

ソフトウェア開発とシンプル・ルール戦略 - ネットサービス開発の事例から - *

関西学院大学 商学部
柿原 正郎

Email: kakihara@kwansei.ac.jp
URL: <http://www.kakihara.org/>

2005 年 11 月

はじめに

現代社会の急速な情報ネットワーク化により、我々の毎日の生活のなかで何らかのソフトウェアの支援を受けることのない場面は皆無と言って良い状況になった。比較的その存在が理解しやすいコンピュータのアプリケーション・ソフトウェアにとどまらず、家電製品、携帯電話、自動車、各種交通網などを支える様々な組込みソフトウェアなど、我々の日々の生活はソフトウェアの働き無しにはまったく回らなくなっている。現代社会におけるソフトウェアの重要性は、日に日に高まっていることは誰の目にも明らかであろう。

このように社会的意義を増すソフトウェアを開発する手法やプロセスの管理や効率化、そしてソフトウェア自体の品質向上のための方法論の科学として「ソフトウェア工学(ソフトウェア・エンジニアリング)」という研究領域が 1960 年代に生まれた。ソフトウェア工学における中心問題を端的に言えば、「品質の高いソフトウェアをいかに効率的に開発するか」というものである。この問いに対する知見の蓄積として、構造化プログラミング、オブジェクト指向開発アプローチ、CASE (Computer Aided Software Engineering) ツールの活用、CMM (Capability Maturity Model、能力成熟度モデル) などの手法が生み出されてきた。

しかしながら、今日激変する社会環境のなかで、ソフトウェア工学のあり方も変わらざるを得なくなってきた。これまでただ単に品質の高いソフトウェアを作り上げることに専念すればよかったソフトウェア開発も、技術環境や市場環境の急速な変化に合わせて、よりオープンかつフレキシブルな開発体制や適切な開発投資効果が強く求められるようになった。つまり、ソフトウェア開発の「戦略化」がいま強く求められるようになってきたのである。常に変化を続ける技術環境と市場環境のなかで、いかに効率よく開発し、いかにタイミング良く市場投入をし、いかに適切な利益を生み出すのかという経営戦略の視座がソフトウェア開発においても極めて重要な意味を持つようになったのである。だが、現在までのところ、ソフトウェア工学研究における戦略概念の扱いは極めて限定的である。「ソフトウェア戦略」、「戦略的ソフトウェア開発」などの言葉が独り歩きするばかりで、経営学の経営戦略論の理論蓄積を本格的にソフトウェア工学分野に導入した研究は極めて少ない¹。

こうした背景を踏まえ、本稿ではソフトウェア工学と経営戦略論との理論的架橋の可

* 本稿は『商学論究』(関西学院大学商学研究会)第 53 巻・第 3 号に掲載予定の論文のドラフトである。Copyright © Masao Kakihara 2005. All rights reserved.

¹ 数少ない例外として、クスマノの一連の研究が挙げられる。例えば、Cusumano (1991; 2004)。

能性を探るひとつの試みとして、アイゼンハートらが提唱する「シンプル・ルール戦略」を取り上げ、ソフトウェア開発の文脈におけるその有効性について議論する。まず第節では、現代のソフトウェア開発を取り巻く環境変化を整理する。第節では、これまでのソフトウェア工学研究における戦略概念の取り扱いについて批判的に考察する。第節では、経営戦略論の既存研究を簡単にレビューし、ソフトウェア開発におけるシンプル・ルール戦略の適用可能性を考察する。第節では、現在ソフトウェア開発の分野で注目を浴びているネットサービス分野の事例を用いて、シンプル・ルール戦略の妥当性を検討する。最後に第節では、本稿の論点と今後の課題をまとめる。

ソフトウェア開発を取り巻く環境変化

我々の現代社会に広く遍く存在するようになったソフトウェア、その社会的重要性は今でこそ自明のこととも言えるが、たった30年前ですら十分に理解されていなかった。1946年にペンシルベニア大学で開発された世界最初のコンピュータENIACから始まり、その後コンピュータの商用化が進んだ1950・60年代までのコンピュータ黎明期には、ソフトウェアの問題はハードウェアと一体化した問題領域であった。ソフトウェアの社会的意義がとりわけ注目されるようになった契機のひとつは、やはりビル・ゲイツとポール・アレンによる1975年のマイクロソフト社の設立であろう。その当時、一介の大学生だった彼らがBASIC言語の開発・販売で成功した時点で、その会社が現在のような世界最大のソフトウェア企業になるとは誰も予想していなかったに違いないが、彼らの成功の軌跡はソフトウェアの発展・発達の歴史と言っても過言ではないだろう。

そのマイクロソフト社の設立から30年、現代のソフトウェアを取り巻く環境には驚くべき変化があった。それは主に以下のような点が挙げられる。

(1) ハードウェアの急速な発展

CPU(中央処理装置)の演算処理能力の劇的な向上、メモリや外部記憶媒体の価格の急速な下落、出入力デバイスの多様化など、ハードウェアの技術発展は劇的に進んだ。半導体の集積密度は18~24ヶ月で倍増するという「ムーアの法則」を出すまでもなく、コンピュータ分野の技術発展は怒涛の勢いで進む。そうしたハードウェアの上で動くソフトウェアもこれら一連の技術発展の影響を強く受け、日進月歩のハードウェアの発展を最大限に活かすために、ソフトウェア開発もそれら技術発展がもたらす可能性を取り入れることを恒常的に要請されるようになった。

(2) 使用環境の拡大

1970年代までごく一部の企業と研究機関のなかでしか動いていなかったコンピュータは、1980年代以降急速に一般的使用環境へと広がっていくこととなる。パーソナルコンピュータと呼ばれるようになった一般家庭用コンピュータは、いまでは7割を超える世帯に普及し²、様々なアプリケーション・ソフトウェアがごく当たり前家庭で使われるようになった。また、携帯電話、家電製品、自動車などに搭載される組込みソフトウェアは、室内・屋外にかかわらず、さまざまな使用環境で正確に動作することを求められる。現代の多様で複雑な使用環境のなかでソフトウェアに求められる品質と性能の要求は必然的に急速に高まっていった。

