革新的ソフトウェア・プロジェクトを目指して

― 無形労働 (Immaterial Labor) [*] の視点から ―

岸田 孝一
SRA
E-mail: k2@sra.co.jp

要旨

これまでのソフトウェア工学は物質的労働 (Material Labor)のパラダイムにもとづく規律 (Discipline) を中心に構成されてきた. この論文ではソフトウェアの開発や保守の仕事を 無形労働 (Immaterial Labor) としてとらえ. 新しいプロジェクト・マネジメントのあり方を探る.

[*] Immaterial Labor の訳語は, 社会学関連の文献では,「物質的労働」(material Labor) に対比して「非物質的労働」となっているが, ここではその訳語に含まれる政治的ニュアンスを避けるために, 伊藤昌夫さんの提案案にしたがって「無形労働」という訳語を使用している.

Immaterial Product は「無形プロダクト」. 音楽やソフトウェアは CD という material な形態をとっているが. そこに記録された「かたちのない」情報が主体である. 介護そのほかのサービスのように, もともと「かたちのない」プロダクトを提供する無形労働も存在する.

1. はじめに

ソフトウェア工学の概念が提唱されたのは1960年代末にヨーロッパで行われた2回のNATO ワークショップでの討論の結果である [1]. 当時は参照すべき既存の工学は機械工学や土木工学などの物質的製品を対象とするものしか存在しなかったので, ソフトウェア工学の枠組みもそうした工業生産のパラダイムに沿って工程の標準化や分業作業の属人性の排除などの概念を用いて構築されることになった.

このアプローチは, あらかじめ仕様が確定した(あるいは比較的安定した)ソフトウェア・プロダクトの開発にさいしてはそれなりの成功を収めた. しかし. ハードウェアの小型化・低価格化やネットワーク技術の進展にともない, 社会のさまざまな分野にコンピュータの利用が浸透してきた現在, われわれが扱うソフトウェア・システムは多様なユーザのさまざまな文化に影響を与えるような情報を内包するプロダクトになり, われわれの仕事, 学習, コミュニケーション, 娯楽などの活動を定義しなおすようなプロダクトと化している. 当然のことながら, その仕様はあらかじめ確定することはできす, また周囲の社会環境ノン変化に対応して 絶えず変化し続ける.

伝統的な工学パラダイムには, そうしたソフトウェア・プロダクトの開発や保守のプロセスを考える上で基本的な何かが欠けているといわざるを得ない. われわれはいま根本的に新しいパラダイムを検討すべき時に来たと考えられる.

2. 無形労働 (Immaterial Labor)

音楽レコードやビデオCDが商品として市場に出回り始めたとき, 多くの人びとはそれらを知的労働の成果にもとづく新しいタイプの工業製品だと受け止めた. しかしマウリツィオ・ラッツァラート [2] は, 状況を別の視点から眺めて, 今日のポスト工業社会における支配的な労働形態に関する考察をおこなった. かれによれば, 現在起こっている社会現象は, 文化や情報を内包するさまざまな商品の生産や消費が, 全世界の経済システムに浸透し, 影響を与えつつあるということなのである. それは, これまでの工業的生産のプロセスが, いまや, 文化や情報が本質的な役割を占める無形労働のプロセスに置き換わりつつあるということを意味する. そうしたプロセスによって. ファッション. 趣味・嗜好, 消費傾向, 芸術的標準, さらには世論といったものが形成され決定される時代が到来したともいえる.

無形労働のプロセスを構成する文化や情報に関する知的活動は, これまで一部のブルジョア階級の手慰みとして扱われ、工場労働と同列の労働とは認められていなかった. しかし, 今日の状況は, IT やネットワークの発達にともなって, ほとんどの人びとがなんらかのかたちで無形労働に従事するようになったといっても過言ではない.

