2017年度　博士学位申請論文

ソフトウェア開発プロセスにおける分業構造と知識労働

―日本の受託ソフトウェア開発の組織問題―

Division Structure and Knowledge Labor in Software Development Process

: Organization Problem of Custom Software Development in Japan

指導教員　亀川　雅人　教授

立教大学大学院ビジネスデザイン研究科ビジネスデザイン専攻

学生番号　12WG003J

平井　直樹
HIRAI, Naoki

第１章　研究の目的と方法 1

(1)　本研究の背景 1
(2)　本研究の目的 3
(3)　本研究の方法 7
(4)　本研究の構成と用語の定義 11

第２章　ソフトウェア産業の現状 18

(1)　日本のソフトウェア産業の現状 18
(2)　日本のカスタムソフトウェアへの偏重 33
(3)　小括 36

第３章　先行研究と問題設定 38

(1)　これまでの先行研究との関係 38
(2)　日本のソフトウェア産業に関する研究 40
(3)　知識労働に関する研究 48
(4)　問題設定・残された分析課題 53

第４章　ソフトウェアの製品設計思想 55

(1)　アーキテクチャーの分類 55
(2)　アーキテクチャーの選択 65
(3)　小括 69

第５章　ソフトウェア開発の特徴と手法 71

(1)　ソフトウェアの開発プロセス 71
(2)　開発手法の変遷 78
(3)　開発手法の貢献と限界 83
(4)　小括 90

第６章　ソフトウェア産業の下請け構造と製造工場化 92

(1)　ソフトウェア産業の成り立ち 92
(2)　ソフトウェア産業の下請け構造 96
(3)　ソフトウェア・ファクトリー 103
(4)　小括 107

第７章　ソフトウェア開発プロセスにおける知識労働 108

(1)　ソフトウェア工学 108
(2)　ソフトウェア開発プロセスにおける知的側面 113
(3)　小括 118

第 8 章　ソフトウェア開発における知識労働と分業の問題－開発事例研究－ 120

(1)　調査概要 120
(2)　事例 1：要件定義、設計、プログラム作成の分業 125
(3)　事例 2：上流工程と下流工程の企業間分業 131
(4)　事例 3：要件定義と設計以降の分業 137
(5)　小括 142

第 9 章　ソフトウェア開発プロセスにおける知識労働と分業の問題－考察－ 144

(1)　開発事例から明らかになった問題 144
(2)　ソフトウェア開発の分業構造と問題解決 146
(3)　ソフトウェア開発プロセスにおける知識創造活動 150
(4)　小括 156

第 10 章　結論と今後の課題 158

(1)　結論 158
(2)　残された研究課題 162

参考文献 168

官公庁・社団法人データ・資料等 182
WEB サイト 184

図 目次

図 1-1 本研究の構成 12
図 1-2 ユーザー企業と IT ベンダー（受発注関係） 15
図 2-1 ソフトウェア業の売上高・従業者数の推移 19
図 2-2 ソフトウェア業の従業員規模別の年間売上高と事業所数（平成 26 年） 20
図 2-3 業種別開廃業率（平成 26 年） 20
図 2-4 企業の IT 投資の推移（ソフトウェア業） 21
図 2-5 プロジェクト工数比率 22
図 2-6 全ソフトウェア行数比率 22
図 2-7 新規ソフトウェア開発行数比率 23
図 2-8 パッケージソフトウェアの種類と範囲 27
図 2-9 カスタムソフトウェア業の資本系列別構成 30
図 2-10 ソフトウェア開発のアウトソーシングの流れ（オフショア開発） 32
図 4-1 パソコンの構成図とインターフェース 56
図 4-2 工程アーキテクチャーにおける連結関係と対応関係 57
図 4-3 コンピューターの垂直統合 59
図 4-4 コンピューターの水平分業 60
図 4-5 モジュラー型アーキテクチャーの関係性の例（パソコンのシステム） 63
図 4-6 インテグラル型アーキテクチャーの関係性の例（自動車） 63
図 4-7 設計情報のアーキテクチャー特性による製品類型 66
図 4-8 垂直統合と垂直分業 68
図 5-1 ソフトウェア開発の工程分離の例 76
図 5-2 一般的な WATERFALL MODEL 80
図 5-3 AGILE（SCRUM 開発手法） 82
図 5-4 WATERFALL MODEL による V 字型モデル 84
図 5-5 コンカレントエンジニアリングのようなソフトウェア開発 86
図 5-6 開発プロセスの採用（2010 年） 90
図 6-1 ソフトウェア開発の請負・下請け関係 96
図 6-2 ソフトウェア産業の下請け構造 99
図 6-3 ソフトウェア産業の取引構造 100
図 6-4 ソフトウェア・ファクトリーのイメージ 104
図 7-1 製造業とソフトウェアのデザインと製造・組立 118
図 8-1 事例 1：開発システムのイメージ 126
図 8-2 事例 1：工程と分業構造（担当） 127
図 8-3 事例 1：メンバー構成 128
図 8-4 事例 1：工程間の連携 130
図 8-5 事例 2：開発システムのイメージ 132

図 8-6 事例 2：工程と分業構造（担当） 133
図 8-7 事例 2：メンバー構成 133
図 8-8 事例 2：工程間の連携 136
図 8-9 事例 3：開発システムのイメージ 138
図 8-10 事例 3：工程と分業構造（担当） 138
図 8-11 事例 3：メンバー構成 139
図 8-12 事例 3：工程間の連携 141
図 9-1 ソフトウェア開発の困難性と試行錯誤 152

表 目次

表 1-1 本研究で使用される主な用語の表記 17
表 2-1 日本標準産業分類（平成 25 年 10 月改定） 25
表 2-2 ソフトウェア業の種類 26
表 2-3 ソフトウェア業の業務種類別年間売上高（平成 26 年） 29
表 4-1 モジュラー型とインテグラル型のアーキテクチャー比較 62
表 5-1 ソフトウェア開発の標準的なプロセスと職種の作業範囲 72
表 5-2 WATERFALL MODEL と AGILE の開発スタイル等の違い 87
表 6-1 コンピューターメーカーのソフトウェア・ファクトリーへの取組み 105
表 7-1 ハードウェア・ソフトウェア開発環境の変化 109
表 8-1 開発事例の分析のフレームワーク（焦点） 121
表 8-2 ソフトウェア開発事例のまとめ 124

第1章 研究の目的と方法

(1) 本研究の背景

ソフトウェアは、我々の身の回りに多く存在し、例えば銀行の ATM や、ビルの空調システム、鉄道や飛行機の運行システム、コンビニエンスストアの商品管理、映画やコンサートのチケット予約など、さまざまな分野と場所で利用されている。

このようなソフトウェアのうち、日本では企業の固有のシステムを対象とするカスタムソフトウェアが 8 割を占めている（経済産業省経済産業政策局調査統計部サービス統計室）。このソフトウェアの開発は、企業の組織戦略として、ハードウェアなどの製造業やソフトウェアに似たようなアーキテクチャーを持つといわれる建築業で培われた知見や管理手法などを取り入れながら発展してきた（妹尾, 2001; 権藤・明石・伊知地・岩崎・河野・豊田・上田, 2009）。

日本のソフトウェア産業は、1960 年代から 1980 年代にかけて金融機関などの基幹システムを対象としたメインフレームと呼ばれる大型コンピューターのソフトウェア開発が主流であった。大規模な開発が中心であり、顧客の提示した仕様に沿ったソフトウェアをスケジュール通りに作成することが求められ、そのようなソフトウェアの開発は、顧客あるいは元請け会社から仕様が提示され､売上高はほぼプロジェクトへの投入人員数に比例し、業績確保のために人の質よりも数優先の経営戦略が採用されてきた（尾裕, 1997）。

こうしたソフトウェアの開発は、機能をモジュールごとに分離するとともに、作業工程を分割することで分業が行われてきた。特に、要件定義や設計などの上流の工程からプログラム作成やテストなどの下流の工程へと流れていく分業構造で構成され、この開発手法は Waterfall Model[1]と呼ばれた（Royce, 1970; Brooks, 1975, 1995; Bell and Thayer, 1976; Larman, 2003; 高橋, 2010; 小椋, 2013）。この分業構造は、安定した画一的なプロセスによるソフトウェアの開発方法であり、品質や納期といったプロジェクトマネジメントの面で欧米と比較して優れていたともいわれている（Cusumano, 2004）[2]。一方で、こうした分業構造に基づく開発においては、下流工程が上流工程からの指示に従って遂行される作業工程として捉えられる傾向にあり、下流工程の作業は代替可能な労働と位置付けられ、しばしばアウトソーシングの対象となってきた。

このような中、1990 年代以降、パーソナルコンピューターの普及により、コンピューターの互換性の普及や小型化、軽量化が進み、ソフトウェアの開発環境は著しく変化した。

---

[1] ウォーターフォール・モデル。Waterfall Development とも呼ばれる開発手法。詳細については本文中で後に述べる。

[2] Cusumano（2004）によると、日本のプログラムは欧米に比べ、2.5 倍程度の差が出ているという。ただし、Cusumano の研究は各国の労働条件などを考慮していない。日本の場合、残業や休日出勤などによる長時間労働を含めた数値であり、必ずしも日本のソフトウェア産業は生産性が高いとは限らない。

ソフトウェアの利用とその重要性が高まるとともに、顧客のニーズは多様化し、企業の中枢を担う基幹システムを含め、システムはより現実のビジネスを反映しようと、複雑、かつ大規模になっていった（阿草, 2009）。くわえて、多様なソフトウェアで構成される情報システムは、当初の業務を制御し自動化することから、企業や組織の問題解決のための意思決定の支援、企業戦略への貢献、ビジネスプロセスの再設計（Hammer and Champy, 1993）といったことまでシステムに組み込むことを求められるようになった。

そして、2000 年代に入り、ソフトウェアは IoT[3]の推進やネットビジネスの登場などの環境の変化に、より一層迅速に対応することが求められ、これまでのコスト削減だけではなく、企業のビジネスに貢献する付加価値までも求められるようになっていったのである（保田・大石・吉田・山田, 2001; 立川, 2005）。

このように、ソフトウェアの利用目的やビジネスモデルが大きく変化していくなか、日本のソフトウェア産業は従来の製造業をモデルにした 1960 年代の大型コンピューター向けの安定した開発手法を維持しようとしていたため、このような変化に十分な対応ができていない（保田・大石・吉田・山田, 2001）。

とりわけソフトウェアをはじめとした IT 産業は、技術の進歩が非常に早く、その時点で注目を浴びているような技術でも数年後には時代遅れになってしまうようなことが多い。また、ソフトウェアの開発や保守のための専門的な技術知識や関連する業務分野の幅広い知識が必要であり、それらを組み合わせた高度な知識創造活動としての側面を有するものでもある。しかし、多くの企業では社内にそのような専門的な技術者を抱えていることは少なく、ソフトウェアを導入、運用するためには、外部のソフトウェア開発の専門企業によるプロフェッショナルなサービスを受ける必要がある。独立行政法人情報処理推進機構編（2012）よると、特に日本国内のソフトウェア開発では、内製は 1/4 に留まっており、3/4 が外部のソフトウェア開発の専門企業に委託されている一方、アメリカでは、パッケージソフトウェアの購入と外部の企業へのソフトウェア開発の委託は全体の約 2/3 であり、約 1/3 が内製しているという。

さらに、ソフトウェアの開発はアウトソーシングが進み、作業工程を分割し、一部では海外の専門企業へ開発作業などを委託するオフショア開発が進展している[4]。このオフショア開発は急速に拡大していくことが見込まれており、中国やインド、ベトナムなどオフシ

---

[3] コンピューターだけでなく、さまざまな物をインターネットなどの通信手段を介して制御や計測などを行うこと（IT 用語辞典 e-Words）。

[4] 2011 年 10 月末時点のオフショア企業の提示価格よると、例えば、オフショア開発の 1 ヶ月に要する金額は、日本で 60～80 万円近くかかるものが、中国の安いところでは 30 万円を下回るケースも存在する。

また、高橋（2013）が 2013 年 6～8 月に行った聞き取り調査によると、詳細設計工程から単体テスト工程までの業務の月額は、東京の中小企業が 50 万円なのに対し、中国の北京や上海で 35 万円、大連で 27 万円、ベトナムのホーチミンで 20 万円だという。

ョア開発先として代表的な国以外にも、カンボジアなどの東南アジアやベラルーシといった東ヨーロッパにまで拡大している状況である。しかしながら、そのようなオフショア開発における分業構造は、旧来の開発手法を維持するために、国内で不足した労働力の確保やコスト削減をもっぱら目的として進められてしまうことが多かった。

特にそうしたアウトソーシングの対象となってきたのは、ソフトウェア開発の下流工程に位置する製造工程、つまりプログラム作成等の作業であった。このことは、日本のソフトウェア産業においては、こうしたプログラム作成業務が企業のコア・コンピタンスに結びつくような知識労働とはみなされておらず、代替可能な労働と位置付けられていることを示唆している。

(2) 本研究の目的

① 研究目的

本研究は、わが国の受託ソフトウェア産業が直面する、「変化の速い市場には対応できず、さらに革新的なイノベーションも期待することができない」という問題に対し、未だ多くの企業が依然として「画一的なソフトウェアを製造する工場モデル」を採用し、その結果、ソフトウェアの開発プロセスを安易に分割してしまうこと、とりわけ開発プロセスにおいて設計工程とプログラム作成工程を分離することで、プログラムの設計と作成の間の連携が分断され、その結果開発プロセスにおける試行錯誤を通じた創造的な問題解決が阻害されることを明らかにする。その上で、本研究は質の高い、革新的なプログラム開発には、こうした設計および作成工程間の連携が不可欠であり、そのためには下流工程を外部調達の容易な代替可能な作業としてではなく、知識労働として捉える必要があることを指摘するものである。

ソフトウェア産業は、これまで高品質のソフトウェアを効率よく開発しようと、製造業から工程管理などの開発手法を援用してきた。しかし、製造業ではライン生産方式の下で工場による大量生産を行うようなところもあれば、町工場のように一品受注生産を行うところも存在する。ソフトウェアの開発は、大量生産を前提とした工場の肉体労働のようなものとは異なっており、むしろ町工場のような一品受注生産型に近い。

ところでプログラミングには、プログラミング言語をはじめとして、ソフトウェア工学を中心とした技術者による専門的な技術[5]と技能が必要であり、質の高い、革新的なプログラムの開発には、顧客ニーズに対する理解、経験的に把握された問題解決に関する知識が要求されることも考えると、高度な知的作業としての側面を備えると考えられる。そのため、ソフトウェア産業は、従来の製造業などとは異なる労働市場、労働力管理が形成され

[5] 本研究では、技術の定義について中川（2011）の定義を援用し、有形、無形を含む「製品や工程に関する、人ないしモノの動作の仕方あるいは動作原理」（中川, 2011, p.13）とする。

ていると考えられるが、こうした知的作業としての側面を備えるということのゆえに、これまでのライン生産方式の製造業を模した開発手法の導入はしばしば限界があると考えられる。さらに、そのような開発手法の導入により、ソフトウェアの開発が価値ではなく、量の取引になっていることが日本の IT 技術者の価値を下げているということも指摘されている（独立行政法人情報処理推進機構, 2015）。

このような日本のソフトウェア開発に対して、これまでも多くの研究がなされてきた。なかには本研究と同様、上流工程を中心として下流工程の知的作業としての側面を軽視するような開発手法について、その問題点を指摘する研究もある。しかしながら、そうした研究においてもどのような経緯によりそのような開発手法が採用され、なぜその開発手法を未だに採用しているのかを明らかにしているものは少ない。さらに、設計とプログラム作成工程を分離することの問題を指摘している研究は一部あるものの、その指摘はソフトウェア開発の複雑性についてのほか、イノベーションや創造性の欠如といった課題を提起するに留まることが多く、ハードウェアとは異なるソフトウェアの特性やソフトウェア開発の知識労働としての側面に注目してその開発プロセスの問題を明らかにした研究はほとんどない。

ソフトウェアの開発プロセスは、分離できない緩やかな分業で成り立っていると考えられ、従来さらに今日においても依然として主流となっているようなソフトウェア開発の上流工程と下流工程を分離し、そこに垂直的な分業を導入するような構造は、効率的に機能しているとは必ずしもいえない。特にソフトウェアの開発は、様々な顧客企業の多様なビジネスをシステム化するための、問題発見やその解決が必要とされる業務であり、そこには担当者の知的な、創造的な能力が必要とされることとなる（妹尾, 2001）。

こうした観点に立てば、ソフトウェアを作成することは「デザイン」であって、製造業のような「製造」ではないと考えられる。しかしながら、これを理解せず、単純な製造工程として外注するように、ソフトウェア開発の下流工程をアウトソーシングすることは、作業のみならずソフトウェア開発の競争優位に関わる知的作業を分離しまうようなこととなり、顧客ニーズの変化に柔軟に、かつ迅速に対応し、革新的なソフトウェアを生み出そうとする企業にとって致命的であると考えられる（Bean, 2005）。

以上のような観点から、本研究は日本のソフトウェア産業が、その開発作業を製造業とみなして設計とプログラム作成の工程を分離することで、プログラム作成工程が備える知識労働としての側面も分離してしまっているために、質の高い革新的なプログラム開発を実現する上での困難に直面しているということを明らかにする。本研究の視点に立てば、プログラム作成を定型的な組立作業のように捉え、下請け企業や海外へアウトソーシングしてしまうような従来の開発方法は、今日必要とされるソフトウェア開発には適用できないと考えられる。むしろ、ソフトウェアの開発は、そこで生じる問題の発見や解決といった試行錯誤を繰り返しながら顧客にとってより価値のあるものを作り出す創作活動でもあ

り、そこに革新の源泉や作業者にとっての仕事の魅力も存在すると考えられる。くわえてこのことは、ソフトウェア開発を納期や費用という点でいかに効率的なものとして管理することをもっぱら重視し、そうした管理のゆえにそこで開発する技術者が単純労働者としてしばしば軽視されがちであるという日本のソフトウェア産業に対して、今日におけるソフトウェア開発の付加価値の源泉がどこに存するのかという問いを改めて提起することになると考えられる。

② 研究対象

本研究は、ソフトウェア産業のうち、顧客企業の固有のビジネスモデル、固有のニーズに合わせてシステムを受託開発するカスタムソフトウェア産業[6]を対象としている。このカスタムソフトウェアは、社会のインフラや企業の基幹システム[7]、流通システム、EC サイトなどで導入されており、日本のソフトウェア業の売上高の 8 割近く[8]を占め、すでに完成したものを導入するパッケージソフトウェアと比較して未だ多くのソフトウェア企業で主要事業となっている[9]。

また、本研究では、カスタムソフトウェア開発のうち、数人から数十人程度の少人数で開発されるソフトウェアを主な対象としている。カスタムソフトウェア開発の中でも、金融機関などの企業の合併や統合のようなシステム開発や基幹システムの更新などは、数百人から数千人単位の技術者を必要とし、数年から数十年かけて行うような大規模な開発であることが多い[10]。

---

[6] このカスタムソフトウェア開発（custom software development）は、受注開発、受託開発、受託ソフトウェア開発、カスタムソフトウェア開発、オーダーメイド開発などさまざまな呼称が存在し、統一されていない。

[7] 販売管理、生産管理、会計など企業活動の中心となる業務。これらをシステム化したものは基幹系システムなどと呼ばれる。

[8] 経済産業省経済産業政策局調査統計部サービス統計室の「平成 25 年特定サービス産業実態調査」によると、日本のソフトウェア業の年間売上は約 10 兆 7,000 億円だが、そのうち約 9 兆 3,000 億円（約 86%）が受託ソフトウェア開発となっている。残りの約 1 兆 4,000 億（約 14%）のうち、パッケージソフトが約 9,000 億円（約 9%）、ゲームソフトが約 3,000 億円（約 3%）、コンピューター等基本ソフトが約 2,000 億円（約 2%）となっている。

[9] このカスタムソフトウェアの開発自体も、各コンポーネントを組み合わせてシステムを作り上げるシステム構築や、システムの設計からプログラミング、各種テストを含めシステム開発などに細分化される場合もある。ソフトウェアの分類については、第 2 章で詳しく述べるが、本研究では、システム構築やシステム開発を含めた企業より受託して行うソフトウェア開発をカスタムソフトウェアとして一括りにしている。

さらに、パッケージソフトウェアや組込みソフトウェアなど他のソフトウェア業とは区別して、産業としては「受託ソフトウェア産業」とし、そこで開発されるものは「カスタムソフトウェア」と記述している。

[10] 大規模開発の例として、世界最大のシステム開発とされる三菱東京 UFJ 銀行の勘定系システムの完全統合プロジェクト「Day2」がある。開発工数は 11 万人月と巨大であり、開発期間は 3 年以上かかり、2,500 億円が投じられ、ピーク時の技術者数は 6,000 人に上った

しかしながら、日本では景気の悪化や 2011 年の東日本大震災の影響などにより企業のIT 投資は減少し、コストを度外視するような開発は少なくなっている。そのため、このような大規模なシステム開発は減少し、その代わりにソフトウェアの機能を縮小し、幾つかの小さなサブシステム単位に分割することで、そのコストを抑えた開発へと主流は移りつつある（斎藤・後藤, 2016）。特に、2004 年ごろより、必要な時に必要なシステムやサービスだけを開発、利用することができるクラウド・コンピューティングの活用が徐々に浸透し始めており、その傾向は今後も拍車がかかると考えられる。このため、多くのソフトウェアの開発プロジェクトは、数か月から長くても 1 年以内に収まることが少なくない（独立行政法人情報処理推進機構編, 2014）。このような中小規模のソフトウェア開発のプロセスや手法は、上に指摘したような数年がかりの大規模な開発とは異なり、それゆえ、本研究の分析枠組みをそのまま適用することは困難である。そのため、上記のような大規模開発については本研究の対象から除外としている。

また、ソフトウェアには、その分類としてカスタムソフトウェア以外にパッケージソフトウェアやゲームソフトウェア、さらに組み込みソフトウェアが存在するが（総務省統計局, 2013）、本研究ではカスタムソフトウェアのみを対象とし、その他のソフトウェアは対象外としている。

こうした限定を設ける理由として、カスタムソフトウェアは、ユーザーの要望に合わせた個別向けにカスタマイズした開発を行うことを目的としているのに対し、パッケージソフトウェアは、IT ベンダー側主導のもと、顧客のニーズを探りながら開発する研究開発的な度合いが強く、一度開発してしまえば量産することが可能となるといったように、両者の間には大きな違いがあるためである。また、パッケージソフトウェアは、特定の企業に対してではなく、不特定多数の企業や利用者を相手に開発するため、顧客との仕様や設計についての擦り合わせを行わないといった違いも存在する。

ゲームソフトウェアについても、大量生産できるという点でパッケージソフトウェアと類似しており、受託開発型のカスタムソフトウェアとは異なる部分が多い。また、ゲームソフトウェアは一般ユーザー向けの娯楽としてのコンテンツであり、企業のビジネス向けのソフトウェアとはその用途が明らかに異なる。

一方、組み込みソフトウェアは、カスタムソフトウェア同様にビジネス向けのソフトウェアであるが、自動車や航空機、電化製品、医療用機器、人工衛星などのハードウェアとソフトウェアを組み合わせた製品であり、ハードウェアに制約される部分[11]が大きく、企

（大和田, 2009）。

一方、2012 年頃より開始したみずほ銀行の勘定システムの統合は、ピーク時の技術者数は 8,000 人、3,000 億円強が投じられたとされ（ITpro, 2016）、完成の時期も延期に延期を重ね、2018 年度以降に持ち越されている（日経新聞電子版, 2016/11/12 2:00）。

[11] 自動車や航空機、家電などに組み込まれるため、熱力学や運動エネルギー、万有引力など実世界の物理法則に支配される。

業システムのソフトウェアとは一線を画している[12]。

(3) 本研究の方法

本研究は、今日の日本の受託ソフトウェア開発における問題を明らかにするべく、日本の受託ソフトウェア開発の発展の歴史的経緯を踏まえ、ソフトウェア開発プロセスがいかなる手法を生み出し、どのような特徴を形成して今日に至っているのかを検討したうえで、実際の開発プロジェクトの事例分析を通じて、今日の受託ソフトウェア開発にいかなる問題が存在しているのかを明らかにする。

分析を通じて、本研究では、今日必要とされる質の高い、革新的なソフトウェア開発には、開発の工程や時には企業の境界を超え適用されうるような幅広い共通知識の形成が必要であること、開発においては工程間の緊密な連携に基づく試行錯誤と問題解決を繰り返す知的な創造的作業が要求されることを明らかにする。そのうえで本研究は、より革新的な、顧客満足を高めるようなソフトウェア開発を実現しようとするならば、従来わが国のソフトウェア産業においてしばしば考えられてきたようなソフトウェアの開発者は代替可能な労働力であると捉えるのではなく、ソフトウェア企業の競争力に直接的に関わる創造的作業者として捉える必要があることを指摘する。

以下､本研究の結論に至る論理や事例調査について､あらかじめ述べておくこととする。

① 日本のソフトウェア開発についての先行研究

本研究は、ソフトウェア開発の分業構造に着目しているが、ソフトウェアに関する研究自体は商用コンピューターが登場した 1950 年頃より行われている。さらに、1968 年に NATO がソフトウェア・エンジニアリング会議でソフトウェア危機を唱えた以降、ソフトウェア工学を中心とした多くの研究が登場し、ソフトウェアに関する名著や古典と呼ばれるもののうち、1970 年代に書かれたものも多い。

そのような中で、特に本研究に関する日本のソフトウェアのビジネスモデルや戦略について古くから進めてきた研究として Cusumano（1991, 2004）があげられる。

当然ながらわが国においても、日本のソフトウェアを対象とした研究は 1960 年代より、現在に至るまで多く行われている[13]。本研究に関する研究としては、今井・安藤・白井・辻・久保・玉置・浜田（1989）が、日本のソフトウェア産業を中心に情報産業の価値がハードウェアからソフトウェアにシフトすることを指摘しているほか、妹尾（2001）は Cusumano（1991）が指摘した日本のライン生産方式のような工場型のソフトウェア開発について触れるとともに、従来のソフトウェア開発の問題点を指摘し、リーダーの役割を強

[12] 小川（2011）によると、組み込みソフトウェア開発においても、カスタムソフトウェア開発と同様、増大する開発量や複雑、曖昧な要求仕様といった問題を抱えている。
[13] 例えば、IT 関連の学会の一つである情報処理学会は、1960 年に設立されている。

調している。また、峰滝（2004）は日本のソフトウェア開発のモジュール化とアウトソーシングの関係を分析しており、高橋（2010）は日本のソフトウェア産業の国際競争力がないとされる点についてその分析を行っている。

これ以外にも日本のソフトウェアを対象とした多くの研究が存在するが、その多くは IT バブル[14]を経て IT 産業としてソフトウェアに注目が集まり始めた 2000 年以降をその研究の対象としており（峰滝, 2005; 杉山, 2008; 田中, 2009; 高橋, 2010; 居駒, 2011 など）、それ以前の問題を論じた研究は少ない。早くから工場型のソフトウェア開発といった日本固有の特徴とその課題を対象とした研究を進めてきたのは、先にあげた今井他（1989）や Cusumano（1991, 2004）などが代表的であり、これら以外にはほとんどないと考えられる。

なかでも Cusumano（1991, 2004）は、1970 年代よりパッケージソフトウェアや組み込みソフトウェアも含めたソフトウェア全般について研究を進めてきており、それまであまり研究されてこなかった日本のソフトウェア産業についても、日本に 7 年近く滞在してその分析を行ってきた。特に、富士通や NEC、東芝といった大手メーカーが主導した日本の工場型のソフトウェア開発に関心を持ち、その手法の分析を進めていった。さらに、米国や欧州、そして日本のソフトウェア開発の特性などを比較し、その中でも日本のソフトウェア開発の優れている部分とその課題を明らかにしている。このように 1990 年代の日本のソフトウェア開発をここまで調査、分析した Cusumano の研究は、非常に価値のあるものと考えられる。

本研究では、この Cusumano（1991, 2004）や今井他（1989）、妹尾（2001）、峰滝（2004）、高橋（2010）の研究を中心に、これまでの日本のソフトウェア産業の先行研究として、特にかつて日本で中心的存在を果たした工場型のソフトウェア開発に関する研究を分析する。

② ハードウェアとソフトウェアの比較

本研究では、ソフトウェア開発の組織問題を分析するにあたり、本研究が対象とするカスタムソフトウェアがどのような分業構造で構成されているのかに関する整理を行う。

ソフトウェア産業は、その製品の特性、特にその根幹において技術・技能といったものが重要視されているという点で知識集約型産業としての性質を有すると考えられる。そのうえで、ソフトウェア開発を知識集約的な性質を持つものとして捉えた場合のその作業上の課題を明らかにするとともに、日本の基幹産業として成り立ってきたハードウェア、製造業や建築業などの開発手法との比較、整理を行い、ソフトウェアの開発がハードウェアの開発とどのように異なるのか、ハードウェアの開発手法を採用することがなぜしばしば

[14] インターネットバブル（Internet Bubble）やドットコムバブル（dot-com bubble）と呼ばれ、アメリカを中心に 1990 年代後半にインターネット関連の株価が高騰した。日本でも 1999 年から 2000 年末頃にかけて株価が上昇した。しかし、2000 年 4 月に NASDAQ で株価暴落が発生し、シリコンバレーの多くのベンチャーが倒産に追い込まれた（元橋, 2005）。

困難に直面するのかを説明する。

さらに、今日の受託ソフトウェア産業における開発プロセスの問題がいかなるものであるのか、なぜそうした問題が生じるに至ったのかを理解するためには、今日のソフトウェア産業がどのようにして現在のような分業構造を築き上げてきたのか、その歴史的経緯やビジネスモデルを明らかにする必要がある。

ソフトウェアの開発は、その黎明期において 1 企業だけですべてを開発するような産業として未発達の状態であった。どのような産業も、その発展の初期においてはイノベーターとなる企業が存在し、その財やサービスが普及するにつれて、専門に特化する企業活動が生まれてくる。ソフトウェアについては、1960 年代から 1980 年代前半における企業情報システムの主役であるメインフレーム[15]を中心として、絶対的強者として一時期世界の 70%以上のコンピューターを独占し（杉山, 2011）、ソフトウェアを含めた IT 産業全般をリードしてきた IBM がその革新的な企業であるといえる。そこで、現在のソフトウェア開発において見られるような企業間の分業と協業の構造が、どのように企業に戦略として取り入れられてきたのか、現在のソフトウェア産業を分析するとともに、イノベーターとして IT 産業をリードしてきた IBM を中心とする IT 産業の歴史を整理する。

③　製品設計思想

ソフトウェアを含め製品を作り上げていく上で、いかにして組織内外の役割を分担し、連携を図っていくのかについては、製品全体の構成を決める製品設計思想であるアーキテクチャーを考慮する必要がある。ソフトウェアは、コンポーネント[16]ごとにモジュール化したもので構成され、その開発において分業が行われている製品であると考えられる。こうしたソフトウェアの開発プロセスの問題を明らかにする上では、そもそもソフトウェアがいかなる製品設計思想に基づくものであるのかが明らかにされる必要があろう。そこで、本研究ではソフトウェアのような製品システムの構成についての分析として、製品・工程アーキテクチャーに関する諸研究（Baldwin and Clark, 2000; 藤本, 2001a, 2001b, 2003; 藤本・武石・青島編, 2001; 浜屋, 2004; 中川, 2011）を取り上げる。

一般に、アーキテクチャーは、製品アーキテクチャーや工程アーキテクチャーに分類することができ、組み合わせによるモジュラー型と、擦り合わせによるインテグラル型が存在する。

---

[15] 大型汎用機。商用に利用され始めた 1950 年代当初では、コンピューターは事務処理用や科学技術計算用などに分かれた互換性のない専門機であったが、1964 年に登場した IBM のコンピューター、SYSTEM/360 からさまざまな用途に利用できる汎用機として利用されるようになった。詳細については、第 4 章で述べる。

[16] 詳細については第 4 章で述べるが、コンポーネントとは、何らかの機能を持ったプログラムの部品や機械のパーツなどを指す。モジュールもコンポーネントとほぼ同義に近いが、規格化、標準化された交換可能な部品を指す（IT 用語辞典 e-Words）。

こうしたアーキテクチャーの視点から捉えたとき、ソフトウェアはしばしばモジュラー型の典型とみなされることが少なくない。しかしながら、本研究の視点に立てば、カスタムソフトウェアの開発は、独立した機能を組み合わせるモジュラー型の特徴を持つだけでなく、そういった機能や工程間の細かい調整が必要な擦り合わせ型の特徴をも備えるものと捉えられる。このような調整を必要とする擦り合わせ型のアーキテクチャーは、多くの部品が機能的かつ構造的に複雑な相互依存関係にある。ソフトウェアの開発においては、製品としてのソフトウェアについてどのようにその機能や工程をモジュール化するのか、さらに、そのモジュール化された機能や工程をどのように調整し、擦り合わせていくのかを決めなければならないのである。

④　ソフトウェア開発の事例研究

本研究では、定性的な調査方法を採用し、カスタムソフトウェアの開発現場での事例研究を通じて、特にその作業プロセスの編成や工程間分業の編成、そしてそれらの連携や調整がいかに行われたのかを分析する。そのうえで、いかなる理由からソフトウェアの開発が成功あるいは失敗するのか、さらにそこではどのような問題が生じ、どのように解決が図られているのか、ソフトウェア開発の成否に関わる諸要因を明らかにする。

事例研究に関する分析方法や検証の焦点など、その詳細については分析に際して改めて説明するが、あらかじめ対象について若干説明しておくこととする。

本研究はソフトウェア開発事例として、民間のソフトウェア企業の開発プロジェクトであり、それぞれ開発において異なる分業構造が採用された 3 つの事例を取り上げている。

