解　説

IPS

アジア・太平洋におけるソフトウェア技術

11. 問題，方法，そして特化†

Michael Jackson ††
(翻訳編集：田村　恭久†††)

1. 願　　望

ソフトウェア工学という学問を創り，実践に活かそうとする願望は2001年で33歳，すなわち1世紀の3分の1を経過する．しかし，現在までの26年間を考えると，あと7年でその願望を達成できると考えるのは難しい．ソフトウェア工学は，すでに確立した工学の他の分野のようにはなっていないし，近い将来そのようになることもないだろう．

この理由の一つには，ソフトウェア工学が難しいということがあげられる．

もう一つの理由は，プロフェッショナリズムの欠如である．ソフトウェア開発は，数学的プログラミングから発生した．これを実践したのは数学者，宇宙科学者，物理学者である．彼らは，プログラミングを，彼らの専門分野に比べて些細なものであるとみなした．やがて，大型汎用機の文化のなかで「ユーザのプログラミング」と「プロのプログラミング」の境界ができたが，後年，この境界はパソコン文化が主導権を握るにつれ曖昧になる．

Visual Basic, Paradox, Lotus 1-2-3 などのツールは，プロの仕事としてのプログラミングを想定しているし，一方エンドユーザの「アマチュアの」プログラミングも想定している．確立した工学分野で，このような二面性をもつ領域はない．自動車や橋梁を「アマチュア」として開発する人はいないだろう．しかしソフトウェア開発では，「アマチュアの」開発と，プロが仕事として行う開発を明確に区別するのは難しい．この結果，プロのためのソフトウェア工学やソフトウェア工学の規則作りを目指した動きもあったものの，ソフトウェア開発は，依然としてアマチュアの仕事と言わざるをえない．

われわれがエンジニアになりたいという願望が果たせないもう一つの理由は，理論の研究者も実践の研究者も，ソフトウェアに対して素朴すぎることにある．すべての物質が生成可能であるような万能物質，すべての問題が解決可能な万能方法論などの特効薬を不断に追求していることは，われわれが対象分野のさまざまな性質すらまだ理解していないことを示唆する．「われわれはプロフェッショナリズムを欠く」と強く不平をもらす人々でさえこの素朴さの例外ではない．たとえば，形式主義（formalism）の主導者はソフトウェアがより一層の注意，数学的な厳格さ，正確さをもって開発されたならば，しばしば起きる言語道断なエラーのいくつかは回避できる，と主張する．しかし，そのような処方が完全かつ万能の解決法として，あまりに安易に提案されていた．われわれの問題点は，もっと深いところにある．

2. 成功の処方

いくつもの処方が，われわれの大変不健康な状態を癒す処方箋として提案されてきた．

一つの強力な薬は，再利用可能なソフトウェア部品を開発し，それを用いて対象のソフトウェアを開発する手法である．これは電子工学などの例を踏まえたものである．いくつかの特定の環境（数値計算のサブルーチンやGUIのオブジェクトクラスはこの代表例である）では，この方法はある程度成功を収めた．

しかし，日常のプログラミングでは，まだソフトウェアを部品の集合にまで分解する努力が実っていない．ある場合は，要求仕様が特殊すぎ，利

† Problems, Methods and Specialization by Michael JACKSON (An Independent Consultant).
†† 独立コンサルタント
††† 上智大学理工学部

用可能な部品のインタフェースや機能が要求とかみ合っていない．あるいは，悪いことに要求が一般的すぎるのである．読者がこの文章を読んでいる間にも，何人のプログラマが線形探索のプログラムを組んでいるだろうか？そして間違いなく，そのうちの何人かはバグを作り込んでいるのだ．