無形労働は, 生産と消費の関係を変えてしまう働きをする. 従来は, 生産がニーズに対応し, プロダクトの消費が価値を生み出すというかたちであった. しかし, 現在の無形労働が作り出すのは, 既知の消費ニーズに応えるためのプロダクトではない. 音楽やファッションなどの「かた

ちのない」プロダクトは，消費者の物質的ニーズに対応するのではなく，からの刺激によって生み出された人びとの非物資的な欲望を満足させることを意図している．つまり無形プロダクトが経済的な価値を持ちうるのはその背後に含まれているアイデアがはっきりと消費者に伝えられ受け入れられた場合だけなのである．そして，消費はさらに新しいニーズを作り出し，それを生産者の側にフィードバックする．こうした社会化のプロセスこそが，無形プロダクトの特徴だといえよう[3]．

無形労働によって作り出されるプロダクトの価値や品質は，その生産に携わる人びとの知識や創造性に大きく依存する．生産者に求められるのは次の3つのスキルである．すなわち（1）知的スキル，（2）コミュニケーションスキル，そして（3）自発的行動力．これらのスキルは，ソフトウェア開発者にとって必須のものだと考えられる．

3. 無形労働としてのソフトウェア開発

ことさらいうまでもなく，ソフトウェアは「かたちのない」ものあり，ソフトウェアを開発・保守する仕事は無形労働の一種である．作業工程の細分化や標準化などの手法にもとづくテイラー方式の科学的管理は無形労働には適用できない．マネージャにとっては，これまでのように生産ラインの職長として部下を監督するのではなく，開発者の自主的行動を尊重し協働作業や創意工夫を促す役割を担うことが必要になる．

ソフトウェア開発組織とは，本来相互にさまざまな関係を持つプロフェッショナルの集団であり，それぞれのメンバーは，そこで行われる開発作業の内容に関心を抱きみずからの貢献に対してそれなりの見返りが得られる場合にのみ，その組織に参加するというのが理想的なかたちである．ラッツァラートは，こうした組織形成のあり方が今日次第に支配的になりつつあると指摘する．多様な自主的労働形態があちこちで誕生し，大小さまざまなサイズの作業ユニットが特定のプロジェクトのために結成され，仕事が完了するとメンバーたちはまた無形労働の人材プールにもどって行く．

思想的な枠組みとしての無形労働は，ソフトェア開発およびソフトウェア産業に対して次の2つの点で深い関係がある．

まず第1に，われわれが現在開発し保守・運用しているシステムは，単なる物質的プロダクトではなく，労働・学習・コミュニケーション・娯楽といったわれわれの社会生活のかたちを変革（再定義）する機能を含んでいる．その社会的な価値は．ユーザがシステムを利用することによって確定されるのである．

そして第2に，ソフトウェア・システムは，旧来の物質的労働を無形労働に変貌させる原動力として働くのだということを忘れてはならない．自動車工場や家電製品工場の製造ラインで働いている人びとの多くは，単に物質的な生産に携わっているのではなく，かなりの時間をソフトウェア・システムとのやりとりに費やしている．そうした生産支援システムを開発したソフトウェア技術者は，単にユーザに仕事を効率化するツールを提供したのではなく，ユーザの生産活動やその社会的形態を変革する役割を演じたのだといえよう．

ソフトウェア・システムの要求仕様は，もはやそこにある情報を把握して分析すればよいというようなものではなくなった．仕様は，何らかのかたちで，われわれ開発者が創りだし設計すべきものである．ソフトェアは既存のニーズを満足させるためや，何らかの現実世界のモデルをコンピュータに載せるために作られるわけではない．ソフトウェアは，曖昧なあるいはまだ存在していないニーズをかたちあるものにし，それをユーザに提示して現実を認識しなおさせるためのツールなのである．

既存のニーズにもとづいて生産が行われるのではなく，生産と消費がからみあったプロセスを通じて，潜在的なニーズが顕在化されて行く．すなわち，ソフトウェア・システムは，単に工場の中で製造され塀の外にいるユーザに投げ渡されるようなものではないのだ．それは，生産者と消費者とのあいだの社会的な関係を確立するためのメディアなのである．ソフトウェア産業にとっての成功の鍵は，ユーザがプロダクトの消費プロセスを通じて生み出す新しいニーズをいかにして的確に把握しそれに応えるべき技術革新を行えるか否かだといえよう．

したがって，絶えず変化し続ける要求仕様を確定させることはソフトウェア工学にとっての主要な研究開発課題ではありえない． むしろそれは潜在的なニーズを探り出し表面化させることによってソフトウェア・システムの価値を高める上での基礎だと考えられる．最近提唱された「いつまでもベータ (Forever Beta)」というアイデアは，無形労働によって作られるタイプのソフトウェアの本質を反映している．近年盛んになってきた新しい製品リリースのアプローチはそうした考えにもとづいている．