(3) ステークホルダーの多様化

開発するソフトウェアの規模の拡大に合わせて、開発に直接的・間接的に関わる人々

² 単身を含む世帯における普及率(総務省『通信利用動向調査報告書』)。

が急速に増加することとなった。マイクロソフト社が開発する Windows 系オペレーション・システムのソースコードの行数は、1992 年の「Windows 3.1」では約 300 万行であったが、2000 年の「Windows 2000」では 3500～6000 万行までに増加した³。こうしたソフトウェア開発の規模と複雑性の増大に対して、近年のコンポーネント志向のソフトウェア開発手法などの広がりにより⁴、企業内外の様々な組織境界を超えた連携のもとで開発されたソフトウェアが、企業の壁を超えた使用環境で様々なユーザーに利用されることで、ソフトウェア開発のステークホルダーが急速に拡大した。

(4) ソフトウェアの継続的な修正

企業間の競争が激化し、競争環境が絶え間なく変化し続けるなか、ソフトウェアに対しても常に新しい機能の追加や修正が要求されるようになった。パッケージ・ソフトウェアでさえ「完成品」という概念が崩れつつあり、頻繁なアップデートや修正プログラムの配布がなされるようになった。こうした完結することのないソフトウェア開発のプロセスが、さらに新たな問題発生のきっかけとなり、全体のソフトウェア開発としての投資対効果の測定を難しくさせることとなった。

こうした急速な環境変化を端的に整理するとすれば、ソフトウェア開発は 2 つの大きな環境変化に直面している。ひとつは、急速な「技術発展」であり、もうひとつは、絶え間ない「市場変化」である。そして、ソフトウェア開発のアプローチをこの 2 つの環境変化を軸に整理することができる。

まずは、急速な技術発展によって実現可能となった様々な機能を活かして、ソフトウェア開発に積極的に取り込んでいくアプローチがある。これをここでは「テクノロジー・プッシュ」型アプローチと呼びたい。ハードウェア、ソフトウェアにかかわらず、技術発展は日進月歩で進んでいく。つい半年前にはできなかったような機能やサービスが、突如として実現できるようになる。それは、単に CPU の処理速度やネットワークのデータ転送スピードの問題だけでなく、新しいソフトウェア開発言語の登場や開発支援技術の登場などにも大きく影響を受ける。こうした技術発展に「プッシュ」されるかたちで、実現するソフトウェアの機能性や品質も大きく向上していくこととなる。

その一方で、開発されたソフトウェアを受容する市場の変化に強く牽引されるかたちで、ソフトウェア開発を捉えることもできる。これを前述のアプローチと対比させるかたちで、「マーケット・プル」型のアプローチと呼びたい。ソフトウェアは、程度や範囲の差こそあれ、何らかの市場のニーズを充足させるために存在している。パソコン上で動く表計算ソフトは、ビジネスシーンにおける様々なデータの計算効率性の向上というニーズに応え、また携帯電話に内蔵されている組み込みソフトウェアは、通話機能だけに留まらない多様な機能の実現のために開発される。こうして、市場からの強い「プル」に応じて、ソフトウェアに求められる機能性や品質も変化していく。

この「テクノロジー・プッシュ」と「マーケット・プル」という 2 つのソフトウェア開発の捉え方は、現実問題としては相対するアプローチではなく、実際のソフトウェア開発は、技術発展と市場変化の両者のインタラクションのなかで行われるものとして捉えるべきであろう(図 1 参照)⁵。現代のソフトウェア開発は、単に急速な技術発展を活

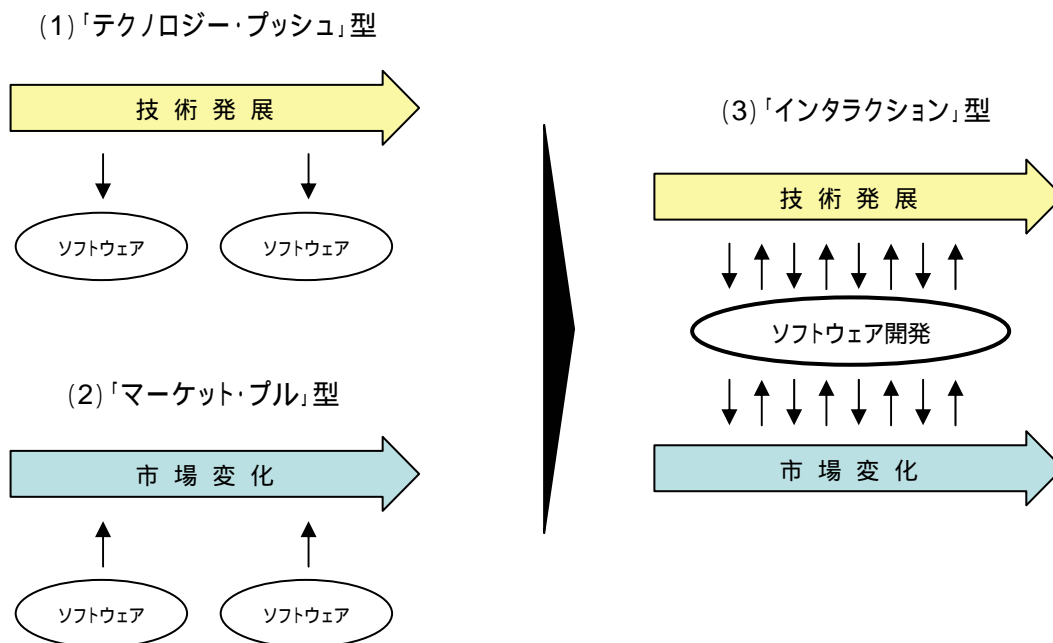
³ 経済産業省『2004 年版組み込みソフトウェア産業実態調査報告書』。

⁴ ソフトウェアを機能単位で分解し、それら部品を組み合わせることで効率的なソフトウェア開発を進める手法をコンポーネント指向ソフトウェア開発 (Component-Based Software Development) という。