形式主義の支持者は，しばしばソフトウェア開発において数学的な側面がしばしば無視されてきたことを指摘する．コンピュータのプログラムが形式的なものだということは疑いの余地がなく，したがって数学的な考察の対象となる．彼らは，確立された他の工学分野にならおうとする．構造工学のエンジニアは，応力を計算する．自動車のエンジニアは，トルクを計算する．彼らは，ソフトウェアのエンジニアも計算をしなければならないと信じている．

正確な計算が可能であるのに，曖昧さ，混乱，不確定なものを許容するのは，明らかに合理的でない．しかし，ソフトウェア開発の多くの分野においては，数学的なアプローチをとるための必要条件を満たしていない．確立された工学分野におけるエンジニアは，明確に定義された，狭いコンテキストのなかで計算を行う．たとえば自動車のエンジニアは，設計に着手する際にタイヤを4本使うか，8本使うかは計算しない．何十年にもわたる経験を基に，標準的な設計のレパートリがすでに存在している．新しい設計の一つ一つも，標準からのわずかな変更に過ぎない．実際には，設計の空間は非常に狭いのである．しかし，ソフトウェア開発の大抵の分野では，このような確立された標準的な設計というものがない．形式的な方法が提示するソフトウェア構造は，設計の空間を絞り込むには一般的すぎるのである．

まず最初に問題を理解し，解の構造化から始めるべきである．それがなされて，初めて計算が可能になるのである．厳密な記述と計算は，2番目にくるべきものである．それは，作業のなかで決定的に重要となる場合があるかも知れないが，それが常に問題の中心的なものであるとは限らない．構造化方法論やオブジェクト指向方法論は，ソフトウェアの構造を設計するための方法とされている．たいがいの場合，これらの方法すら一般的すぎるのである．ソフトウェアの一つ一つの断片は，ある意味ではデータフローダイアグラムで記述され得る．すべてのプログラムは，多かれ少なかれオブジェクト指向言語で記述され得る．オブジェクトやデータフローは一般的な計算のパラダイムである．どのオブジェクトが必要なのか？どの関数とデータストリームが？これらの方法論が与える設計空間は広すぎて役に立たない．

3. 問題と解

David Garlan と Mary Shaw は，そのアーキテクチャアプローチ[1]のなかで，共通のアーキテクチャのスタイルを分類し，研究することで，設計空間を絞り込む方向を示している．アーキテクチャの例として，パイプ，フィルタ，黒板を共有する複数プロセスなどがある．彼らのアプローチは，プログラミングのクリシェ[3]，ソフトウェア方法論の研究家のお気に入りの建築家である Christopher Alexander の研究[4]に基づくパターン言語のアプローチ，などと共通点がある．

これらすべての研究は，解がもっている特徴や構造に焦点をあてることで，確立された他の工学分野において見られる標準的な設計の確立を目指す．しかし，それは，問題の特徴を明らかにするという補完的であるが多分もっと重要な作業を無視している．解に着目する方法は想像以上に広く普及している．

いくつかの方法は，解の記述に傾注する時にも問題を分析し，構造化すべきと主張している．Johnson はこの理由として，解の構造は問題の構造より容易に理解できる，と述べている[5]．プログラミング言語と環境は，必然的に概念と用語を与える．たとえば，関数，手続き，プロセス，データ構造，型，メッセージ，呼び出し，ファイルなどがある．これらの概念と用語は，われわれがソフトウェアによる解を記述しようとしたとき，いつでも使えるものである．問題を取り巻く環境は，これに比べてあまり役に立たないし，示唆するものも少ない．

問題から目をそらさせる他の要因は，ソフトウェアシステムはしばしば問題ドメインの部分的なシミュレーションであるということである．よって，開発者は，ソフトウェアの記述は問題の記述である，という主張に容易に納得してしまう．しかし，それは自動車のハンドルの記述が，ドライバの腕と手の機能的な記述であると言うに等