ソフトウェア開発は、一企業の中でも多くのプロジェクトが稼働している。さらに、プロジェクトによってソフトウェアの開発方法も大きく異なり、技術者もさまざまなプロジェクトチームに所属している（東京大学社会科学研究所, 1989)。そのうえ、そのようなプロジェクトの進行は､企業や部署､プロジェクトのリーダーの考え方にも左右されるほか、開発の規模や難易度、メンバーの熟練度などにも強く影響される。こうした多様な性質を帯びる開発プロセスの実態やそのプロセスにおいていかなる問題が生じるのかといった点を明らかにするためには、プロジェクト・リーダーやメンバーに対する聞き取りやプロジェクト報告書などの書面による調査により、一つ一つの事例を精査していく必要があると考えられる。

しかしながらその一方で､このようなソフトウェア開発プロジェクトの多様な性質から､その実態を反映するような定量的なデータを取得することは難しく、さらにほとんどの企業は、ソフトウェア開発について守秘義務を持っており、また顧客企業との関係も考慮することから、顧客名や金額などの数値は元より、プロジェクトの成否やその諸要因に関する詳細なデータまで公開することはほとんどないため、インタビューによる調査もしばしば困難を伴う。

こうした調査上の問題から、本研究では、各事例の分析にあたって企業や個人が特定できないよう処理が施されるとともに、対象企業において調査、開示可能な範囲での分析とならざるを得なかった。分析される事例は、プロジェクト単位で選定された事例であり、それぞれの事例についてはプロジェクト概要やメンバーとその分業構造、開発の特徴などが分析されるとともに、分析の焦点として開発プロジェクトの成否を判断する基準となる品質（Quality）、コスト（Cost）、納期（Delivery）の 3 点に加え、人的・技術的サポート（Service）を中心に、プロジェクトとしてどのような結果、成果が導き出されたのかを分析し、そうした結果、成果をもたらすに至った要因を明らかにする。

⑤ 知識マネジメント

本研究は、ソフトウェア開発が単なる加工組み立てのような作業ではなく、作業者やプロジェクトメンバーが備える専門的・技術的知識や顧客や関連作業に対する幅広い知識に依存すると共に、プロジェクト進行における試行錯誤や創造的な問題解決活動に依存する、いわゆる「知識依存的」な活動であると捉えるものである。こうした観点から、本研究では、事例研究の対象であるソフトウェア開発の各プロジェクトについて、そこで明らかにされた工程間の分業や連携の成否について、なぜそうした有効性が発揮できたのか、または発揮できなかったのかという点について、「知識マネジメント」の視点から考察する。ソフトウェア開発の作業を進めていく過程において、設計工程には設計の変更や検証といった問題解決サイクルを何度も繰り返す必要性が存在し、そうした設計工程に含まれる多くの問題を下流工程で発見し解決していくことが求められるが、ここにおいては各工程の担当者や作業者が保有する専門的、技術的な知識や業務上のノウハウといった様々な知識の活用が不可欠であり、そうした知識の活用如何が開発プロジェクトの成否を左右しうると考えられるのである。

こうした観点より、守島（2001a, 2001b, 2002, 2011）や妹尾（2009）、三輪（2014）などの知識マネジメントの視点からの問題発見と問題解決に関する研究を整理する。さらに、Ferguson（1992）や藤本（2001a; 2001b, 2007）などの製造業に基づいた製品設計とその問題解決に関する議論を手掛かりに、ソフトウェア開発における問題発見や解決といった知的労働についての考察を行う。

(4) 本研究の構成と用語の定義

① 章の構成

本研究の構成は、この第 1 章を含め、10 章で構成されている（**図 1-1**）。

第 1 章では、導入部として、本研究の背景や目的、アプローチ方法、論文の構成、そして研究の範囲について述べる。そのほか、本研究で使用する用語の定義や表記方法の統一についても説明する

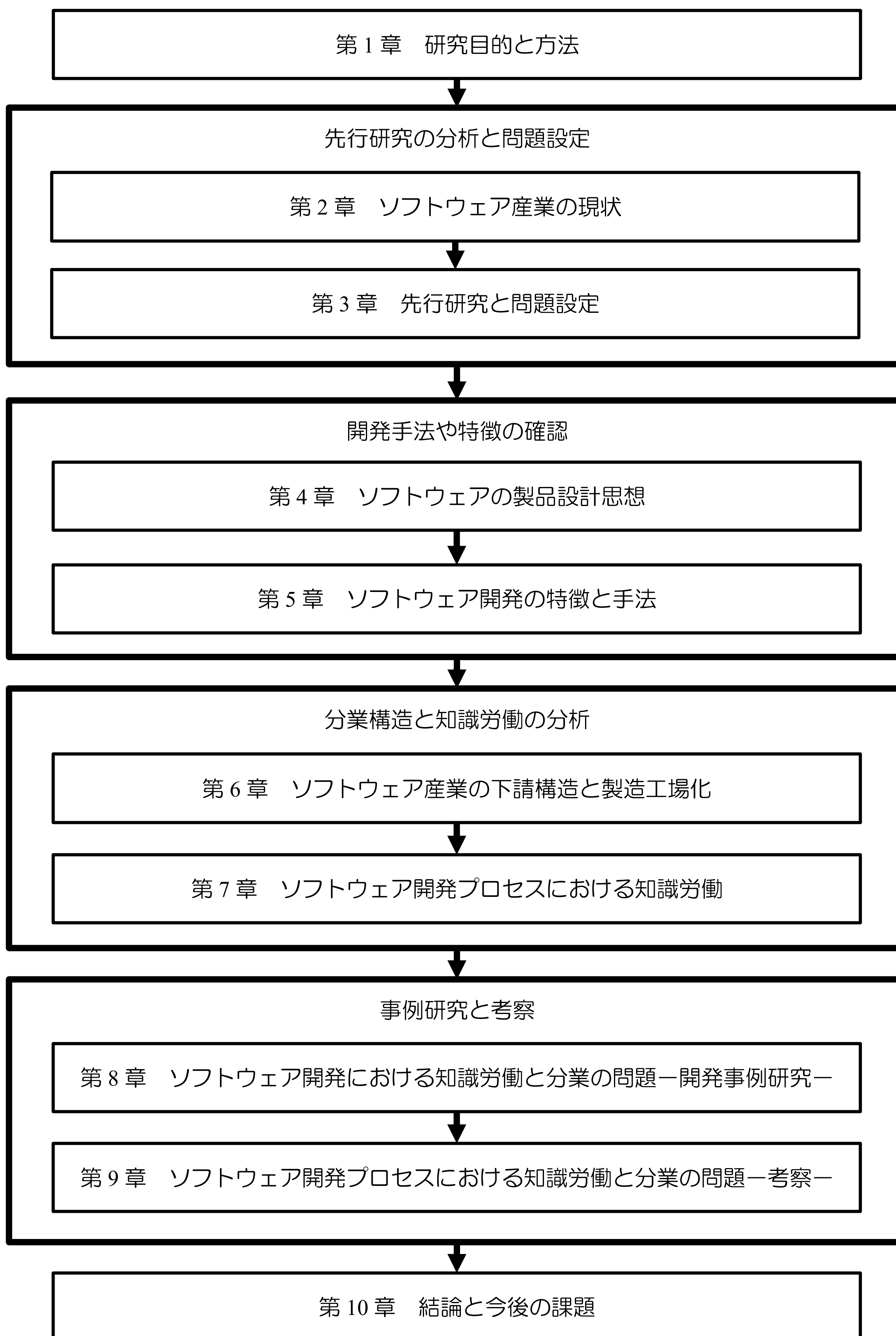


図 1-1 本研究の構成

第 2 章と第 3 章では、先行研究の検討と問題設定について述べる。

第 2 章では、先行研究の検討に入る前提、予備的な考察として日本のソフトウェア産業の現状やソフトウェアの分類について確認する。また、日本のソフトウェア産業の海外へのアウトソーシングについて述べる。さらに、本研究が対象とするカスタムソフトウェアについて、その特徴や日本の多くの企業で導入されている理由を確認するとともに、パッケージソフトウェアとの違いを確認する。

第 3 章では、先行研究の検討とそれを踏まえた本研究の問題設定を行う。本研究の視点によれば、ソフトウェア開発、特にその下流工程は従来の理解とは異なり、担当者の保有する様々な知識に依存するところの大きい知識労働としての側面を備えるものと考えられる。しかしながら、日本のソフトウェア産業において、これまでこうした捉え方がなされてきたかというと、必ずしもそうではない。本章では、日本のソフトウェア産業において、ソフトウェア開発における開発作業に対し、これまでどのような評価や分析が行われてきたのか、そこでのソフトウェア開発がどのような業務、労働であると捉えられてきたのか、先行研究の検討を通じて確認するとともに、そこから導き出される問題設定について述べる。

第 4 章と第 5 章では、ソフトウェア開発の開発手法や特徴について述べる。

第 4 章では、ソフトウェア開発を製品設計思想としてのアーキテクチャーの視点から検討する。ソフトウェア開発におけるモジュール化の重要性を確認するとともに、モジュラー型アーキテクチャーとインテグラル型アーキテクチャーの議論に基づいて、そういった機能や工程の分割がソフトウェア開発にどのように影響するのかを確認する。

第 5 章では、これまでのソフトウェア開発がどのように行われてきたのか、また今日どのように行われているのか、Waterfall Model と Agile[17]と呼ばれる手法を中心に取り上げ、検討する。特に、1970 年ごろから現代に至るまで、日本のソフトウェア開発で主流として機能してきた Waterfall Model が、時代の流れとしてどのように Agile に取って代わられつつあるのか、その開発プロセスの特徴と限界がいかなるところに見られるのかについて検討する。

第 6 章と第 7 章では、ソフトウェア開発の分業構造と知識労働を検討する。

第 6 章では、日本のソフトウェア産業に焦点をあて、同産業に特徴的に見られる分業構造を踏まえ、かつて日本のソフトウェア開発で中心的存在を果たしたソフトウェア・ファクトリーを取り上げ、その貢献と限界について述べる。

第 7 章では、前章で述べた日本のソフトウェア・ファクトリーを生み出したソフトウェア産業が、どのように発展し、いかに現在のビジネスモデルを作り上げてきたのかを確認する。その上で、現状のソフトウェア･ファクトリー構造に基づいてプログラム開発を行う

[17] アジャイル。agile software development。軽量型開発とも呼ばれる開発手法。アジャイルソフトウェア開発の詳細については本文中に後述する。

ことが、いかなる問題をもたらしうるのか、こうした問題の克服において従来の分業構造を再編する必要があるとすればそれはどのような再編であるのか、こうした論点に対する分析的視点および枠組みを検討する。

第 8 章と第 9 章では、ソフトウェア開発の事例研究の分析とその考察を行う。

第 8 章では、前章の議論を踏まえ、ソフトウェア開発の事例分析を行う。本事例分析では、それぞれのソフトウェア開発プロジェクトについて、その開発プロセスにおける分業の編成、さらに開発の現場で行われた作業プロセス間の連携や開発担当者の知的活動とプロジェクトの成否との関係に関する分析を中心として、ソフトウェア開発プロジェクトの成否を左右する諸要因を明らかにする。

第 9 章では、前章の事例分析から得られた結果の考察を行う。そのうえで、ソフトウェア開発プロジェクトの成否に影響を及ぼす諸要因が、なぜ、またどのようにそうした開発プロジェクトの有効性の発揮の有無と関係しているのか、ソフトウェア開発プロセスで見られた試行錯誤を繰り返す「創造的な問題解決過程」と、それに携わる技術者の「知識マネジメント」の視点から改めて検討する。本章における考察を通じて、有効なソフトウェア開発プロジェクトにおいては、開発に従事する技術者が単なる作業者としてよりはむしろ創造的問題解決を担う知識労働者として機能しているということが示される。

第 10 章では、これまでの各章の内容を要約すると共に、本研究の結論、本研究の学術的貢献、実務的貢献が提示される。そのうえで、本研究の今後の課題が提起される。

② 用語の定義について

本研究における用語の定義や略語や類似した用語などの表記方法について、あらかじめまとめておくこととする。

本研究が対象とする受託ソフトウェア産業に従事する技術者は、システムエンジニア[18]やプログラマー[19]と呼ばれることが多く、さらに、さまざまなソフトウェアの開発会社が存在する。しかし、システムエンジニアとプログラマーの職務上の定義は非常に曖昧[20]であり、本研究では、そのような実態に合わせ、システムエンジニアやプログラマーを分けず、ソフトウェア技術者として統一して捉える。

さらに、ソフトウェアの開発会社についてもソフトウェアハウス、ソフトハウス、ソフ

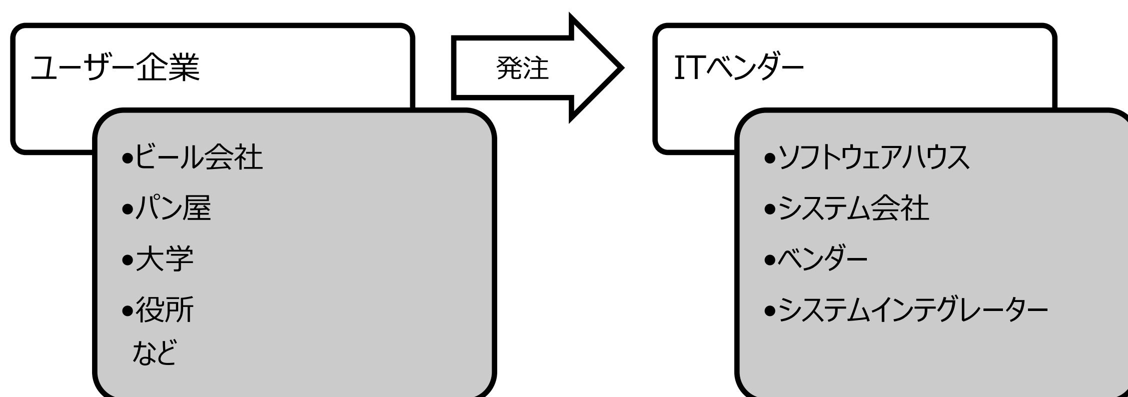
[18] SE, System Engineer。主に情報システムの開発から運用、プロジェクトの管理などに従事するコンピューター技術者。

[19] PG, Programmer。主にソフトウェアなどのプログラムを作成する技術者。

[20] システムエンジニア（System Engineer）は和製英語であり、日本のシステムエンジニアに相当するものは海外では developer や engineer などが使われる。企業や団体によってシステムエンジニアの職務範囲は大きく異なっており、明確な定義がされていない。概ね要件定義や設計などの上流工程を担当するソフトウェア業務従事者をシステムエンジニアと呼び、開発作業に従事する者をプログラマーと呼ぶことが多いが、その境界も曖昧である。

トウェア企業、ベンダーなど多数の呼称が存在するが、受託ソフトウェア産業は特にシステムインテグレーター[21]を指すことが多い。しかしながら、本研究ではこれらをまとめて、「IT ベンダー」と記述する。

一方、IT ベンダーの顧客、つまりソフトウェアやそのシステムの発注元は、銀行や保険会社、証券会社などの金融業から、小売業、ビール会社やパン屋などの製造業、建設業、大学などの教育機関、そして官公庁・役所などの公的機関まで存在するが、本研究ではこれらをまとめて「ユーザー企業」と記述する（**図** 1-2)。



出所：筆者作成

図 1-2 ユーザー企業と IT ベンダー（受発注関係）

本研究が対象とする受託ソフトウェア開発はシステム開発と呼ばれる場合がある。ソフトウェアは、厳密にはパーソナルコンピューター上で稼働するアプリケーションソフトウェア[22]を指すことが多い。例えば、Word や Excel、Web ブラウザー、ゲーム、会計ソフトなどが身近なものであろう。

一方、システムは、そのようなアプリケーションソフトウェア自体を指すこともあれば、複数のソフトウェアで構成されたものやそれに付随する環境を指す場合もあり、システム開発はソフトウェア開発を含んだ広いものとして扱われることが多い。例えば、企業の会計システムや販売管理システム、在庫管理システムなど、複数のソフトウェアを組み合わせたものや、ハードウェアやネットワークなども含んだ環境全体などがそれにあたる。

ただし、開発の現場では、このようなソフトウェア開発とシステム開発とを明確に区別していないことも多い。本研究では、システム開発についてもソフトウェアの作成を行うという点から、ソフトウェア開発と同義のものと捉えている。

このように、ソフトウェアは多くの工程に分かれて開発が行われているが、本研究では

[21] SI, SIer, System Integrator。システムインテグレーション（System Integration）と呼ばれる、システム構築と開発の全体を通して一括してサービスを提供する企業、事業所。
[22] application software。応用ソフトウェアやアプリケーションプログラムなどとも呼ばれる。

このソフトウェア開発工程のうち、特に設計と、一般的に製造と呼ばれているプログラムの作成との分業関係に着目している。この「プログラム作成」工程とは、プログラム作成に関わるコーディング（プログラミング）作業や、仕様・設計の確認、レビュー、デバッグなどの作業全般を指す。Bean[23]（2005）は、このプログラムを作り込む工程について、「デザイン」と呼び、製造業における「製造」とは明確に区別している。こうした Bean の区分に基づけば、本研究でも、「プログラム作成」を単なる「製造」作業とみなすことは難しい。本研究もこうした Bean の視点に従い、プログラムを記述しソフトウェアを作成する工程について、これを製造業における「製造工程」と明確に区別するために、「プログラム作成工程」と表記している。

本研究は、Bean が指摘するようにソフトウェア開発を単なる「製造」作業にとどまらない活動として捉え、これをある種の「知識労働」として捉える視点に立つものである。その上で、本研究は、革新的、または創造的なソフトウェアの開発を行うためには、試行錯誤といったものが重要であるという仮説的認識に立つ。

ここにいう革新的なソフトウェアとは、今までにない問題や新しい領域に取り組んでいくようなソフトウェアであり、例えば、都市の安全・環境シミュレーションシステムや創薬・バイオ新基盤技術開発へ向けたタンパク質反応全電子シミュレーションなどであり、これまでの企業のビジネスを支えてきた基幹システムや、コスト削減のために単に IT 化しようとするソフトウェアとは異なるものである。

本研究ではソフトウェアを中心とした IT 産業を対象としているが、IT を専門とした技術者にとってもコンピューター用語は難解なものが多い。ソフトウェア産業も業界特有の言葉づかいが多く、例えば「AWS」[24]や「ST」[25]などの略称、略語が多く使われることもあり、なかには非常に分かりにくいものもある。このような略称、略語は混乱の元となるため、本研究では、IT のような一般に浸透しているような用語を除き、システムインテグレーターを指す SIer などの略称は原則として使用しないこととする。また、これ以外にも情報通信産業特有の技術用語が出てくるが、それらについては、脚注にて随時説明していくこととする。

そのほか、「コンピューター」という単語一つでも、一般社会で使用される「コンピューター」と、工学・技術の様式に基づいた「コンピュータ」のような表現の違いも存在する（表 1-1）。本研究では、より自然な読み方である一般の社会生活における表記として、「コ

[23] アメリカのソフトウェア会社の社長であり、ソフトウェア開発者。

[24] Amazon Web Services の略。Amazon.com によるクラウドコンピューティングを利用した Web サービス。

[25] System Test（システムテスト）の略。ソフトウェア開発の最終段階で行うテストで、ソフトウェアが要求仕様を満たしているかの確認や、本番と同じ条件で正常に稼働するかなどの確認を行う。

ンピューター」といった表記で統一する[26]。

表 1-1 本研究で使用される主な用語の表記

外来語（英語綴り）	内閣告示	工学・技術系表記
Computer	コンピューター	コンピュータ
Server	サーバー	サーバ
Architecture	アーキテクチャー	アーキテクチャ
Supplier	サプライヤー	サプライヤ
Parameter	パラメーター	パラメータ
maker[27]	メーカー	メーカ
Vendor	ベンダー	ベンダ
Integrator	インテグレーター	インテグレータ
Programmer	プログラマー	プログラマ

出所：筆者作成

[26] 「コンピューター」等外来語は、工学・技術系の「JIS Z 8301 規格票の様式及び作成方法」(JISC 日本工業標準調査会）の様式に基づいた「コンピュータ」という表記の仕方と、内閣告示第二号（1991 年 6 月 28 日）（文部科学省, 1991）による、一般の社会生活における外来語の正式な表記としての「コンピューター」が存在する。ただし、JISZ8301 においても、長音符号を付けるか、付けないかについて厳格に一定にすることは困難であるとしており、長音符号を用いても略しても誤りでないとしている。
　本研究では、原則として、引用部分を除いて、一般社会に馴染みがあると考えられる内閣告示の「コンピューター」表記で統一する。

[27] 日本では、製造業や製造企業のことをメーカーと呼ぶことがあるが、maker は作り手や製造者などを意味することが多く、製造業（製造企業）という場合は、manufacture（manufacturing industry）を通常使用する。

第2章 ソフトウェア産業の現状

本章では、次章の先行研究の検討と問題設定を行う前提として、予備的な考察として日本のソフトウェア産業の現状について確認する。特に、本研究が対象とするカスタムソフトウェアの特徴や同じソフトウェアであるパッケージソフトウェアとの違いなどを確認する。

(1) 日本のソフトウェア産業の現状

① ソフトウェア産業の推移

受託ソフトウェアを中心とした日本のソフトウェア産業は、ハードウェアであるコンピューターの歴史とともに、その周辺サービスや、周辺機器を提供する産業として発展してきた。IT の重要性が認識されるとともに、ソフトウェア産業は、経済活動や社会生活を変化させ、企業の経営のあり方を変えてきた。

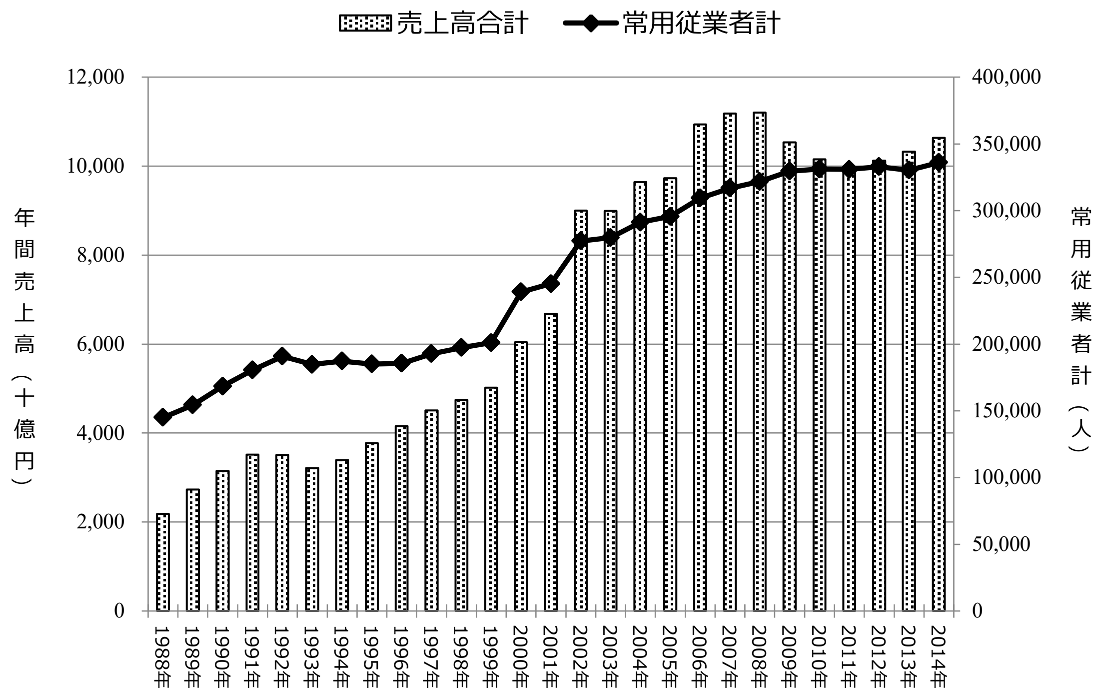
日本のソフトウェア産業は、2000 年前後の IT バブルやそれに伴う IT 化の導入が活発であった頃に、従業者や売上を急速に伸ばしてきた。しかし、リーマンショックが発生した 2008 年以降は成長も鈍化し、2011 年の東日本大震災以降は横ばいになっている（図 2-1)。

このソフトウェア産業は、大手企業と中小零細企業に二極化した寡占が進んでおり（湯野川, 2010）、従業員規模が 500 人以上の事業所は 1%程度にすぎず、そのほとんどが中小零細企業で構成されている（図 2-2)。また、年間売上高でも 100 億円を超える事業所が 1%にすぎないのに対し、半数以上の事業所が 1 億円から 10 億円の売上高に留まっており、大企業を中心とした下請け構造となっている。

『中小企業白書 2011』（中小企業庁, 2011）によると、2004～2006 年の経済センサス-基礎調査において多くの業種で開業率が廃業率を下回る中、ソフトウェア業を含む情報通信業は、医療・福祉とともに、開業率が廃業率を大きく上回る結果が出ている。例えば小売業の開業率 4.8%、廃業率 9.7%や、飲食店・宿泊業の開業率 7.0%、廃業率 8.7%に比べ、情報通信業は開業率 15.6%、廃業率 11.5%と開業率が上回っている。

2006～2009 年の経済センサス-基礎調査では、2004～2006 年の調査と比較して、全体的に開業率より廃業率が高くなっている[28]。その中でも情報通信業は、開業率がトップの 3.9%となっており、廃業率に関しても金融・保険業の 9.5%に次ぐ 9.3%と、2004～2006 年と同様に他業種と比較して高い割合となっている。

[28] 2006～2009 年の基礎調査は、商業・法人登記などの行政記録を活用して、事業所・企業の捕捉範囲を拡大しており、また、本社などの事業主が、支所などの情報も一括して報告する「本社等一括調査」を導入しているため、2006 年以前の事業所・企業統計調査と単純に比較することはできない。



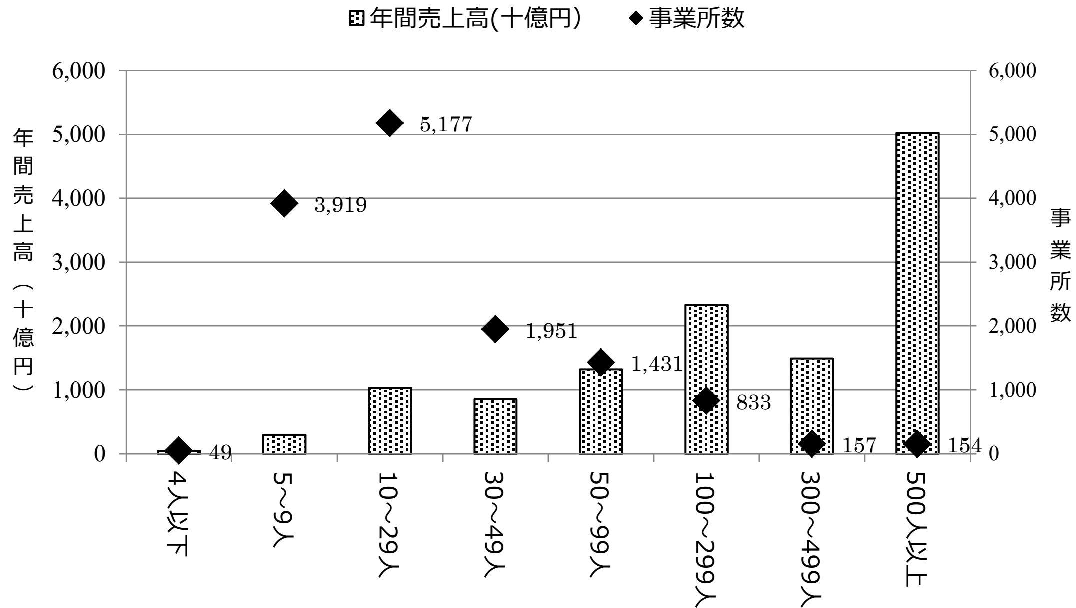
出所：経済産業省「特定サービス産業動態統計調査」、総務省統計局「経済センサス」をもとに筆者作成

図 2-1 ソフトウェア業の売上高・従業者数の推移

さらに、『2017 年版中小企業白書』（中小企業庁, 2018）によると、2015 年の中小企業の業種別開廃業率において、情報通信業は、宿泊業・飲食サービスや建設業、生活関連サービス・娯楽業に次いで開業率が高く、一方で宿泊業・飲食サービスや建設業に次いで廃業率が高くなっている（図 2-3）。

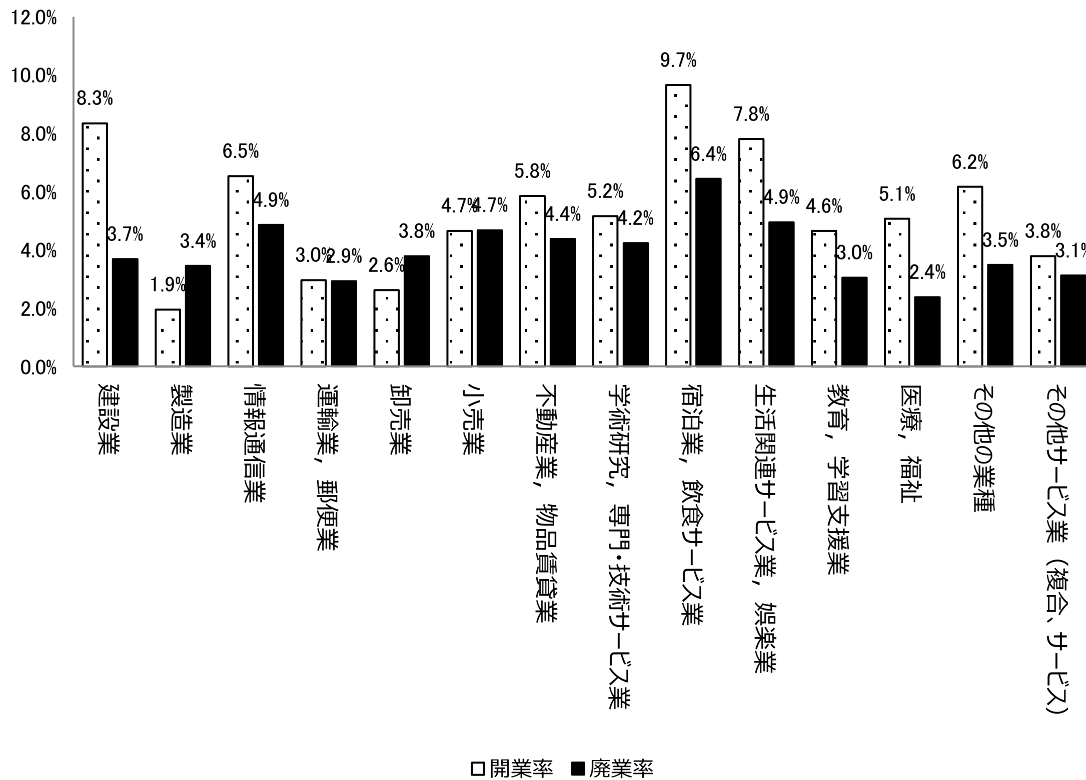
これらデータより、IT と呼ばれる情報通信業は比較的新しい産業のために新規参入がしやすい分野であるが、それゆえに市場競争も激しく、撤退していく企業が多いことが予想できる。

このようにソフトウェア業を含む情報通信業は、非常に高い開業率と廃業率を抱えた競争の激しい産業となっているが、『情報サービス産業白書 2010』（一般社団法人情報サービス産業協会編, 2010）によると、日本の情報サービス産業は過去に 2 度の大きな問題に直面しているという。1 度目は 1990 年代のバブル経済の崩壊による不況であり、2 度目は 2001 年頃の IT バブル後の不況である。この IT バブル後の不況時には、事業のリストラや雇用調整、人件費を中心としたコスト圧縮、アウトソーシングや派遣社員への切り替えが進み、その結果、重層的な下請け構造を助長する要因の一つとなっている。



出所：経済産業省「平成 26 年特定サービス産業実態調査（確報）」をもとに筆者作成

図 2-2 ソフトウェア業の従業員規模別の年間売上高と事業所数（平成 26 年）

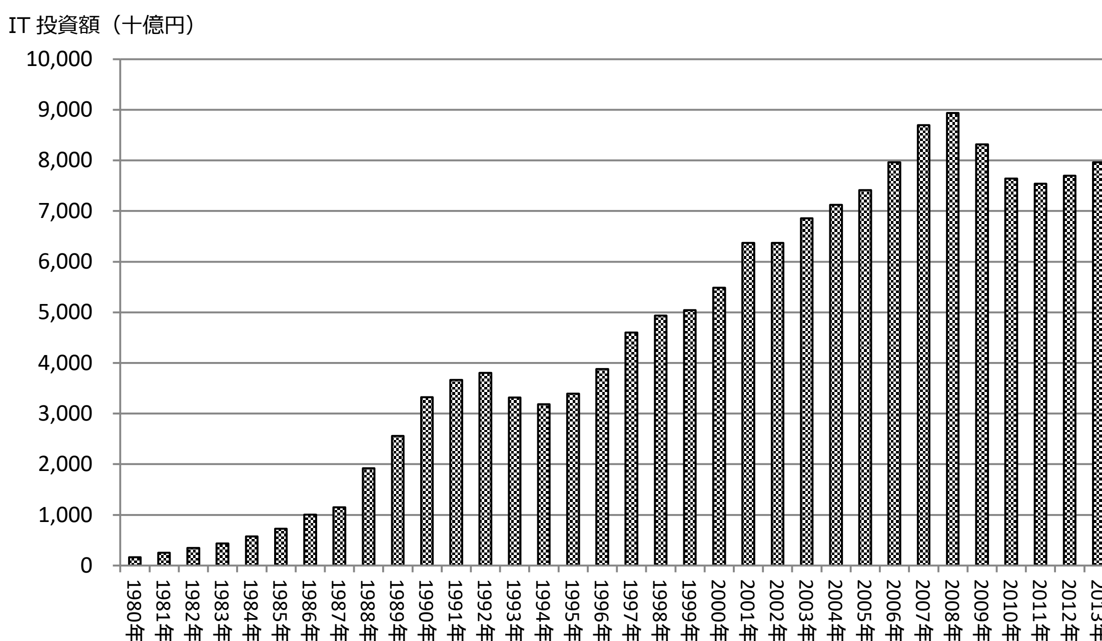


出所：中小企業庁（2018）「2017 年版中小企業白書」をもとに筆者作成

図 2-3 業種別開廃業率（平成 26 年）

② IT 投資とソフトウェアの開発規模

2001 年の IT バブル崩壊、2008 年頃に発生したサブプライム問題やリーマンショック、2011 年の東日本大震災などにより、ユーザーの IT 投資は減少する一方であり、ソフトウェア開発の規模も小さくなってきている。特に、2008 年頃を境に IT 投資が減少しており、リーマンショックによる日本経済への影響が、IT 投資にも影響していると考えられる（斎藤・後藤, 2016）（図 2-4）。