もちろん，これまでソフトウェア工学のメイン・テーマであった開発技法やプロセスの革新は，引き続き重要な課題であり続けるが，しかし，何を作るか (What?) よりも，どうやって作るか (How?) や，なぜ作るのか (Why?) そして，どうなるのか (Where to?) のほうに焦点が移って行くであろう．

無形労働の視点からソフトウェア開発という仕事を眺め

てみると，それはある種の設計作業であることがわかってくる．そこには2つの側面がある．ひとつは，プロダクトとしてのソフトウェア・システムの設計である．これは，従来の意味でのソフトウェア設計にあたる．そしてもうひとつは，そのソフトウェア・システムを利用（消費）するユーザとのコミュニケーションによって生まれる価値の設計である．どちらがより重要かという問題ではない．両者はソフトウェア設計の2つの側面であり，うまく統合される必要がある．無形労働としてのソフトウェア開発に課せられた課題は，こうした設計の二面性にどう対処するかであろう．その意味からすれば Participatory Design の方法論[4]や Socio-Technical Theory[5]について，さらに深い研究が進められることが望ましい．

これまでのソフトウェア工学界における支配的な傾向は，「ソフトウェア・システムを与えられた仕様を満足して正しく開発することが主要な任務であり，そのための技法が不足している」ということであった．しかし，無形労働の視点から見れば，そもそも「正しい」システムなどは存在しない．システムのほんとうの価値は，ユーザによってそれが使用されるプロセスを通じて，初めて確定されるものだからである．

4. ソフトウェア開発プロジェクトの組織

無形労働としてのソフトウェア開発プロジェクトを組織するにあたって，特に，プロジェクトの実践を通じての技術革新を考える上では，たとえば，次にあげるような指導原理（Principle）が重要だと考えられる．

Principle 1: 動機付け(Motivation)

開発技術者たちに明確な動機付けを与えそれを維持して行くことが，まず第1のプライオリティである．開発プロジェクトにおいて主役を演じるのは開発者たちであり，マネージャではない．

ソフトウェア・システムの価値は，その設計や具体化にさいして，さまざまな技術的選択肢のうちのどれを採用するかによって決まる．それには，開発者たちの個人的嗜好や感覚が大いに関係している．イギリスの研究者がおこなった調査によれば，ソフトウェア開発者の動機付けに影響を与える因子は，挑戦，変化，利益，問題解決，チームワーク等であり，なかでも，変化する仕様を相手にして挑戦的な課題に直面した場合に，開発者のモチベーションが大きく高まるということが報告されている [6].

動機付けは，技術的な革新のみならず，ソフトウェア開発に固有な社会的協同作業にとっても重要な意味を持っている．開発プロセスの任意の時点において，開発者たちは，それぞれの技術的ニーズを相互に交換し合い，情報の共有を円滑におこなうための，フレキシブルなメカニズムを必要としている．

20世紀ドイツの美術界を代表するコンセプチュアル・アーティストとして知られたヨーゼフ・ボイスは，かつて，「すべての人間は芸術家である」と述べた[7].

> どのような人間の内部にも創造的能力が潜在している．しかし、だからといって,だれもが画家や彫刻家になることができるといいたいわけではない．そうではなく,人間のあらゆる仕事の領域には潜在的な創造性があるといいたいのだ．どのような仕事も何らかの意味で芸術に関係している．そのような芸術は,芸術を理解する一部の人間が行うバラバラで孤独な作業によってなされるものではない．そして,そうした少数の人間以外は芸術とは別の仕事をしなければならないというわけではない．わたしがいっているのは,あらゆる人間活動や労働形態における創造性のことであり,それは芸術にしか見られないものではない．創造性はあらゆる活動に属している．

ボイスのことばは，人間のすべての活動には何らかの創造的機会が潜んでいるという重要な（しかし，ともすれば忘れられがちな）事実を指摘している．ソフトウェア・プロジェクトのマネージャは，そのことを常に留意し，メンバーの動機付けを心がけなければならない．

Principle 2: 多声 (Polyphony)

近年その評価が高まりつつあるロシアの思想家ミハイル・バフチンが提唱した「多声」 (Polyphon) および「非完結性」 (Unfinalizability) という2つのの概念は，ソフトウェア・プロジェクトの技術的マネジメントにとって，きわめて重要な意味を持っている．バフチンは，小説家ドストエフスキーの作品群を綿密に分析して，この2つの原理を発見した[8].