⁵ この概念化は、マーケティングの文脈で一般的に議論される「プロダクト・アウト」と「マーケット・イン」の概念と共通する部分が多いが、往々にして「プロダクト・アウトからマーケット・インへ」という時代的方向感と共に議論される概念と比較して、ここでの概念化は純粋な並列概念である。

用するだけでなく、日々刻々変化する市場の変化にも対応していかなければならず、その逆もまた然りである。現代のソフトウェア開発は、技術環境だけでなく市場環境とも密接な関係を持つものであり、それら両方の環境変化と「対話」し、柔軟に適應していく必要に迫られているのである。

図1：2つの環境変化のなかでのソフトウェア開発の捉え方



ソフトウェア開発における戦略視座

このように、これまで比較的閉じられた環境のなかで行なわれていたソフトウェア開発が、現代の急速な技術発展と市場環境の変化のなかでダイナミックな対応を求められるようになった。この変化を端的に表現すれば、まさにソフトウェア開発の「戦略化」が求められるようになったと言える。

一般的な家庭用パソコンのハードウェア性能はほぼ高位平準化し、一部のプロフェッショナル向けソフトや業務用ソフトを除いて、ハードウェア性能がソフトウェア性能を制約することは少なくなった。携帯電話や家電製品なども、各製品間でハードウェア性能面での決定的な違いは少なくなり、ソフトウェアで提供する機能やその拡張性による差別化が多くなってきた。ソフトウェアはハードウェアを目的通りに動かすだけの二次的要素ではなく、ソフトウェアの機能性と品質が、その商品やサービスの売上や評価を直接的に左右する要素になったのである。つまり、これまで愚直に機能的安定性を追求するだけで良かったソフトウェア開発に、その機能がどのようなハード技術によって支えられ、市場のどのようにニーズに答えようとし、そしてその成果として適切な収益が上がるか否かという、まさに「戦略的な視座」が求められるようになってきたのである。

しかし、これまでのソフトウェア開発に戦略の視点がまったく無かったわけではない。ソフトウェアを具体的にどのような要件に基づいてどのような仕様にするのか、またそのソフトウェアの開発プロジェクトをどのように管理するのかなど、より良いソフトウ

ウェアの開発における目的達成の視点としての戦略の概念はこれまでのソフトウェア開発にもあった。

一般的にソフトウェア開発とは、「要件定義」から始まり、「システム仕様」、「ソフトウェア仕様」を決めて、「プログラミング」を行い、「ソフトウェアテスト」、「システムテスト」を経て、全体の「運用テスト」を行うまでの一連のプロセスを指すことが多い。この流れをあたから順を追って進む古典的な開発手法が「ウォーターフォール型」と呼ばれるものである。こうしたリニアな開発プロセスをもとにして、ソフトウェア開発の現場で試行錯誤を続け、「手戻り」⁶による修正コストをできるだけ下げて、より効率的な開発が実現できる新たな方法論が数多く生み出されてきた。「インクリメンタルモデル」、「RAD モデル」、「プロトタイピング」、「スパイラルモデル」、「コンカレント開発モデル」などがそれらの一部である（Pressman, 2000）。しかし、依然開発の現場で参照モデルとして広く活用されているのは、ウォーターフォール型モデルである。国内の主要ソフトウェア・ベンダが取り組んだ約 1000 の汎用コンピュータ向けソフトウェアの開発プロジェクトに関する調査では、全体の 97.2 パーセントにもものぼるプロジェクトがウォーターフォール型の開発プロセスを採用していたことが明らかになっている（情報処理推進機構, 2005b）。

ソフトウェアの開発プロセスを考える際に国内で一般的に用いられているのは、ウォーターフォール型モデルを開発フェーズとテストフェーズで分けて折り曲げた通称「V字モデル」である（図 2 参照）。この V 字モデルをもとにすれば、ソフトウェア開発における戦略の視座は、開発プロセスの最初の段階である「要件定義」、さらにはそれよりも上流の「システム化計画」や「システムの方向性」の過程、すなわち「超上流プロセス」にあるとされる（情報処理推進機構, 2005a）。近年、ソフトウェアの戦略的開発やそれを実現する新たな手法の開発を促進すべく、産官学連携でソフトウェア工学の本格的な研究推進がなされており⁷、そこではまさにこの超上流プロセスにおける戦略視座の重要性が強く説かれている。このソフトウェア開発の超上流プロセスとは、「経営層、業務部門、情報システム部門というステークホルダが、きちんとした合意形成を図る場」であるとされる（同: p.51）。特に、企業向けの情報システム開発（エンタープライズ系ソフトウェア）では、こうした経営陣が積極的に開発に関与する姿勢が極めて大事だと指摘されている。

しかしながら、こうした超上流プロセスにおける戦略視座の重要性の指摘は妥当なように見えるが、前述したようなソフトウェア開発を取り巻く環境変化を鑑みれば、はたして現実的な指摘だと言えるだろうか。たしかに、超上流プロセスにおけるソフトウェアの基本設計（アーキテクチャ）や事業化の方向性の策定には、技術発展と市場変化、両方の環境変化を見据えた的確な意思決定と合意形成が必要なのは間違いない。だが、技術発展はますます進み、消費者ニーズなど市場環境もますます不明瞭になり、さらにはこれまで以上のステークホルダーがソフトウェア開発に複雑に関与するようになってきている。言わば、ソフトウェア開発における戦略の不可視性・不確実性がますます高まっているのである。こうした先の見通しがまったく利かない環境のなかで、開発プロセスの初期段階、それよりさらに前段階の超上流プロセスにおいて、市場投入後の効果も見据えた的確な戦略を策定せよとの指摘はあまりに非現実的とは言えまいか。さら