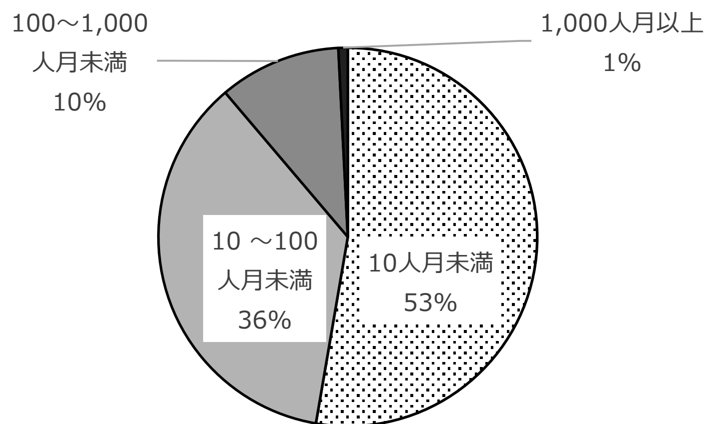
出所：総務省 情報通信統計データベース「ICT の経済分析に関する調査」をもとに筆者作成

図 2-4 企業の IT 投資の推移（ソフトウェア業）

このような IT 投資の減少に伴い、ソフトウェア開発の規模も 1990 年代以前に多かった何年もかかるような大規模な開発は少なくなり、小規模な開発が多くなっている。例えば、独立行政法人情報処理推進機構編（2014）が調査したソフトウェア開発プロジェクトの作業工数ごとの比率によると、10 人月未満が 52.7%、10～100 人月未満が 36.1%、100～1,000 人月未満が 10.4%となっている（図 2-5）。

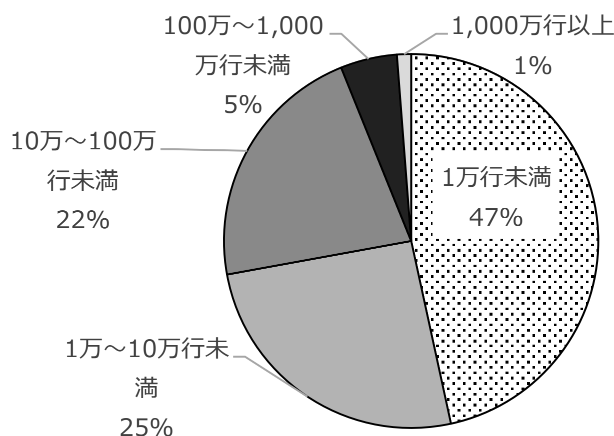
また、新規開発においてもプログラムのソースコードの開発量は、1,000～1 万行が 4 割弱となっており、開発対象となるソフトウェアが小規模化していることが確認できる（図 2-6）（図 2-7）[29]。

[29] ただし、プログラムの開発量が少ないことが単純に開発規模の小さいことを表すわけではない。ソフトウェアはプログラムによって動作するが、プログラムには多種多様な言語があり、記述方法も異なる。そのため、プログラム言語によってソースコードの量、つま



出所 : 独立行政法人情報処理推進機構編 (2014)『ソフトウェア開発データ白書 2014-2015』より筆者作成

図 2-5 プロジェクト工数比率

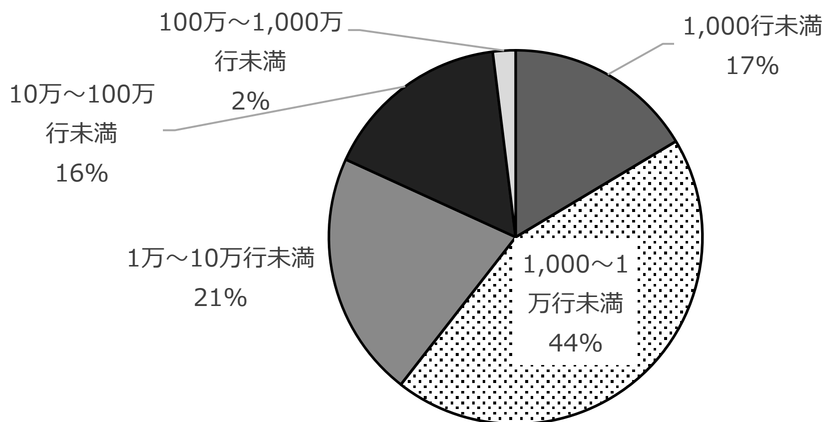


出所 : 独立行政法人情報処理推進機構編 (2014)『ソフトウェア開発データ白書 2014-2015』より筆者作成

図 2-6 全ソフトウェア行数比率

りプログラムの行数は大きく異なっており、後の時代に開発されたプログラム言語ほど、これまでの技術の積み重ねや進歩により、共通に利用できる簡易なモジュール機能を備えていたり、単純な記述でより高度な処理が可能となっていたりすることが多く、結果的に必要とされるソースコードの量も少なくなる。

また、あるプログラム言語でその最終的な計算結果が同じであっても、ソフトウェア技術者の技量によって、そこに至るまでのプログラムの記述の仕方は大きく異なり、より高度なロジックや記述方法になるほどソースコードの量が少なくなる場合がある。さらに、ソースコードの量が多ければ多いほど、そのプログラムに含まれるバグなどの欠陥が増えることにも繋がる。



出所：独立行政法人情報処理推進機構編（2014）『ソフトウェア開発データ白書 2014-2015』より筆者作成

図 2-7 新規ソフトウェア開発行数比率

2000 年頃の IT バブルの時代とは異なり、ソフトウェア開発自体も企業のコスト削減の対象となり、要望の挙がったソフトウェアのすべての機能を開発するのではなく、必要な機能だけを選別して開発するようになっていった。さらに、IT バブルの崩壊やリーマンショックを経た 2010 年代に入ると、クラウドサービスが利用されるようになり、必要な機能を必要な分だけサービスとして利用できるようになった。一方で、ネットワークを利用し、企業の業務プロセスごとに開発されたソフトウェア同士を連携させた複雑な企業間の情報システムの構築も求められるようになっていった。

このように、ソフトウェア開発は多額の投資と労力を必要とする数年間かかるような大規模開発は少なくなっており、システムや機能単位ごとに分離した数か月間の小規模開発が多数を占め、その上で企業のビジネスに直結したより高度で複雑なソフトウェアが求められるようになってきている。

③　ソフトウェアの分類

次にソフトウェアの種類を整理する。情報通信関連の産業を一括りに IT 産業と呼称することが多いが、製造業のようにその内容は多岐に渡っている。平成 25 年 10 月に改定された「日本標準産業分類」（総務省統計局）によると、IT[30]、または ICT[31]と呼ばれている「情

[30] Information Technology。情報技術。コンピューター処理などの情報処理やデータ通信などの通信処理に関する技術の総称。政府の IT 基本戦略、及び e-Japan 戦略により、2000 年頃からこの呼称が普及し始めた（首相官邸, 2000, 2001）。

[31] Information and Communication Technology。情報通信技術。IT の別称であり、内容もほぼ同義である。IT にコミュニケーションが加えられており、IT よりも情報や知識の共有に重点が置かれている。2005 年より、e-Japan 戦略の後身である総務省主導の u-Japan 政策以降、

報通信業」は、「通信業」、「放送業」、「情報サービス業」、「インターネット付随サービス業」、「映像・音声・文字情報制作業」の 5 つに分けられている（表 2-1）。

本研究が対象とするカスタムソフトウェアは、その中の「情報サービス業」の「ソフトウェア業」に属し、その中の「受託開発ソフトウェア業」に該当する。

「ソフトウェア業」は、「受託開発ソフトウェア業」のほかに、「組み込みソフトウェア業」、「パッケージソフトウェア業」、「ゲームソフトウェア業」の 4 つに分類されている（表 2-2）。

『情報サービス産業白書 2015』（一般社団法人情報サービス産業協会編, 2015）によると、ソフトウェア業の各業態は次のように定義されている。

a) 受託開発ソフトウェア業

「受託開発ソフトウェア業」は、顧客の委託により、電子計算機のプログラム作成およびその作成に関して、調査・分析・助言などを行う事業所を指す。

この中に、本研究が対象とするカスタムソフトウェアの開発をはじめ、プログラム作成、情報システム開発、ソフトウェア作成コンサルタントが含まれる。製造業の生産管理システムや販売管理システム、会計システム、銀行企業の基幹システムなどがこれにあたる。

この受託開発は、ユーザー企業からカスタムソフトウェアとして開発の発注依頼を受けるビジネスモデルであり、ユーザー企業の固有のビジネスモデルに合わせて開発するため、その開発内容について受託側の IT ベンダーの恣意性を入れる余地は基本的にない。発注元のユーザー企業が定義した仕様通りに開発する必要がある。

b) 組み込みソフトウェア業

「組み込みソフトウェア業」は、情報通信機械器具、輸送用機械器具、家庭用電気製品などに組み込まれるソフトウェアを作成する事業所を指す。具体的には、スマートフォンなどの電化製品に組み込まれているシステムや、心電計などの医療器具、エンジンのコントロールシステムやカーナビゲーションなどの自動車関連システムがあげられる。特にハードウェアとソフトウェアを組み合わせた製品であるため、熱力学や運動エネルギー、万有引力など実世界の物理法則による制約が大きいことが特徴である。

c) パッケージソフトウェア業

「パッケージソフトウェア業」は、電子計算機などのパッケージソフトウェアの作成やその作成に関しての調査・分析・助言などを行う事業所を指す。

IT に代わりこちらの呼称を使うことも多い（総務省）。

表 2-1 日本標準産業分類（平成 25 年 10 月改定）

大分類	中分類	小分類	細分類
A 農業，林業			
B 漁業			
C 鉱業，採石業，砂利採取業			
D 建設業			
E 製造業			
F 電気・ガス・熱供給・水道業			
G 情報通信業			
	├ 37 通信業		
	├ 38 放送業		
	├ 39 情報サービス業		
	│	├ 390 管理，補助的経済活動を行う事業所（39 情報サービス業）	
	│	**├ 391 ソフトウェア業**	
	│	│	**├ 3911 受託開発ソフトウェア業**
	│	│	├ 3912 組込みソフトウェア業
	│	│	├ 3913 パッケージソフトウェア業
	│	│	└ 3914 ゲームソフトウェア業
	│	└ 392 情報処理・提供サービス業	
	├ 40 インターネット附随サービス業		
	└ 41 映像・音声・文字情報制作業		
H 運輸業，郵便業			
I 卸売業，小売業			
J 金融業，保険業			
K 不動産業，物品賃貸業			
L 学術研究，専門・技術サービス業			
M 宿泊業，飲食サービス業			
N 生活関連サービス業，娯楽業			
O 教育，学習支援業			
P 医療，福祉			
Q 複合サービス事業			
R サービス業（他に分類されないもの）			
S 公務（他に分類されるものを除く）			
T 分類不能の産業			

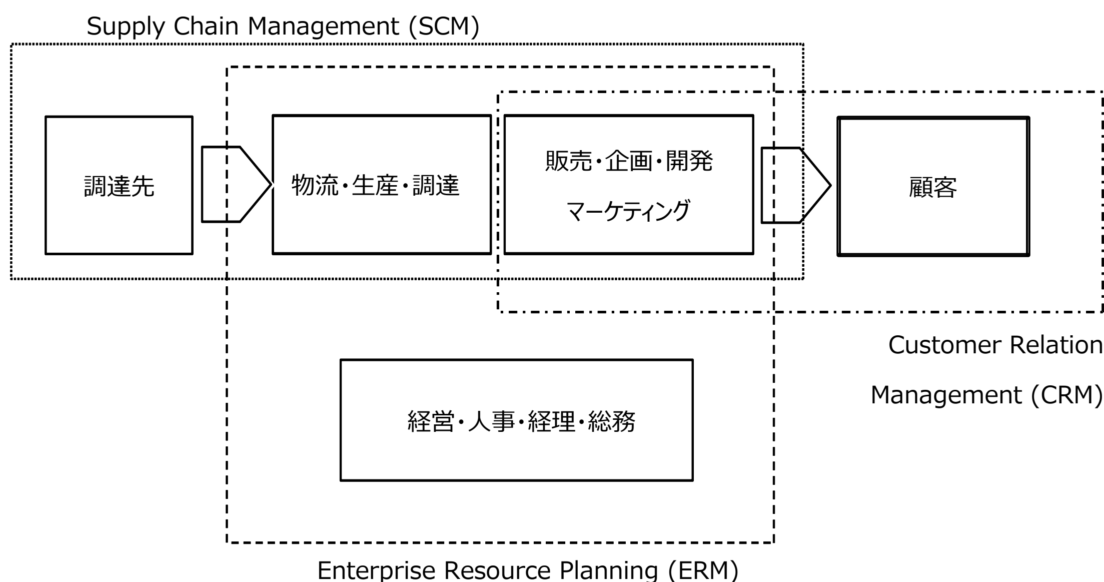
出所：総務省統計局（2013）「日本標準産業分類（平成 25 年 10 月改定）」をもとに筆者作成

表 2-2 ソフトウェア業の種類

細分類	業務内容	ソフトウェアの例	代表的な企業[32]
3911：受託開発ソフトウェア業	顧客の委託により、電子計算機のプログラム作成およびその作成に関して、調査・分析・助言などを行う。	製造業などの生産管理・販売管理システム・会計システム、銀行・保険の金融業の基幹システム等。インターネット上の EC サイト・システム、チケット販売サイト・システムなど。	IBM, 富士通, NEC, 日立, NTT データ, 日本ユニシスなど。
3912：組み込みソフトウェア業	情報通信機械器具、輸送用機械器具、家庭用電気製品に組み込まれるソフトウェアの作成などを行う。	スマートフォン・iPhone のシステム、心電計、カーナビゲーションシステムなど。	キヤノンソフト、富士ソフト、日立ソフト、東芝情報システムなど。
3913；パッケージソフトウェア業	電子計算機等のパッケージソフトの作成やその作成に関しての調査・分析・助言などを行う。	量販店の店頭にある会計ソフト、販売管理・在庫管理パッケージソフ、ERP パッケージソフトなど。	Oracle、SAP、日本インフォア、弥生株式会社など。
3914；ゲームソフトウェア業	家庭用テレビゲーム機やパソコンゲーム、携帯電話のゲームやそのソフトの作成、及びその作成に関しての調査・分析・助言などを行う。	家庭用テレビゲームソフト、携帯電話ゲームソフト、SNS ゲームソフトなど。	任天堂、ソニー、セガ、カプコン、スクウェア・エニックス、マイクロソフトなど。

出所：総務省統計局（2013）「日本標準産業分類（平成 25 年 10 月改定）」、一般社団法人情報サービス産業協会編（2015）『情報サービス産業白書 2015』をもとに筆者作成

32 実際のソフトウェアの開発作業は、系列の子会社や協力会社などの下請け企業が行うこともある。



出所：古殿（2006），島田・高原（2007）をもとに筆者作成

図 2-8 パッケージソフトウェアの種類と範囲

具体的には、量販店の店頭で売られている会計ソフトウェアや給与計算ソフトウェアから、ERP パッケージ[33]と呼ばれる企業システムの統合管理用のソフトウェアなどがあげられる（図 2-8）。

しかし、OS[34]やミドルウェア[35]、ERP、SCM[36]などのパッケージソフトウェアは、その大部分がドイツの SAP 社やアメリカの ORACLE 社に代表されるような欧米製のソフトウェアで占められており、国産のソフトウェアは少ない。

---

[33] Enterprise Resource Planning package。ERP は、企業の経営資源を一元管理することで、経営資源の効率的な効用を図ろうとする方法であり、企業内部の生産、物流、財務・会計などの効率化に焦点を置いている（古殿, 2006; 島田・高原, 2007）。

[34] Operating System。コンピューター機器の管理や制御など、パソコンを動かすため基本的な機能としてシステム全体を管理するソフトウェア。初期の頃のコンピューターは、OS を持っておらず、IBM の SYSTEM/360 が世界初の商用 OS とされる。その後、UNIX や MS-DOS といった OS が次々登場していった。

2017 年時点で、個人向けのものとしては、Microsoft 社の Windows シリーズや Apple 社の Mac OSX などがある。また、企業向けの業務用パソコン向けとしては、AT＆T 社のベル研究所が開発し、Sun Microsystems 社などが商用に販売している UNIX 系 OS や、Microsoft 社の Windows Server などが存在する。そのほか、2010 年以降に利用者が増加しているスマートフォン向けとして、Apple 社の iOS や Google 社の Android OS などが存在する。

[35] OS（operating system）とアプリケーションソフトウェアの中間のソフトウェア。データベース管理システムやサーバー管理システムなどが該当する。

[36] Supply Chain Management。製品とサービスの供給業者からの効率的な調達を目的としており、全体最適化の観点から、需要の予測や在庫管理・生産計画などの情報を企業間で共有するシステム。この需要予測の精度を上げるには、サプライチェーンを構成する企業間の情報を共有する必要がある（古殿, 2006; 島田・高原, 2007）。

情報サービス産業協会が 2003 年 12 月に発表した調査結果によれば、日本における 2002 年のパッケージソフトウェアの輸出額は 93 億 1,300 万円なところに対し、輸入額は 2,962 億 5,200 万円と、30 倍以上もの輸入超過となっている。海外で開発されたこれら OS やソフトウェアのプラットフォームは、グローバルで利用できるように多言語化されており、日本でも普及が進みつつある。

d) ゲームソフトウェア業

「ゲームソフトウェア業」は、家庭用テレビゲーム機やパソコンゲーム、携帯電話のゲームやそのソフトの作成、及びその作成に関しての調査・分析・助言などを行う事業所を指す。

具体的には、ファミリーコンピューターやプレイステーションなどの家庭用のテレビゲームソフト、iPhone などのスマートフォンのゲームや SNS[37]上のゲームなどがこれにあてはまる。

これらゲームソフトウェアのうちテレビゲームのソフトウェアは、日本においてスマートフォンのゲームソフトウェアにシェアを奪われつつあり、その成長に陰りが見えてきたものの、題材となる漫画やアニメの人気が高いことがその国際競争力に貢献しており、日本のソフトウェアの中でも唯一海外に輸出されている分野ともいえる（高橋, 2010）。

本研究では、このうちカスタムソフトウェアを対象としているが、このカスタムソフトウェアの開発には、開発にかかる一過性のコストと、開発後の保守・運行にかかるランニングコストが存在する。特に受託ソフトウェアの開発は、システム開発の受注時の価格、つまり開発コストが注目されるが、システムを利用し続ける限りその後の保守・運用が必須となるため、ランニングコストが開発コストを上回ることが多い（飫富・廣松・小林, 2009）。

このような保守・運行に関わるランニングコストが大きく発生することに目を付けて、システム開発の受注競争で、他の企業の予算を遥かに下回る安値で落札し、その後数十年の保守を契約するようなケースも存在する（田中, 1988a）[38]。

---

[37] Social Networking Service。ソーシャルネットワーキングサービス。インターネット上に構築された、社会的ネットワーク。参加型コミュニティサイトなどが代表的であり、Facebook や mixi などが有名である。

[38] 東京都の総合文書管理システムを日立製作所が 750 円で落札したケースがある（ITpro, 2001）。通常、大規模システムの開発は数年かかることが多い。しかし、代金が開発完了時に払われるような場合、それまでに経営体力を維持し続けられる企業は少なく、自然と規模の大きい企業が落札することが多い。しかし、このような限度を超えた安値で買い叩くような行為は、ソフトウェア開発を経営体力がある大手企業しか受注できなくなってしまう弊害がある。

さらに、官公庁によるソフトウェアの調達は、年間売上高や自己資本額、営業年数などの外形的要素で入札の参加資格が絞られてしまい、長年営業を続け、売上高が大きい大手

また、OS などの基本ソフトウェアは総じて開発規模が非常に大きく、そのようなソフトウェアを開発するために巨額の開発資金の負担や多数の技術者を擁することのできるのは大企業に限られる（田中, 1988b）。これに対して、応用ソフトウェア（アプリケーションソフトウェア）は相対的に開発規模が小さいため、開発する企業は必ずしも大企業である必要はない。もっとも、これら基本ソフトウェアの開発自体も、オープン・ソース・ソフトウェア[39]などを利用することでその開発費用を抑えるケースもでてきている（谷花・野田, 2012; 神戸, 2014）。

一方で、パッケージソフトウェアは、カスタムメイドの受託開発とは異なり、大量にコピーし販売することで規模の経済を生かすことができる。このパッケージソフトウェアのビジネスモデルの特徴は、一度ソフトウェアが完成できれば複製して販売できることにある。開発コストは多額に上り、固定費がそのほとんどを占める。ただし、ソフトウェアが完成してしまえば、それ以降開発コストはかからないため、販売数量が損益分岐点を超えれば､その売上高は利益となり､売上高の伸張に伴って利益率も拡大していくことになる。

「平成 26 年特定サービス産業実態調査（確報）」（経済産業省経済産業政策局調査統計部サービス統計室）によると、この情報サービス産業の概況として、ソフトウェア業全体の事業所数は平成 26 年の時点で、14,000 ヶ所近くにのぼり、従業員数は 67 万人、年間売上高は 10 兆円前後にまで伸びている（**表 2-3**）。

表 2-3 ソフトウェア業の業務種類別年間売上高（平成 26 年）

	事業所数	年間売上高（百万円）	年間売上高構成比
ソフトウェア業　計	13,670	10,088,605	100.0%
受注ソフトウェア開発	12,425	8,322,392	82.5%
ソフトウェア・プロダクツ	4,144	1,766,213	17.5%
業務用パッケージ	3,648	1,008,678	10.0%
ゲームソフト	458	501,745	5.0%
コンピューター等基本ソフト	508	255,790	2.5%

出所 : 経済産業省経済産業政策局調査統計部サービス統計室「平成 26 年特定サービス産業実態調査（確報）」,「「ソフトウエア業」統計表一覧　事業従事者 5 人以上の部」をもとに筆者作成

企業に有利に働いている（ソフトウェア産業研究会, 2005）。

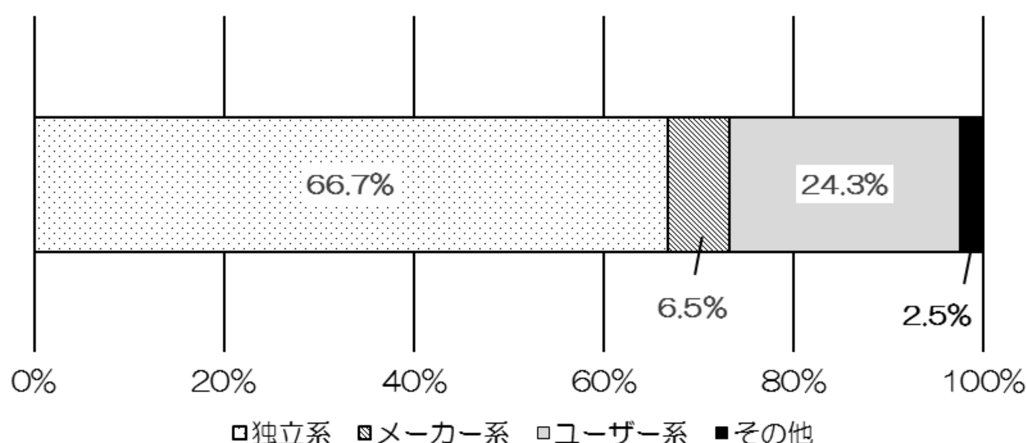
[39] Open Source Software。インターネットなどを通じて公開された、誰でも無償で自由に利用や複製、改変、再配布が可能なソフトウェア。詳しくは、第 10 章の本研究の課題において説明する。

これらソフトウェア業の構成比のうち、82.5%が受託ソフトウェア開発となっており、それ以外はソフトウェア・プロダクツとしてまとめられ、17.5%となっている。さらに、ソフトウェア・プロダクツの17.5%のうち、10.0%が業務用のパッケージソフトで、ゲームソフトは5.0%、コンピューターなど基本ソフトに至っては、2.5%にすぎない。

このように、日本のソフトウェア業は、「受託ソフトウェア開発」、つまりカスタムソフトウェアが非常に多く、ソフトウェア業としての売上の大部分を占めていることが確認できる。

④　ソフトウェア開発企業の分類

日本のカスタムソフトウェア開発に関わる企業は、主にメーカー系、ユーザー系（商社系）、独立系の3つに分類できる（**図 2-9**）[40]。



出所：一般社団法人情報サービス産業協会（2016）『2015年版基本統計調査報告書』p.11をもとに筆者作成

図 2-9 カスタムソフトウェア業の資本系列別構成

メーカー系のITベンダーは、富士通やNEC、日立、IBMなど、かつてコンピューターメーカーとしてハードウェアの販売を兼ねてきた企業が多く、これら企業はソフトウェア業界における大手ITベンダーに位置している。また、各メーカーの子会社などもこの系列に含まれる。

ユーザー系（商社系）企業には、野村総合研究所や、新日鉄ソリューションズ、伊藤忠テクノソリューションズ、住商情報システムといった商社や金融などの子会社が多く、そのほとんどは、商社や金融会社の情報システム部門だったものが子会社として分離・独立

[40] グラフ中の「その他」は、海外企業の出資比率が50%以上の企業、各種団体、または、独立系・メーカー系・ユーザー系のいずれの系列にもあてはまらない企業を指す。

したケースが多い。そのほか、キリンビジネスシステム[41]やジェイアール東海情報システム[42]といった企業のグループ会社として情報システムの開発を担うケースも多い。

独立系は、大塚商会や CSK ホールディングスなど、これまであげたメーカーやユーザー系列に属しておらず、親会社を持たない資本的に独立した企業が対象となる。特に独立系企業の多くは、日本のソフトウェア産業の勃興期である 1960 年代後半から 1970 年代にかけて設立されたものが多く、その当時、最も普及していた IBM のマシンに対する受託計算を行う情報サービス業が活発であった（一般社団法人情報サービス産業協会編, 2015）。

このように、カスタムソフトウェア開発の企業は、大まかに 3 つのグループに分かれているが、その分け方に厳密な定義があるわけではなく、例えば NTT データなど、業種・範囲が広く、どの系列にも分類しにくい企業も存在する。

⑤　オフショア開発

ソフトウェア開発では、そのプロセスの一部、あるいはすべてを開発会社にアウトソーシングすることが多い。プログラム作成などの下流工程はアウトソーシングし、要件定義や設計など重要な部分は自社や情報システム子会社、もしくは大手などの信頼あるソフトウェア企業に委託する上流工程重視の戦略である。

さらに、日本では、2000 年頃の IT バブルでソフトウェア開発の需要が増えてから、海外のソフトウェア開発企業にアウトソーシングするオフショア開発[43]が進展しており、その潮流は日本に限らず世界的なものとなっている（**図 2-10**）。例えば、中国やインドなど海外企業に委託する場合、低い価格が開発における大きな魅力となる。

このオフショア開発の主な目的は、国内で不足している人材の確保や開発コストの削減であり、国内のソフトウェア技術者の不足が原因の 1 つとなっている（高橋, 2013）。

こうした国内の人材事情について、「情報経済革新戦略～情報通信コストの劇的低減を前提とした複合新産業の創出と社会システム構造の改革～」（経済産業省商務情報政策局産業構造審議会情報経済分科会, 2010）によると、2001 年の IT バブル崩壊後も情報システムやそのソフトウェアの大規模化や複雑化に加え、あらゆる社会活動で IT が活用される環境が出現したことにより、情報処理に係る人材需要が急激に拡大しているという[44]。一方で、

---

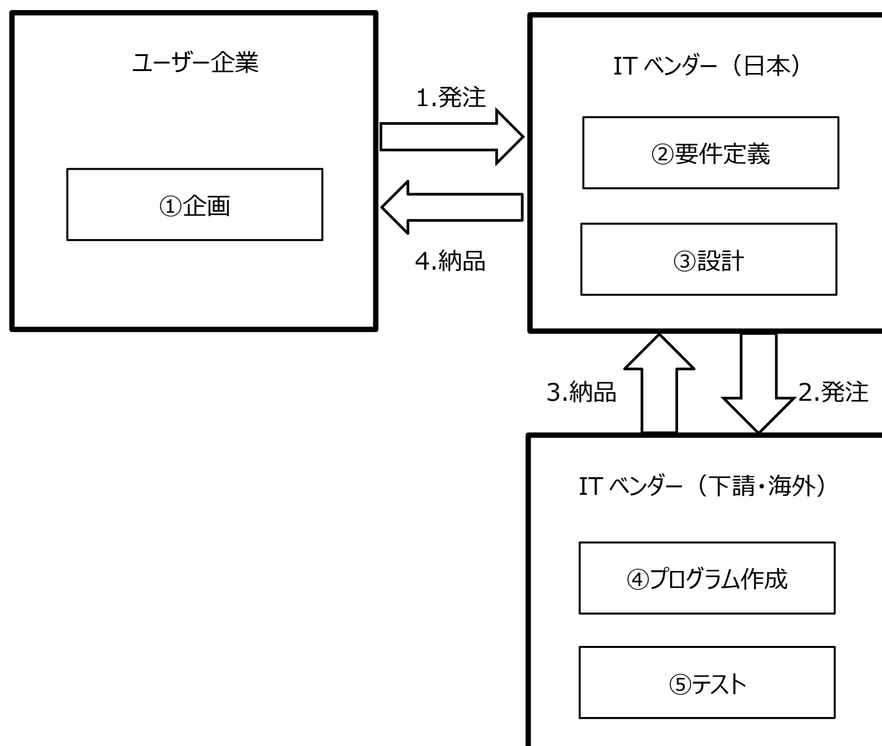
[41] キリングループの情報機能会社。キリングループのシステム開発・運用・保守・ユーザー支援などを提供している。なお、完全子会社ではなく、発行済株式の 49%は株式会社 NTT データが取得している（キリンビジネスシステム, 2016）。

[42] ＪＲ東海の完全子会社。情報システム会社として、ＪＲ東海やＪＲ東海グループ各社に対し、情報システムとそのサービスを提供している（ジェイアール東海情報システム, 2016）。

[43] Offshore Development。ソフトウェア開発や運用管理を、海外事業者や海外子会社に委託すること。特にソフトウェアのプログラム作成工程やテスト工程を、人件費の安い中国やインド、ベトナムなどの新興国に委託することが多い。

[44] 企業が抱える情報システムやその資産は多くなる一方であり、例えば、キリングループのシステム開発を請け負うキリンビジネスシステムによると、社内には、約 400 のシステ

足下は景気後退の影響により人材の量的な不足感が解消されてきているものの、質的な不足感は依然根強いという。つまり、人材の量も不足しているが、それ以上に質の面で解消ができていないのが日本のソフトウェア産業の現状である。



出所：筆者作成

図 2-10 ソフトウェア開発のアウトソーシングの流れ（オフショア開発）

さらに、八尋・片山（2008）によると、特にハイレベルの IT 技術を担う高度 IT 人材が質的にも量的にも圧倒的に不足していると述べており、有効求人倍率でも、2007 年度は全職種が 1.0 倍を下回っている一方、情報処理技術者は 3.5 倍程度と、深刻な状況となっているという。今後もこうした国内のソフトウェア技術者の状況が急速に改善されることは考えにくく、技術者の確保のため、オフショア開発による海外の人材の活用が強まることが考えられる。

ただし、オフショア開発がもたらすものは、このような期待された側面ばかりではない。オフショア開発は、ブリッジ SE と呼ばれる現地と日本の連携を行なう技術者やマネジャーが必須である。言語の隔たりもさることながら、商習慣や文化も異なる国同士のやり取りには、高度な調整能力を持つ人材が必要となる。しかし、日本企業は、日本の下請け構

ムが稼働しており、保有するサーバーの数も 2,000 を超えるという（ITpro Active, 2016）。また、2012 年末時点で、クラウドサービスの利用を除く企業が所有する日本国内のサーバー数は、200 万台以上とされる。

造をそのまま海外にあてはめようとし、オフショア先の企業をビジネスのパートナーとしてではなく、下請け企業扱いする傾向が強い。さらに、このブリッジ SE の育成が間に合わないことや、日本企業がブリッジ SE も置かずに日本の仕様書をオフショア先企業に丸投げして開発を依頼するようなことも多く、このことは日本のオフショア開発の比率が最も高い中国で失敗する原因の一つとしてあげられてきた。特に、日本の中小 IT ベンダーで、英語の仕様書などの言語の壁や文化の隔たりを乗り越えて、世界に対してサービスを輸出できるような人材を擁する中小企業はほとんどないと考えられる。