ドストエフスキーの小説には，さまざまな人物が登場して，人間が生きることについての真理とは何かをめぐって「深遠な」討論を続けて行く．作者としてのドストエフスキーは，これらの登場人物の誰かひとりに肩入れしたりすることなく，全員を平等に取り扱っている．小説のストーリーは，どこかの時点で終わるのだが，しかし，それまでに続けられてきた対話や討論は，決して結論には到達しない．世界で最初の「対

話型小説」としてのドストエフスキーの作品のユニークさはそのあたりにあるとバフチンは指摘する：

> 世界の究極の問題である真理は，単独の個人的意識の中では解明できないと，ドストエフスキーは考えていました．真理は，1つの意識には収まりきれないのです．真理は，常に部分的にですが，多くの対等の意識が対話的に交流するプロセスの中で解明されて行きます．この対話は，真理について考え，真理を求める人間が存在する限り，終わることも完結することもありません．対話の終わりは人類の破滅を意味します．
>
> ドストエフスキーは，ポリフォニック（多声的）な小説，つまり，究極の問題をめぐる緊迫した激しい対話として構成された小説の創始者です．作者はこれらの対話を完結させたり，作者としての自分の結論を提供したりはしません．かれは，いかなるもかたちであれ，完結というものを認めませんでした．
>
> たとえいくつかの小説（たとえば「罪と罰」）が完結しているようであっても，それはただ形式的な文学的完結にすぎません．「カラマーゾフの兄弟」はまったく完結していません．そこでは，あらゆるものが未解決つまりオープンなままになっています．あらゆる問題が問題のまま残り，一定の結論を示すものは何もありません．大切なのは，異なる人間によって具体化された思想が，同時並行的に多数存在することなのです．重要なのはそれらの間の対話，明らかに完結されない対話なのです．対話の完結，論争の終了は，外からの野蛮な物質的な力を導入すれば実際には可能であることを，ドストエフスキーは作品の中で一度ならず示しています．しかし，本質的には，そうした対話的思考やその意味というものは，完結されません．

われわれに与えられた課題は，文学作品を対象に展開されたバフチンの分析手法をソフトウェアに置き換えて，さまざまな視点や思惑が混在する多声的（Polyphonic）な開発プロジェクトのあり方についての理論を整備することである．本来何らかの時間的制約のもとにあるプロジェクトは，「外からの」強制力（たとえば納期）によって完結させられるが，そこで開発されたソフトウェア自体が内包する諸問題の解決は，決してそこでは終わっていない．すなわち，そのソフトウェアに関係する開発組織やユーザ組織がその問題を抱え続けることを意味する．ソフトウェア開発者たちは，これまでにそのプロジェクトで行われた対話や討論の中間的成果を携えて，次のプロジェクトに進んで行くのである．

バフチンはまた，フランス中世の民話文学（フランソワ・ラブレーの作品群）の分析から，「カーニバル文学」という独自の視点を提示している．このメタファは，オープンソース/フリー・ソフトウェアの社会的問題を考えるさいに，エリック・レイモンドが提示した「伽藍とバザール」 [9] よりも役に立つように感じられる．

Principle 3: 差異 (Difference)

これまで，ソフトウェア工学の方法論は，規律や統制を中心とする物質的労働（工業生産）のパラダイムにもとづいて構築されてきた．たとえば，その典型的な例として，ソフトウェア開発組織の成熟度を示す CMM では，開発プロセス遂行の第2レベルを「反復可能」（Repeatable）と呼んでいる：

> このレベルにおけるプロセスの特性は，それが「反復可能」であり，予期される一手の成果をもたらすということである．プロセスの規律は，決して厳密不可侵なものではないが，しかし現実のプロセスが所定のかたちに維持されることを確実にするものでなければならない．

CMM は，それぞれが反復可能であって，所定の成果をもたらすような一連の Key Practice Area から構成されている．そこで示されている規律は，もし似たようなプロジェクトを繰り返し実施する場合には，開発プロセス内のそれぞれの活動について，きちんとしたドキュメントが整備され，教育訓練が行われ，効率的に実施されるべきだということである．

そうした規律は，工場生産における物質的労働には適しているが，しかし 無形労働としてのソフトウェア開発においては，ややもすると開発者の創造的行為を抑圧する結果をもたらす．いわゆる SPI 活動がしばしば現場からの抵抗に遭遇することの理由は，そのあたりにあると考えられる．