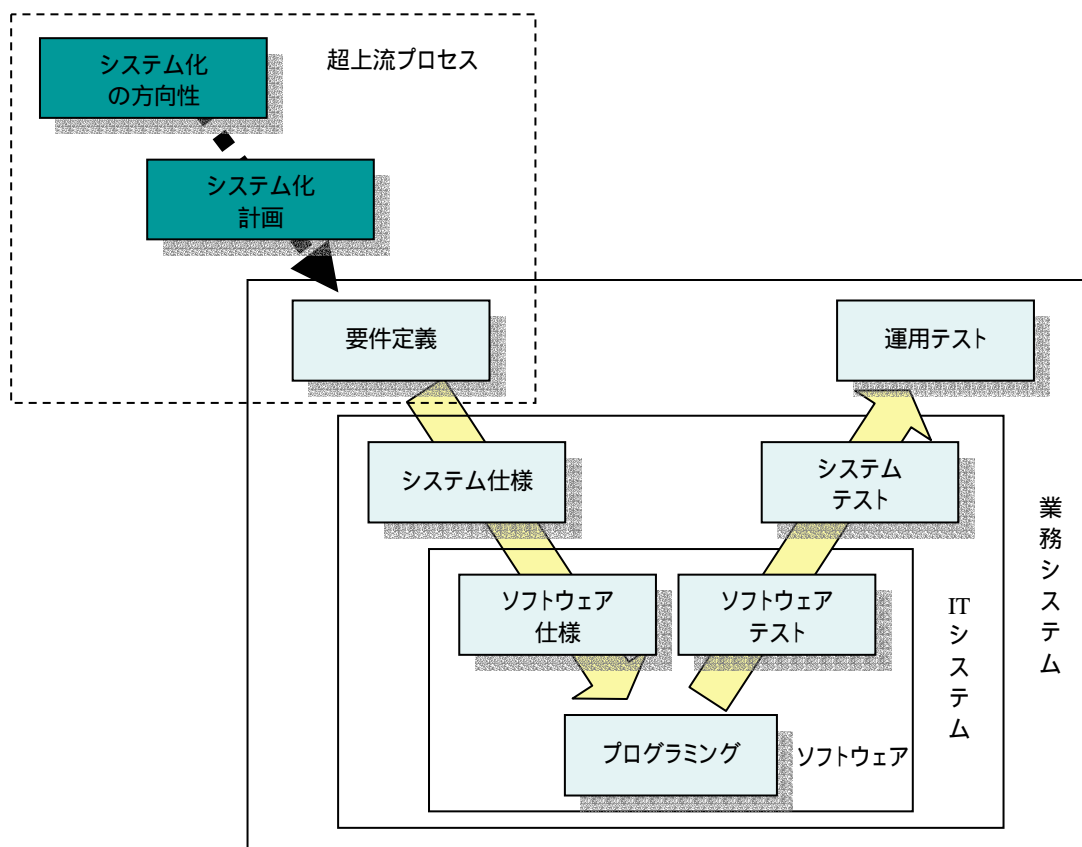
⁶ ソフトウェア開発の一連のプロセスにおける「作業のやり直し」を指す用語。業界用語として一般化している。

⁷ 詳しくは、情報処理推進機構・ソフトウェア・エンジニアリング・センター（SEC）のウェブサイトを参照。http://sec.ipa.go.jp/

には、不確実性の高い環境のなかで超上流プロセスでの戦略策定に無理に拘れば、それこそ「勘」やこれまでの「成功体験」に過度に頼ることになり、結果的に暗闇で鍵を落とした者が街灯の下の明るい場所ばかり探しているという逸話⁸が指し示すような状況に陥る危険はないだろうか。

これまで見てきたように、従来のソフトウェア工学研究における戦略概念の扱いは、開発の超上流プロセスにフォーカスするかたちで議論されてきてはいるものの、近年の環境変化のスピードを鑑みれば、いささか非現実的なものと言わざるを得ない。開発するソフトウェアの戦略的位置づけや役割について、超上流プロセスにおいていかに経営陣と開発現場が合意形成しようが、その後の技術環境や市場環境の変化によっては、その戦略的妥当性が上がることもあれば下がることもある。こうした環境変化のスピードの速いなかで実践されているソフトウェア開発により合致する戦略アプローチは、これまでソフトウェア工学で取り入れられてきたような古典的な計画先行型の戦略アプローチではなく、まったく別の戦略アプローチではないかという仮説を筆者は持っている。この点について、次節で議論を進めたい。

図 2：ソフトウェア開発の V 字モデルと超上流プロセス
(情報処理推進機構 (2005a) をもとに筆者作成)



⁸ ある晩一人の男が街灯の下で何かを探している。通りすがりの人がどうしたのかと聞くと「鍵を落としたので探している」とのこと。この辺りで落としたのか聞くと「いや、あちらの暗いところで落とした」と言う。なぜ落としたあたりを探さないのか聞くと「ここが明るいから」と答えたという逸話。人は往々にして「すべきこと」よりも「できること」を優先してしまいがちであるという隠喩。

経営戦略論の系譜と「シンプル・ルール戦略」

本節では、経営学における経営戦略論の研究蓄積を簡単にレビューし、ソフトウェア開発に関するこれまでの議論との接続を試みる。

経営戦略論は、経営学の諸分野のなかでも最も歴史の浅い領域のひとつである。現代の企業経営における経営戦略の考え方を体系化し研究領域として確立させたのは、1980年に刊行されたポーターの「競争の戦略」(Porter, 1980)であることに対して疑義は少ないであろう。ポーターは、経済学における産業組織論をベースにして、ある産業のなかで他を圧倒する競争優位性を持つためには、その産業構造を分析する必要があると説き、その産業構造の分析枠組みとして、有名な「5つの要因(five forces)」を提示した。それは、産業内の同業者間での競争の激しさ、新規参入の脅威、代替的な製品・サービスの脅威、供給業者の交渉力、買い手の交渉力、の5つである。こうした産業構造の分析を踏まえ、その産業内での自社の優位なポジションの獲得の大切さを説いた。このことから、ポーターに代表される経営戦略論の流れは一般的に「ポジショニング・ベース」の戦略論と呼ばれる。この戦略論の流れは、それまでの業界特殊な事例の集積でしかなかった経営戦略論に、確固たる分析フレームワークを与えたという点で画期的なものであった。