(2) 日本のカスタムソフトウェアへの偏重

① パッケージソフトウェアの導入

企業がシステム導入を行うにあたり、パッケージソフトウェアか、カスタムソフトウェアかの選択が考えられる。前節で述べたように、日本の情報サービス産業の売上高のうち、82.5%が受託ソフトウェアであるのに対し、パッケージソフトウェアは 10.0%程度にすぎず、日本のソフトウェアは、カスタムソフトウェア偏重となっている（経済産業省経済産業政策局調査統計部サービス統計室)。

工藤（2009）によると、アメリカでは ROI による判断基準から、システムの機能の取捨選択を行っており、日本よりも ERP のようなパッケージソフトウェアの利用が多いという[45]。日米のソフトウェアの投資で大きく異なる部分は、この ERP などのパッケージソフトウェアの割合が、アメリカでは 3 割を占めているのに対し、日本では全体の 1 割以下と小さいことである。

このような日本のユーザー企業がパッケージソフトウェアの利用ではなくカスタムソフトウェアによる受託開発を選択する理由として、高い社内の調整コストのほか、ユーザー企業の能力不足などがあげられる（田中, 2009)。

この社内の調整コストとして、アメリカと日本企業との業務習慣、文化の違いが考えられる。アメリカでは、パッケージソフトウェアを利用する際もカスタマイズや拡張をせず、企業のビジネスモデル自体をパッケージソフトウェアに合わせていく傾向がある。一方、日本では、パッケージソフトウェアに合わせて自社のビジネスモデルを変えるようなことはせず、逆にカスタムソフトウェアのようにパッケージソフトウェアを大きくカスタマイズすることで対応していくケースが多い。

このような理由として、日本では現場の力が強く、パッケージソフトウェアに合わせてこれまでの方法を変えることに大きな抵抗があることが考えられる。特に業務システムを

---

[45] ただし、工藤によると、アメリカでは ERP 本来の使い方である基幹業務のすべてをパッケージソフトウェアで行うことが多いが、日本では自社業務のうち会計システムの部分だけに ERP を導入したものもカウントされており、その実態には大きな違いがあるという。

統合的に管理する ERP パッケージソフトウェアの導入には、CIO[46]や経営トップが指導力を発揮し、トップダウンで導入を進める必要があるが、日本企業では、現場の意見が非常に強いボトムアップ型の組織構造となっていることが多い。そのため、情報システムの構築に際しても、現場の意見が尊重されやすく、それゆえそれまでの業務の手続きをまったく変えてしまうような全体最適化は難しく、部分最適やカスタマイズを求める傾向が強いといえる（田中, 2009）。さらに、日本語や日本特有のビジネス慣行も海外のパッケージソフトウェアを導入する際の障壁となっている（高橋, 2010）。

このようなパッケージソフトウェアは Microsoft や Oracle、SAP など海外の IT ベンダーにより、すでに世界で標準化されているものが多い。世界中で多くの利用者が存在するため、新たにソフトウェアを利用するための教育を行う手間が減ると同時に、他企業との協業も行いやすくなるといったスケールメリットがある。しかしながら、日本では、これまで言及されてきたような理由により、カスタムソフトウェアからパッケージソフトウェアの導入に踏み込めず、パッケージソフトウェアによるこのようなネットワークの外部性による恩恵にあずかることはできていない（田中, 2009; 稲村・渋谷, 2013）。

朴・藤本（2016）は、このような欧米型の ERP パッケージソフトウェアが世界で標準化できた理由として、1970 年代にソフトウェア産業として成立してきた頃より、長い時間をかけて顧客との緊密な関係の中で産業ごとのベストプラクティスを構築してきたことを指摘している。一方で、日本の IT ベンダーは、顧客企業ごとの要求に沿ったカスタムソフトウェアの開発を行ってきた。そのため、産業ごとのベストプラクティスが IT ベンダー内に蓄積されにくいことを指摘している。

そのほか、日本のユーザー企業がカスタムソフトウェアの導入に偏重している理由として、システムを採用する側であるユーザー企業自身の問題に起因する部分が考えられる。例えば、システム開発を発注するユーザー企業の担当者が仕様書を作成できない問題や、ソフトウェアに関する基本的な知識が欠けているため作成されたソフトウェアに対する適正な評価が行えないといった問題である。また、IT ベンダーにある程度依存できる受託開発とは異なり、パッケージソフトウェアの導入はユーザー企業の担当者がその利用の責任を負うことになるため、自社の業務や社内のシステム、導入するパッケージソフトウェアに深く精通している必要がある（田中, 2009; 中川, 2011）。

さらに、ソフトウェア導入時のリスクを回避しようとする問題も考えられる。金融機関の基幹システムや鉄道の運行管理システム、空港の管制システムなど企業に導入されているシステムの多くは､いまや企業のビジネスだけでなく社会インフラの中枢を担っており、一度障害が発生すれば、その損失は多大なものとなる[47]。業務の要望を次々と反映してい

[46] Chief Information Officer。最高情報責任者。
[47] 例えば、2002 年 4 月のみずほ銀行の合併で発生した ATM の障害により、公共料金の引き落としなどができなくなったほか、日本航空や全日本空輸のシステム障害により、飛行

った結果、システムは複雑になり、その改修や更新自体も難しくなってきている。そのため、これまで利用してきたカスタムソフトウェアからパッケージソフトウェアに変更することで大きなトラブルが発生した場合、システム担当者はその職歴に致命的なダメージを負うことになる。そのような経営リスクをシステムの担当者レベルで負うことは難しく、それゆえ大きなリスクを伴うようなソフトウェアの変更には慎重になると考えられる（田中, 2009）[48]。

② カスタムソフトウェアの導入

このように、日本のユーザー企業がカスタムソフトウェアを選択せざるを得ないネガティブな要因が多く存在するが、一方でカスタムソフトウェアをあえて選択する合理的な理由も存在する。

例えば、日本企業の強みは社内に蓄積するノウハウであることが多く、カスタムソフトによってその強みを実装できるという点で一定の合理性があるとする考えである。

田中（2009, 2010）は、カスタムソフトウェアを選択することが、日本の長期取引関係などを生かし、企業の競争力を維持する戦略として合理的であると述べている。カスタムソフトウェアは、その企業固有に作られたものであり、つまり企業のノウハウといったものを取り込むことが可能となる。そのため、企業の競争能力がそのようなノウハウに存在するとすれば、それを維持、活用するためにもカスタムソフトウェアを導入することが考えられるのである。

一方、アメリカがパッケージソフトウェアを重視している理由も、このノウハウの有無に起因することが考えられる。アメリカでは、市場の変化にすばやく対応することを重視しており、労働市場の柔軟性の高さやプロジェクトごとに技術者を採用するようなソフトウェアの開発スタイルのため、企業特有のノウハウが溜まりにくく、それゆえ企業のノウハウを取り込むことが必要となるカスタムソフトウェアより、標準化やネットワークの外部性を持つパッケージソフトウェアを重視していることが考えられる（田中, 2009, 2010）。

これまでの研究の中には、ここまで言及されてきたようなカスタムソフトウェアのデメリットを説明し、生産性の高いパッケージソフトウェアへの転換を推進するようなものも多い。しかし、本研究の目的は、カスタムソフトウェアとパッケージソフトウェアの優劣を決めることではない。確かに、地方銀行のように銀行同士の共通のシステムとしてパッケージソフトウェアを利用して開発した事例も存在する（山沖, 2011, 2012, 2014）。一方で、

機の遅延や欠航が発生するなど、社会に多大な影響を及ぼしてしまう。

[48] このようなリスクを回避するため、カスタムソフトウェアを選択する際に、IBM や富士通といった大手や著名な IT ベンダーに依頼する傾向にあるとも考えられる。無名のソフトウェア開発企業ではなく、こういった著名な IT ベンダーにシステム開発を依頼することで、仮にそのシステム開発が失敗したとしても大義名分が立つことになる（田中, 2009）。

1 万台近くのコンピューター端末を安定的に管理している JR の座席予約システムや、世界最高水準の高速性と信頼性を誇る東京証券取引所の株式売買システムのように、カスタムソフトウェアによって企業の競争力に大きく貢献しているものも存在する。

このような特殊なソフトウェアは標準的なパッケージソフトウェアでは実現することは難しく、仮にパッケージソフトウェアを採用したとしても、大きくカスタマイズすることが必要となってしまう。さらに、他の企業と同一のパッケージソフトウェアを導入した場合、その企業のシステムに関連するノウハウの反映や、システムを利用した差別化ができなくなることを意味し、企業はシステム以外のところで差別化を図る必要性が出てくる。

もっとも、日本ではカスタムソフトウェアに対し、2000 年前後の IT 投資が活発でその投資も制限されてこなかった IT バブルの時代に、必要とされない多くの機能を盛り込んだ過剰なシステムが作られてきたこともあり、批判の対象にもなってきた。一方で、パッケージソフトウェアとカスタムソフトウェアを柔軟に組み合わせるようなケースのほか、クラウド・コンピューティングを利用したアプリケーションのみを利用する方法なども出てきている。重要なことは、目的や規模、そしてビジネスに合わせてソフトウェアを選んでいくことであるといえる。

(3) 小括

本章では、次章の先行研究の検討に入る前提として、予備的な考察として日本のソフトウェア産業の現状を確認してきた。日本では、ソフトウェアを開発する際、そのプロセスの一部、あるいはすべてを開発会社にアウトソーシングすることが多く、ソフトウェア産業は大企業を中心とした下請け構造で構成されている。

多くの企業では、ソフトウェアを開発するために社内に専門的な技術者を抱えていることは少なく、ソフトウェアを導入、運用するためには、外部のソフトウェア開発の専門企業によるプロフェッショナルなサービスを受ける必要があり、カスタムソフトウェアも外部の IT ベンダーにアウトソーシングすることで開発されてきた。

こういったソフトウェアは、カスタムソフトウェアとパッケージソフトウェアに大きく分類できるが、日本のソフトウェア産業はカスタムソフトウェアに大きく偏っている。このような日本のユーザー企業がカスタムソフトウェアを選択する理由には、一定の合理性が存在しており、例えば、日本企業の強みは社内に蓄積するノウハウであることが多く、企業固有のシステムとしてそのノウハウを実装できるため、日本の企業がカスタムソフトウェアを採用するというものである。

パッケージソフトウェアは、企業のビジネスモデルをソフトウェアに合わせる必要があるものの、標準化やネットワークの外部性といった恩恵を与えてきた。一方で、カスタムソフトウェアは、逆にビジネスモデルにソフトウェアの設計を合わせるため導入に手間やコストがかかるものの、標準的なパッケージソフトウェアでは実現できないようなカスタ

マイズが可能であり、それゆえ企業の競争力に大きく貢献してきたのである。
　次章では、このような日本のソフトウェア産業の前提を踏まえ、日本のソフトウェア産業についてこれまでどのような評価や分析が行われてきたのか、先行研究を検討するとともに、本研究の問題設定を行う。

第3章 先行研究と問題設定

本章では、ソフトウェア産業について、これまでの先行研究のレビューとそれを踏まえた本研究の問題設定を行う。

本研究の視点によれば、ソフトウェア開発、特にその下流工程は従来の理解とは異なり担当者の保有する様々な知識に依存するところの大きい知識労働としての側面を備えるものと考えられる。しかしながら、日本のソフトウェア産業においてこれまでこうした捉え方がなされてきたかというと、必ずしもそうではない。

本章では、本研究が主張する、わが国のソフトウェア産業が直面する変化の速い市場に対応できず、さらに革新的なイノベーションが期待することができていないという問題が引き起こされていることと、その問題の克服のためにはソフトウェア開発業務の知的作業あるいは知識労働としての側面を重視し、開発従事者の知的能力の発揮を阻害しない開発プロセスの編成が必要となることを説明するにあたり、「知識マネジメント」の問題解決を中心とした考え方に関する整理を行う。その上で、日本のソフトウェア産業が先行研究において、これまでどのような評価や分析が行われてきたのか、そこでソフトウェア開発がどのような業務、労働であると捉えられてきたのか、先行研究の検討を通じて確認するとともに、そこから導き出される問題設定について述べる。

(1) これまでの先行研究との関係

本研究は、ソフトウェア開発の分業構造を対象とし、アーキテクチャーやビジネスモデルについての先行研究を検討している。しかし、ソフトウェアを対象とした研究の多くが、その技術的な特性からソフトウェア工学やそれに類するものであり、ソフトウェア開発における分業の問題や、日本のソフトウェア開発企業の戦略やビジネスモデルなどの経営学的視点より研究したものはほとんどない。

① 分業に関する研究との関連

分業に関する研究は自動車産業などを対象とした研究（浅沼, 1997; 武石, 1999; 藤本, 2001a, 2001b など）が多く、ソフトウェア産業を対象とした研究は少ない。

当然ながら、アウトソーシングを含めた企業間関係とその境界を対象とする研究や著書は数多く存在し、現在でもその研究は活発に行われている。さらに、企業の競争優位に関わる知的作業として、企業の中核となる資源であるコア・コンピタンスに注目した研究（Prahalad and Hamel, 1990）などもある。

そのほかに、企業間の分業関係に関わらず、従業員間といった企業の内部のコンテクストの共有化に関する研究も多い。例えば、青島編（2008）は、従業員間でコンテクストの共通化が図られることにより、組織内のコミュニケーションがスムーズになるとしている。

Clark and Fujimoto（1991）は、企業内における部門同士の密接な協力を部門間統合と呼び、それが日本の自動車メーカーが高品質で、顧客ニーズに合った製品を開発できる要因の一つとなっているとしている。

一方、企業の内外のそのような分業関係の問題は、単純にコミュニケーションの良し悪しだけに起因するものではない。例えば、企業間提携として、相手組織の行動に対する組織間の信頼の質についての研究も進んでおり（Sako, 1992; 真鍋, 2002, 2004; 川崎, 2014）、英米では契約関係の誠実な履行を信頼しているのとは対象的に、日本では長期的で互恵的な善意に基づく信頼があることが指摘されている。また、真鍋（2004）は、トヨタ自動車とそのサプライヤー間の信頼関係を定量調査し、コミュニケーションは知識の共有を進めるものの、信頼関係の構築には協力会などによる組織間学習や長期的な取引関係などがその要因と考えられるとしている。

青島編（2008）は、日本企業の従業員の職務範囲の境界が曖昧であり、従業員が助け合うことで多能工化が進みやすくなっており、そのことで顧客ニーズへの対応のほか、業務の繁閑にも柔軟に対応できていると述べている。そのほかに、企業のドメイン固有の知識間にある境界について、その橋渡し役としてのプロジェクト・リーダーの役割の重要性についての研究（藤本, 2001b; 椙山, 2001; 林, 2008 など）も進んでいる。

一方で、Taylor（1911）に代表される科学的管理法のような高度な専門分化と作業の単純化の研究から、そういった行きすぎた分業と機械化に対する批判として、1950 年代の炭鉱労働の研究に端を発する技術的要因と社会的要因の両方を充たす社会・技術システム論（Trist and Bamforth, 1951; 上林, 2001; 森田, 2010）のほか、1970 年代に始まった Quality of Working Life や労働の人間化（小林, 2008; 菊野, 2010）などの人的資源管理の分野で多くの研究が蓄積されている。

このように、分業に関する研究は多くの知見が存在するが、企業間関係の研究としてソフトウェア産業を対象としたものは少ない。さらに、ソフトウェア産業の分業構造を対象とした研究においても、その表面的な特性や問題を述べるに留まっていることが多く、ソフトウェア開発のプロセスまで深く踏み込んだ研究はほとんどない。

② ビジネスモデルに関する研究との関連

本研究では、日本のソフトウェア開発を製造工場化しようとしたビジネスモデルについて分析しているが、このようなソフトウェアのビジネスモデルに関する研究としては、2000 年以降、企業の枠に囚われないオープン・ソース・ソフトウェア開発の研究が盛んとなっている（国領, 2004; 竹田, 2005; 谷花・野田, 2012, 2013; 野田・丹生・コークラン, 2012, 2013; 野田・丹生, 2014; 神戸, 2014, 2015）。しかしながら、このオープン・ソース・ソフトウェアの開発手法は、仕様書もないまま、インターネットを介して多くの開発者によって自由に開発が行われており、本研究が対象とするカスタムソフトウェアといった企業の受託に

よるソフトウェア開発とはその方法や目的が大きく異なっている[49]。

また、ソフトウェア開発に従事する知識労働者としての技術者に焦点をあてた人的資源管理の分野の研究も多く、例えば、ソフトウェア技術者の能力の活用やキャリア形成についての研究（今野・佐藤, 1987, 1990; 梅澤, 1996, 2000; 三輪, 2009, 2010, 2012, 2014; 古田・藤本・田中, 2013）がある。そのほか、システム開発の技術の多様化やステークホルダーの増加に対するマネジメントとして、プロジェクトチーム内のコンフリクトの研究（床井・妹尾, 2010）もある。

このように、ソフトウェア産業のビジネスモデルに関する研究は、オープン・ソース・ソフトウェアを中心に、遠隔地を通したソフトウェア開発や、コミュニケーションに関する研究が確認される。しかし、ソフトウェアに関する研究の多くは、その技術的性格から、ソフトウェアの利用方法やその導入方法について、ソースコードの行あたりのバグなど欠陥の数やアルゴリズムの解析といったソフトウェア測定法[50]について、そしてプログラムを効率よく作成する方法についてなど、ツールや手法の開発に傾斜した技術や工学の分野が中心となっており、人間的な部分や組織的な部分など社会学や経営学に関する研究は非常に少ない（DeMarco and Lister, 1987, 1999, 2013）。

(2) 日本のソフトウェア産業に関する研究

本節では、これまでの日本のソフトウェア産業がどのように捉えられてきたのか、本研究に特に関係するものとして、5 つの先行研究を検討する。

ソフトウェア産業や IT 全般について、特にその技術に関する研究が非常に多いものの、日本のソフトウェア産業の分業や組織構造の問題を題材とした研究は非常に少ない。特に 2000 年以前の日本のソフトウェア産業について研究されたものは少なく、その中の重要な研究結果として、Cusumano（1991, 2004）や今井・安藤・白井・辻・久保・玉置・浜田（1989）が存在する。

[49] オープン・ソース・ソフトウェアの開発コミュニティとして、OS（オペレーションシステム）である Linux の開発がある。この Linux は、Microsoft の Windows などの企業が開発する OS 製品に匹敵する機能、品質、信頼性を持っている。自発的に技術者が参加したり、メンバーが入れ替わったりするような集合体によって、高度なソフトウェアの開発が可能であるとともに、全体としてのソフトウェア生産能力を維持しうることを証明した（竹田, 2005）。

オープン・ソース・ソフトウェアについては、本章ではこれ以上取り上げないが、第 10 章の本研究の課題で説明する。

[50] プロセスの評価・改善、工数の見積り、信頼性の評価などソフトウェアの品質や生産性の改善を目的とし、ソフトウェアとその開発や利用過程を対象とした定量的な評価尺度（松本, 1998; 阿萬・野中・水野, 2011）。

① Cusumano によるソフトウェア・ファクトリーを中心とした研究

Cusumano（1991, 2004）は、1980 年頃よりソフトウェア産業を対象とした研究を進めており、7 年近い日本での滞在の中で、1970 年代から 1980 年代当時の日本のソフトウェア開発の中心となっていたソフトウェア・ファクトリーと呼ばれる、製造業の工場を模した開発方法の調査を行っている。Cusumano は、その中で日本と北米などの海外との間でソフトウェアのビジネスモデルの違いを分析し、当時の日本のソフトウェア開発企業は米国に次ぐ規模のソフトウェアを生産しており、そのソフトウェアは品質管理や技術者管理、製品テスト技術、顧客の高い要求への対応により、ソフトウェアの欠陥を最低限に抑えながら生産してきたと述べている。なかでも、日本のプログラマーは多くのプログラムを生産しており、プログラムを生み出すプロセスの点で優秀であり、その生産指標は米国企業と同等かそれ以上だと述べている。

一方で Cusumano は、日本のソフトウェア企業が日本国内のユーザーに向けて対応するあまり、海外に向けた英語によるインターフェースを備えた製品を設計せず、グローバルで標準化された製品の開発をほとんど行ってこなかったことを指摘している。さらに、日本企業の多くは、IBM に代表される米国企業に追従しながら、メインフレームと呼ばれるコンピューターを特定の顧客向けにカスタムまたはセミカスタムのシステムの設計と構築を労働集約型で行ってきた。しかしながら、こうしたシステムは設計上のイノベーションといえるようなものはほとんどなく、日本企業の多くは、革新的能力や野心がさほど強くなかったことが指摘されている。

このような日本のソフトウェア産業でイノベーションが見られなかった原因として、Cusumano は、1969 年以降、日立や東芝、NEC、富士通などの日本の主要なコンピューターメーカーが、技術者や専門家を大きな施設に集めることで、ハードウェアの製造と同様に統制が可能となるアプローチを採用し、製造と品質を管理しようとしていたことを明らかにしている。この方法はソフトウェアのライン生産方式のような製造工場化、すなわちソフトウェア・ファクトリーと呼ばれており、Cusumano は標準化された設計パターンや元の条件からほとんど変更しないようなソフトウェア・システムを構築するためには非常に適しているが、それ以外には適していないと述べている。

このソフトウェア・ファクトリーについては後の章で詳しく述べるが、Cusumano は、アメリカの Microsoft や Netscape のような世界を変えてきた企業を比較対象にあげ、ソフトウェア開発は製造活動ではなく製品設計であり、その設計こそが製品であるとし、日本のソフトウェア・ファクトリーのようなシステムでは欠陥ゼロのソフトウェアを作成できても、世界を変えるようなものは生まれず、それによって大きな利益を生むこともないと述べている。

以上のように、Cusumano の研究は、日本のソフトウェア産業の特徴として 1970～1980 年代に注目を浴びたソフトウェア･ファクトリーを中心に分析を行っている。このことは、

本研究において、当時の日本のソフトウェア産業がどのような方法でソフトウェア開発の効率化を図ろうとし、今日の分業構造と組織問題を生ずるに至ったのかを紐解く鍵となる。この製造業と同様の品質管理手法を模した方法は、確かに当時需要が高かった類似したソフトウェアの開発には向いていたのである。

ただし、Cusumano はこれ以降、戦略やプラットフォームビジネスの研究にシフトしている（Gawer and Cusumano, 2002; Cusumano, 2010）。そのため、Cusumano（1991, 2004）において、日本のソフトウェア・ファクトリーの特徴と問題点までは明らかにしたものの、その後そのソフトウェア・ファクトリーが失敗してしまったことについてはほとんど触れていない。このソフトウェア・ファクトリーのような方法は、1990 年代のソフトウェア開発の大規模化や複雑化、さらに 2000 年以降の変化と速さを中心としたソフトウェア開発が中心となるにつれて、適合できなくなっていった。これまでの日本のソフトウェア産業は、ソフトウェア開発を製造業とみなし、高品質のソフトウェアを作成することに力を注いできたが、その前提が崩れてしまったのである。

②　今井他による組織風土を中心とした研究

今井・安藤・白井・辻・久保・玉置・浜田（1989）の研究は、ソフトウェアに関連する研究者と実務者で構成されており、ハードウェアからソフトウェアへとその価値が移りつつあることを述べるとともに、ソフトウェア開発のプロジェクトの成功要因を組織構造に求め、特にその組織の風土にあるとした。

今井他は、Cusumano（1991, 2004）と同様に 1970～1980 年頃の日本のソフトウェア産業を対象とし、その開発現場の著しい後進性として、IT のような情報化をリードする先端技術産業の名にはおよそそぐわない苛酷な肉体労働が日夜続けられていることや、ソフトウェア開発が 1960 年頃に考え出されたものから基本的に変わっておらず、こつこつと入力することで開発されるような手工業的な性格が強いスタイルが続いていることを指摘している。また、ソフトウェアの特徴として、物的生産物のように固定化されず、連続的に更新される変動性を持つことをあげており、それゆえソフトウェア産業にとって日々の生産と消費から得られる経験を集積する足場が定まらず、産業として発展していくうえでの難点であることを指摘している。

さらに、今井他は、このような日本のソフトウェア産業について、プログラミングがシステム・エンジニアリングより低位の機能であるとする偏見が定着してしまっていることを指摘している。例えば 1988 年当時、政府による高度情報処理技術者育成指針として、プロダクション・エンジニアと呼ばれる上級プログラマーを意味する区分[51]を作らざるを得

[51] 国家資格である情報処理技術者試験では、高度情報処理技術者試験の区分としてスペシャリストを設定することで技術者のクラス分けを行っている。プロダクション・エンジニアは、プログラマーへの指導的立場となるシステムエンジニアを想定しており、その後、

なくなったことをあげ、このことがプログラマーを高度に知的な生産に従事する人と認めてきたアメリカと異なり、「広義のソフトウェアの生産に従事する職業人を大事にしない日本の伝統文化を象徴するものである」[52]と述べている。

このほか、今井他は日本のソフトウェア産業の組織構造について、Cusumano（1991, 2004）が述べたようなソフトウェア・ファクトリーをあげ、そのような階層型の組織に対してクラフト型の組織を用いる場合には、かなりまとまった仕事を任せられる管理者の能力や技術者の能力をどう揃えるかという問題があると述べている。その上で、この管理者の能力の問題として技術力を持つ管理者を必要とするが、有能なプログラマーが良い管理者になるという保証もなく、その養成も困難であることを指摘している。また、ソフトウェアの生産の問題として、製造業のような小集団による助け合いとは異なり、能率の良い技術者が能率の悪い技術者をカバーしつつチームを組むということが難しく、単なる集団主義的な共同作業では効率を発揮できないと述べている。

このため、今井他は、ソフトウェアの生産が標準的に確立した仕事ではなく、進化的に展開する技術をこなすという性質を有しており、組織の規律よりは組織の意向を伝達し得るような組織風土が重要であると述べている。しかしながら、そのような組織風土は、組織デザインとして容易に作られるものではなく、その対処の仕方が経営戦略をも左右することとなるとも述べている。

以上のように、今井他の研究は、組織構造の点から日本のソフトウェア産業に従事する技術者、特にプログラマーといった技術者を軽視することを問題視しており、この指摘は本研究に近いものがある。このようなプログラム作成といった下流工程やそこに従事する技術者に対するアメリカと日本の考え方の違いは、現在もそう大きく変わっていない。

一方で、今井他は日本のソフトウェア・ファクトリーが抱える問題についても言及しているが、今井他が研究を行った 1980 年代当時は、当初の計画通りにソフトウェアを完成させるためにその開発を納期や費用という点でいかに効率的になるように管理する Waterfall Model と呼ばれる開発手法が主流として機能してきた。しかし、Waterfall Model に代わって 21 世紀のソフトウェア開発で主流となっている変化を前提とし、ソフトウェアの完成よりはユーザーの要望を満たすことを目的とした新しい開発手法はまだ登場していなかった[53]。そのため、今井他の研究も当時主流であった Waterfall Model を前提とし、その開発手法の特徴であるソフトウェアのマネジメントが中心となっており、特に管理者からの視点

アプリケーションエンジニアなどに分割、統合され、現在ではシステムアーキテクトなどに置き換えられている。

[52] 今井他（1989）p.61

[53] このような開発手法として Agile（アジャイル）がある。この Agile については第 5 章で詳しく述べるが、Waterfall Model に対する軽量な幾つかの開発手法の総称である。そのうち最も早いものが、1986 年に登場した Scrum と呼ばれる開発手法であり（平鍋・野中, 2013）、それ以外のほとんどの開発手法は 1990 年代に登場した。

に偏ってしまっている。

③　妹尾による旧来の手法とリーダーシップの観点による研究

妹尾（2001）は、ソフトウェア開発におけるプロジェクト・リーダーの支援に焦点をあて、その開発手法やリーダーの行為について論じている。

妹尾は、Waterfall Model のようなソフトウェア開発を管理、標準化する開発手法を中心に、Cusumano（1991）が取り上げた日本のソフトウェア・ファクトリーを例にあげ、生産性や品質、再利用率などの向上には貢献したものの、ソフトウェアの競争に繋がる「機能」の向上には結びついていないことを指摘している。

さらに、妹尾は、Waterfall Model 以外の手法を anti-Waterfall Model と位置付けてその比較を行っており、anti-Waterfall Model に準拠したソフトウェア開発プロジェクトの特徴として、「リーダーや開発者が突発的事項に対して即興的に対応するための自由度が高く保たれている」[54]と述べている。また、Waterfall Model に代表される工業製品の製品観と単純労働者という開発者観は、ソフトウェア開発の実態には適用できないことも指摘し、日本ではいまだ Waterfall Model が主流として機能してきた一方で、アメリカなどでその Waterfall Model が支配的地位を喪失している理由について、開発プロセスの移り変わりや開発手法の洗練ではなく、リーダーシップ・スタイルに求めている。

そのうえで、妹尾はこれまでの製品開発の研究を分析し、そこでは開発リードタイムを短縮するという目的のもと、開発プロセスをどのようにデザインするかという議論が進められてきた（Clark and Fujimoto, 1991）とし、Waterfall Model と Anti-Waterfall Model の関係について、「順次プロセス vs. 重複プロセス」の比較と、「初期計画の協調 vs. 試行錯誤的反復」の比較であるとしている。そして、そのような議論には、プロジェクトを「問題解決の束」と捉える観点や、プロセスモデルについて複数のモデル間の優劣を論ずる議論図式、そしてこの「問題解決の束」を上手にデザインすることでより大きな効果をもたらすという信念が共有されてきたと述べている。

これら議論のうえで、妹尾は、リーダーシップを円滑に発現させるためには、「絶えず変化する状況を把握し、偶発的に起こった出来事に即興的に対処するための工夫が必要」[55]であるとし、熟練リーダーの知識の蓄積であるナレッジマネジメントシステムの構築や、プロセス改善の試み、開発活動の予測や制御可能性を高める CMM 手法に対して、リーダーの仕事を一面的にしか捉えていないと批判的している。さらに、このような経験豊富な熟練者が持つ変化と異常に対する「知的熟練」（小池, 1977）の技能の本質について、過去の類似条件のような試行錯誤を必要としない定石の知識を蓄積したものではなく、相互作用のためにアクセス可能な範囲が広いことにあるとしている。

[54] 妹尾（2001）pp.73-74
[55] 妹尾（2001）p.74

以上のように、妹尾の研究は、ソフトウェア開発が Waterfall Model から anti-Waterfall Model へと開発プロセスが移り変わりつつあることを指摘している。ここで特に重要なことは、この 2 つの手法を製品開発のプロジェクトに照らし合わせ「順次プロセス vs. 重複プロセス」の比較と、「初期計画の協調 vs. 試行錯誤的反復」の比較として捉えており、このような比較の背景としてプロジェクトを「問題解決の束」と捉える観点があることを提示していることにある。妹尾はこの問題解決についてリーダーに視点をあてており、小池（1977）の「知的熟練」と結びつけている。

この問題解決とリーダーによる熟練の視点や、Waterfall Model によりソフトウェア開発者を知的な、創造的な能力が必要とされない単純労働者としてみなしていることへの指摘は、本研究を進める上で重要な示唆を与えている。特にこれまでの研究の多くは、ソフトウェア開発のマネジメントにのみ焦点をあてていることが多く、妹尾の研究はソフトウェア開発の現場において発生した問題に対する試行錯誤の取り組みや、さらにそういった問題解決に関わる熟練を結びつけたことに特徴的がある。

一方で、妹尾の研究はリーダーシップを中心に論じられている。プロジェクトを進める上で、リーダーの役割が重要であることに異論はないであろうが、ソフトウェア開発においてリーダーは必ずしも開発作業まで行うとは限らない。むしろ、ソフトウェア開発の進捗の把握や円滑なチーム運営など、管理者や責任者としての立場が重要視され、実際の開発作業よりもユーザー企業の担当者との折衝が中心となることが多い。そのため、本研究は妹尾の研究視点とは異なり、リーダーの役割についてではなく、実際の開発現場の技術者の分業に焦点をあて、ソフトウェアを作成するうえでの試行錯誤や問題解決といったプロセスに着目している。

④　峰滝によるソフトウェア開発のモジュール化の研究

峰滝（2004）は、日本のソフトウェア開発の生産工程や組織のモジュール化とアウトソーシングの関係を分析している。峰滝は、モジュール化について、ハードウェアであるコンピューターの生産性に劇的な上昇をもたらした[56]ものの、その成功モデルを他の産業に機械的にあてはめて考えることはできないと述べている。

そのうえで、峰滝はモジュール化が日本のソフトウェア産業の生産性向上に寄与していない理由として、ソフトウェア開発の各企業が担当する工程に、ゆるやかな分業関係がみられることを指摘している。また、日本ではソフトウェア開発のアウトソーシングを進める際、プログラム作成工程とテスト工程の分業が十分になされていないことが多く、さらに工程間の連結ルールが明確に安定しておらず、途中で微調整を行っていくことが実情であるとも述べている。そして、ソフトウェアの生産工程をモジュール化する場合、一定の

[56] このようなモジュール化の成功例は、シリコンバレー・モデルと称された（峰滝, 2004）。

連結ルールに基づいて、設計やプログラム作成、テストのそれぞれ工程が独立したモジュールになっていなければならないと述べている。

峰滝は、このようなモジュール間の調整をプロジェクトのマネジャーの役割に結び付けている。ここでのプロジェクト・マネジャーとは、妹尾（2001）が述べたリーダーとほぼ同義と考えられるが、峰滝はプロジェクト・マネジャーが個々の工程モジュール間の情報処理や伝達を媒介する役割を担わされていると述べているところに特徴がある。つまり、顧客のニーズに合わせながら、ソフトウェアの品質や納期を守るため各工程を調整していくことが、プロジェクト・マネジャーの役割であるとしている。さらに、そういった調整力として求められているものが、アウトソーシング先の品質や進捗管理のほか、スタッフに関する十分な知識や動員力、そして技術が把握できているスタッフがプロジェクトに参加することであると述べている。