しい．

解決すべき問題に焦点を当てることは，すでに確立された他の工学分野では必然的である．というのは，標準的な材料，技術，レパートリは，非常に限定された分野にしか応用できないからである．自動車のエンジニアは自動車の設計をするものであり，橋梁を設計してくれと頼んでも無駄であろう．しかしソフトウェア開発の素材と技術は，標準的な設計がないため，応用分野が非常に幅広い．このことを，当たり前だと考えてはいけない．問題は，明確に認識され，分析されるべきである．

4. 問題のコンテキスト

Fred Brooks[6]の「銀の弾丸はない」という講演に対し，Wlad Turski[7]はソフトウェアの開発は形式的なコンピュータシステムの開発だけでなくさまざまな考慮が必要であると指摘した．問題をそのコンテキストに沿って認識，理解，分析することは，不可欠かつ本質的な作業である．したがって，ハードウェア開発と異なり，ソフトウェア開発は，銀行業務，電話交換機，航空機の管制や内部制御，プログラムのコンパイル，表計算，テキストの整形と表示などに関わる．すなわち，プログラムやシステムに関するすべてのものなのである．

一目すると，これはソフトウェアの開発者がアプリケーションドメインに精通しているか，あるいはこれから精通しなければならないと受け取られるかも知れない．実際にはそうではない．ソフトウェアの開発者はアプリケーションドメインのうちソフトウェアの設計や構成に影響する事柄に精通しているべきである．屋内暖房のエンジニアは，建築家であることは期待されていないが，建物のさまざまな特性のうち暖房に関する事柄，すなわち外界の気候の影響，ドアや窓の位置などに精通している必要がある．これらの事柄は，暖房システムの設計上，必要な熱量と設計上の長所と問題点を明らかにする．

ソフトウェアの開発者にとって，アプリケーションドメインの重要な側面は，そのドメインを記述するのがどの程度困難なのか，また必要ならばシミュレートしたり制御したりすることがどの程度困難なのかである．よって，ソフトウェアの開発者は，現象学（ドメインの主な現象とその関係を認識すること）に秀でていなければならない．また，記述する技術（因果関係などの難しい属性を認識し，それらを必要十分に記述すること），形式化の能力（非形式的な現実に対して，用途に適した十分に知的な制御をもたせるために，必要に応じて抽象化や一般化を用いて，表現すること）も要求されている．

結局，当然ながら，問題は解の仕組みを構成することにある．ソフトウェア開発でも同様で，その仕組みは実際の世界のなかの特定のコンテキストに適合する必要がある．ここで，かかるコストとそれによって得られる利益が感得され，評価される．

5. 問題フレーム

問題のコンテキストを認識することは，問題自体を理解することの第1段階に過ぎない．古代ギリシャの数学者は，問題の研究に多くの注意を払ってきた．また彼らは，それを問題の解や解法とは切り離された主題として扱っていた．Polyaによる立派な本[8]には，これに関する記述が載っている．ギリシャ人たちは数学の問題を，発見的あるいは構成的問題と証明問題に分類した．たとえば，発見的問題として次のような問題がある．

長さ a, b, c が与えられたとき，3辺の長さがおのおのの長さになる三角形を作れ．

また，証明問題としては次のようなものがある．

四辺形の4角が等しいならば，その角度はすべて90度であることを証明せよ．

さらに，すべての問題は主要部と解を持つ．証明問題の主要部は，四辺形の4角が等しいといった前提とその角度はすべて90度であるといった結論から成る．解は，前提から結論を導くことである．構成的問題の主要部は未知である．三角形とそのデータ（長さ a，b．c），それにデータとして与えられた辺の条件である．解は，データとして与えられた条件を満たすものを構成することである．主要部は，問題の一部であって，解や解決の段階ではない．問題の主要部によって解くべき問題の言葉で解決方法を議論できる．

Polyaは問題の種類に応じた方法論を提案している．証明問題に対し，次のように示唆している．「同じ，あるいは類似した結論をもつ定理を考えてみなさい」，「仮説が与えられたとき，結論が真か