その一方で、こうしたポジショニング・ベースの競争優位の捉え方に対する批判も徐々に出てくることとなる。ポーターの戦略論は、産業構造が企業のあり方や行動規範を規定するというS-C-P(Structure-Conduct-Performance)図式に依拠している。この図式に従う限り、ある企業の競争優位は、産業という外部競争環境により決定され、その競争環境をいかにして優位な状況に持っていかかが勝敗の分かれ目になるとされてきた。しかし、個々の企業にはそれぞれ固有の資源があり、その資源が独自の競争優位をもたらしているのだという説明も可能である。1980年代における日本の製造業の躍進を分析したアメリカの経営学者の研究により、それまでのポジショニング・ベースの戦略論の枠組みでは、日本企業の優位性を十分に説明できないということが徐々に明らかになった(e.g. Prahalad & Hamel, 1990)。そうした研究をもとに、競争優位の源泉として着目されるようになったのが、事業を支える中核技術やそれを実践していく暗黙的ノウハウや組織能力などの「資源」である。ある企業の競争優位は、業界内の他のステークホルダーとの関係によって決まるのではなく、企業内部にある差別性が高く模倣困難な資源によって構築されるのだという説明がなされるようになった。こうした経営戦略論の流れは近年「リソース・ベース」の戦略論と呼ばれている。

この流れの代表的研究者であるバーニーは、企業の競争優位の分析にあたり「VRIOフレームワーク」という枠組みを提唱した(Barney, 2002)。すなわち、経済価値(Value)、希少性(Rarity)、模倣困難性(Imitability)、組織(Organization)の4つの側面からその企業の保有する経営資源やケイパビリティを分析し、継続的な競争優位の構築を目指すという枠組みである。こうしたリソース・ベース戦略論は、これまでポジショニング・ベース戦略論が主に扱った産業構造というスタティックな要因からの分析では扱われてこなかった「長期的な資源蓄積を促すメカニズム」(青島・加藤, 2003: p.109)に光を当てた点で高く評価され、経営戦略論の新たな柱として確固たる地位を獲得した。

このポジショニング・ベース戦略論とリソース・ベース戦略論以外にも、近年様々な戦略論のアプローチが提唱されているが⁹、この2つが現代の経営戦略論の二大潮流を形成していることに異論は少ないであろう。そこで、まずはこの2つの経営戦略論のアプ

⁹ 現代の経営戦略論の様々なアプローチの比較は、河合(2004)に詳しい。

ローチを現代のソフトウェア開発の文脈において、その適用可能性を検討してみると、ポジショニング・ベース戦略論もリソース・ベース戦略論も、その前提認識に決定的な問題があることがすぐに分かる。それは、戦略立案のプロセスにおける時間設定が極めて緩慢なことである。ポジショニング・ベースであろうが、リソース・ベースであろうが、基本的な視座としては、比較的变化が少ない競争環境や安定的なステークホルダー関係を前提にしている。しかし、ネット時代のソフトウェア開発の世界は半年先の状況も読めないほど、猛烈なスピードで変化を遂げていく。「ドッグ・イヤー」で進む技術発展や活発な新規参入などの市場変化のなかで、新たなソフトウェアの開発と市場投入をスムーズ且つ迅速に行なっていかなければならないとすれば、それまでポジショニング・ベース戦略論やリソース・ベース戦略論が前提にしてきた戦略立案のタイムスパンは長すぎるのである。急速な環境変化に絶えず直面しているソフトウェア開発の現場にしてみれば、他の競合商品/サービスとの関係のなかで適切なポジションを狙って開発に取り組みたり、模倣困難な差別性や参入障壁を生み出せるような資源を蓄積したりする時間の余裕はまったくない。こうした現実を見れば、ソフトウェア開発において妥当な戦略フレームワークは、ポジショニング・ベースでもリソース・ベースでもない、別の戦略アプローチであると考えるのが妥当であろう。

そこで取り上げるもう一つの経営戦略論の流れが、アイゼンハートとサルによる「シンプル・ルール戦略」である(Eisenhardt & Sull, 2001)。彼女らがとりわけ注目したのは、変化のスピードが速く不透明な「高スピード市場 (high-velocity markets)」における競争優位の形成メカニズムであり、その事例として扱ったのが Yahoo!などのネット企業であった。Yahoo!の成功について、ポジショニング・ベース戦略論の視座から分析すれば、競争環境としては参入障壁も少なく新規参入の脅威も高かったため優位なポジション戦略は採り難く、単にネット市場そのものが魅力的であっただけで、Yahoo!は市場の成長に引っ張られただけだという結論ぐらいしか導き出せない。一方、リソース・ベース戦略論の視座から見ても、Yahoo!が取り立てて差別性の高い模倣困難な経営資源を保有していたわけではないため、Yahoo!の成功の要因は説明し難い。ポジショニング・ベース、リソース・ベース、双方の戦略論の枠組みに依拠する限り、Yahoo!やその他のネット企業には「戦略が無かった」という結論に至ってしまう。