以上のように、峰滝の研究は、日本のソフトウェア開発をアウトソーシングする際、モジュール化が生産性の大幅な向上に寄与しなかった理由として、工程間のモジュール化が不十分なことと、プロジェクト・マネジャーによるモジュール間の独立性の管理や調整する機能が十分になされていなかったことに原因を求めていたのである。ここで重要なことは、峰滝はモジュール化の議論としてソフトウェアの工程間がモジュールごとに独立していることを前提としているが、実際の工程間はゆるやかな分業関係がみられ、工程間の連結ルールが明確に安定しておらず、途中で微調整を行っていくということである。ソフトウェアを開発する際には、他の機能から干渉されないように、機能ごとにモジュールとして分離、独立化することで分業が行われているが、実際にはその工程を峰滝が述べるように完全に切り離すことは難しいのである。本研究は日本のソフトウェア産業が、その開発作業を製造業とみなして設計とプログラム作成の工程を分離することで、プログラム作成工程が備える知識労働としての側面も分離してしまっているために、質の高い革新的なプログラム開発を実現する上での困難に直面していることを明らかにしようとしており、この峰滝のモジュール化の議論と､その工程間が曖昧であるという指摘は重要な視点となる。

⑤ 高橋によるソフトウェアの国際競争力の欠如に関する研究

高橋（2010）は、日本のソフトウェア産業の国際競争力がないとされる点について、その要因分析を行っており、日本から世界をリードするようなソフトウェア企業が現れる可能性は皆無に近いと主張している

高橋は、日本のソフトウェア技術者の能力や意欲不足をあげ、創造的なアイデアや製品に欠けているとし、Cusumano（2004）も指摘したような日本の技術者の市場獲得意欲は海外と比較して差があることを述べている。また、日本のソフトウェア市場が外国企業にとって参入障壁が高く、閉鎖的である理由として、強い国内発注指向や日本語、日本特有のビジネス慣行をあげており、その閉鎖的な状態に守られてきたためソフトウェアの価値を

ユーザー企業に理解させる努力を怠り、その結果「国内のソフトウェア産業には独創性や価格を競うビジネス環境が育たなかった」[57]と述べている。

さらに、高橋は、国際競争力欠如の要因として、妹尾（2001）も指摘した、今までにない問題や新しい領域に取り組んでいくような創造的なソフトウェアの開発に適していないWaterfall Modelが日本では主流として機能しており、Agileのような要件定義や設計を開発の序盤で確定しない開発方式の導入が遅れていることを指摘している。その上で、日本のソフトウェア開発企業がこのWaterfall Modelを採用し続ける理由として、Waterfall Modelが日本のソフトウェア開発の多重下請け構造とその作業の流れに整合的である点や、設計とプログラミングの役割がはっきりと分かれていることで「ソフトウェアを設計図通りに作れなかったときは製造側の責任となるので、ソフトウェアを設計する側にとって都合がいい」[58]という、問題が発生した時の責任の所在がはっきりしている点をあげている。

高橋はWaterfall Modelに関するソフトウェア技術者の熟練度についても言及し、Waterfall Modelでは比較的単純な工程であれば熟練度の低い技術者でも開発を行うのに問題はないが、Agileでは設計とプログラム作成を繰り返すため、熟練した技術者が多数必要であるとしている。その上で、日本のソフトウェア開発は、海外へのアウトソーシングが増加しており、Waterfall Modelはそうしたプログラム作成などの一部工程のみを外国企業に委託することに適しているとも述べている。

以上のように、高橋の研究は、日本の国際競争力欠如について、技術者の能力や意欲の不足、日本市場の閉鎖性、そしてWaterfall Modelのような旧来の開発手法に原因があるとしている。この高橋が述べている問題は、Cusumano（1991, 2004）や今井他（1989）、妹尾（2001）なども述べていることであり、日本のソフトウェア産業が独創性や価格を競うビジネス環境を育ててこなかったという重要な部分を指摘しており、このことは本研究が対象としているソフトウェア開発を標準化した単純労働に置き換え、そこでの開発者の知的な、創造的な能力が必要とされる側面を軽視している問題にも繋がっている。

さらに、高橋はソフトウェア産業に独創性や価格を競うビジネス環境が育たなかったことを指摘しているが、このことは日本のソフトウェア産業が一品一品を作り上げる受注型のカスタムソフトウェア開発が主流であるため、発注側の企業にとっても価格を判断しづらく、適正な価格による競争ができなかった面が考えられる。

一方で、高橋はこの議論をカスタムソフトウェアだけでなく、市販のパッケージソフトウェアも含めて論じており、その違いを明確にできていない。企業固有のシステムを作成するカスタムソフトウェアと、複製を前提としたような市販のパッケージソフトウェアとではその目的も異なっており、異なった種類の製品やシステムに同じようなアプローチを利用することは誤りである（Cusumano, 2004）。

[57] 高橋（2010）p.159
[58] 高橋（2010）p.162

特にカスタムソフトウェアは企業固有のシステム開発であり、他の企業のシステムに転用しにくい。当然ながら海外には、海外現地に合わせたシステム開発の企業が存在し、それゆえ日本企業向けに開発したシステムは、基本的に海外へ輸出することは難しい。このことから、カスタムソフトウェアを高橋の述べるような対外輸出を目的とする国際競争力の対象として単純に捉えるべきではないと考えられる。

さらに、高橋は Waterfall Model に日本の国際競争力欠如の原因の一端を求め、Agile の優位性を述べており、特に日本企業が Waterfall Model を採用する原因として、下請け構造や分業の責任の観点から指摘したことが特徴である。たしかに、Agile は Waterfall Model よりも柔軟性があり、変化に強いとされ、Waterfall Model から Agile へと開発の主流は移りつつある。しかし、高橋の述べているような設計とプログラム作成を繰り返すような作業は決して Agile 特有のものとして限られているわけではなく、実際のソフトウェア開発の現場では Waterfall Model でも工程間でフィードバックを行っており、高橋はソフトウェア開発手法そのものに問題を絞りすぎている点で、現状把握とその分析が不十分である。

(3) 知識労働に関する研究

本節では、知識労働に関わる先行研究を確認する。

本研究は、ソフトウェア開発が単なる加工組み立てのような作業ではなく、作業者やプロジェクトメンバーが備える専門的･技術的知識や顧客や関連作業に対する幅広い知識に依存すると共に､プロジェクト進行における試行錯誤や創造的な問題解決活動に依存する､いわゆる「知識依存的」な活動であると捉えている。

さらに、本研究では、ソフトウェア開発をライン生産方式のような工場で行われるようなものとは異なるものであり、一品受注型の専門製品の開発として捉えている。こうした工場の大量生産や、専門製品の生産に関する研究は自動車産業を中心に多く行われてきている。

こうした観点から、ソフトウェアの開発に関わるそのような知識労働としての側面やそこで働く知識労働者の知識の伝達や協働について先行研究を検討するとともに、大量生産や専門製品の生産についての先行研究に関する整理を行う。

① 企業の内部資源に関する研究

1980 年代より､企業の内部資源の研究（Rumelt, 1984; Wernerfelt, 1984; Barney, 1986 など）が進み、外部資源との相互作業を通して企業の内部資源が強化されることが明らかとなっていった（相原, 2000）。

こうした企業内部の資源の中でも特に注目されているものが、企業が持つ知識といった情報資源であり、そこには企業内部の技術や組織の行動パターン、マニュアル、さらには従業員の個人的知識､文化､問題解決手法といった多くのものが含まれている（中川, 2011）。

企業における情報資源とは、マニュアルなどの文章として形式知化されているもののほか、個々の従業員の記憶など暗黙知的なものまで、企業内部に蓄積されている。しかし、暗黙知的なものであるため、こういった情報資源の価値は分かりづらく、企業の会計情報などにも表れてこない（中川, 2011）。

特に、この情報の中心にあるものが知識であり、ヒトやモノ、カネとは異なり、希少かつ模倣が困難なものとされる（伊丹, 1984; Barney, 2002）。また、知識は、単独では機能せず、企業はこの無数にある知識をまとめることで、その意思決定や行動を行っている。そのような知識は企業が独自に蓄積、構築してきたものであり、企業の行動や思考はこの知識によって規定されるといえる（中川, 2011）。

さらに、知識の連結構造は、企業内部に限られず、取引先や提携先企業と製品開発のような知的なリンゲージが形成されることもあり、そのような企業間関係が構築されると、技術や組織文化など企業の知識の結合関係ができあがる（中川, 2011）。

このような知識について、林（2008）は、科学的、公共的な知識がコード化や記述化されることで公共財として入手できる一方、業界固有の知識のようなものは科学的・公共的な知識より移転は困難であるが、サプライヤーなどを含む専門家によって移転されていくことを指摘している。そのうえで、企業固有の知識について、長い時間をかけて企業内に蓄積、構造化された暗黙知として企業のシステムに埋め込まれており、科学的・公共的な知識や業界固有の知識と比較して、容易には複製できず、特に個人が保存する知識はより移転の困難性が強いと述べている。

Langlois and Robertson（1995）は、企業を能力の束として捉えており、企業の境界に関する理論の分析対象は企業の能力や知識の問題が中心となるとしている。Langlois and Robertson が述べる企業の能力とは、組織が保有している生産やマーケティング、ファイナンスのほか、管理するうえでの知識やスキル、経験などの能力を意味し、暗黙知を含んでおり、企業の境界決定のために重要なものである（Penrose, 1959）。さらに、企業間において、文化や意思疎通、能力の非対称性により知識の伝達は困難であり、企業間で知識を移転するよりも企業内部で知識移転を行ったほうが効率的となる（Teece, 1986）。

こういった暗黙的知識を単純に保持するだけでは競争優位には繋がっていかず、組織として、生み出された知識を蓄積、活用していくような知識をマネジメントする活動が企業の競争優位に繋がっていくといえる（一條, 2003）。

特に本研究が対象とするソフトウェアは、目に見えないデータで作成されており、技術や開発プロセス、手法といったもののほか、システム化の対象となるユーザー企業のビジネスに関わる知識も必要とされ、そういった知識は個々の企業によりその内部に積み重ねられてきた。さらに、ハードウェアのような物質は劣化していくが、ソフトウェアはデータであるため劣化せず、むしろ品質は向上していく傾向にあり、特にソフトウェア技術は、本来累進的な性質を強く持っており、過去のアイデアや知識を再利用するケースが非常に

多く、そういった知識の蓄積により品質が向上していく（鈴木, 2009）[59]。

このように、企業の内部資源の一つとして知識が重要であることが示されるが、こうした企業固有の知識は暗黙知的なものとして蓄積されてきた。また、非定型的な作業、規格化されない情報や知識、作業者の行為や経験を離れてしまうことで容易に機能しないような知識の移転は、取引コストを著しく高めることになり、企業間における知識の移転は困難であることが指摘される。

ユーザー企業は、ソフトウェアを開発するためにソフトウェア開発の専門企業である IT ベンダーを利用するが、その際、ユーザー企業と IT ベンダーの企業間で知識を含む情報のやり取りが発生する。ソフトウェア開発のために、IT ベンダーはユーザー企業固有の知識を必要とするが、そうした企業固有の知識の移転は簡単ではなく、それゆえソフトウェア開発は困難に陥りやすい。さらに、ユーザー企業から IT ベンダー内に知識が移転できたとしても、ソフトウェア開発は複数人のチームで作り上げる協働作業であるため、IT ベンダー内部でも知識移転が生ずるのである。

このように、ソフトウェアの開発は、業務や技術を含む多くの知識の蓄積と移転が発生し、それゆえソフトウェア開発の分業に大きな問題を与えているのである。

② 大量生産と専門化に関する研究

前節の日本のソフトウェア産業の先行研究において、ソフトウェア開発の製造工場化として、ソフトウェア・ファクトリーが導入されたことが確認された。このソフトウェア・ファクトリーは、Taylor（1911）の科学的管理法のような頭脳労働と手労働を分離する方法であり、それによりソフトウェアを大量生産することで、増え続ける需要に応えようとしたものである。こうした大量生産について、生産管理の分野で Taylor（1911）の科学的管理法やフォードの T モデルによる大量生産、さらにトヨタのリーン生産方式などを中心に多くの研究が行われている。

Taylor の科学的管理法では、熟練工の作業を細分化、標準化、単純化することで、標準的な作業時間や作業方法を設定し、熟練工以外の労働者でも標準レベルの作業ができるようにした。さらに、熟練工の知識や現場での工夫といった頭脳労働を取り除き、管理者側に集中させることで、計画と執行の分離を図った。

また、フォードの大量生産システムは、自動車のような数千点から数万点の部品を持つ製品を組み立てるために、さまざまな取り組みを行った。例えば、部品の加工機械を導入することで部品の標準化や共通化を行い、誰でも部品の生産ができるように熟練作業者を

---

[59] ただし、1960 年代に主流であった COBOL 言語のように、あまりに古いプログラムや言語を使用している場合、対応できる技術者も高齢化しており、少なくなっている。そのため、過去の技術の利用は、再利用や保守ができないなどの問題が発生し、かえって品質が悪化する場合もある。

不要にするとともに、生産性の向上を図った。さらに、そういった機械をベルトコンベヤー式の組立ライン化することで、作業工程を単純な工程に細分化し、作業の専門化を図った（豊田, 2012; 伊達, 2013)。

こうした大量生産への取り組みは、一方で市場の変化への対応が難しく、過剰な分業や機械化が問題になり、前述の社会・技術システム論（Trist and Bamforth, 1951; 上林, 2001; 森田, 2010）や労働の人間化（小林, 2008; 菊野, 2010）などにより批判されている。競争環境が大きく変化してスピードが速まり、多様な問題が発生するような状況（守島, 2016）や新しく工夫していくような製品に対して、このような大量生産の方法は適用しにくい。さらに、過剰な分業により、作業を細かく専門化しすぎると、生産システムが硬直化し、そのための調整コストやムダが発生してしまう可能性もある（藤本, 2003)。

製造業でも、このようなライン生産方式による大量生産ではなく町工場などの受注型の生産方式があるように、本研究が対象とするカスタムソフトウェアも受託による開発、つまり一品受注生産型であり、こうした大量生産とは本来異なるものである。

受注型の生産方式は、専用化された特注品であり、一般化されたものとは異なる。また、そういった特注品は、大量生産方式のようなマニュアル化することが難しく、部品をモジュール化して単純に組み合わせるのではなく、そうした部品同士を擦り合わせながら作り上げていく。モジュール化や擦り合わせといったアーキテクチャーについては次章で詳しく述べるが、こうした擦り合わせによる製品は、構造的に複雑な相互依存関係にあるため、設計変更やソフト検証といった問題解決サイクルを何度も繰り返す必要がある（藤本, 2007)。さらに、部門間同士や企業同士による協働で開発することが必要となり、その設計などを綿密に調整することが必要となるのである。

③　知識労働者の協働と専門性に関する研究

受注生産型の製品が協働で開発されていることを述べたが、ソフトウェアの開発もかつての手工業的な時代とは異なり、その規模は現在では非常に大きくなっている。もはやソフトウェアは個人で開発することは困難であり、そこではソフトウェア技術者による協働の作業が必要となる。

このような協働作業について、Wakefield（1833）は同じ仕事を協力し合う単純な協働と、異なる仕事を協力し合う複雑な協働とに区別している。そのうえで、単純な協働の特徴として、同じ場所で同じ仕事に従事することで作業時間の短縮や身体的適応力の克服などが考えられ、それにより余剰生産物が得られるようになるとしている（村田, 2012)。

また、仕事の区別について、Smith（1776）がピンの製造から、作業の適切な分割と結合として労働者の学習効果による技術の向上や作業の変更の際の無駄の減少、そして機械の発明や改良により、熟練形成の効率化や知識の専門化といった分業の生産性の向上を説明した（若田部, 1991; 村田, 2012)。つまり、作業を標準化した単純労働にすることで、労働

者はその作業に専念でき、そのことにより作業の無駄の減少や機械の発明改良が促進されるという過程が強調されている。

Alvesson（2004）は、知識労働者について、典型的なプロフェッショナルに必要とされる倫理や教育、資格要件といったものが重視されていないことを指摘している。

知識労働者の働き方について、Davenport（2005）は協働の度合いと業務の複雑さの程度によって大きく異なると述べている。特に業務が複雑な場合は高度な専門能力が必要であり、知識労働者には高度な思考力を発揮する仕事もあれば、比較的定型的な仕事も存在する（Davenport, 2005）。

Babbage（1832）は、個人間の能力の違いを取り上げ、労働者の能力に応じた最適な配置について指摘した。それによると、生産工程ごとに必要となる技術力の違いが存在し、そのため高い技術を持ち、高い賃金が必要となる技術者には、簡単な仕事よりも相応の高度な仕事に専念させることで生産費の無駄を省くことができるとしている。Babbage のこの理論は、「バベッジの定理」と呼ばれており、企業内の分業において、知的労働の分化が存在することが示された（村田, 2012）。

ソフトウェア開発も仕事の内容によって、難易度や複雑性がかなり異なっており（三輪, 2014）、ソフトウェアをモジュール化することで作業を分割してきたが、その作業の専門性はモジュールごとに異なっており、そこでの知識も分業されてしまう。ソフトウェアは担当者の保有する様々な知識に依存する面を備えており、こうした大規模、複雑化していくソフトウェアに対応するためには、単なる作業従事者ではなく、専門性に富んだ知識労働者が必要となる。さらに、こうした知識労働者の仕事は、その目的や内容が所与されるマニュアル・ワーカーとは異なり、提案力や問題解決力を必要とし、専門的な知識だけでなく幅広く文脈的な知識が必要となる（三輪, 2014）。

一方で、発生した問題の安定性や変異性に応じて、その問題解決に必要となる行為も異なる。特に生じた問題が変異性に富む場合は、多様な例外を処理する必要がある。例外の処理には十分な知識が必要であるが、知識が不足する場合には、個人の直観や経験、推測といったものに依存した問題解決活動が必要となる（Perrow, 1967; 加護野, 1980）。

このような、変異性に富んだ異常事態への対応能力について、小池（1977）はふだんと違った作業を処理する能力やスキル、つまり問題や変化といった異常事態に対応する能力を「知的熟練」と呼び、その「知的熟練」の程度が人材の価値を決定するとしている。

ソフトウェアの開発は、ユーザー企業の要望を製品化するために、試行錯誤を繰り返しながらその要望を取捨選択して具体化する必要があり（大日方, 1971）、デザイン思考に基づく製造業の製品設計に近いものといえ（妹尾, 2001; Cusumano, 2004）、そこに現れてくる不確実で複雑な問題に対して、発見と解決のサイクルを繰り返していく高度な知識創造活動としての側面を有するのである。

(4) 問題設定・残された分析課題

ここまでの先行研究の検討より明らかになったことは、まずソフトウェア産業の分業構造を対象とした研究は、その表面的な特性や問題を述べるに留まっていることが多く、ソフトウェア産業の知識労働という側面にまで深く踏み込んだ研究はほとんどないということである。また、ソフトウェアに関する研究の多くも、その技術的性格から、ソフトウェアの利用方法やその導入方法について、ツールや手法の開発に傾斜した技術や工学の分野が中心であり、人間的な部分や組織的な部分など社会学や経営学に関する研究は非常に少ない状況である。

そのような中、数少ない日本のソフトウェア産業に対する研究では、日本のソフトウェア産業の多くの問題を浮き彫りにしており、それぞれの研究者が、組織風土やリーダーシップ、モジュールなどの視点からその問題を指摘している。

なかでも、Waterfall Model と呼ばれる旧来のソフトウェア開発の管理を主体とした手法に問題があることが先行研究において共有されており、特に Cusumano（1991, 2004）がその研究対象として取り上げた日本のソフトウェア・ファクトリーがその中心に存在する。確かにこのソフトウェア開発の製造工場化は、Cusumano が指摘するように標準化された設計パターンに基づいて開発されるようなソフトウェアには有効であった。しかし、それゆえ、今までにない問題や新しい領域に取り組んでいくような革新的なソフトウェアの開発には向かず（妹尾, 2001）、企業のソフトウェアの価値を軽視するような戦略の欠如（高橋, 2010）を引き起こしたことが強調される。

また、今井他（1989）や峰滝（2004）、高橋（2010）は、ソフトウェア開発の上流工程と下流工程を分離し、下流工程をアウトソーシングしてしまうような方法を批判的に捉えているところに共通点がある。そして、ソフトウェア開発は、工場における製造のようなものではなく、多様かつ高度なソフトウェア生産のための知的な、創造的な能力が必要とされる作業であるということが、Cusumano（1991, 2004）や今井他（1989）、妹尾（2001）の共通する意見でもある。

一方で、Cusumano（1991）や今井他（1989）は、その研究時期が古いこともあり、妹尾（2001）や高橋（2010）などが提示した Agile のような変化を前提としたソフトウェア開発の分析にまで踏み込めていない。また、これら先行研究では、ソフトウェア開発を管理することで問題の解決を図ろうとするところに重点が置かれており、管理される側、つまりソフトウェアを作成している現場の視点があまり反映されていない。実際のソフトウェア開発の現場では、必ずしも手順通りに秩序だった開発ができているわけではない。むしろソフトウェアの開発は、要件が定まらず混沌とした中で行われているのである。

知識マネジメントの先行研究から指摘されるように、こういったソフトウェアを開発するには、その目的や内容が所与されるマニュアル・ワーカーとは異なり、提案力や問題解決力を必要とし、専門的な知識だけでなく幅広く文脈的な知識が必要となる。ソフトウェ

アの開発は連続性や漸次性のもと、変更や改善のために試行錯誤を繰り返しながら作られる製品設計のような知識労働としての側面があり、これまでの日本のソフトウェア産業の研究では、そういった事実への指摘が不十分といえる。そして、そのようなさまざまな障壁を乗り越えていく部分に、革新の源泉や作業者にとっての仕事の魅力が存在し、今日におけるソフトウェア開発の付加価値の源泉が存在するといえる。

ここまでの先行研究の検討を踏まえ、本研究では、ソフトウェアのプログラム作成工程を定型的な組立作業のように捉えて標準化した単純労働に置き換えることで、下請け企業や海外へアウトソーシングしてしまうことがソフトウェア開発にどのような影響を及ぼしているのか、そして、ソフトウェア開発においてそこで生じる問題の発見や解決といった試行錯誤を繰り返しながら顧客にとってより価値のあるものを作り出す創作活動がどのように行われているのか、という問いのもと、次のような手順で論じていく。

これまでのソフトウェア開発がどのように行われてきたのか、また今日どのように行われているのか、Waterfall Model と Agile と呼ばれる手法を中心に、その特徴と限界がいかなるところに見られるのかについて検討する。また、同産業に特徴的に見られる分業構造を踏まえ、かつて日本のソフトウェア開発で中心的存在を果たしたソフトウェア・ファクトリーを取り上げ、その貢献と限界について明らかにする。そして、これら議論を踏まえ、事例の検討より、開発の現場で行われた作業プロセス間の連携や開発担当者の知的活動とプロジェクトの成否との関係に関する分析を中心として、ソフトウェア開発プロジェクトの成否を左右する諸要因を明らかにする。また、ソフトウェア開発プロジェクトの成否に影響を及ぼす諸要因が、なぜ、またどのようにそうした開発プロジェクトの有効性の発揮の有無と関係しているのか、ソフトウェア開発プロセスで見られた試行錯誤を繰り返す「創造的な問題解決過程」と、それに携わる技術者の「知識マネジメント」の視点から改めて検討する。

次章では、まずソフトウェアといった製品を作り上げていく上で、どのように作業を分担し、連携を図っていくのか、そしてどのような関係性を持たせるのか、部品の構成や相互関係のあり方を決める設計思想について検討する。

第4章 ソフトウェアの製品設計思想

本章では、ソフトウェアのような製品がどのようなつくりになっているのか、製品設計思想としてのアーキテクチャーの視点から検討する。

ソフトウェアは、多くのソフトウェア部品が寄せ集まることで、一つの製品となっている。このような製品をつくるとき、外見や機能などをどのような構成にするのか、そしてそれぞれの部品にどのような関係性を持たせるのかを決めなければならない。このような部品の構成や相互関係のあり方を決めるのがアーキテクチャーである（Simon, 1957; Ulrich, 1995; 中川, 2011）[60]。

とりわけハードウェアは、多くの機能を組み合わせて設計、開発していくモジュラー型アーキテクチャーの開発の事例として取り上げられることが多い。一方で、本研究が対象とするカスタムソフトウェアはモジュラー型の開発スタイルを持つが、モジュール同士の擦り合わせや調整を必要とし、インテグラル（擦り合わせ）型のアーキテクチャーによる設計、プログラム作成の側面を持っていると考えられる。こうした観点から、ソフトウェアを開発する際に設計とプログラム作成間にどのような関係が生ずるのかという本研究を進めていく上での議論の下地となりうる、モジュラー型、インテグラル型の 2 つのアーキテクチャーの特性について、分析を進めていく。

ソフトウェアに限らず、複数のサブシステムで構成された製品は、このアーキテクチャーのでき方次第でうまく動作するかが決定されるため、非常に重要なものとなっている。また、そのような複数のサブシステムから構成される製品は、組織内外のさまざまなステークホルダーが関係している。ソフトウェアを含む製品を作り上げる上でどのように組織内外の役割を分担し、連携を図っていくのか、そうした協業や分業の境界デザインの決定にはこのアーキテクチャーが関わってくるのである。

(1) アーキテクチャーの分類

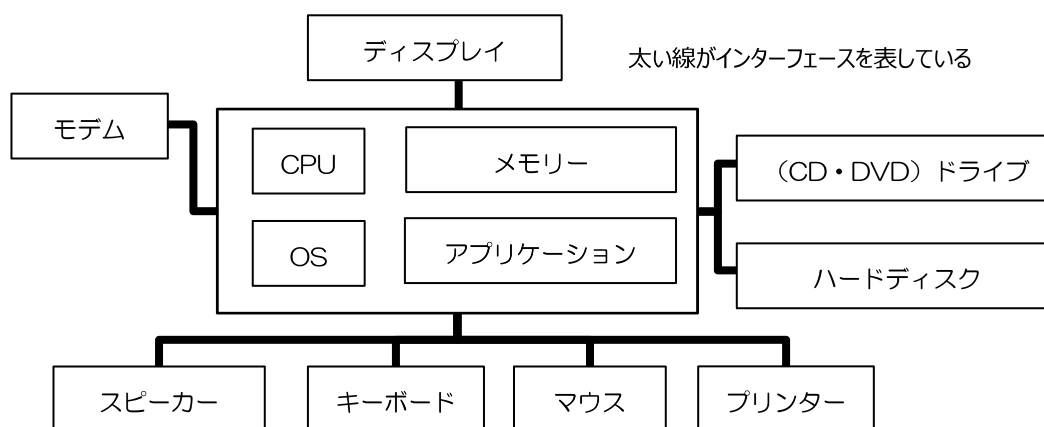
① 製品・工程アーキテクチャー

一般に、アーキテクチャーは、製品アーキテクチャーと工程アーキテクチャーに分類できる。このうち、製品アーキテクチャーとはシステム全体に関する知識といった具体的な設計思想を指す。アーキテクチャーにより、コンピューターやソフトウェアなど、複雑なシステムで構成された製品を複数の機能要素や部品など単純な下位システムへ分割することで、そのシステムの複雑性を軽減するのである。特に製品の設計において、製品をどの

[60] ソフトウェアに関連する技術的なアーキテクチャーとして、コンピュータ・ハードウェアの基本構造を示すコンピューター・アーキテクチャーやネットワークの論理構造などを示すネットワーク・アーキテクチャーなどが存在するが、本研究では、Ulrich（1995）らが説明する製品の機能や開発工程の設計思想を表すアーキテクチャーを対象とする。

ような構成部品や工程に分割し、そのインターフェース[61]をいかに設定、調整するかが極めて重要となる。製品のサブシステム同士は、このインターフェースのもとで統合、協調して動作することで、製品システムとしてまとまった機能となる（藤本・武石・青島編, 2001; 柴田, 2003; 藤田, 2013）。

このようなインターフェースを通じて構成される製品の代表例としてパーソナルコンピューターがあげられる。パーソナルコンピューターはマザーボードや CPU、キーボード、ディスプレイなど多くのサブシステムによって構成されている。そして、それらは公開、共有されたインターフェースによって繋げられ、一つのパソコンとして機能している（**図 4-1**）。



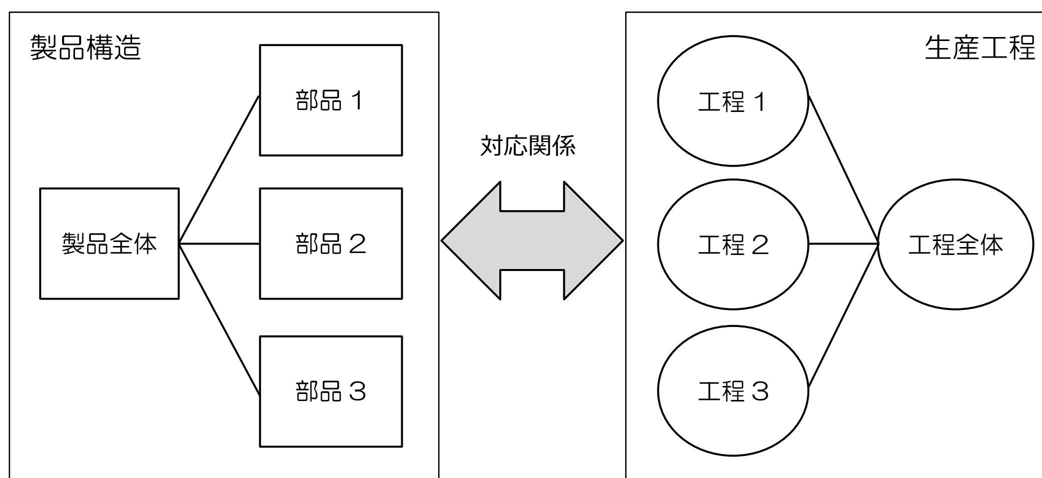
出所：藤本・武石・青島編（2001）、柴田（2003）、藤田（2013）をもとに筆者作成

図 4-1 パソコンの構成図とインターフェース

一方、工程アーキテクチャーとは、製品の生産における工程の編成、つまり工程設計の思想を指し、「製品構造と工程との対応関係、および工程間の連結関係によって構成」[62]されている（**図 4-2**）。

工程アーキテクチャーは、この部品構造と工程の対応関係が複雑になると、組立精度や作業条件などを調整する必要が出てくる。設計された製品や部品に対し、品質や納期、コストを目標通りに達成するため、生産設備のほか、作業者、作業方法、治工具といったものをどのような工程の流れ、つまりプロセスやレイアウトで繋げていくのかを考えなければならないのである（藤本, 2002; 目代, 2012）。

[61] 部品やプログラムなどが接続、接触する箇所や、情報やデータなどをやりとりするための手順や形式、規約を定めたもの（IT 用語辞典 e-Words）。
[62] 目代（2012）p.641



出所：目代（2012）p.641 をもとに筆者作成

図 4-2 工程アーキテクチャーにおける連結関係と対応関係

この製品アーキテクチャーと工程アーキテクチャーは表裏一体であり（Abernathy, 1978; 藤本, 2002）、本研究でも、製品アーキテクチャーと工程アーキテクチャーを明確には分離せず、製品・工程アーキテクチャーとして扱っていく。

製品のデジタル化が進展するにつれ、その製品は複数のサブシステムで構成される傾向が強まっており、製品の設計思想であるアーキテクチャーが極めて重要になってきている。さらに、こういった複数のサブシステムで構成される製品は、取引相手など多くのステークホルダーが関係しており、どのように業務や役割を分担していくか、その調整や連携の仕組みをシステムとしてうまく構築することが必要となる（柴田, 2003）。

このアーキテクチャーには、代表的な分類方法として、機能にその分割の焦点をあてた、組み合わせによるモジュラー型と、擦り合わせによるインテグラル型が存在する。また、複数企業間の提携に分割の焦点をあて、社会全体にインターフェースの設計情報を公開するオープン型と、関連する企業にのみインターフェースの設計情報を公開するクローズ型も存在する（Baldwin and Clark, 2000; 藤本・武石・青島編, 2001）。

企業は、製品・工程アーキテクチャーにより、コンポーネント間の相互関係を定めており、それに基づき組織を設計しているため、組織の知識体系や分業方法、設計などもそれに依存する。さらに、そこに子会社や下請け企業、同業他社など、外部企業との関係も関わってくる。そのため、製品・工程アーキテクチャーの変更は、企業にとって全面的な組織変更に繋がり、困難を伴うのである（中川, 2011）。

② モジュール化

ソフトウェアを開発する際、他の機能や工程に干渉されないようにモジュール化が進められているが（峰滝, 2004）、こうしたモジュール化とは、部品と部品の間にあるインター

フェースが 1 対 1 のように単純化、標準化されており、非常に独立性が高く、そういった複数の部品の単純な組み合わせによって商品の機能を実現することである（Ulrich, 1995; 藤本, 2001a, 2001b; 藤本・武石・青島編, 2001; 浜屋, 2004 など)。また、モジュール化することで、そのモジュールの交換することができ、顧客のニーズに合わせた機能の拡張が可能となる[63]。

ただし、モジュール化したとしても、そうしたモジュールの調達のほか、求める機能を実現するための適切な組合せに関する知識が必要となり（延岡, 2006)、すべてのニーズに応えた無数の製品を作成することは難しい。こうしたモジュール間は、ルール化されたインターフェースにより相互作用が抑えられ、環境の変化にも対応しやすく、不確実性への対応にも重要な役割を果たしている（Langlois and Robertson, 1995)。