ポスト構造主義を代表するフランスの哲学者ジル・ドゥルーズは，「同じことが反復されることは決してない．差異だけが反復されるのだ」と主張する．その著書「差異と反復」の序章は次のようなパラグラフで始まっている[10]．

> 反復は一般性ではない．それは，いくつかの仕方で一般性と区別されなければならない．両者の混同を含む表明は厄介な代物である．たとえば，わたし

たちが，「一般的なものについてしか科学は存在しない」というのと，「反復するものにしか科学は存在しない」というのを，同一視するような場合である．差異は，その本質上，反復とその類似（たとえそれがどのように近い類似であろうと）との間に存在する．

邦訳で2段組500ページを越えるこの哲学的大著の全容を解説することは，残念ながらわたしの能力を超えている．ここでは，何冊かの注釈書を参考にしながら，ドウルーズの思想を，ソフトウェア開発との関連を考えながらトレースしてみることにしよう．

差異について考えるさいに陥りやすい誤りは，すでに同一のものとして特定されているものを基準にして差異を規定することだ，とドウルーズはいう．たとえば，プロジェクトAを基準にして，それと異なる他のプロジェクトについて差異を考えるとか，AとBという2つのプロジェクトを取り上げて両者の差異を論じる場合などが，それにあたる．このとき，差異は同一性との関連で定義されることになる．差異について考えるはずだったのに，結局，同一性が思考の基礎に据えられてしまうのである．

では，同一性を経由しない差異とはいかなるものか．それにはまず，同一性をできあがったものとして考えるのをやめ，同一性という観念や同じものとして認識されるものが，いったいどんな思考経路によって作り上げられたかを考えることである．

同一性は，ある思考過程の途中で生まれる中間結果（あるいは効果）にすぎず，決して何かの前提や原理にはなりえないのである．そのようにとらえられる同一性は，むしろ自分自身に対して異なるものになることを，その本質として持っている．同一性は，ただそのように見えるだけの視覚的な特殊効果にすぎない．正確に同じ状態が，観念としてであれ，プロセスの状態としてであれ，再現されることはありえない．反復は，差異をともなわずになされることはないからである．

ソフトウェア開発プロセスにおいては，同じような活動が繰り返し行われるように見えることが多い．それらを同一なものとして把握し定義するというのが，いわゆるソフトウェア工学におけるさまざまな技法やモデルの原理になっており，それからの「差異」を規制したり分析したりすることが，いつのまにかわれわれの生活習慣と化してしまっている．

ドウルーズがいうように，同一性の反復が単なる視覚的効果に過ぎないのだとしたら，われわれは，そうした反復の原理を捨てて，「差異」それ自体についての分析を行い，それにもとすく方法論を構築すべきであろう．

振り返ってみれば，これまでのソフトウェア工学における技術的革新の多くは，まさに開発プロセスにおける「差異」を認識することから生まれてきたのだといえる．

たとえば，いくつかの「構造化プログラミング」技法を考えてみてもよい．それらの技法は，一般には，単にプログラム構造の機能的モジュール化を目指した（そうした同一性を持つ）ものだと誤解されているが，本質はそうではなかった．いずれも，プログラムの実行プロセスに注目し，あのイタリア人数学者によって証明された「構造化プログラム定理」に準拠しながら，いくつかの意図的な差異にもとづいて提案されたのであった．

実行プロセスがソースコードから読めるように（ダイクストラ），処理対象のデータ構造を反映するように（ジャクソン），あるいは処理の意味とは独立した標準構造に帰着するように（岸田，といったそれぞれ「異なる」意図の実現を目指していたのであった．

Principle 4: リゾーム (根茎, Rhizome)

フェリックス・ガタリとジル・ドウルーズの共著になる「千のプラトー」では，これまでの西欧哲学に共通であった「樹木」型の階層構造の思考モデル（根・幹・枝・葉）に代わるものとして，「リゾーム」（Rhizome: 根茎）のメタファが提案されている [11].