偽のどちらに近いかを考えなさい」，「その仮説から導かれる他の結論も考えてみなさい」．発見的，構成的問題に対しては，次のように示唆している．「条件を分割しなさい」，「すべてのデータを使用していることをチェックしなさい」，「未知数のバリエーションを考え，データに近づけなさい」，「条件が未知数を決定するのに十分かどうか吟味しなさい」．

問題の主要部と解は，問題をシステマティックに考え，適切な解法を選べる構造を形成するはずである．このような構造を，問題フレームと呼ぶ．

問題を理解するということは，ある問題の主要部と解を見出し適切な問題フレームに当てはめることである．これが，問題分析で行われる主要な活動である．

6. ソフトウェア開発の問題フレーム

問題フレームの考え方は，ソフトウェア開発に直接適用可能である．もちろん，問題のコンテキストから始める必要がある．すなわち，実世界の関連要素を認識し，その属性と関係を評価する．数学では対象領域の豊富な知識を，一言で表現できる．たとえば，それは三角形と言えば，それ以上説明しなくても読者はそれが何を意味しているか分かると期待できる．また三角形の性質について多少の知識はあるだろうと期待する．

しかしソフトウェア開発では，数学の抽象的な対象領域に比べ実世界の領域ははるかに豊かで多様である．それらの性質は，注意深く記述を行うことで明らかにされなければならない．Turskiが指摘するように，それはしばしば形式からはずれたものであり，どのような言語システムを用いても性質は記述しきれない．それゆえ，ソフトウェア開発者は，実世界の領域における重要な性質を探し出し，それを適切な言語で形式化する技術をもたねばならない．

Polyaの2種類の問題は，彼が議論している小さな問題のクラスの問題を構造化することさえ不十分である．ソフトウェア開発の問題はもっと多様で，二つ以上の問題フレームが必要である．しかし，明確に記述された問題フレームはほとんどない．ここでいくつかの問題フレームを抽出し，概要を記述する．それらはジャクソン法フレーム，ワークピースフレーム，環境－効果フレームである．

（1） ジャクソン法フレーム．

ジャクソン法[9]は情報システムを開発するために適した問題のフレームを用いる．この主要部は以下のものである．

・実世界：システムが情報を生成するところの，特定の世界である．これは問題のコンテキストのドメインである．また動的なものであり，イベントやそれによって状態遷移が起こるような，時間軸上の挙動を含む．実世界は自律的である．イベントの発生や状態遷移は自然に起こり，外部から引き起こされたり，制御されたりはしない．特に，機械によって制御できない．この問題のフレームを用いる開発者にとって，実世界は変更や制約を加えることはできず，単に観測，記録するだけのものである．

・情報出力：実世界について必要な情報の出力．問題のコンテキストの一部である．

・要求：情報システムのユーザが要求する情報．これはコンテキストの一部である．この要求は時間に沿ったイベントの構造化されていない系列である．ユーザはこの構造化されていない系列の元としてのみ捉えられる．たとえば，個々のユーザは区別されず，ドメインはいかなる内部状態ももたないものとして扱われる．

・システム：構築されるべき仕組みである．これは実世界，情報の出力，要求と結ばれている．システムは要求に対応して，あるいは実世界の状態や挙動によって自律的に情報を出力する．

・機能：システムが実世界，情報の出力，要求の間で果たすべき関係．情報は，実世界であるイベントが発生した場合やある条件が成り立った場合に，あるいは，要求に応答して出力される．

ジャクソン法フレームの解とは，実世界をモデル化，あるいはシミュレートし，機能を満足するようなシステムを構成することである．

（2） ワークピース・フレーム．

これは機械をテキスト形式やグラフィック形式の書類を生成するツールとして扱う．単純なワープロなどのアプリケーションに適している．これは次の主要部からなる．

・ワークピース：図形や表が埋め込まれているテキスト文書のような作業対象のオブジェクトを指す．問題のコンテキストのなかで，ワークピース

は実体のないドメインである．この状態を変化させることはできるが，外部からの働きかけの結果によってのみ変わり，自律的な挙動はない．

・操作要求：システムのユーザがワークピースに対する操作の要求．これは問題のコンテキストのドメインである．ジャクソン法フレームにおける要求のように，操作の要求は非構造的なイベント系列として扱われる．ここでは，個々のユーザやドメインの状態は区別されない．