藤本・延岡（2004）は、この環境の不確実性について、「選択すべき事業、製品、技術の範囲が広く、しかもどれを選択するのが良いのかわかりにくい状況」[64]と定義している。そして、モジュラー型製品の特徴として、取捨選択や組み合わせ方に関する不確実性の高さがあり、それゆえその選択能力が重要であるとしている。

ソフトウェア開発の不確実性の理由の一つとして、ソフトウェアを設計する際に、手法が確立して信頼性が証明されている枯れた技術を使わず、あえて新しい手法や新しいモジュールなどのバージョンを採用することがあげられる（中尾, 2009)。ソフトウェア産業を含む IT の分野では、技術や機器などの発達が非常に早く、同じような設計図で何十年間も生産し続けることは安全性や安定性の面でメリットがあっても、開発効率や計算速度、顧客の要望に沿ったソフトウェアの実現などで劣ってしまう。そのため、ソフトウェア開発者が、新しい手法やバージョンを採用しようとすると、そのことでこれまで経験しなかったような問題や失敗に直面するようになる。

ここで注意すべき点は、モジュール化は、設計に関する基礎的なパラメーターが必要であり、つまり設計に存在する知識に基づいていることである。そのため、例えば半導体の

---

[63] モジュールとほぼ同じ意味を持つ言葉として、コンポーネントが存在する。この 2 つの違いとして、コンポーネントとは、何らかの機能をもった部品を指すが、それ単体では利用することができず、他の部品と組み合わせたりすることで機能を実現するものとして考えられる（IT 用語辞典 e-Words)。

一方、モジュールはシステムを構成する要素であり、本文でも述べたように規格化、標準化されており、複数の部品を組み合わせたものとなる（IT 用語辞典 e-Words)。しかしながら、モジュールもコンポーネントも、企業やプロジェクト、産業ごとに捉え方や使い方に違いがあり、研究者の間でも明確な共通の定義が存在せず、混同されていることが多い。

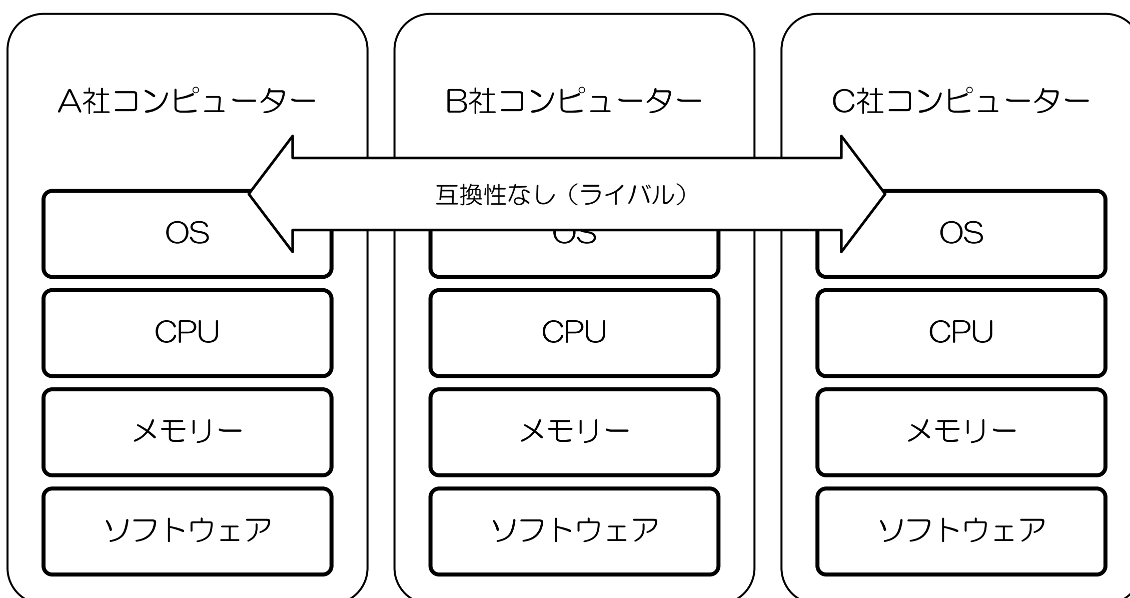
中川（2011）によると、部品、サブシステム、コンポーネント、ブロックなど研究者によって呼称はさまざまであり、中川はコンポーネントとして統一している。ソフトウェアでは、コンポーネントが使われることが多く、複数のモジュールが組み合わさって一つのコンポーネントを構成するとされる。

[64] 藤本・延岡（2004）p.18

モジュール化を行うには、チップの設計と実際の製造のオペレーションに関する非常に深くて詳細な知識が必要となり（Baldwin and Clark, 2000）、モジュール化による分割は容易ではないのである。

このような製品アーキテクチャーによるモジュール化の成功例として、IBM の商用コンピューターである SYSTEM/360 がある。本研究が対象とするソフトウェアが産業として成り立ったのは 1970 年前後であるが、それより前のコンピューターが商業化された 1950～1960 年代においては、まだハードウェアのアーキテクチャーが統一されておらず、コンピューターはモデルごとに異なった仕様で構成されていた。さらに、ハードウェアとソフトウェアの間にも互換性が無いため、コンピューターのモデルが変わる度にソフトウェアを作り直す必要があった（杉山, 2008）。

さらに、IBM の SYSTEM/360 が登場する以前のコンピューターは、企業別に部品間の互換性がない異なったインターフェースを採用していたため、他の企業のコンピューターと接続することが難しく、そのためにハードウェアの OS や CPU、メモリー、そしてソフトウェアまですべてを自社で提供する垂直統合であった（図 4-3）（図 4-4）。

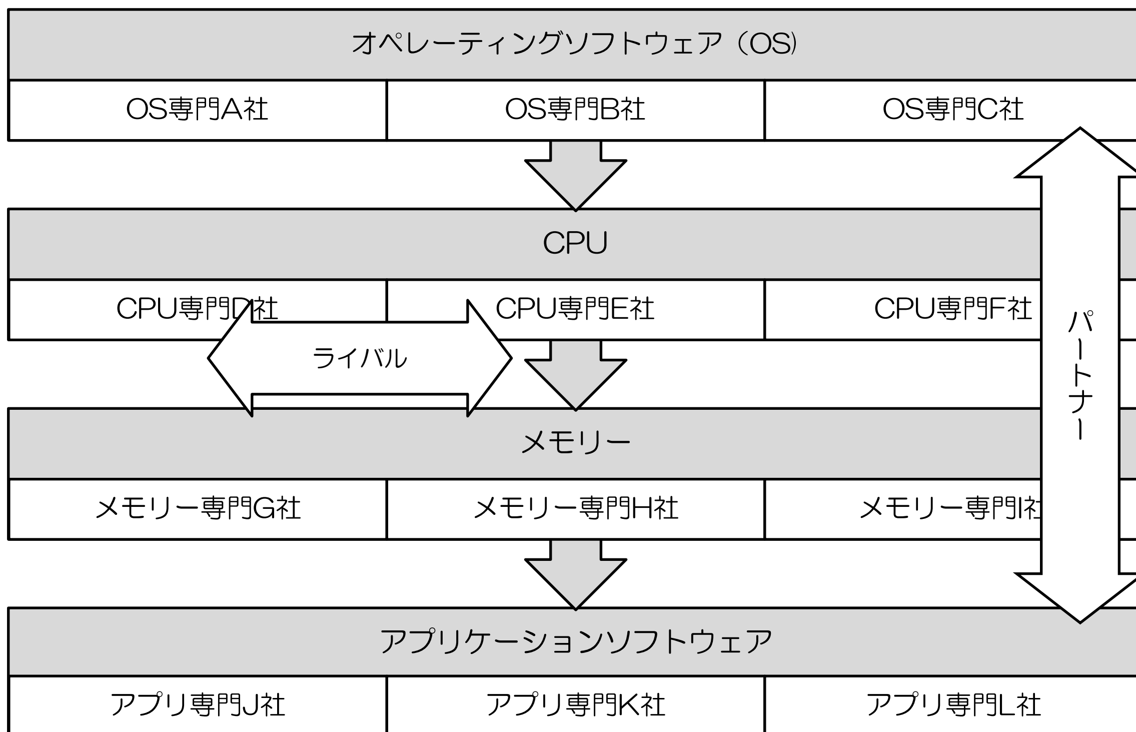


出所：杉山（2009）をもとに筆者作成

図 4-3 コンピューターの垂直統合

この SYSTEM/360 のアーキテクチャーは、複雑化していたコンピューターを特定の機能が発揮できるコンポーネントに分割しており、ハードウェアがモジュール化する経緯となった（峰滝, 2004; 野田, 2006; 杉山, 2009; 中川, 2011）[65]。

[65] ただし、モジュール化したとはいえ、SYSTEM/360 により IBM1 社がそのインターフェ



出所：杉山（2009）をもとに筆者作成

図 4-4 コンピューターの水平分業

一方で、このようなモジュール化は、製品性能と製品コストの間のトレードオフがあり、標準的なインターフェースを採用するため、個別にカスタマイズや改善することができず、必ずしも最適な設計とはならない可能性がある（青島, 2009）。

例えば、モジュール化は、製品開発の効率性と柔軟性の代償として、短期的に製品性能の低下を招くこともある。例えば、生産工程のモジュール化が十分に整備されておらず、アウトソーシングが効率的に行われていないような場合には、生産性に対して負の影響を与えるとされる（峰滝, 2004）。

先行研究の検討で言及されたように、峰滝（2004）はソフトウェア開発のモジュール化とアウトソーシングの関係を分析し、一定のインターフェースによる連結ルールにより、ソフトウェア開発の工程が設計、プログラム作成、テストのそれぞれでモジュールとして

ース設計や機能設計などを決定する独占した状態となっており、高度にモジュラー型であるだけでなく、後述するクローズ型アーキテクチャーの製品でもあった（藤本, 2002; 齊藤, 2014）。

齊藤（2014）は、IBM の SYSTEM/360 について、本文でも説明したモジュラー型かつクローズ型のアーキテクチャーに分類される場合のほか、ソースコードや仕様、構造といったものが公開されていない IBM 社により制限されたシステムなため、インテグラル型アーキテクチャーに分類される場合があることを指摘している。また、論者によってそうしたアーキテクチャーの区別、範囲も異なっていることを指摘している。

独立する必要性を指摘している。しかし、現実にはプログラム作成とテストの分業が十分になされておらず、日本のソフトウェア開発は、各企業が担当する工程間は、ゆるやかな分業関係がみられることも指摘している。

このようなモジュール化について、延岡（2006）や藤本・朴（2015）は、顧客ニーズの視点から差別化の重要性を論じている。延岡（2006）は、コモディティ化による製品のモジュール化の進展が差別化を困難にし、過当競争に繋がると考えられることに対し、顧客ニーズの観点から意味的価値による差別化が重要であると述べている。さらに、延岡は差別化ができたとしても、顧客がそれに見合った対価を支払ってくれるとは限らず、機能的な価値だけでは顧客ニーズは頭打ちしてしまうことも指摘している。

また、藤本・朴（2015）も、デジタル化の進展によって、すべての付加価値がソフトウェアを媒介に半導体のチップの中に埋め込まれるようなものとなり、そうした状況では単なる組み立てでは付加価値がほとんど生まれなくなることを指摘し、Apple や Google のような顧客のニーズを解決してくれるソリューションを提供するビジネスを例にあげている。

モジュール化を進める際には、どのように機能や工程をモジュール化し、それを調整し、擦り合わせていくことが重要であり、単純なモジュール化や組み合わせだけでは価値を生むとは限らないのである。

③　擦り合わせ

次に、組み合わせによるモジュラー型アーキテクチャーと対比される擦り合わせによるインテグラル型アーキテクチャーの視点から検討し、それぞれのアーキテクチャーがいかなる仕組みで、製品開発にいかなる影響をもたらしているのかを論じていく（表 4-1）。

モジュラー型アーキテクチャーでは、コンポーネント間の複雑な相互関係を分解してモジュール化することで、コンポーネントが独立し、総合に調整を行う必要がなくなり、分業が行いやすくなる（図 4-5）。

ここでコンポーネント間の依存関係が残ってしまうと、モジュール化の効果も低下してしまう。生産プロセスや技術の相互依存が低いようなモジュラー型アーキテクチャーでは、それによってできあがる製品もモジュラー型の部材となる（中川, 2011）。そのため、綺麗に独立したコンポーネントに分割する必要があり、設計の最初から相互依存が生じないよう製品アーキテクチャーを取り入れていかなければならない。さらに、個別のコンポーネントごとに標準化することで規模の経済が働き、コンポーネントの生産コストが低下し、コンポーネント間の調整がなくなることで、部品企業と完成品企業は別々に開発を行うことが可能となる。

他方、インテグラル型アーキテクチャーでは一つのモジュールが多くの機能を担っており、コンポーネント間を最適化することを必要とするが、コンポーネントの分割は必要ないためモジュール化に伴う困難さは少なく、生産コストを下げることも可能となる（Abernathy,

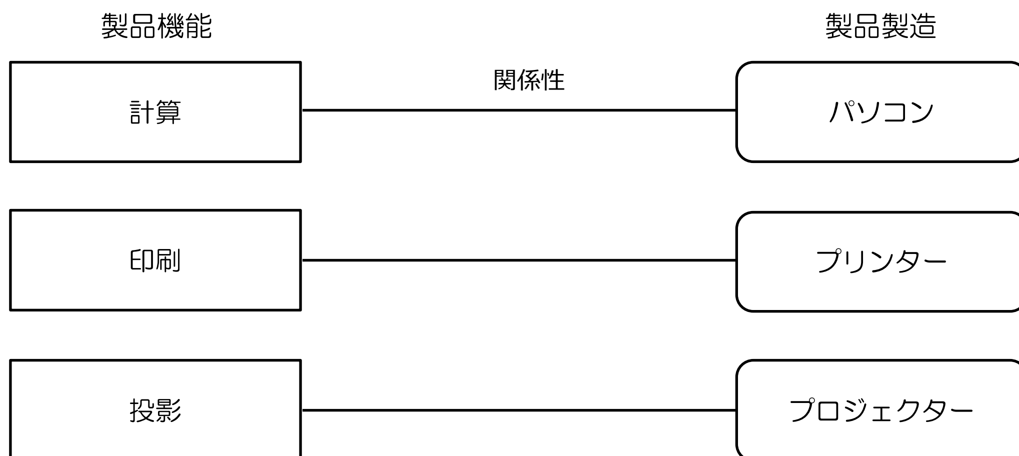
1978; Langlois and Robertson, 1995; 秋池, 2012)。しかし、コンポーネントが分割されないことにより、機能と部品が 1 対 1 ではなく多対多の関係となってしまう（**図 4-6**)。

表 4-1 モジュラー型とインテグラル型のアーキテクチャー比較

	モジュラー型アーキテクチャー	**インテグラル型アーキテクチャー**
コンポーネント	1：1（分離・独立） （卓越したコンポーネントと その組み合わせの知識が必要）	多：多（提携） （連携、調整のための 知識や構造が必要）
品質	標準化・規格化 されている	最適化・改善（カスタマイズ） されている
調整力・コスト	少ない （単純化するため調整が楽）	多い （複雑なままのため、 後で調整が難しい）
開発形態	個人・分離	チームワーク・結合
外注／内製 の傾向	外注 （分離されているため）	内製 （チームワークが必要なため）
サプライヤー との関係	それぞれ独立して開発	共同で開発
必要な能力	不確実性への対応	問題解決
知識	モジュール化のための 設計の深い知識が必要	システム全体の知識が必要
垂直統合度	低い	高い

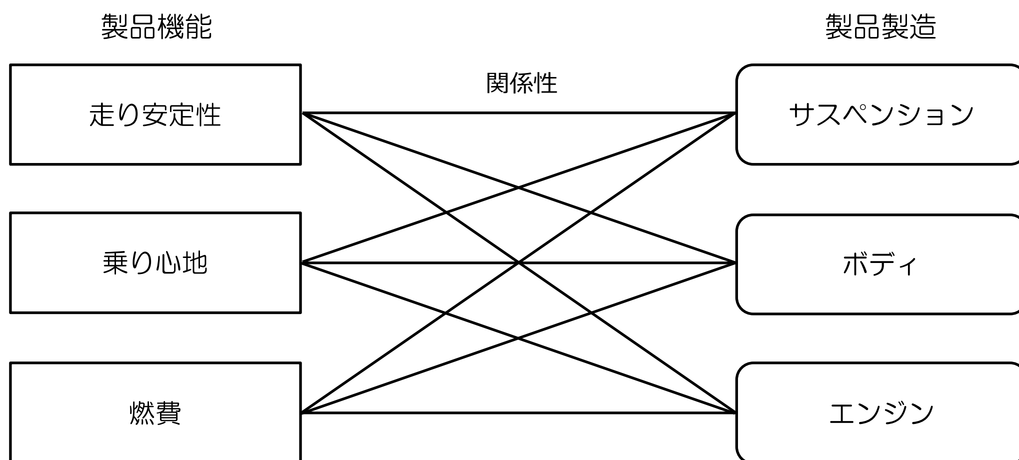
出所：藤本（2001a, 2001b, 2004, 2007），藤本・武石・青島編（2001），浜屋（2004），青島（2009），中川（2011）をもとに作成

そのため、インテグラル型アーキテクチャーではコンポーネント間の関係が複雑な状態のまま開発することになり、後の段階でコンポーネント間の調整が必要となる。つまり、企業内部のコンポーネント部門と部品企業であるサプライヤーとで共同で開発することが必要となり、それゆえ設計の条件を綿密に調整する能力が必要となる（藤本, 2007)。このように企業内部ですべての開発活動を行うか、企業間で開発活動と知識を共有化していくか、その境界デザインが問題となる。



出所：藤本（2004）をもとに筆者作成

図 4-5 モジュラー型アーキテクチャーの関係性の例（パソコンのシステム）



出所：藤本（2004）をもとに筆者作成

図 4-6 インテグラル型アーキテクチャーの関係性の例（自動車）

さらに、インテグラル型アーキテクチャーでは、部門間の協調的な知識創造活動や、企業間の連携により、システム全体に関する知識を醸成する必要がある（中川, 2011）。特にソフトウェアは､モジュール化されているとはいえ､その機能を削ることは容易ではない。多くのソフトウェアでは、機能よりもシステムの信頼性が重要視され、そのため多くの機能が密接に結合しており、モジュールごとに分解して機能を削るようなことは難しい（Cusumano, 2004）。

このようなインテグラル型アーキテクチャーの特徴について、藤本（2007）は、新しい独自の技術や設計には既存の知識だけでは不十分であることを指摘し、設計パラメーターの調整を細かく行うようなインテグラル型アーキテクチャーの製品を作成するのに統合型

製品開発の組織能力が有効であることを指摘している。また、そのような製品は、「ニーズもその変化の程度も異なる多様な市場や顧客に容易に対応できない」[66]が、多くの部品が機能的に、構造的に複雑な相互依存関係にあるため、設計変更やソフトウェアの検証といった問題解決サイクルを何度も繰り返す必要があり、そういった点では、日本企業の統合型製品開発力が適していることを指摘している。

また、大鹿・藤本（2006）は、日本の組立製品・プロセス製品を対象とし、製品・工程アーキテクチャーの視点から輸出競争力を統計分析したところ、インテグラル型に近いほど輸出競争力が高かったとしている。このことは、日本が企業間および企業内部での人的な連携・調整に長けているとされおり、それゆえインテグラル型アーキテクチャーの製品のほうが競争力は高い可能性があることを示していると考えられる。

インテグラル型アーキテクチャーは、コンポーネントが強固に組み合わさって相互に依存しているため、開発スピードが遅くなってしまうが、最適化された特注のレーシングカーのように優れた性能を生み出すこともある（Cusumano, 2004）。

浜屋（2004）も製品アーキテクチャーとビジネスモデルの関係性について、ゲームソフトのようなインテグラル型アーキテクチャーの性質を有する製品の開発を取り上げ、内製を中心とした開発のほうがパフォーマンスは高いとしている。

このようにモジュラー型アーキテクチャーは、機能要素と活動要素が 1 対 1 の明確な関係性を持っているため、個人の専門技術が活きるが、一方で、機能要素と活動要素が多対多の関係であるインテグラル型アーキテクチャーでは、チームワークが活き、長期雇用と長期取引を背景とした日本企業は、そういったチームワーク型において一定の競争力を持つ傾向にある（藤本, 2007）。

もっとも、現実には企業は合理的であり、組織能力や製品アーキテクチャーといったものは固定的ではなく、環境に合わせてダイナミックに変化していく（藤本・武石・青島編, 2001; 浜屋, 2004; 藤本, 2007）。そのため、モジュラー型かインテグラル型かの選択についても**表 4-1** で示したようなメリットとデメリットがあり、どちらかが優れているとは一概にはいえない。どちらかのアーキテクチャーに転換したり、両方を合わせた複合的なアーキテクチャーで構成されたりすることもあるため、どちらが適しているかを一方に決めつけられるものではなく、日本企業が擦り合わせ型に強いという単純な図式も通用するわけではない（藤本, 2007）。

[66] 藤本（2007）p.85

(2) アーキテクチャーの選択

①　企業間のアーキテクチャー

製品アーキテクチャーの選択は、製品の開発のほか、製品生産におけるコストにも影響が発生する。

中川（2011）は、製品アーキテクチャーによるコンポーネントと組織能力の関係について、モジュール化によってコンポーネント間を独立させるように、企業も組織能力をコンポーネント別に分離させていく必要性を主張している。一方、インテグラル化によるコンポーネント間の依存性が高まることに対しては、企業が組織内外のコンポーネント部門を繋ぎ合わせ、組織能力を結合させていく必要性を述べている。

製品アーキテクチャーが異なることにより、企業間の取引関係も変化してくる（藤本・武石・青島編, 2001）。

コンポーネントの整合性をとるような知的結合は、企業間で行うより、自社内で行うほうが容易となる。自社であれば同じ目的の中で協業が行いやすいが、他社との間ではそもそも企業ごとにその目的も異なっており、協業関係にも機会主義が生じる恐れがある（中川, 2011）。

モジュラー型アーキテクチャーとインテグラル型アーキテクチャーに、企業間の提携という軸を加えることで、オープン型アーキテクチャーとクローズ型アーキテクチャーへの分類ができる。

オープン型アーキテクチャーは、モジュラー型アーキテクチャーの一種であり、モジュール間のインターフェースが業界レベルで標準化される。そのため、企業間のモジュールを組み合わせることで、パーソナルコンピューターのような機能性の高い製品ができあがる（藤本, 2002）。

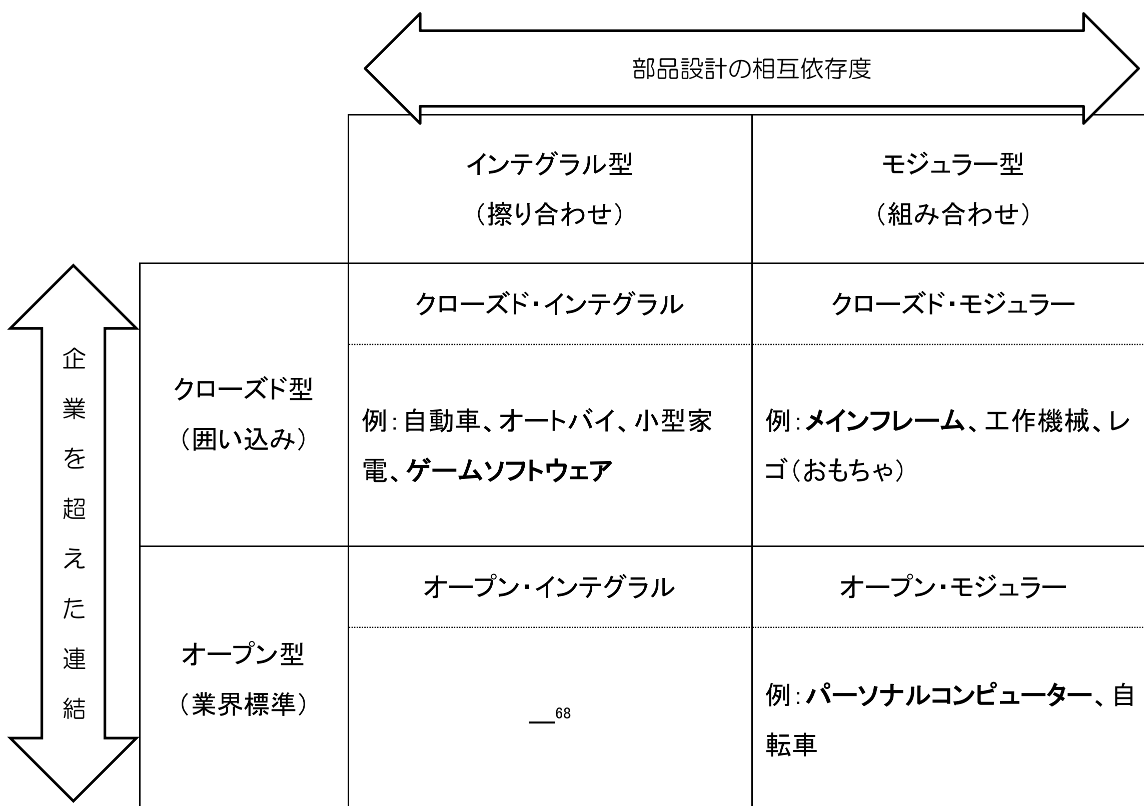
一方、クローズ型アーキテクチャーは、モジュール間のインターフェースが 1 企業内で囲い込みされている。さらに、自動車のように、インターフェースが 1 企業内で決められていながら、擦り合わせで作られるようなものはインテグラル型かつ、クローズ型のアーキテクチャーといえる[67]。

---

[67] 宇山（2013）によると、インテグラル型の代表格である自動車産業も 1990 年代頃からその開発にモジュール化が進められてきている。自動車産業では、1990 年代から 2000 年代にかけて、モジュール化としてプラットフォームを単位に部品を共通化することに主眼が置かれており、「作業負担の軽減や品質向上、機能統合などによるコスト削減や性能向上に用いられた」（宇山, 2013, p.32）。さらに、2010 年前後になると、フォルクスワーゲンがモデルを跨いだモジュールの共通化を行い、モジュールによって構成される基本骨格により製品の多様化と構成要素の共通化を果たしている。

日本では、日産自動車やトヨタ自動車がモジュール化を進めており、特にトヨタの TNGA（Toyota New Global Architecture）が注目を浴びている。

例えば、前述の IBM の SYSTEM/360 のようなものは、モジュラー型であるが、やはりそのインターフェースは 1 企業内で決められているため、モジュラー型かつ、クローズ型のアーキテクチャーといえる（藤本, 2002; 斎藤, 2014）（図 4-7）。



出所：藤本（2002, 2004）をもとに筆者作成

図 4-7 設計情報のアーキテクチャー特性による製品類型

ただし、インターネットのようなネットワーク技術の発達により、モジュラー型の代表格であるパーソナルコンピューターも、そのコアにある CPU は容易に模倣ができないものであり、モジュラー製品といえども、内部はインテグラル型アーキテクチャーの場合が少なくない（藤本・朴, 2015）[69]。

[68] オープン型のインテグラル型アーキテクチャーは、左下の部分のように存在しないとされている。その理由として、オープン型、つまり業界標準となっているもののため、擦り合わせる（インテグラル型）必要が無いためである。

[69] ここまで、パーソナルコンピューターをオープン型モジュラー型の代表格のように扱ってきたが、その OS や CPU などそれぞれの部品自体は専門企業によるインテグラルな製品として作成されている。そのため、統一されたインターフェースのもとモジュラー型の組み立てが行われても、部品同士、またはソフトウェア同士の特定の組み合わせにより、うまく動作ができないといった互換性や相性の問題が存在する。そのうえ、各部品やソフトウェア自体は高度に複雑化した製品となっているため、部品同士の相性が悪い場合でも根本的な原因が突き止められないことも多い。

このように、製品アーキテクチャーの違いによって、製品を設計する際に自社内部だけでなく企業間関係も考慮していかなければならないのである[70]。

モジュラー型アーキテクチャーでは、それぞれのコンポーネントが独立するため、企業同士の調整コストが少なくなり、コンポーネントを専門とする企業は競争力を持ちやすいとされる。ただし、そういったコンポーネント間の取引コストはゼロになるわけではなく、その取引コストが高いのであれば、垂直に統合したほうが有利となることも考えられる（Langlois and Robertson, 1995; 中川, 2011）。

浜屋（2004）の調査によると、モジュラー型アーキテクチャーを採用している企業は、その製品の組み立てをアウトソーシングしている場合が多いという。その上で、アウトソーシングを行っている企業ほど組み立ての自動化などが進んでおり、熟練技術者への依存も低くなっていると述べている。

すでに述べたように、インテグラル型製品アーキテクチャーの場合、コンポーネント間の連携を強めるため、製品システム全体の設計やその構造に関する知識が必要となる。そうした知識を獲得するために、自社で内製することで知識を蓄積していくほか、外部のコンポーネントを持つ企業を買収することで内部化する方法をとることになり、垂直統合度が高くなる。一方で、モジュラー型アーキテクチャーの場合、コンポーネントごとの専門の知識が必要となる。そのため、コンポーネント全般ではなく、特定のコンポーネントに資源を集中させるほうが有利になり、垂直統合度も低くなる。

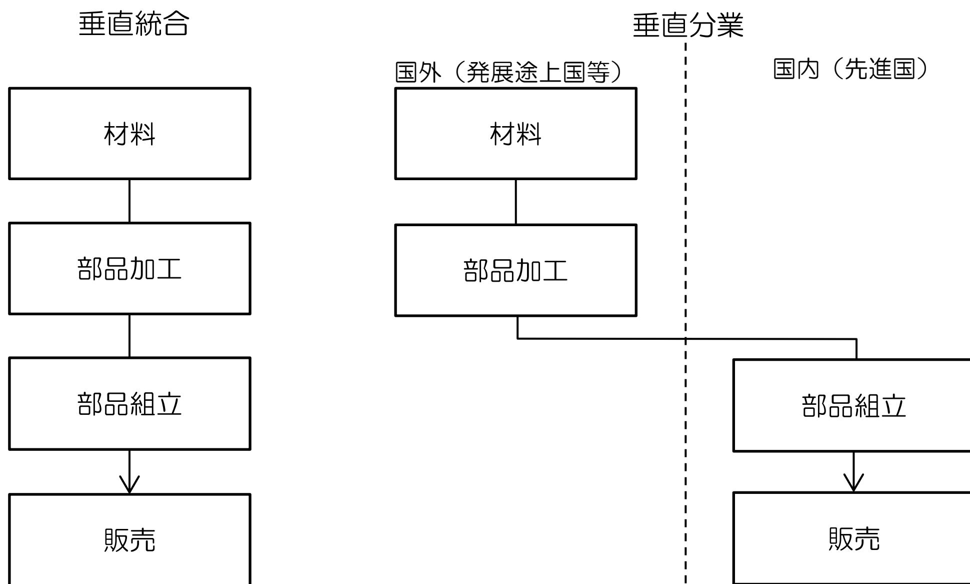
こうしたコンポーネントがそれぞれ市場で取引されるようになることで、垂直統合型の企業は次第に分解され、コンポーネントの専門企業によるネットワーク構造へと産業構造は変化していく（中川, 2011）。

さらに、中川（2011）によると、垂直統合型の企業はモジュール化により製品の情報が明文化され、コンポーネント間のインターフェースが標準化されることで、垂直統合の強みが弱まることになる。その結果、企業ごとにコンポーネントに特化し、垂直分業型企業群によるネットワーク構造が成立するとしている（**図** 4-8）。

---

70 藤本・延岡（2004）は、製品アーキテクチャーの「インテグラル度」「モジュラー度」「オープン度」「クローズ度」といったものを厳密に区別する方法はなく、製品がどれに属するかを測定することは困難であることを指摘している。そのため、安易に組立製品を製品アーキテクチャーの軸で分類することは、危険といえる（藤本・朴, 2015）。

藤本・桑島編（2009）は、プロセス産業を対象とし、生産プロセスや材料の技術的な相互関係である工程アーキテクチャーの傾向から、その製品のアーキテクチャーを把握することを主張している。すなわち、生産プロセスが技術的に複雑で相互に依存し合っているインテグラル型工程アーキテクチャーの場合、そこから生産される製品も複雑性の高いインテグラル型製品アーキテクチャーとみなしている。一方で、生産プロセスが技術的な相互依存が低いようなモジュラー型工程アーキテクチャーの場合は、そこから生産される製品もモジュラー型製品アーキテクチャーであるとしている。



出所：伊丹・松島・橘川編（1998），中川（2011）をもとに筆者作成

図 4-8 垂直統合と垂直分業

② ソフトウェアのアーキテクチャー

このように、インテグラル型のアーキテクチャーはモジュラー型アーキテクチャーと比較して、コンポーネント間の調整が必要となり、自社内部にとどまらず、企業間で開発活動と知識を共有化していく必要がある。

本研究が対象とする日本のカスタムソフトウェア開発はインテグラル型に近いとされるが、各企業が担当する工程間は、分業が十分になされていないゆるやかな分業関係がみられ（峰滝, 2004）、その開発活動にも、企業間の知識をいかに共有化していくのか、知的結合を含んだ境界をいかにデザインしていくのかが問題となる。

特にソフトウェアは競争が激しい分野でありながら、その技術は目に見えないためその差がわかりにくい。そのような技術的な差別化が難しい分野では、顧客の特殊な知識を製品開発のプロセスにどれだけ取り込めるかが重要となる（藤本, 2007）。