リゾームとは，植物学では，地面の下に水平に生え広がる根を指す．典型的な例は竹の地下茎である．ガタリとドウルーズは，このリゾームが，人間の思考のモデル，さらには知識・文化や社会を考えるための新しい概念モデルとして有用ではないかと主張している．

文化のモデルとしての樹木型の階層構造は，常にその出発点である「根」に注目し，そこから時系列的に立ち上がってくる「幹」や「枝」，そして「梢」や「葉」について考えるかたちになっている．このフレームワークが，これまでの西欧世界における現実認識やすべての思考（植物学から動物学，解剖学，さらにまた認識形而上学，神学，存在論，全哲学）を支配してきた」と，ガタリ&ドウルーズは指摘する．

> 書物の第1のタイプは根としての書物である．樹木はすでに世界のイマージュになっている．根は，世界としての樹木のイマージュなのである．それは，有機的，意味作用的，主体的な美しい内面性としての古典的な書物である．

そのことは，たとえばわれわれの周囲にある SWEBOK ほかのソフトウェア工学文献のスタイルをみればあきらかであろう.

リゾームは，それとは異なり，水平にあちこちの方向につながった線が，どこにも中心をもたず，どこからどこへでも自由に連結し，あちこちから幹を立ち上がらせる．それは，さまざまな意味論的連鎖，権力構造，状況などを，芸術，科学，あるいは社会組織とランダムにつなぎ合わせるオープンな多様的構造体なのである.

リゾーム・モデルの基本的特徴は次の通りである:

1) 連結性および異種混合性: リゾームは，その任意のポイントにおいて他のものそして他のポイントと連結される．そのようなことが可能なければならない．多声構造を持つソフトウェア・プロジェクトにおいては，さまざまな新しいアイデアが，任意の地点（あるいは時点）から発生する．それらを有機的に連結することにって技術革新が可能になるのである.
2) 多様性: リゾームは多様体である．それは，いかなる場合にも何らかの「一」なるものと関係付けられることがないという意味で多様なのである．すなわち，統一的な原理や規律を追及するという「樹木型構造」の思考フレームワークを捨て，「多声」をベースとする「非完結」的な対話や討論を喚起することが，技術革新を行うさいの出発点である.
3) 断裂性: リゾームは任意の場所で断裂する可能性がある．そしてそこから，あるいは他の場所から，すぐに再生し，同じラインあるいは別の新しいラインを進んで行く．ソフトウェア開発における新奇なアイデアは，多くの場合，これまでのプロセス・モデルとは相容れず，プロセスにおける「断裂」効果をもたらす．その断裂をポジティブにとらえて，新たな地下茎の成長を促進させることが重要なのである.
4) 地図性: ドゥルーズたちのいう「地図」は、いわゆるロードマッツとは異なる。それは現実のコピーではなく、現実認識そのものを意味している．すなわち,リゾームは，いわば「道のない地図」だと考えられる．たくさんの Point ofInterest がそこにはマークされている．どこから出発しどこへ行くのかといった問いは無用であり，始まりも終わりもなく，いつも中間である．それはいかなる構造的なモデルあるいは生成的なモデルにも帰着されない.

-

リゾームの持つこうした特徴は，特に創造性を重視したソフトウェア・プロジェクトの組織構成を考える場合にきわめて重要だと考えられるが、具体的にそれを適用したプロセス・モデルの構築には，さらに突っ込んだ考察が必要であろう.

Principle 5: 開放性(Openness)

ソフトウェア・プロジェクトは，外の世界に向かって開かれた技術的な窓を持つことが望ましい．そしてその窓の役割を演ずる人間が必要である．プロジェクトは，その窓を通じて，内部の活動の成果を外に向かってアピールすることでき，また外から新しいアイデアを導入することができる．古代中国の都市国家において，すぐれた統治能力を持った諸王が，その城壁の外の荒野を旅する諸子百家の知恵を取り入れて政治の革新を図ったことは周知の事実である．典型的な例は,斉国の首都・臨淄の城門の近くに集められた「稷下の学士」（孟子もその1人だった）であろう．秦の始皇帝の成功が，そうしたオープンな統治姿勢を徹底したことにあったのはいうまでもない.

さらにいえば，GNU や Linux などのフリーまたはオープンソース・ソフトウェアは，まさにラッツァラートが暗示したのと同様なプロジェクトによって開発されてきた．そうしたプロジェクト・チームは，インターネット上に分散されたメンバーから構成され，各人が自分の責任において特定の問題に取り組んで行く．協働はボトムアップに行われ，マネージャの管理下ではなく，メンバー相互の意志にもとづいて開始され管理される．すべてのメンバーが，ある意味で事業経営者なのである[12].