・操作：ユーザが機械に対しワークピース上で実行するよう要求する操作．操作要求とワークピースドメインの状態を関係づけるもの．

・機械：構築されるべき機械．ワークピースを具体化したものを含み，操作要求に応じた操作をワークピースに施す．自律的な挙動はない．

ワークピース・フレームにおける解とは，操作要求に対してワークピース上に操作を施す機械を構成することである．

（3） 環境－効果の枠組み．

これはDavid Parnas と Jan Madey が提唱したアプローチ[10]と共通するものをもっている．これは組込み型システムで，外部のドメインを制御するものに適している．この主要な部品は以下のものである．

・環境：制御されるべきドメイン．状態と挙動をもち，部分的に自律的で，部分的に外部からのイベントに反応する．

・接続：構成されるべき機械と環境との間の関係．機械は状態とイベントを検知でき，影響を及ぼす．問題のコンテキストのドメインである．

・機械：構築されるべき機械．この挙動は半ば自律的であり，半ば環境のイベントや状態遷移に反応するものである．

・要求：機械が実現し，維持すべきドメインの性質と挙動（環境において起こる事象間の関係）．

環境－効果における解とは，接続を介して環境の状態を検知し，制御するような機械を組み立てることである．これは要求を満たしている必要がある．

以上，いくつかのフレームを簡単にスケッチしてきたが，これらのフレームが相互に変換可能でないことは明らかである．問題フレームと問題を適合すべきである．問題のコンテキストのドメインは，対応する主要部の特徴に反映しているべきである．また，直接/間接いずれかの方法で相互に，かつ機械と接続されている必要がある．フレームは，すべての要求されている性質と関係に適応するような，適切な要素をもつ必要がある．よって，ワークピースの枠組みは化学プラントの制御システムの開発には役に立たない．なぜなら，ワークピースのドメインは実体のない，化学作用などのないものを仮定しているからである．ジャクソン法のフレームは組込み型のシステムには不適切である．なぜなら，実世界のドメインは自律的であることを仮定しており，組込み型のシステムの環境は機械によって制御されるからである．

7. フレームと方法

問題フレームの主要部は，開発方法の素材を与える．一つの開発方法は一つの問題フレームを規定し，問題のコンテキストのドメインとそのフレームの主要部を認識する指針を与える．また，主要部の記述から始まって記述すべきことを規定する．さらに，記述の順番や，使うべき複数の言語，抽象化，合成，変換，修飾，洗練といった操作も規定する場合がある．この操作によって，既存の記述から新しい記述が導かれる．もちろん，最終的な記述は組み立てられるべき機械の記述となる．

良いソフトウェア開発方法は，非常に絞り込まれた問題フレームを規定し，その性質をフルに利用する．主要部の特徴を知ることにより，記述に用いる言語を選択できる．たとえば，ジャクソン法フレームの実世界は，図表現された正規文法としてCSP（Concurrent Sequential Processes）で表現される．問題のコンテキストのドメイン間の関係を知ると，簡単な方法で記述をすることができる．たとえば，ワークピース・フレームのワークピースは，抽象データ型のインスタンスとして記述でき，操作はその型に対する操作となる．潜在的な難しさが分類され，具体的な解が得られる．記述のための技術と表記方法は，記述対象の主要部とその間の主要な関係に当てはまるよう，鋭く研ぎすまされている．

問題フレームが問題により適合すればするほど，また解決方法が問題フレームと主要部の性質をより明確にすればするほど，問題解決のための障害を識別し，克服する力が増す．単純な問題と