ソフトウェアの開発は細かいインターフェース部分の設計の擦り合わせが必要となるが、必ずしも全体を一括して開発するわけではない。むしろ比較的関連性の低い機能ごとにモジュールへと分割してそれぞれの機能ごとに開発を行う。機械製品が生産工程によってモジュールが分割されることが多いのに対し、ソフトウェアの開発はこの機能ごとに開発作業が分割される必要があり（竹田, 2005）、作業の手順を分割する開発プロセスのモジュール化と、関連性の強さによる機能ごとのモジュール化が行われている。

特にソフトウェアは高度に複雑化していくにつれ、そのプログラムコードも何千、何百万行にも増大してしまうため、コンポーネントごとに独立させるモジュラー型アーキテクチャーの重要性が増してくる（Cusumano, 2004）。

峰滝（2005）は、ソフトウェアのモジュール化に関して「特定サービス産業実態調査」を用いて、モジュール化の成功により生産性が劇的に上がったコンピューターなどハードウェアを比較対象として、ソフトウェア開発にもそのモデルがあてはまるかどうか実証分析を試みている。峰滝によると日本の情報サービス産業全体の問題として、労働集約的な方法のため、アウトソーシングが効率的ではなく開発規模の経済性が働いていないことを指摘している。その上で、情報サービス産業の生産性向上としてアウトソーシングを効果的に行うためには、ソフトウェアの機能とその開発プロセスがモジュール化されていることが必要であり、硬直的なピラミッド型の産業構造ではモジュール化という概念が馴染まないと述べている。このことは、日本企業がモジュール化を極端に進めず、擦り合わせによって差別化を図ってきた（藤本, 2004; 延岡, 2006）こともその一因と考えられる。

しかしながら、藤本・武石・青島編（2001）や藤本（2007）らが指摘するように、現実にはどちらか一方のアーキテクチャーのみ採用するとは限らず、中間的、もしくは双方を取り合わせて構成されることも多い。藤本・延岡（2004）も、一方的なインテグラ化もモジュール化も好ましくなく、アーキテクチャーは消費者の選好パターンによって決められると述べている。さらに、変化・多様性を好む消費者はモジュラー型製品を、統合性・洗練性を好む消費者はインテグラル型を好む傾向にあるとし,「製品アーキテクチャーの選択は、企業組織の製品の設計能力と消費者の製品評価能力の相互作用および共進化の経路によって決まる」[71]と述べている。

モジュラー型アーキテクチャーを採用することによって、垂直統合であった企業が部品部門などを分離、独立化させたとしても、その部門自体が卓越したコンポーネント知識を保持していなければ分離した強みを活かすことができない。また、擦り合わせ型アーキテクチャーを中心とした垂直統合を進めたとしても、その連携と調整が適切でないかぎり、強みが活かせないといえ、企業が保有する知識や構造にこそ、その本質があると考えられる（中川, 2011）。

(3) 小括

本章では、ソフトウェアの開発プロセスの問題を明らかにする上では、そもそもソフトウェアがいかなる製品設計思想に基づくものであるのかを明らかにすべく、ソフトウェアのような複数のサブシステムで構成された製品について、部品の構成や相互関係のあり方を決めるアーキテクチャーの視点から、モジュラー型アーキテクチャーとインテグラル型アーキテクチャーの特徴と問題を検討してきた。

モジュラー型アーキテクチャーをもとにした製品は、インターフェースが 1 : 1 で構成されるよう分割することで複雑な製品の開発はある程度容易になるものの、そのモジュール

[71] 藤本・延岡（2004）pp.14-15

を調達し、求める機能としてどのような組み合わせが適切であるかの知識が必要となる。一方で、インテグラル型アーキテクチャーの製品は、インターフェースが多：多で構成されており、擦り合わせて開発しなければならず、コンポーネント間の連携を強めるためには、製品システム全体の設計やその構造に関する知識が必要となる。

ソフトウェアは個々の機能をモジュール化しているが、それはすなわちコンピューターや工業製品のように単純な組み合わせで作成できることを意味しているわけではなく、ソフトウェアは機能と性能が複雑に組み込まれた構造であり（藤本・朴, 2015）、一般的な工業製品のように、個々のモジュールが完全に独立することは難しい。

ソフトウェアもいかにその機能や工程のモジュール化を進めていくのか、さらにそうしたモジュール化された機能や工程をいかに調整し、擦り合わせていくかのという点が重要となり、モジュールを単純に組み合わせるだけでは新たな価値を生みだすことは難しい。特に顧客のニーズが複雑で、変化が激しいような環境において、ソフトウェアを決まった組み合わせにより作り上げることは難しく、モジュール同士の組み合わせをどう擦り合わせるか試行錯誤していく必要が出てくるのである。そして、このことはモジュールを中心に、作業プロセスの編成、工程間分業の編成、それらの連携や調整をいかに行うかという問題にも繋がる。

次章では、このようなソフトウェアの開発プロセスの特徴と限界について検討する。

第5章 ソフトウェア開発の特徴と手法

ソフトウェア開発において、1970 年代より現代に至るまでその中心を担ってきたものが Waterfall Model と呼ばれる開発手法である。Waterfall Model は今日の日本のソフトウェア開発でも主流の開発手法として多くのプロジェクトで採用されており、多くの貢献をしてきた。特に Waterfall Model は、次章で検討するかつて日本のソフトウェア開発で中心的存在を果たしたソフトウェア・ファクトリーと呼ばれる工場型の開発モデルのような開発方法やその分業構造と親和性が高かった。

一方で、この Waterfall Model は、今日必要とされる質の高い、革新的なソフトウェアの開発を行うには限界があり、海外ではこの Waterfall Model に代わる方法として登場した Agile が開発手法の主流になりつつある。

本章では、ソフトウェア開発の分業構造やその知識労働について検討する前提として、これらソフトウェアの開発プロセスに関する整理と比較を行う。未だ日本の多くの企業で Waterfall Model が採用されている要因について、さらにこうした Waterfall Model が Agile へと取って代わられつつある要因について、その開発プロセスの特徴と限界がいかなるところに見られるのか検討する。

(1) ソフトウェアの開発プロセス

① 基本的な開発プロセス

Waterfall Model と Agile といった開発手法について検討する前提として、ソフトウェアの基本的な開発プロセスを確認する。ソフトウェアの開発プロセスは、当初は個人の作業スタイルで行われていたものの、品質管理などの点から製造業のような工程ごとに分離し、その工程を順番にこなしていく方法が定着している（表 5-1）。

このソフトウェア開発プロセスの詳細は次のとおりである。

・提案依頼、見積もり

開発プロセスに入る前に、開発の元請けとなる企業の選定のため、コンペティションが行われる。ユーザー企業から提示された提案依頼書[72]を元に、コンペティションに参加する IT ベンダーはシステム設計のための提案書や見積書などを作成する。このコンペティションの結果、ユーザー企業と IT ベンダーの間でシステム開発などの契約が締結され、以降の開発プロセスへと着手することになる。

[72] Request For Proposal。RFP と略されることが多い。発注企業が、IT ベンダーに対して、業務委託や情報システムの導入の際、具体的な提案を依頼するドキュメント。

表 5-1 ソフトウェア開発の標準的なプロセスと職種の作業範囲

プロセス	Systems Engineer	Programmer[73]	作業人数の目安[74]	大まかな作業分類・概要
見積もり	管理・作業		少(1～3 人)	業務の現状や問題点の洗い出し開発の全体の規模を決める
分析	管理・作業		少(1～3 人)	
要件定義	管理・作業	※	少(1～3 人)	基本的な事項を決定。システム、プログラム、データの構成と仕様などを決める（定義する）
基本設計	管理・作業	※	少(1～3 人)	
詳細設計	管理・作業	作業	多(7～10 人)	プログラムの詳細（動作や処理の流れ）を決定し、作成する
プログラム作成	管理	作業	多(7～10 人)	
単体テスト	管理	作業	多(7～10 人)	できあがったプログラムを動かし､1 つのソフトウェア・システムとして設計や要件に合致すること（仕様を満たしていること）を確認する
結合テスト	管理	作業	中(3～6 人)	
システムテスト	管理・作業	※	中(3～6 人)	
本番移行	管理・作業		少(1～3 人)	システムを本番稼働させる（旧システムとの切り替えなど)。 稼働後の日々のメンテナンス
運用・保守	管理・作業		少(1～3 人)	

出所：筆者作成

[73] 便宜上、システムエンジニアとプログラマーの職務担当を分けているが、開発プロジェクトによって、作業範囲は異なる。実際には、プログラマーも要件定義やシステムテストなどに従事する場合もあり（表中の※）、システムエンジニアとプログラマーの職務上の定義は曖昧である。

さらに、プログラマーに対し、プログラム作成専門のコーダーや、テスト工程を担当する作業者であるテスター、ゲームソフトウェアでさまざまな操作などを行いプログラムのバグを発見する作業者であるデバッカーなども存在する。中でもコーダーはプログラマーとは異なり、コーディング、つまりプログラム作成作業として設計書通りのプログラムを記述するような作業者を指すことが多い。ただし、日本ではプログラマーとコーダーの違いは明確ではなく、また、その定義も企業、プロジェクト、研究者で異なっている。

[74] 表では例として 10 人程度のチームの人数を示している。開発に関わる作業人数はソフトウェア開発の規模によって増減するが、基本設計までの作業人数は少なく、プログラム作成とそのテスト工程で一番人数が多くなり、その後、徐々に減少していく傾向にある。

例えば、10 人程度でチームを組む開発プロジェクトの場合、おおよそ少=1~3 人、中=3~6 人、多=7~10 人程度で構成され、より大きなプロジェクトの場合、少=2~4 人、中=5~8 人、多=8~20 人といった具合で構成されることになる。

また、提案依頼書を作成するベンダーを選出するコンペティション自体が行われる場合もある。官公庁による発注では、企業の売上や営業年数、自己資本、流動比率などの外形的要素によりランク付けを行い、予定価格の大小に応じて入札への参加資格を分類しており、これら外形的要素に弱い中小企業は参入しにくい場合もある。

・分析

「分析」は、「システム分析」とも呼ばれ、ソフトウェア開発に着手するための事前の調査、分析作業であり、ソフトウェアの現状や問題点を洗い出す基礎となる。ユーザー企業のシステムへの要望は、不完全で曖昧であり、かつ矛盾している部分が多く存在し、実際の業務に適していない要望が含まれている場合もある。そのため、まずはシステムの現状を把握、分析していくことが重要となる。

・要件定義

「要件定義」は、分析結果を基に、実際の業務と照らし合わせ、具体的にどのようなソフトウェアを作成するのかを決める重要なプロセスであり、ここでソフトウェア開発プロジェクトの成否を左右する諸要因が決まるといっても過言ではない（萩森, 2007; 中尾, 2009）。

ユーザーの要望は不完全で曖昧なだけでなく、広大で多岐に渡る。しかし、その要望のすべてをソフトウェアとして実装するには、莫大な予算や数年単位に及ぶ開発期間が必要となり、現実的には不可能なことが多い。特にビジネスのスピードが上がるにつれて、ソフトウェアの開発スピードも求められており、不要な要望や機能を削ぎ落として、適度な期間でシステムを作り上げることが重要視されてきている。そこで、要件定義として業務フローの精査やユーザーヒアリングなどを通じて、どのような機能が必要で何が不必要かを選定していく。この選定を失敗してしまうと、要求に満たないソフトウェアや、想定外のソフトウェアができあがってしまうのである。

このようにユーザー要件は曖昧なため、後の工程で要件の変更や追加が発生することが多く、納期が遅延する原因となる。曖昧な条件の中、少しでも正確な見積もりや要件定義を行うには、先のリスクを予見できるような経験が重要であり、くわえて、業務に精通していることも必要となる。

・設計

「設計」は、要件定義で確定した内容を仕様書や設計書に落とし込む工程である。次のプログラム作成工程のために、ここで可能な限り詳細な開発内容を決定する必要がある。例えば、Web サイトの開発であれば、どのような画面を表示するのか、どのように画面の遷移や処理を行い、どういったデータや帳票ができあがるようにするのか、といった具体

的な仕組みに関してこの工程で決定する。

このとき、次のプログラム作成工程を海外へアウトソーシングするような場合には、設計書も英語などアウトソーシング先に合わせた言語で作成する必要があり、外国人技術者が理解できるレベルまで詳細に作成する必要がある。しかし、外国人技術者は日本特有の仕様や商業慣習までは理解できない。そのため、設計書が日本人であれば理解できるような曖昧な部分を残していると、誤りや齟齬を生じさせ、プログラムの不具合を招き、ひいてはソフトウェア全体の品質が低下してしまう原因となる。

また、こうした設計書は必ず作成されるとは限らず、簡易なものや概要程度でまとめられたものしか作成されていなかったり、過去に作られたが最新のソフトウェアの情報に更新されていなかったりなど、不十分なことが多く、次工程のプログラム作成を難しくする要因の一つとなっている。

・プログラム作成

「プログラム作成」は、設計書を元に実際にシステムを開発、構築する工程である。「製造」や「実装」、「開発」とも呼ばれ、プログラムのソースコードを作成していく。このプログラムをコーディング[75]する作業者はプログラマーと呼ばれ、その専門性のため、非常に難解で、かつ複雑で高度な作業であると捉えられることが多い[76]。

しかしながら、日本の大半のソフトウェア開発の現場においては、このプログラム作成作業は、要件定義や設計で定められた仕様書や設計書に基づいてプログラムを記述することが目的とされ、ソフトウェア開発における工程のうちそれほど重要なものではないとされている。そのため、プログラム作成工程は、設計書を元にコーディングするだけの単純作業や軽度の作業とみなされ、それゆえ下請け企業への委託や海外へのアウトソーシングの対象となってしまうことが多い[77]。ただし、設計工程で述べたように、設計書は不十分であることが多く、設計書の内容次第では想定外のプログラムができあがってしまうため、安易な外部への委託はシステム開発の失敗の要因となる。

---

[75] coding。プログラム言語を使用し、ソースコードを作成する。プログラミング、プログラムを書く・作る・組む、実装などの呼称がある。

[76] 極めて高度なプログラマーとして、天才プログラマーと呼ばれる人間も存在する。彼らは、高度で革新的なソフトウェアを多く生み出すことができる。特に、Windows といったパソコンの OS など、非常に複雑で難度が高いソフトウェアは、そのような非常に高度なプログラマーが関わっていることが多く、一般には理解されにくい部分でもある。

また、2000 年代半ばから、クラウド・コンピューティングといった技術に注目が集まるに伴いその専門性も日々高まっており、プログラマーを中心としたソフトウェア技術者はそうした新しい技術にも深く精通している必要がある。

[77] 脚注ですでに述べたが、設計書通りのプログラムを記述するだけの作業者のことをコーダー（Coder）と呼ぶ。日本では、明確にはコーダーと呼ばないものの、プログラマーの作業がコーダーと変わらないような場合もある。

特に、ソフトウェアの開発は、数人から数十人、もしくは数百人といった多くの技術者が関わるため、全員が優秀な技術者であることは現実的ではなく、さらにスーパープログラマーを確保することや養成することは容易ではない。ソフトウェア開発は、さまざまな能力や異なるスキルを持った人材で成り立っており、その適切な業務分担をどのように行っていくかが重要となる（小川, 2011）。

・テスト

「テスト」では、ソフトウェアのプログラムが正しく動くかを確認する。このテスト工程でプログラムのバグ[78]などの欠陥を除去しており、プログラムの正確性や、操作性、レスポンスなど、細かな部分から総合的な部分までの品質が保証され、その後、ユーザー企業に納品され、ソフトウェアは本番稼働されることになる。

テスト方法も、単純なプログラムの稼働確認から、細かいロジックの確認のほか、例外ケースの確認、インターフェース同士の結合部分の確認、ユーザーの実際の操作による確認など、さまざまなものが行われる。テストは、工程の中では比較的大きい工程であり、大規模なシステム開発の場合、1 年以上にわたりテストを続けることもある。

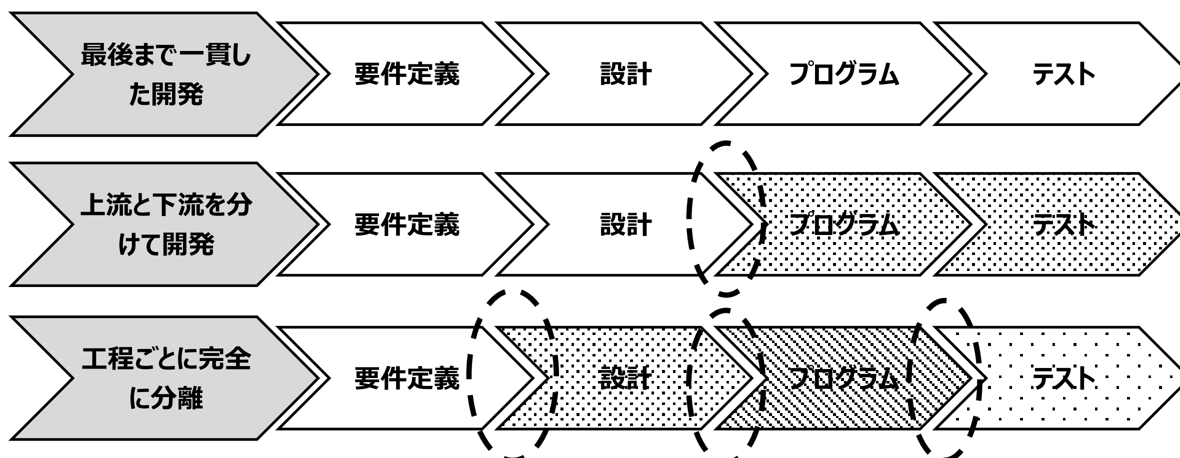
・運用、保守

「運用・保守」では、ソフトウェアの開発が終わり、本番サービスを開始した後の定期的な安定運用やソフトウェアの改善・追加などを行う。特にソフトウェアは、ユーザーの利用状況やアクセスの頻度、利便性など、実際に運用してみることで初めて分かる部分が存在し、日々の状況や現場の変化に対応していくため、ソフトウェアは定期的にメンテナンスしていかなければならないのである。

これらの工程のうち、システム分析から、運用・保守まで一貫して IT ベンダーが受託する場合や、工程ごとに別の IT ベンダーが担当する場合もある。そのうえ、一貫して開発を請け負った場合でも、実際に開発に携わる下請け企業が工程ごとに入れ替わることもある（**図** 5-1）。

さらに、システムの開発と、その後の運用・保守作業は別物として扱われることが多く、システム構築や開発を行った IT ベンダーが、その後の運用・保守まで契約できるかについては、それまでの開発実績や納品したシステムの品質次第といえ、システム開発の継続性に関わる問題となる。

[78] コンピュータープログラムに含まれる記述ミスといった誤りや欠陥。ソフトウェアが仕様通りに、または正常に動かなくなることの原因となるが、よほど小規模なものでもないかぎり、バグがないソフトウェアの開発は不可能といえる。



出所：筆者作成

図 5-1 ソフトウェア開発の工程分離の例

また、こうした工程間の分業は企業間に限られず、1 つの企業の中でも部や課、チームやプロジェクト間で作業工程を担当するメンバーが入れ替わる場合がある。

このような開発工程は、ソフトウェアの開発手順として基本的なモデルであり、Waterfall Model や Agile などどのような開発手法であっても変わりはない。企業や開発プロジェクトによって、多少工程の名称が異なることもあるが、概ねこのような手順を踏んでソフトウェアは開発されていく。実際の開発現場では、これらの工程も開発規模の大小により、さらに細かく分類されていることも多く、例えば、設計作業でも基本設計、外部設計、内部設計といった具合に分割されることもあり、テストも単体テスト、結合テスト、システムテストなどに分割されることもある。

ソフトウェア開発では、これら多くの作業工程を経由しながら完成を目指すことになるが、いずれかの工程で問題が発生した場合、開発作業全体に影響する可能性がある。ソフトウェア開発プロセスは、こうした問題を防ぎ、工程のスケジュールや管理していくため、工程ごとに分離されたアーキテクチャーをとることが多い。

一方、これらの作業プロセスはゆるやかな分業でてきており（峰滝, 2004）、それぞれ密接に関連している。例えば、プログラム作成には設計情報が必要となるなど、前工程に関わる技術を含む多くの情報が必ず必要となるため、隣接する工程から完全に独立した開発は不可能であり、工程間の情報の共有が不可欠となっている。

さらに、ソフトウェア開発の作業をアウトソーシングするため、ソフトウェアの機能だけでなく作業工程も分割し、それぞれの工程ごとに IT ベンダーを選出する場合がある。こ

のような工程ごとに分離した開発方法は、ソフトウェアが小規模や簡易であれば、その工程間の関係も複雑ではないため問題は少ないと考えられる。しかし、ソフトウェアが大規模、かつ複雑になるにつれ、工程ごとに異なる IT ベンダーを採用することは、品質や生産性、納期の調整や達成が難しくなる。さらに多重下請けなどに関するコンプライアンスの問題も生じることから、一括してシステム分析から運用までを任せられるような元請け企業として特に大手 IT ベンダーに任せられることが多い。

このようなユーザー企業の要望に応えるため、大手 IT ベンダーは大規模な開発に備えて系列会社や子会社を設立し、そこに続けて発注することで工程間の切れ目を無くし情報共有を高めている。また、アウトソーシングをする場合でも、受注依存度が高い専属型の下請け企業に発注するケースが多い。

独立行政法人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（2013）によると、受託開発の課題として、仕様や計画の変更が多いといった問題のほか、要求仕様や設計仕様の共有が難しいことが指摘されている。ソフトウェア開発では、工程ごとに分割する手法は不十分であり、特に工程ごとにアウトソーシングするような方式では効率の良い開発ができるとは限らないのである。

② 要件定義工程を中心とした上流工程の重要性

ソフトウェア開発が工程ごとに分割されることを説明してきたが、これら開発工程のうち、どのようなソフトウェアを作成するのかを決める要件定義工程や設計工程の重要性をあげる研究も多い（萩森, 2007; 中尾, 2009 など）。特に要件定義は、要求提示元となるユーザー企業によって仕様の品質が大きく左右される（北村, 2009）。

日経コンピュータ（2003）の調査によると、プロジェクトの QCD（品質、コスト、納期）について当初の計画を達成できなかった最も大きな理由として、要件定義の曖昧さによる問題があげられている。後述する Waterfall Model のような上流工程から下流工程に流れていくような開発スタイルでは、計画を予定通りに進めるために、要件定義が重要となる（萩森, 2007）。このために、ソフトウェア工学のもと、要件定義を明確化し、問題を早めに見つけるような取り組みが研究されてきた。しかし、企業の戦略やビジネスモデル、環境の変化により、要件定義は常に変わる可能性がある。

特に日本のソフトウェア開発では、要件定義を曖昧にしたままにしておくことが多い。この理由として、萩森（2007）は、日本のソフトウェア開発が工程を進めていく際に要件定義の内容やそれぞれの責任を曖昧なままにし、ソフトウェアを作り込む過程で徐々に要件の決定や修正を行うことを指摘しており、これにより曖昧な部分を明確化するとともにユーザーの要求を満たすソフトウェアの完成を目指しているとも述べている。

神岡・細谷・張（2006）も同様に、日本のソフトウェア開発が、要件定義を含む上流工程で作成される仕様書や設計書などのドキュメントが曖昧になっており、後工程に進むに

つれて最適な設計を工程間で擦り合わせをしながら開発を進めていると指摘している。

一方、神岡・細谷・張（2006）は、ユーザー企業が要件を含む仕様を明確にできないため、そのような明文化されない部分について IT ベンダー側が業務知識を含む過去の開発経験に基づいた知識やコミュニケーションで補う必要があることを指摘している。そのほか、要件定義などの上流工程を担当する IT ベンダーの能力不足も、その要件が曖昧になる理由の一つとしてあげられている。こうした上流工程の IT ベンダーの能力が不足しているような場合には、下流工程の IT ベンダーが上流工程の要件定義や設計に一緒に立ち会ったり、ユーザー企業が直接下流工程の IT ベンダーとコミュニケーションを取ったりするようなケースも出てきている。

また、ソフトウェア開発は技術者の専門的な知識や技術が不可欠であるが、ユーザー企業側に必ずしも技術やシステムに精通している人材がいるわけではない。そのため、ソフトウェア開発の能力が不足しているユーザー企業は、IT ベンダーに要件定義のすべてを一任してしまうことが多く、一方 IT ベンダー側も要件の変更や追加、スケジュールの遅延を避けるために、要件定義を主導することが多い。その結果、ユーザー企業が十分に関わってこなかったことによる要件の齟齬が発生することもある（松村・吉田・井手・森崎・戸田・松本, 2011）。特に、業務の分析が不十分だとできあがったシステムの機能も乏しいものとなってしまう。

こうした要件定義の難しさには、顧客の要望する機能要件だけでなく、顧客が明確に意思表示をしていないが、開発していく上で考慮しなければならない非機能要件がある。例えば、システムが特定条件でもダウンしないようにすることや、処理の負荷が高まっても速度の低下を抑えることや、セキュリティ精度のことに対する考慮であり、顧客自身が必要であってもそこまで要件として提示できなかったり、考えが及ばなかったりすることが多く、技術者側で補完しなければならないものである。この非機能要件数が多いほど、システムの信頼性も向上するのである（中尾, 2009）。

このように、要件定義を含む上流工程は完成することはなく、続く下流工程には明確にできなかったり、気づかなかったりした要件や設計の要素が残っているのである。

(2) 開発手法の変遷

① 計画と管理を目的とした Waterfall Model

ソフトウェアの開発において、要件定義を曖昧にしたまま後の工程で擦り合わせながら決めていくことを述べたが（神岡・細谷・張, 2006）、多くの製造業においても開発の途中で製品の仕様に対する変更が発生するように、ソフトウェアの開発にも仕様の変更が発生する。さらに、ソフトウェア開発は、物理的な製品が関わる製造業と比較してデジタルデータといった変更しやすい特性がある。そのため、開発の途中で仕様変更が発生することが多く、その変更を加えることでスケジュールの遅れや新たなバグの発生など品質の低下

に繋がっている[79]。

このような仕様変更で生じる問題に対する方法として、仕様変更を極力防いで当初予定していた通りのソフトウェアを完成させる方法と、仕様変更をできるだけ完成物に反映させる方法が考えられる。特に、製造業や建築業では、仕様変更を極力防ぐ方法が取られてきた。製品の仕様の決定や概念設計などの上流工程が完成してから、実際の製造などの下流工程へと入る方法である。その理由として、製造業や建築業では、一度製品の製造や建設作業に着手すると、工程を後戻りした場合、設計図の変更、金型の再作成、原材料・資材の再調達などが発生し、莫大なコストが発生してしまうためである。例えば、家屋やビルを建てる場合、一度工事が始まってからの大きな変更は難しいであろう。このため、下流工程から上流工程への後戻りを極力行わないような手順になっているのである。

この仕様変更を極力防ぐ設計手法は、開発の初期段階で問題を発見することで、後工程での設計変更を減少させ、開発コストや開発期間の短縮を図ることを目的としている。そのため、特に上流に位置する基本設計や概念設計に重きを置いている（竹村, 2001）。

上流工程を重視し仕様変更を防ぐ開発方式は、製造業をモデルとした日本のソフトウェア開発に広く浸透してきた。このようなソフトウェアの開発を工程ごとに分離し、上流工程を重視して工程を遵守することで、納期や費用という観点で効率的に管理する開発手法として定着したものが Waterfall Model である（図 5-2）。

この Waterfall Model では、顧客からの要望を取り入れて、仕様を明確にして設計に落とし込む上流工程と、その仕様や設計を元にプログラムを作り上げてテストを行う下流工程とに分離される。

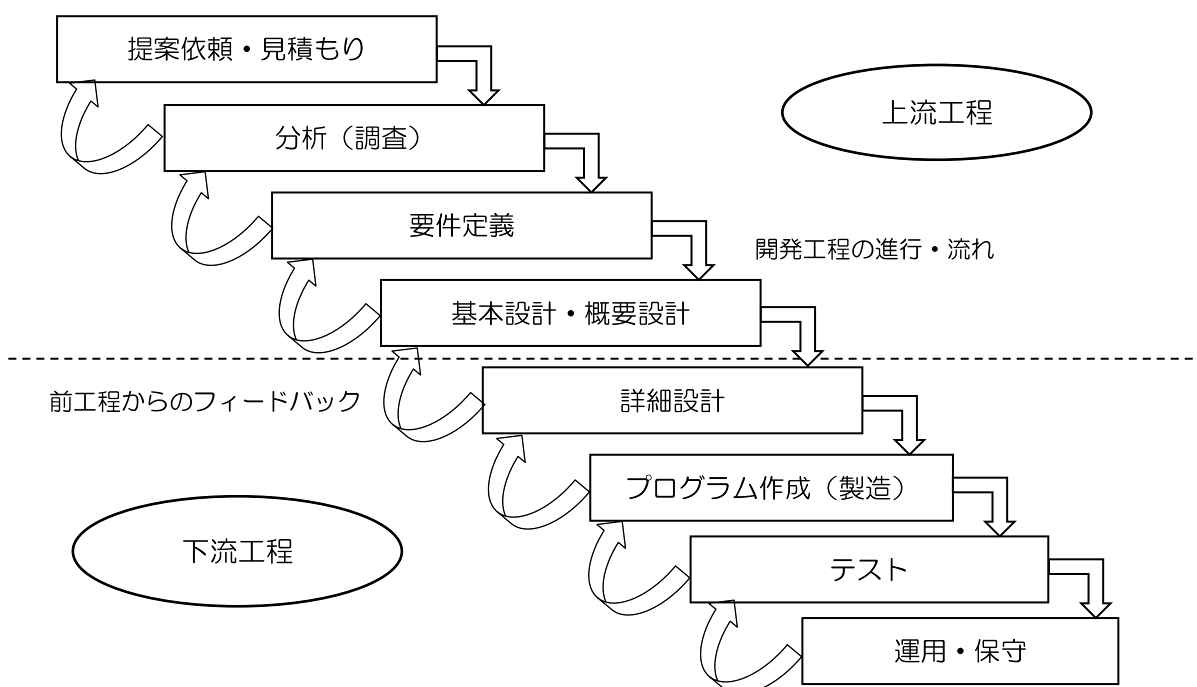
このような上流に位置する工程では、高度なスキルを持ち、業務に精通した熟練技術者が設計図を作成する。一方で、下流に位置する工程では、コストが高いベテラン社員より、スキルや経験が浅いもののコストが安い新入社員などの技術者[80]や、自社の社員よりコストがかからないアウトソーシング先の企業に割り振る方法が、Waterfall Model に準拠した標準的なソフトウェアの開発手法である。

この開発工程の大きな特徴は、原則として作業の順番が決まっており、見積もりや分析から順番に工程が進んでいき、必ず前の工程が完了してから次の工程へ進まなければならない。この工程の遡りが発生しないように、水が滝を流れ落ちるような手順となっている

[79] ソフトウェアは、最終的に 0 と 1 に変換されるプログラムによるコンピューターへの命令、またはアルゴリズムでできている（Cusumano, 2004）。ソフトウェアは、このようなデータでできているため、変更しやすい。

[80] 新入社員や若手社員のような技術スキルの低い者がソフトウェアを開発することが可能な理由として、ソフトウェアの標準的な処理がある程度モジュール化、コンポーネント化されていると考えられる。こうした状況では、複雑な部分を一から作成する必要はなく、品質にある程度目をつぶれば、それらモジュールやコンポーネントを単純に組み合わせることで簡易なソフトウェアを作成することが可能となる。

ため、Waterfall Model と呼ばれている（高橋, 2010）。



出所：Royce（1970）, 高橋（2010）, 小椋（2013）等をもとに筆者作成[81]

図 5-2 一般的な Waterfall Model

このうち、システム分析や要件定義から設計までの工程を「上流工程」と呼び、運用・保守を除くプログラム作成以降の工程は「下流工程」と呼ばれる。しかし、現実には上流工程で作成された要件や設計に誤りが含まれていることも多く、頻繁に遡りが発生するため、納期の遅延やプロジェクトの失敗を招いてしまうことも多い。そのため、Waterfall Model による開発では、IT ベンダーや各工程の間で高度な情報共有を行うことで、工程の遡りが発生しないよう努める必要がある。

この現在の Waterfall Model の概念となるものは、Royce（1970）が提唱した。ただし、Royce の開発手法は、現在の Waterfall Model とは異なり、隣接する工程ごとにフィードバックすることを必要とした。さらに、Royce は大規模ソフトウェアの開発には製造業の製造工程のようなトップダウンのアプローチが必要な一方、この手法をそのまま適用することは難しく、暫定的なプログラム設計を先行するなど、幾つかの対応が必要なことを指摘

[81] ソフトウェア開発における工程の区分けと名称は、統一されたものが存在しない。そのため、上記の図は一般的な Waterfall Model であるが、その中の細かい工程と名称については企業やその開発プロジェクトによって異なる。例えば、概要設計や詳細設計をまとめて一つの設計としてしまう場合や、基本設計・概要設計のみ行い、詳細設計そのものは飛ばして開発（実装）へ進めてしまう場合もある。

していた。しかし、後に Royce の手法が Waterfall Model として広まった頃には、このフィードバックの概念は無くなってしまい、現在のような後戻りのできない開発手法として定着してしまった（妹尾, 2001; 高橋, 2010; 小椋, 2013）[82]。

この Waterfall Model は、前の工程に戻らないことを前提としているため、ソフトウェア開発プロジェクトの全体を見渡せることができ、スケジュールの立案や資源配分、進捗状況の理解が容易となり、それゆえ高い信頼性が要求されるようなシステム開発に有効とされてきたのである（新井, 2016）。

② 変化とイノベーションを目的とした Agile

ここまで計画と管理を中心とし、手順を追って開発すると同時に仕様変更を認めない Waterfall Model モデルについて説明してきたが、このような Waterfall Model に対し、仕様変更を是とし、部分的に機能を完成させていく新しい開発手法が考えられていった。

