全国情報サービス産業厚生年金基金(JJK)会館 7F B会議室 (東京都中央区築地 4-1-14)
   http://www.jjk.or.jp/info/guidemap.html
4. 定員: 50名 (申込み順, 定員になり次第, 受付を締め切ります)
5. 参加費: SEA 正会員 2,000円, SEA 賛助会員 3,000円, 一般 4,000円
6. 申込み方法: 下の申込み票に必要事項を記入し, SEA事務局 sea@sea.or.jp まで e-mail(テキストメールのみ: HTMLや添付ファイルは不可)でお申込みください. 折り返し受付確認の mail を返信します. 当日会場1階の受付デスクで, 受付番号とお名前を申し出てください. 入館証(バッジ)をお渡しします. 参加費は当日現金でお支払いください. 領収書をさしあげます.

SEA Forum June (6/16 @ JJK 会館) 参加申込み (2006年 月 日)

----------(Mail to: sea@sea.or.jp テキストメールに限る)----------

氏名:_____ ふりがな(_____)
所属:_____
住所:_____
電話・FAX:_____/_____
E-mail:_____
種別(いずれかにチェック・記入): □ SEA会員(No:) □ SEA賛助会員 (会社名:) □ 一般

*注:賛助会員企業一覧:http://www.sea.jp/sanjo.txt

ソフトウェア技術者協会
〒160-0004 東京都新宿区四谷3-12 丸正ビル5F
Tel:03-3356-1077　Fax:03-3356-1072
E-mail:sea@sea.or.jp
URL:http://www.sea.jp/