は，密接に関係した問題フレームを持ち，効果的な開発方法を持っている問題である．

8. 複雑さと統合

ソフトウェア開発の問題は，少数の問題フレームの集合では捉え切れないほど多様で内容が豊富である．各問題に対して，二つ以上の問題フレームが必要である．CASEツールの統合は，ワークピースにきれいに当てはまるかもしれない．しかしユーザがこのツールを使って作業の進捗情報を得ようとすると，ワークピース・フレームとジャクソン法フレームの両方が必要となる．ワークピース・フレームはツールの基本的な機能を構築するため，ジャクソン法フレームはツールユーザの進捗に関する情報を生成するために使われる．一つの問題に対して，二つのビュー，二つの開発方法が適用でき，二つの解を統合する必要がある．もしCASEツールがユーザに制約を課す必要があるならば，環境一効果フレームが適用されるべきである．ワークピース，要求，操作といったものが，このフレームの環境を構成するだろう．

もちろん，こういった複雑さは適用可能な問題フレームとそれに対応する開発方法論のレパートリに関係する．この複雑さを認識し，解消することは，問題を部分問題に分解していく本質となる．しかしこの分解作業は，何が部分問題を構成しているか，という明確な考えに基づくべきである．部分問題とは，適切な問題フレームと有効な解が知られているような問題をいう．

伝統的に，分解するということは，自己充足的立場に立っている．しかし，それは正しくない．もし問題が部分問題に分解されるならば，部分問題の解は，一つの解へ統合できるはずである．分割して征服したからには，統治するために，統一する必要がある．この分割の結果は，解が単一の計算パラダイムの中で分解されるまでは明らかでない．手続きの呼出しにより，全体は無限の深さと広さをもつ階層へ分解可能である．メッセージパッシングにより，全体は無数のオブジェクトとインスタンスへと分解可能である．単一のパラダイムのなかであれば，統合は単純な作業である．

しかし，もう少し特化した問題フレームと解の分解は，重大な統合の問題を提起する．同じドメイン（例えばワークピース）に対し，2つの異なる問題フレームの2つの異なる主要部が現われ，それぞれの解は問題の異なる特性を明らかにしたり，異なる特性に依存したりする．機械による実現方法は両方の特性を満たす一つのドメインでなければならない．このような統合は，伝統的なエンジニアがShanleyの原理と呼ぶもので，Pierre-Arnoul de Marneffe[11]はソフトウェア開発においても有効であると指摘した．

もしあなたがドイツのV2ロケットの断面図を書くとしたら，外壁，構造支柱，タンク壁などがあるのが分かるだろう．もしサターンB型月ロケットの断面図であれば，外壁であり，構造支柱であり，かつタンク壁となるものがあるだけである．ロケットのエンジニアはShanleyの原則を適用し，タンク内の燃料の圧力が外壁の剛性を増すことを利用したのである．

9. 特　　化

スペシャリストは，比較的少数の問題と，それらを理解し，解くための方法に着目する．彼らは，その問題の典型的なコンテキスト，いつも接しているドメインを解析し，記述する能力に長けているのである．スペシャリストはまた，対象としている範囲のどのような問題もカバーしてしまうような枠組みに慣れ親しんでおり，対象とする主要な部分をその枠組みの中で認識するのである．

スペシャリストの構造化に対する能力は，部分問題から解を構成してゆく作業によるところが大きい．ここでの部分問題は，似たような問題の枠組みから認識され，当てはめられる．この焦点の鋭さが特化された知識を構成する．そして，適切な「理論的基礎」や，「実際の指針」は，現実的なものとなる．これが，工学のある分野が，他の分野と分岐する際の原点となる．自動車のエンジニアは，自動車の設計に必要な知識枠組みの集合や，完成度の高い自動車を作り上げるために解決方法を適用する方法を知っているのだ．