しかしながら、Yahoo!はネットポータルビジネスで 1990 年代後半以降明らかにずば抜けた業績をあげており、これを受けてアイゼンハートらは、Yahoo!には戦略が無いのではなく、ネットのような「高スピード市場」に特有の戦略論が存在したとする。それが「シンプル・ルール戦略」である。他の2つの戦略論のアプローチと比較すれば、「優位なポジションの確立」を目指すポジショニング・ベース戦略、「経営資源の効果的活用」を目指すリソース・ベース戦略に対し、シンプル・ルール戦略における戦略目的は「チャンスの追求」である(表1参照)。シンプル・ルール戦略では、戦略的に重要ないくつかのプロセスを選択し、それを推進するためのシンプルなルールを設定する。そのルールは主に5つのカテゴリーに分けられるとされる。

市場のチャンスを捉えるための枠組みを提供する「ハウ・トゥ・ルール」
数あるチャンスを素早く取捨選択するうえで必要な「バウンダリー・ルール」
複数のチャンスに対する資源配分の意思決定に役立つ「プライオリティ・ルール」
主要な戦略プロセスのスピードを決める「タイミング・ルール」
過去のものとなったチャンスから抜け出す時に役立つ「イグジット・ルール」

アイゼンハートらは、Yahoo!のサービス開発には主に4つのシンプル・ルールがあっ

たと指摘する。それは、開発中の各商品の優先順位を把握する、各エンジニアは必ずどのプロジェクトでも働けるようにする、ユーザーとのインターフェースは常にYahoo!ならではの体裁を保つ、新商品は目立たぬよう導入する、の4つである。このルールに従ったことで、Yahoo!は怒涛の変化を見せるネットビジネスの世界で圧倒的な地位を確立するまでに至ったのである。

変化の速い「高スピード市場」においては、企業は素早くチャンスを発見し柔軟に動かねばならない。ただし、柔軟に動くといっても、無秩序に動くのではない。いくつかの主要な戦略のプロセスと単純なルールに焦点を絞ることで、複雑で変化し続ける市場のなかで「次の一手」を素早く発見し迅速に実行に移すことができるのである。当然ながら、これら3つの経営戦略論のアプローチはそれぞれ長短がある。しかし、短期間のうちに大きく変化する技術環境と市場環境とのインタラクションのなかで、品質の高いソフトウェアを作り上げ、的確に市場投入していくことが求められている現代のソフトウェア開発に対し、時間設定の緩慢なポジショニング・ベース戦略論もリソース・ベース戦略論も有益な示唆を与えることは難しい。変化の激しいIT産業においてますます巨大化・複雑化が進むソフトウェア開発の現場の問題にもっとも的確に適用できるのはこのシンプル・ルール戦略なのではないだろうか。このことは、「アジャイル開発」や「適応型開発プロセス」などの最新のソフトウェア開発のアプローチとの親和性も高いと思われる（Cockburn, 2001；Highsmith, 1999）。

表1：戦略の3つのアプローチ
（Eisenhardt & Sull, 2001）

	ポジショニング戦略	リソース・ベース戦略	シンプル・ルール戦略
戦略目的	ポジショニングを確立する	経営資源のフル活用	チャンスの追求
戦略ステップ	<ul style="list-style-type: none"> 魅力的な市場を特定する 防御可能なポジショニングを定める 補強し防御する 	<ul style="list-style-type: none"> ビジョンの確立 経営資源の蓄積 各市場で経営資源をフル活用 	<ul style="list-style-type: none"> 混乱に身を投じる 動き続ける チャンスをつかむ ラストスパートをかける
戦略的な問いかけ	我々はどこにあるべきか	我々は何であるべきか	どのように前進すべきか
優位性がどこで生まれるか	きっちり統合された行動体系を伴った、ユニークで価値のあるポジショニング	模倣困難な希少資源	主要なプロセスと企業独自のシンプル・ルール
最も有効な市場	変化が緩慢で構造が安定的な市場	適度に変化し、構造が定まった市場	とどまることなく変転し、先の見通しのたたない市場
優位性の継続性	持続可能	持続可能	持続不能
リスク	状況が変化した時にポジショニングを変更するのが困難になる	動きが鈍くなり、状況の変化に合わせた新たな資源の蓄積に遅れる	有望なチャンスにしっかり腰を据えて取り組めない
業績目標	収益性	当該市場の長期的な支配	成長

事例：ネットサービス開発

本節では、現代のソフトウェア開発の実践におけるシンプル・ルール戦略の枠組みの適用可能性を考察する。本格的な実証研究は別稿に譲り、ここでは主に筆者のヒアリング調査をもとにした簡単な事例研究に留める。

ここで扱うソフトウェア開発の事例は、近年活発に市場に導入・展開されている「ネットサービス」の分野である。ここで言うネットサービスとは、インターネットを介して各種の機能やサービスを提供するビジネスモデルを指す。企業の SCM (Supply Chain Management) や CRM (Customer Relationship Management) 業務の支援するアプリケーションをネットを介して契約企業に提供するアプリケーション・サービス・プロバイダ (ASP) サービスから、一般のネットユーザーが利用するブログやソーシャル・ネットワークワーキング・サービス (SNS) など、近年その多様化が急速に進んでいる。一般向けのネットサービスは無料で提供されるものが少なくないため、ネットサービス全体での市場規模の測定は難しいが、ASP サービスに限って言えば、2000 年の国内市場規模は 20 億円であったが、2005 年には 975 億円に達すると予測されている¹⁰。

ネットサービスのソフトウェアは、従来のアプリケーション・ソフトのように単体のパッケージ製品として存在するものではない。ネットサービスの機能は、あくまで「サービス」としてユーザーに提供され、ユーザーはそれをウェブ・ブラウザ上で使用する。これは、パッケージ製品としてのソフトウェアから「サービスとしてのソフトウェア (Software as Service)」への進化とも言える。