それと似たプロジェクト編成は，別に,そうした特殊なプロジェクトだけではなく,通常の企業組織の内部でも取り入れることができる．数年前の EuroSPI 会議の基調講演で，ヨーロッパの有名な通信機メーカが，世界中に分散したアジャイル開発チームをオープンな形で運営し成功を収めたという事例が報告されている．それは，無形プロダクトを大規模組織の中で開発するためにプロジェクト編成の新しいスタイルだといえるだろう.

5. 終わりに

ソフトウェアは，そこに含まれる文化的情報を本体とする無形プロダクトであり，われわれの仕事や学習，コミュニケーション，あるいは娯楽のスタイルを変革する力を持っている．しかし，これまでに構築されてきたソフトウェア工学の方法論は，旧来の工業文明に固有の物質的労働を規律正しく管理するというパラダイムにもとづいて構築されてきた．われわれは，いま，無形労働の視点に立ってすべてを考え直すべきであろう．この小論では，技術革新を目指すプロジェクト・マネジメントについていくつかの新しい指導原理の提案を試みた.

これまで,ソフトウェア開発組織やプロジェクトのマネジ

メントについては,人びとの精神的規律を重視するという工業文明社会のパラダイムにしたがって,社会あるいは組織の統制を意図するというかたちでの議論が主流を占めてきた.

たとえば,ハンフリーが提案した PSP・TSP・CMM という階層構造は,「修身・斉家・治国・平天下」という儒教的マネジメント思想の現代版でしかないと考えられる. それは,「名正からざれば言したがわず,言したがわざれば事成らず」という孔子の正名論にも一脈通じている[13]. .

ヨーロッパに目を転ずれば,オブジェクト指向方法論の思想的背景が,19世紀末にマックス・ウェーバーが提唱した理想的環境制度のそれと一致するということが指摘されている[14].

情報化社会においては,そうした旧時代的な思考フレームワークが,もはや,多様化し複雑化した時代の状況に適応できなくなっていることは明らかであろう. インターネットの発展や普及にともなって,ソフトウェア・システムが人間の社会生活を支える基本的なインフラストラクチャと化した現在、われわれはそうしたシステムの開発・保守・運用を行うための新しいマネジメント・パラダイムを必要としている. この小論がそうした方向への第一歩として役立てば幸いである.

参考文献

[1] Naur,P. and Randall, B., "Software engineering: A report on a conference sponsored by the NATO science committee," NATO, 1969.

[2]Lazzarato,M., "Immaterial labour," in Radical thought in Italy: A potential politics, P. Virno and M.Hardt, Eds. Minneapolis, MN: University of Minnesota Press, 1996, pp. 133-147.
http://www.generation-online.org/c/fcimmaterialla bour3.htm

[3] Lazzarato,M., "European cultural tradition and the new forms of production and circulation of knowledge," 2004 .
http://multitudes.samizdat.net/article1292.html

[4]Ehn,P., Work-oriented design of computer artifacts. Stockholm, Sweden: Almquist & Wiksell International, 1988.

[5]Ye, Y., Yamamoto, Y., and Nakakoji, K., "A socio-technical framework for supporting programmers," in Proceedings of FSE2007, pp. 351-360.

[6]Hall,T., Sharp,H., Beecham,S., Baddoo,N., and Robinson,H., "What do we know about developer motivation?" IEEE Software, vol. 2008, pp. 92-94, 2008.

[7]水戸芸術館現代美術センター編,「ヨーゼフ・ボイス: よみがえる革命」, フィルムアート社, 2020.

[8]ミハイル・バフチン,「ドストエフスキーの詩学」, ちくま学芸文庫」, 1995.

[9]エリック・レイモンド,「伽藍とバザール」, 光芒社, 1999

[10]ジル・ドウルーズ,「差異と反復」, 河出書房新社, 1992.

[11]フェリックス・ガタリ & ジル・ドウルーズ,「千のプラトー」(上中下), 河出文庫, 2010.「リゾーム」はその序章(上巻に所収)

[12]Terranova.T, "Free Labor: Producing Culture for the Digital Economy Social Text - 63 (Volume 18, Number 2), Summer 2000, pp. 33-58
http://www.electronicbookreview.com/thread/technocapitalism/voluntary

[13] 岸田孝一「分散型ソフトウェア環境の概念モデル」(SS96 in 広島,招待講演)
http://www.sra.co.jp/people/k2/papers/paper3.html

[14] Kouichi Kishida, "The Maturing of a Software Project Manager", IEEE Software, July 1966.
http://www.sra.co.jp/people/k2/essays/essay1.html