ソフトウェア開発全体を，工学の確立された一つの新しい分野としたいという大きな願望は，誤った考えである．われわれの願望は，ソフトウェア工学の中に特殊な専門分野の製品を育成することに向けられるべきである．

この各専門分野それぞれが確立された工学分野と並んで真価を発揮するに値する．ソフトウェア

におけるこのような特殊化には，すでにいくつかの例がある．コンパイラの記述は有名な例である．文脈依存文法など，問題領域に固有の難しさが認識され，分類されてきた．文法からLALRパージング表を得るような記述の変換は，標準的なレパートリの一つである．文法解析，意味解析，コード生成，局所的あるいは大局的な最適化などが研究され，それらの機能を役に立つ製品に統合する方法が開発された．

ソフトウェアの他の分野でも特殊化を進めなければならない．これは，パソコン用のシュリンクラップソフトウェアの開発において特に重要である．雑誌での競合製品の批評のため，開発社間で各社は競合他社の製品を調査し，他社のすぐれた機能を自社製品に取り入れようとする．ユーザは相互運用性やlook and feelの面から標準化を強く要求している．それは，資格さえあれば自動車のドライバはどの車にも乗れて，すぐ運転できることと同様である．しかし，ソフトウェアプロジェクトが一つしかない多くの分野では，こういった圧力はない．

われわれがエンジニアの水準に到達したいと願望するなら，われわれの技法を分割できるような専門分野を識別し，研究し，確立する必要がある．特殊化と，問題指向の理解のみが，ソフトウェア工学をアマチュアからプロへと進歩させていくのである．

謝辞　John Dobson, Daniel Jackson, Ralph Johnson, Wlad Turski からは，本稿の初期の版に対して多数の有益なコメントを受けた．彼らが関心をもって助言してくれたことに感謝する．

参考文献

1) Garlan , D. and Shaw, M.: An Introduction to Software Architecture, in Advances in Software Engineering and Knowledge Engineering, Vol. 1, Ambriola, V. and Tortora, G. eds., World Scientific Publishing, City, New Jersey (1993).
2) Rich, C. and Waters, R.C.: Formalizing Reusable Software Components in the Programmer's Apprentice, in Software Reusability, Volume II, Biggerstaff, T. J. and Perlis, A. J. eds., Addison-Wesley, Reading, Mass., pp. 313-343(1989).
3) Johnson, R.E.: Documenting Frameworks Using Patterns, Proc. OOPSLA, ACM Press, New York (1992).
4) Alexander, C., Ishikawa, S. and Silverstein, M.: A Pattern Language, Oxford University Press, New York (1977).
5) Johnson, R.E.: Why a Conference on Pattern Languages ? , ACM SE Notes, Volume 19 , Number 1, pp.50-52 (Jan. 1994).
6) Brooks, F. P. Jr.: No Silver Bullet—Essence and Accidents of Software Engineering, Proc. IFIP 10 th World Computer Congress, North-Holland, Amsterdam, pp. 1069-1076(1986).
7) Turski, W. M.: And No Philosopher's Stone, Either, Proc. IFIP 10th World Computer Congress, North-Holland, Amsterdam, pp. 1077-1080(1980).
8) Polya, G. A.: How to Solve It; Princeton University Press, Princeton, N. J.(1957).
9) Jockson, M.: System Development; Prentice-Hall, Englewood Cliffs, N. J. (1983).
10) Parnas, D. L. and Madey, J.: Functional Documentation for Computer Systems Engineering (Version 2); CRL Report 237, McMaster University, Hamilton Ontario, Canada (1991).
11) de Marneffe, P.-A.: Holon Programming: A Survey, Universite de Liege, Location, Service Informatique (1973).

(平成6年11月7日受付)

Michael Jackson
独立コンサルタント．
本稿に関する問合せは下記まで．
101　Hamilton Terrace, London NW8 9QX
e-mail:attmail.att.com