今回このネットサービスの分野におけるソフトウェア開発を事例として選んだ理由は、その圧倒的に柔軟で素早い開発プロセスにある。従来のパッケージ型ソフトウェアは、入念な設計、プログラミング、テストを経て、満を持して市場投入されるものであったが、ネットサービスは市場投入した後も容易に修正や機能追加が可能である。すなわち、顧客の多様なニーズや市場環境の変化に対して、素早く柔軟に適應することができるのである。こうした柔軟性ゆえに、ネットサービス・ソフトウェアの開発は、従来のソフトウェア開発と比べて、格段に素早い開発と市場投入、そして技術と市場 (ユーザー) とのダイナミックな対話を求められる。ネットサービス市場はまさに「高スピード市場」であり、現代のソフトウェア開発においてもっとも変化の激しい環境にあると思われる¹¹。

ここで具体的な事例として取り上げるのは、国内のソーシャル・ネットワークワーキング・サービス (SNS) 大手の「グリー (GREE)」である (写真 1 参照)¹²。SNS とは、ユーザーが互いに紹介し合いながら、ネット上で新たな友人関係を広げることを目的にしたオンライン・コミュニティ・サービスであり、その多くは無料で提供されている。グリーは当時まだ楽天株式会社の社員だった田中良和氏が個人的なプロジェクトとして 2004 年 2 月に運営を始めたサービスである。運営開始後約 1 ヶ月でユーザー数は 1 万人にまで増え、2005 年 7 月の時点でユーザー数は 20 万人を超すまでに至った。現在の運営は、2004 年 12 月に田中氏が設立したグリー株式会社が行なっている。

¹⁰ 総務省『情報通信白書』平成 13 年度版。

¹¹ こうしたネットサービスの開発事例は、従来の情報システム開発やパッケージ型ソフトウェア開発の数と比較すれば、依然極めて限られたものであろう。しかしながら、これまでパッケージ型ソフトウェアによって提供されてきた機能を、Ajax (Asynchronous JavaScript + XML) 技術を用いた Web アプリケーションとして商品化/サービス化する事例が昨今急速に増えており、こうした最新のネットサービス開発の実践を戦略的な視座から分析することの意義は大きいと思われる。

¹² <http://gree.jp/>

写真 1：ソーシャル・ネットワーキング・サービス「GREE」の画面サンプル



グリーの開発と運営において特徴的なのは、2005年10月にサービス内容をそれまでのアルファ版(開発初期版)からベータ版(開発途上版)にバージョンアップした際に、グリーのサービスを「永遠のベータ版(Perpetual)」と位置づけたことである。すなわち、常に進化を続けるネットサービスとして、いつまでたっても完成版とはなり得ず、永遠にベータ版が続いていくという意味である。この言葉は、ネットサービス開発の本質を捉えている。絶え間ない技術発展と市場変化のなかで、絶えず双方の環境変化と対話しながら、ダイナミックな開発とサービス導入をしていくということである。

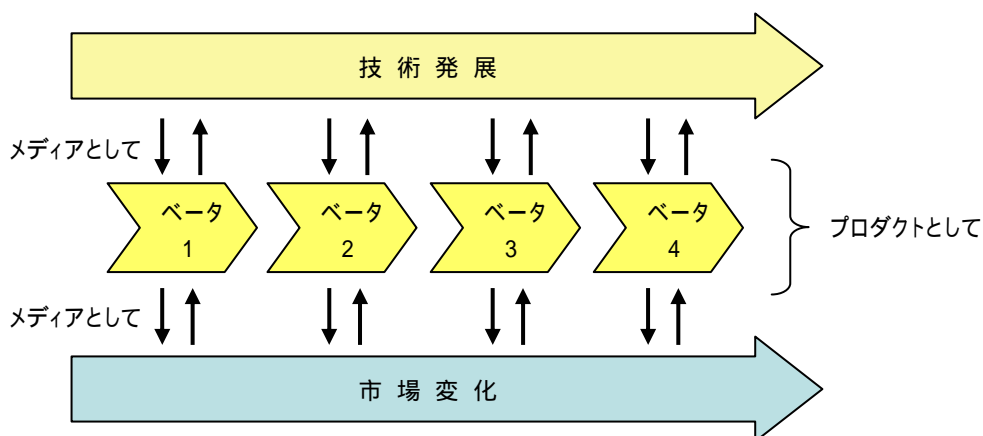
この点に関連して、田中氏は「常にユーザーの使い勝手やベネフィットを考え、最適な技術を活用していく」と語っている。また、こうした速い環境変化のなかでサービスを提供する立場として、「瞬間瞬間の素早い判断が極めて大事」で、目前のチャンスを的確に捉え最大限に活かす姿勢を重視している。また、そうした変化のなかでの「意思決定の効率化」を常に考えており、「ある機能やサービスができあがるまでの時間」が事業全体に対して非常に大きな影響を与えるという。また、そうしたネットサービスを開発し運営していく企業としては、「変化に柔軟に対応できる体制」が極めて重要だとも語っている¹³。こうした姿勢や考え方は、まさしくシンプル・ルール戦略であり、それをネットサービスという急成長する領域でいままさに実践している事例と言えよう。田中氏の言葉にはすべて「スピード」と「柔軟性」に関連する内容が含まれており、それがグリーの開発アプローチにも密接にリンクしているのは想像に難くない。

¹³ 一連の田中氏のコメントは、2005年9月16日に筆者が田中氏に対して行なったヒアリングから。

ここでネットサービスの開発アプローチとして特に注目したいのが、「ベータ版」の概念とその役割である。上記したように、グリーのサービスは「永遠のベータ版」という位置づけであるが、グリーに限らず、その他のネットサービスに関しても同様の考え方が取り入れられている。もともとベータ版の役割とは、完成直前のバージョンとして公開して、テストユーザーからの評価を得ながら、最終的なバグフィックス（問題あるプログラムの箇所の修正）を行なうというものであるが、ネットサービスではそのベータ版が最終製品のかたちでもある。言い換えれば、ネットサービスのベータ版は、それがユーザーに提供される製品であるという意味での「プロダクト」であると同時に、技術環境と市場環境を結び、それを最終サービスの形態へと繋ぐ「メディア」でもある。このベータ版の役割の「二重性（duality）」は、これまでのソフトウェア開発にはなかった概念であり、このベータ版の働きがサービスの品質を左右するばかりか、様々なステークホルダーとの関係を構築し、ひいては最終的な収益性にも影響を与える重要なファクターとなる可能性がある。さらに、上記したように、このベータ版の在り方そのものが、ネットサービス開発の戦略形成に極めて重要な意味を持っている（図3参照）。

グリーの事例に見られるように、昨今のネットサービス分野の開発・導入競争は、これまでのソフトウェア開発とはまったく異なる時間軸でなされている。「サービスとしてのソフトウェア」や「永遠のベータ版としてのネットサービス」など、これまでのソフトウェア開発の考え方を当てはめることができない事態が今後ますます増えていくだろう。その場合、経営戦略の視座からもう一度ソフトウェア開発を捉え直し、技術と市場とのダイナミックなインタラクションのなかで、どのような状況の際にどのような戦略実践が必要なのか、今後さらなる実証研究が求められていると言えよう。

図3：ネットサービス開発におけるベータ版の役割の二重性



おわりに

本稿では、ソフトウェア工学と経営戦略論の理論的架橋という大きなテーマを念頭に置き、ソフトウェア開発の実践における戦略概念に位置づけについて考察を行った。まず、環境変化のスピードがますます速くなっているネット時代のソフトウェア開発において、技術と市場としっかり対話して開発に活かす戦略的視座の重要性は高まってきているという事実、にもかかわらず、これまでソフトウェア工学と経営戦略論の両分野間

の研究の相互交流はこれまで皆無に等しく、「ソフトウェア戦略」や「戦略的ソフトウェア開発」という言葉がまったく空虚なものになってしまっているという事実を指摘した。それを踏まえ、経営戦略論の系譜をおおまかにレビューした後、特にアイゼンハートらが提唱する「シンプル・ルール戦略」の枠組みが現代のソフトウェア開発の実践にうまく適合するという点を考察した。そうした考察を、近年活発な展開を見せるネットサービス分野に当てはめて、事例として SNS 大手のグリーを取り上げ、彼らの戦略には暗黙的にシンプル・ルール戦略が採用されているという点を指摘した。また、ネットサービス開発においてベータ版が果たす役割の「二重性」について若干の考察を行なった。

本稿で展開した数々の議論は、依然として仮説レベルに過ぎない点が多々ある。また、ソフトウェア開発をすべて一括りにしており、その開発規模や運用環境などの場合分けをした精緻な議論にまでは至っていない。こうした問題の解決は今後の課題としたい。今後更なる事例研究の追加や掘り下げ、また実証データの収集・分析などを行なうことで、ソフトウェア工学と経営戦略論の架橋が実現に向かうように、本稿で扱われた論点を今後も引き続き考察していきたい。

[謝辞]

本稿執筆の事前調査の段階で、グリー株式会社の田中良和社長にはご多忙のなか快くヒアリングに応じて頂いた。ここで改めて御礼申し上げたい。また、2005 年 11 月 12-13 日に開催された経営情報学会・2005 年度秋季全国研究発表大会(中村学園大学)において本稿の骨格となる内容を発表した際に、諸先生方から貴重なコメントを頂いた。重ねて御礼申し上げたい。

参考文献

- 青島矢一・加藤俊彦(2003)『競争戦略論』, 東洋経済新報社.
- Barney, J.B. (2002). *Gaining and Sustaining Competitive Advantage*. (2nd edition) Pearson Education, New Jersey. (邦訳:岡田訳「企業戦略論(上)(中)(下)」, ダイヤモンド社, 2003 年)
- Cockburn, A. (2001). *Agile Software Development*. Addison-Wesley, Reading, MA. (邦訳: テクノロジックアート訳『アジャイル・ソフトウェア開発』, ピアソン・エデュケーション, 2002 年)
- Cusumano, M.A. (1991). *Japan's Software Factories: A Challenge to U.S. Management*. Oxford University Press, New York, NY. (邦訳: 富沢・藤井訳『日本のソフトウェア戦略: アメリカ式経営への挑戦』, 三田出版会, 1993 年)
- Cusumano, M.A. (2004). *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*. Free Press, New York, NY. (邦訳: サイコムインターナショナル訳『ソフトウェア企業の競争戦略』, ダイヤモンド社, 2004 年)
- Eisenhardt, K.M. and D.N. Sull (2001). Strategy as Simple Rules. *Harvard Business Review*. Vol.79, No.1, pp. 107-116. (邦訳: 「複雑な市場環境を生き抜く『シンプル・ルール戦略』」, ダイヤモンド・ハーバード・ビジネス, 2001 年 5 月号)
- Highsmith, J.A. (1999). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing, New York, NY. (邦訳: ウルシス

- テムズ訳『適応型ソフトウェア開発』, 翔泳社, 2003年)
- 情報処理推進機構(ソフトウェア・エンジニアリング・センター)(2005a)『経営者が参画する要求品質の確保～超上流から攻めるIT化の勘どころ』, オーム社.
- 情報処理推進機構(ソフトウェア・エンジニアリング・センター)(2005b)『ソフトウェア開発データ白書2005』, ソフトウェア・エンジニアリング・センター編, 日経BP.
- 河合忠彦(2004)『ダイナミック戦略論: ポジショニング論と資源論を超えて』, 有斐閣.
- Porter, M.E. (1980). *Competitive Strategy: Techniques for Analyzing Industries and Competitors*. Free Press, New York, NY. (邦訳: 土岐ほか訳『競争の戦略』, ダイヤモンド社, 新訂版, 1995年)
- Prahalad, C.K. and G. Hamel (1990). The Core Competence of the Corporation. *Harvard Business Review*. Vol.63, No.3, pp. 79-90.
- Pressman, R.S. (2004). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, New York, NY. (邦訳: 西ほか訳『実践ソフトウェアエンジニアリング: ソフトウェアプロフェッショナルのための基本知識』, 日科技連出版社, 2005年)