例えば、システム全体を一気に作成するのではなく、幾つかの機能に分割し、その単位ごとに開発サイクルを回していくスパイラルモデル（Boehm, 1988）と呼ばれる反復型開発方法などが存在する[83]。この反復型開発方法で行われるサイクル内の作業工程は、Waterfall Model と同様であり、工程やスケジュール管理を厳格に行っていく Waterfall Model の拡張版といえる。

工業製品の開発では、試作としてプロトタイプを作成することが重要でもある。特に、製品開発では顧客価値のための設計情報に対する要求が高くなっており、実際のプロトタイプを作成する前のシミュレーション力が重要となっている（藤本・朴, 2015）。同様に、ソフトウェアでも技術的な部分の裏付けのためや、ソフトウェアの一部機能の動きの確認のため、プロトタイプ的にソフトウェアを作成する場合もある[84]。

---

[82] 小椋（2013）は、Waterfall Model の起源を調査し、Royce（1970）が提唱したとされている開発手法が、フィードバックを必要としており、現在のような一方通行の Waterfall Model とは異なっていること、さらに Royce 自身はその論文で「Waterfall Model」という用語も使用されていないことを明らかにしている。

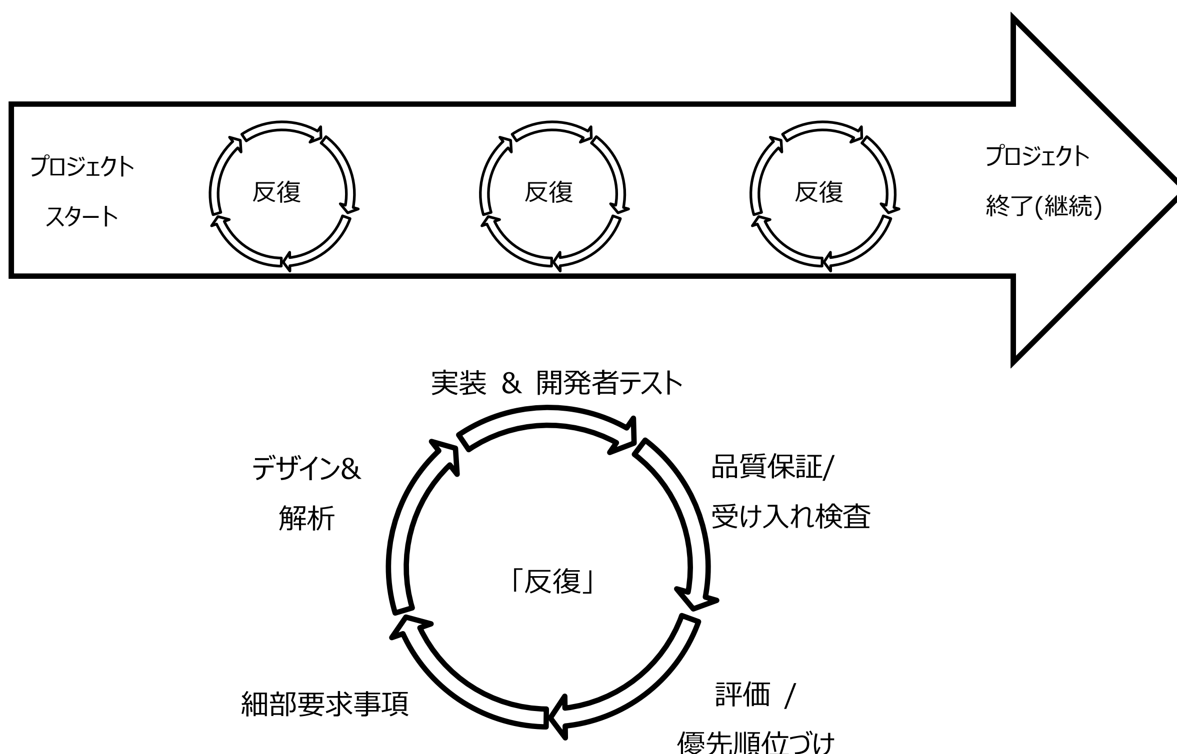
小椋によれば、アメリカの TRW 社に属する Bell and Thayer（1976）によって“Waterfall”という用語が初めて論文に登場し、アメリカの弾道ミサイルによる防衛のためのソフトウェアプロジェクトで使用されたという。

Bell and Thayer（1976）は、Royce（1970）の論文について取り上げ、“he introduced the concept of the "waterfall" of development activities”（Bell and Thayer, 1976, p.62）と、Royce が Waterfall 型の開発の概念を導入したと述べている。

[83] 非 Waterfall Model の代表的なものとして、Agile を本研究で取り上げている。反復型開発モデルの発展したものが Agile という見方もある。本文で説明しているように、Agile は管理よりも変化に重点が置かれている。しかし、Agile は、日本で導入する企業がまだ少なく、そのノウハウも企業内に蓄積されていないため、それを利用する技術者は、独学に近い状況となっている。

[84] ソフトウェアの反復型の開発方法は大きく 2 種類の方法が存在する。1 つは、反復型開発モデルで、プロトタイプとしてシステムを作成し、そのサイクルを繰り返すことで機能

さらに、Waterfall Model などの従来のソフトウェア開発手法に対して、1990 年代後半に Agile が提唱された（Coplien and Harrison, 2004; James, 2010; 設楽・中佐藤編, 2012）（図 5-3）。



上部の左から右へ流れる矢印は、Agile(SCRUM)開発の全体の流れを表す。その中で「反復」を繰り返している。
下部の円を描く矢印は、「反復」の内容を表している。

出所：James（2010）をもとに筆者作成

図 5-3 Agile（SCRUM 開発手法）

この Agile は、正確には特定の手法を指しているのではなく、Waterfall Model のような多くの手順に従って進んでいく重量級な開発手法と比較され、Scrum[85]や XP（extreme

---

を深めていく使い捨てのプロトタイプモデルである。もう 1 つは、段階的開発モデル（インクリメンタルモデル）で、小さい部分からシステムを作り、段階的に機能を拡張していくといった平行で開発できる拡張型プロトタイプモデルである。

[85] 平鍋・野中(2013)によると、野中郁次郎と竹内弘高による 1986 年の論文“The New Product Development Game”において、日本の製造業の新製品開発からヒントを得て「スクラム」と名付けられ、その後、1990 年年代前半に、Jeff Sutherland、John Scumniotales、Jeff McKenna によって、ソフトウェア開発手法として開発されたという。

Takeuchi and Nonaka（1986）によると、新製品開発のプロセスを 3 つのタイプに分類しており、TypeA は工程が連続的にリレーしている方式、TypeB はプロセスの前後が重複した方式、そして TypeC はプロセスがよりオーバーラップした方式としている。このうち TypeC についてラグビーをもとに「Moving the Scrum Downfield」と名付けている。

programming）と呼ばれる、幾つかの軽量なソフトウェア開発プロセスの総称である。

Agile もある程度要件を決める必要があり、優先順位の設定や変更があるが、仕様について厳密な決定をせず、開発プロセスを進める中で仕様を擦り合わせ、ある程度決まった部分だけを先に開発していくスタイルである。Agile は、厳密な仕様を必要とする大規模な基幹システムなどには適していないが、中小規模のシステムなどに適しているとされる。

Waterfall Model に対し、Agile ではすべての機能を一度に取り込むのではなく、幾つかの機能を選択して開発を行う。さらに、その開発も 1 週間から 1 か月程度の短い期間で反復して行い、それぞれの反復期間の終了ごとにソフトウェアを本番稼働させることを目指している。反復作業も要件の中で最優先の高いものからできるだけ早く作成し、少しでもできあがったソフトウェアをユーザーに確認してもらうことで、早期にフィードバックを得て、再度反復作業に戻るのである。

Agile は、フィードバックといった顧客の参画の度合いが強く、そのため、人と人とのコミュニケーションやコラボレーションを重視している。さらに、変化を前提とし、開発前の仕様の固定を前提としておらず（独立行政法人情報処理推進機構, 2011）、完成してからソフトウェアを動かすのではなく、ある程度動くソフトウェアを成長させながら作成する、反復・漸進型な部分が大きな特徴である。

(3) 開発手法の貢献と限界

① Waterfall Model の適用と課題

ソフトウェアは、ユーザー企業が求めるものとともに変化し、より高度に複雑化してきており、処理の自動化や制御といった画一的で機械的なものから、意思決定[86]や戦略策定の支援[87]といった個々の企業に合わせたものへと成長してきた。ビジネスを取り巻く環境の変化により、ソフトウェア開発も組織戦略として取り入れてきた開発プロセスを順に追っていくような手法では対応できなくなってきた。これにより、特にこれまで主流として機能してきた、仕様を最初に確定させるという Waterfall Model の根本的な問題が明らかとなった。

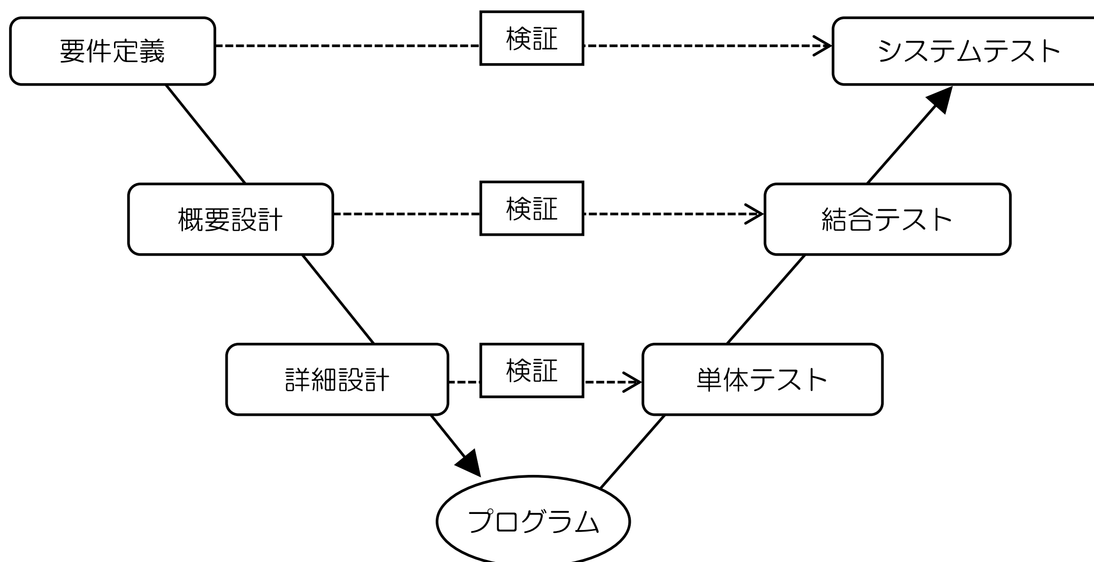
---

[86] 意思決定システム（Decision Support System）。1970 年代になると、企業の創造活動が重要視されるようになり、企業や組織の問題解決のための意思決定を支援するシステムとして作られた。この意思決定の支援が目的とされた理由として、企業の創造活動が重要視されるようになったことが考えられる（立川, 2003, 2005）。

2010 年以降では、IBM の人工知能による質問応答・意思決定支援システムである Watson が代表的である。

[87] 戦略情報システム（Strategic uses of Information Systems、もしくは、Strategic Information System）。1980 年頃から企業戦略として競争が重要視されるようになった結果、企業経営において、情報システムを管理的発想によるものではなく、自社に優位性の獲得を目的とする戦略的発想に基づいて構築されたシステム（Wiseman, 1988; 土屋,1990; 立川, 2005; 古殿, 2006）。

ソフトウェアは完全にできあがってからテストを行うため、テスト工程の最後のほうにならないと、設計や要件が正しく実装されているか検証することができないという限界が存在する（**図 5-4**）[88]。



出所：設楽・中佐藤（2012），一般社団法人情報サービス産業協会（2014, 2015）をもとに筆者作成

図 5-4 Waterfall Model による V 字型モデル[89]

Waterfall Model は厳格な手順を守ることを重視しており、正しいプロセスさえ踏めば正しい結果が得られることを想定している。つまり、「顧客の要求は、プロジェクトが始まる時点ですでに存在し、プロジェクトが完了するまでそれは変わることがない」[90]ことを前提としているのである。しかし、すでに述べたように、この方法では変化の速い市場には対応できず、さらに革新的なイノベーションも期待することができない。現実には、時間をかけても完璧な要件定義を行うことは不可能であり、そのような方法ではユーザーが満足するシステムを作成することはできない。

Waterfall Model を代表としたソフトウェア開発手法では、仕様に関わる重要な決定を開

---

[88] 各モジュール、部品のテストを行う単体テストや、そのモジュールや部品を繋げた結合テストも行われるが､すべてのモジュールを製品として完成した状態でテストを行うには、最後になってしまう。

[89] V 字型モデルは、開発工程（左半分）とテスト工程（右半分）を V 字型に整理したものであり、対応関係を矢印で示している。矢印の対応関係は、実施されるテストがどの開発工程の内容を検証するのかを示しており、最後のテスト工程（システムテスト）でようやく要件定義工程の内容が確認されることになる。

[90] 設楽・中佐藤（2012）p.10

発の序盤に要件定義工程として行う必要があり、仕様決定の遅れはコストの増大やスケジュールの遅延に直結する。特に下流工程で仕様変更が発生した場合には、いくつもの工程を巻き戻る必要があるため、多大なコストとスケジュール遅延が発生しやすい。

例えば、ソフトウェア開発を一括請負契約[91]で受注したような場合、仕様決定の遅れや変更が生じても、納期やスケジュール、請負価格が変わらないことも多い。そのため、納期やスケジュールを守るために、毎日の残業、徹夜、休日出勤などソフトウェア開発現場の労働環境を過酷にし、開発者を疲弊させる一因ともなっている。さらに、このような長時間労働は常態化することが多く、ソフトウェア産業のイメージを悪化させ、人材確保にも負の影響を与えている（高橋, 2010）。

また、ソフトウェア開発では、厳密に工程を分離することはできず、分業が曖昧で不十分である。しかし Waterfall Model に代表される手法は、各工程で 100%の進捗、完成を行い、仕様書や設計書などのドキュメントに基づいて、次工程に引き渡すことを前提としている。現実には、ソフトウェアの複雑性から完璧な設計書の作成は不可能であり、ドキュメントだけの工程の引継ぎでは不十分であり、各作業工程を跨った作業が発生する。つまり、Waterfall Model が想定しているような完成度の高い設計書は、ソフトウェア開発がすべて完了した後でなければ作成することはできないのである。

特に、高度なソフトウェアで作成された製品は、「機能と性能が複雑に組み込まれた構造となっており、それぞれの相関関係を見つけ出すことが困難」[92]であり、想定外の問題が発生することにより設計負荷や多大な工数が発生し、コストが高くなる要因となっている。

特に、仕様の曖昧さには明文化できないような暗黙の依存関係が存在する。そういった仕様に関する知識が暗黙的であると、外部に伝達することを困難となり、調整などのコストは高くなる（Langlois and Robertson, 1995）。そして、そのような暗黙的なものが急に費用として顕在化する可能性がある（峰滝, 2004）。

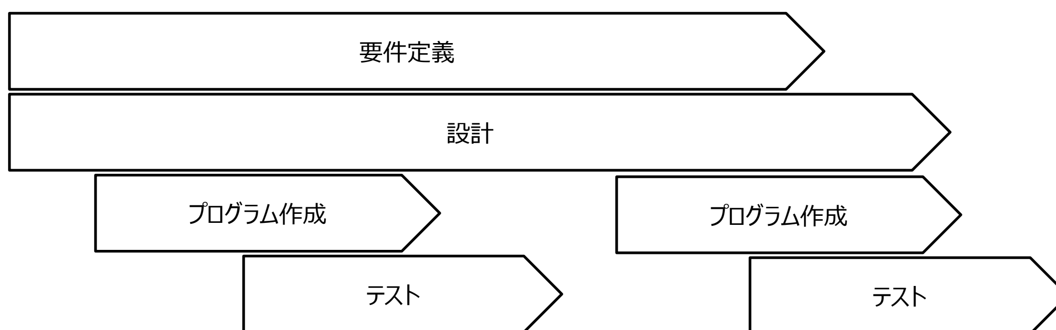
ソフトウェア開発も前工程、もしくは後ろ工程に何かしらの関連をもち、そのコミュニケーションといった人的部分に工程間の連携は大きく左右されてしまうのである。

こういった曖昧さのため、実際には、ある程度仕様が確定した時点や先行して、部分的に設計やプログラム作成、テストと開発を進めることも多く、製造業のコンカレントエンジニアリングのように、ある程度作業工程が重なることも多い（図 5-5）。

---

[91] ソフトウェア開発の要件定義から開発、運用・保守までの全工程を、契約時に見積もった期間と費用で、一度に請け負う方式。完成責任や瑕疵担保責任があり、ここまで述べてきたようなソフトウェアの要件の曖昧さのため、何をもって完成したのか判断がしにくく、受注側にとって不利になる場合も多い。

[92] 藤本・朴（2015）p.46



出所：Cusumano（2004）をもとに筆者作成

図 5-5 コンカレントエンジニアリングのようなソフトウェア開発

特にビジネスの高度、複雑化により、それを実現するソフトウェアも日々高度、複雑化しつつあり、仕様の確定はますます遅れている。もはや開発工程の序盤に仕様が確定することは困難であり、中盤や終盤になってからの仕様変更が前提となりつつある。

こうした仕様の決定について、Ferguson（1992）は工学的科学を例に取り上げ、その目的を「測定できる諸特性ー長さ、重さ、温度、速度等々ー間の関係を正しく示して、技術のシステムを数学的に解析できること」[93]であると述べている。その上で、設計の決定を下す際に理想化された過程を含む計算にどの程度近づけるかという決定は、幅広い知識に基づいて判断する必要があることを指摘している。

このように、仕様変更を極力防いで当初予定していた通りのソフトウェアを完成させるような開発プロセスである Waterfall Model は、工程間の技術情報の共有が不可欠であり、工程間の分割が難しいといった問題を抱えているのである。

② Agile の適用と課題

Waterfall Model は、もはや現在の高度に、かつ複雑化したソフトウェア開発には耐えられなくなり、代わりに登場した Agile といった新しい手法により、ビジネスや市場に合わせて臨機応変に仕様変更していくことが可能となった。また、最低限の仕様のみを決め、最速でビジネスを展開、ニーズに対応していくことも可能となった。

こうした Agile のアプローチに共通するのは、ソフトウェアに対する顧客のニーズを理解することは難しく、その技術の移り変わりも早いため、そのような複雑なソフトウェアを前もって完全に設計することは無理があるという点である。ユーザーからのフィードバックや市場の変化に応えるような開発中の設計の変更は、成功に結び付くようなものであれば、必ずしも悪いものではない（Cusumano, 2004）。

[93] Ferguson（1992）p.28

Agile は開発のスピードが上がるだけでなく、ある程度完成したものが見えてくるためユーザー企業の確認も行いやすく、仕様変更にも対応しやすいなどの利点がある。Waterfall Model は、工程の遡りが発生しないように必ず前工程が完了することが前提となっているため、それぞれの開発プロセスが長期化してしまうなどの問題がある。特に 2000 年以降、インターネット環境が整備されビジネスのスピードが上がるとともに、ソフトウェア開発にもスピードが求められており、開発するソフトウェアによっては Agile が採用される場合もある（表 5-2）。

表 5-2 Waterfall Model と Agile の開発スタイル等の違い

	Waterfall Model	Agile（Agile に共通するもの）
基本スタイル	予測型	適応型
計画	100%正確な計画を最初に立て、それに従う	正確な予測は不可能
変更	変更は（基本的に）しない	ユーザーの要望を満たし、良いものを作るために必要なもの
要求	最初に確定させて吸い上げる	変化するものとして、随時対応する
開発目的	最初の要求通りソフトウェアを作り上げること	変化するユーザーの要望を満たすこと
人的要因	人為的な部分を排除、管理する	人の要素を活かす
環境	安定した環境が前提	変化の多い環境
開発人数	数人から数百人、数千人（自社で足りない分は下請け企業等を導入）	10 人未満（コミュニケーションが取れる範囲内）
開発体制	階層型	チーム型（少人数）
開発期間	数か月から数年間	数日から数か月の繰り返し
製品のリリース	1 回のみ	最短最小のリリースを繰り返していく

出所：Glass（2006），平鍋・野中（2013）をもとに筆者作成

特に、ソフトウェアに対して、昨今のビジネス環境やマーケットの変化、さらに技術の発達や競合他社の戦略などに対応していくことが要請されており、そういった不確実なものへの対応が重要とされる。ソフトウェアを開発していく過程で、ユーザーからのフィー

ドバックに柔軟に応えていくためには、終盤でも設計を変更できるような開発プロセスが重要となるのである（Cusumano, 2004）[94]。

この Agile は、指揮命令系の管理手法とは根本的に異なる手法であるが、ソフトウェア開発の成功確率を劇的に高め、開発スピードと品質を改善する革命を起こしただけでなく、ラジオ番組の企画や、新しい機械の開発、戦闘機の生産、ワインの生産など幅広い業界や部門で広く取り入れられているという（Rigby, Sutherland, and Takeuchi, 2016）。

ソフトウェア産業以外の分野で Agile のような方法が取り入れられている理由は、イノベーションのためであり、決まりきった業務や作業工程にはあまり役立たないものの、こういった企業は非常に動きの激しい環境に置かれており、ただ新しい製品やサービスを開発するだけでなく、各部署の基本的業務にもイノベーションを起こす必要があるためである（Rigby, Sutherland, and Takeuchi, 2016）。

ただし、ソフトウェアの開発を行う際は、必ずしも特定の手法のみ有効ということはない。例えば、Agile で最も普及しているといわれる Scrum は、7 人前後の少人数のコミュニケーションを重視したチームを想定しており、Waterfall Model のような数十人から数百人、さらには数千人規模といった大人数による開発体制は想定していない。

たとえば、高木（2007）は北米では 4 人以下のスモールサイズの企業が多いことを明らかにしており、そういった地域では Agile による開発が適応されやすい側面もある。一方、日本の基幹系システムのような特に大規模な開発では、Agile のような少人数向けの開発スタイルは適用しにくく、その結果、Waterfall Model による手順を追った開発スタイルが中心となっている。

また、日本のソフトウェア産業では、開発プロセスを上流工程と下流工程に分業し、下請け業者が作業を請け負う階層型になっている。さらに、企業の人材の流動性の低さや、次章で説明する人月による工数計算や契約の問題などが存在しているため、Waterfall Model に適した産業構造となっている。Waterfall Model はその開発作業が工程ごとに分離されているため、ユーザー企業がソフトウェア開発企業へ委託する際、例えば、要件定義は準委任契約[95]で、設計工程は準委任契約または請負契約[96]で、プログラム作成工程は請負契約または派遣契約で行なうといったような、工程別に契約しやすくなっており（内布, 2005）、日本のソフトウェア産業の構造に適している。

---

[94] Cusumano（2004）は、Microsoft の Windows95 の開発を例にあげ、プロジェクトの終盤に大きな変更を加え、ブラウザーを追加することで、OS 市場の獲得に繋がったことを指摘している。

[95] 準委任契約では、受注側に業務を処理することが約束される。請負契約と異なり、完成責任はなく、瑕疵担保責任もなく、そのため、要件定義のように仕様が確定しにくく、成果物が完成しにくい工程に適している。

[96] 請負契約では、受注側に成果物を完成させることが約束される。つまり、設計書やプログラムを完成させる必要があり、さらに、それら成果物が問題なく動作する瑕疵担保責任がある。

一方で、Agile はイノベーションを目的としており、決まりきった業務や作業工程にはあまり役には立たず（Rigby, Sutherland, and Takeuchi, 2016）、Agile の開発方法をこのような日本のソフトウェア産業の開発スタイルに単純に適応させることは難しい[97]。

また、Agile は、ユーザーからのフィードバックが必要となるが、そのためには、ユーザー自身にも自身のビジネスは当然ながら、ソフトウェアに関するある程度の知識が必要となる。しかしながら、日本のユーザー企業は、そういった IT に関する専門的な技術者を確保していることは少なく、そのため内製ではなくアウトソーシングを行い、その内容も IT ベンダーに任せてしまうことが多く、Agile が日本に適応できない理由の一つとなっている。

さらに、Waterfall Model は上流工程からのトップダウン型であり、Agile は下流工程からのボトムアップ型と捉えることができるが、そういったボトムアップ型の開発を行うには、そこに裁量権が必要となる。しかし、日本の分業構造は下請け構造になっており、下流は上流に対する裁量権を持っておらず、それゆえボトムアップ型の Agile に対応ができないともいえる。

West and Grant（2010）の調査によると、アメリカを中心とした世界のソフトウェア開発では、この Agile と反復型モデルをそれぞれ 35.4%、20.6%が採用しており、合計すると 56%に達する。一方、Waterfall Model はわずか 13.4%のみという調査結果が出ており、「特定の開発手法を採用しない（Do not use a formal process methodology）」の 30.6%よりも低い（図 5-6）。

Rigby, Sutherland, and Takeuchi（2016）は、Agile は万能薬ではないが、ソフトウェア開発の現場で発生することが多い次のような環境において、効率的にかつ簡単に導入できる開発手法であるとしている。

「解決すべき問題が複雑で、当初はソリューションが不明で、しかも要求される製品仕様は変更される可能性が高く、仕事を分解してモジュール化でき、エンドユーザーとの緊密な共同作業（と彼らからの素早いフィードバック）が可能であり、上意下達の軍隊型チームよりクリエイティブチームのほうが概してよい結果を生む―そのような環境である」[98]。

このように、Waterfall Model もしくは Agile か、または双方を混在した手法が適しているのかについては、プロジェクトの規模や開発対象のソフトウェアの内容に起因するため、

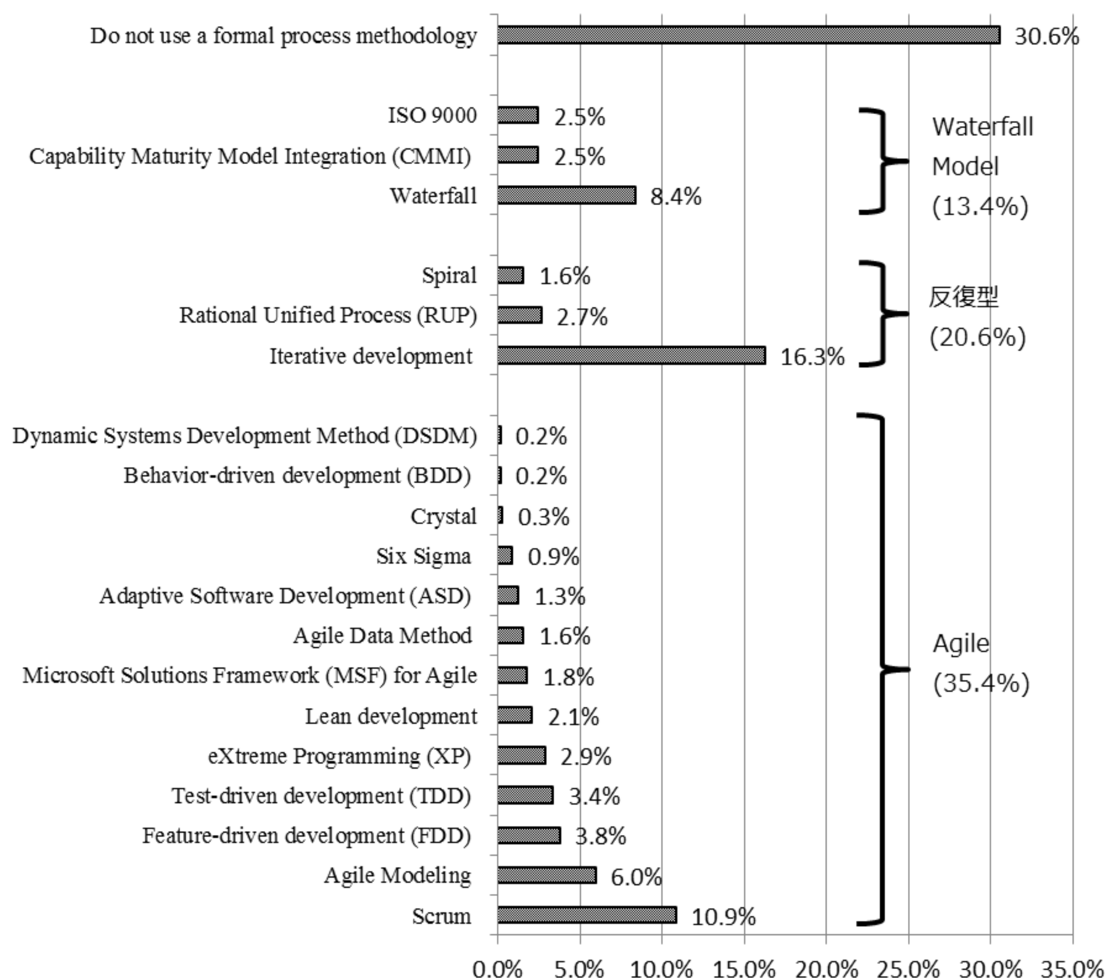
---

[97] 一般社団法人 PMI 日本支部アジャイルプロジェクトマネジメント研究会（2015, 2016）の調査によると、日本国内で Agile による開発手法を導入しているプロジェクトの割合は約 3 割とされている。

ただし、Agile による開発手法導入の実態調査のため、Waterfall Model の導入率についての回答はなく、また、「アジャイルプロジェクトの実態」というテーマで調査を行っているため、Agile に関心の高い回答者、もしくはすでに Agile を利用している回答者が多いと予想され、Agile の導入率も高い結果となっている可能性がある。

[98] Rigby, Sutherland, and Takeuchi（2016）pp.97-98

どちらの手法が正しく、優れているといったものではない。また、実際のソフトウェア開発の現場では、厳格な Waterfall Model、もしくは Agile を採用するのではなく、それぞれを柔軟に採用することも多い。すべての企業のニーズに合致するような唯一のアプローチは存在せず（Cusumano, 2004）、双方の視点を持ち、それをバランスよく組み合わせる方法が重要なのである（Glass, 2006）。



出所：West and Grant（2010）p.2 をもとに筆者作成

図 5-6 開発プロセスの採用（2010 年）

(4) 小括

本章では、これまでのソフトウェア開発がどのように行われてきたのか、また今日どのように行われているのか、検討するとともに、1970 年代より現代に至るまで中心を担ってきた Waterfall Model とそれに取って代わりつつある Agile について、その開発プロセスの特徴と限界を検討してきた。

ソフトウェア開発では、計画とその管理を重要視しており、その代表的な開発手法がWaterfall Model である。この手法は、ソフトウェア開発に一定の成果を上げてきたが、そのような計画重視の方法では、技術の発展によるソフトウェアの大規模・複雑化と、ユーザーのビジネス環境変化の速さに適応できていない。

このような管理を主体としたソフトウェア開発に対し、変化を前提とした、ビジネスや市場に合わせて臨機応変に仕様変更していくことが可能な Agile のようなソフトウェア開発プロセスが生まれてきたのである。

ソフトウェアの開発手法は、この Waterfall Model と Agile を中心に行われているが、日本のソフトウェア産業は Waterfall Model に偏重している実態が存在する。Waterfall Model は日本のソフトウェア産業と非常に相性が良く、分業構造や次章で説明する工場型の開発モデル、工程別の契約方法に適している。また、日本のソフトウェア開発の分業構造は、トップダウン型の Waterfall Model に適した下請け構造となっており、下流工程は上流工程に対する裁量権を持っていない。そのため、ボトムアップ型の Agile のような方法には対応できていない。

ソフトウェアを含む IT 産業は変化が激しい分野であり、積み重ねてきた知識といった経営資源が 1 年後には活用できなくなる可能性がある。このことは、ソフトウェア産業が、Waterfall Model を中心とした製造業の開発プロセスや手法を援用し、仕様を最初に確定させ、後の工程をアウトソーシングするようなこれまでの方法が、根本的な問題を抱えていることに繋がる。

次章では、日本のソフトウェア産業に焦点をあて、同産業に特徴的に見られる分業構造を踏まえ、Waterfall Model を中心とした分業構造と相性が良く、かつて日本のソフトウェア開発で中心的存在を果たしたソフトウェア・ファクトリーを取り上げ、その貢献と限界について述べる。

第6章 ソフトウェア産業の下請け構造と製造工場化

先行研究の検討において、Cusumano（1991, 2004）や今井他（1989）、妹尾（2001）らが明らかにしてきたように 1970〜1990 年代にかけてソフトウェア開発手法の中心にソフトウェア・ファクトリーと呼ばれる工場型の開発モデルが存在した。この開発モデルは製造業のライン生産方式の工場を模しており、日本のソフトウェア産業の Waterfall Model を中心とした分業構造と相性が良く、当時一定の成果を上げていた。しかしながら、欠点も多く、特にソフトウェア開発の下流工程を標準化した単純労働に置き換えるような土壌の形成にも繋がっていたのである。

本章では、日本のソフトウェア開発の下流工程が標準化した単純労働に置き換えられていることの問題に論点をあてている。このような日本のソフトウェア開発にみられる特徴的な分業構造を確認し、ソフトウェア・ファクトリーの貢献とその限界、さらになぜ日本企業がこのソフトウェア・ファクトリーのような開発方法を採用しようとしてきたのかについて検討する。

(1) ソフトウェア産業の成り立ち

1968 年に NATO が開催したソフトウェア・エンジニアリング会議で、コンピューターの高性能化とソフトウェアの複雑化からソフトウェア危機が叫ばれ、ソフトウェア工学の必要性が指摘された。その約 20 年後、Brooks[99]（1986, 1995）が、ソフトウェアの開発が高度に複雑化していくのに対し、これらをすぐに解決し、生産性を高めるような手段は存在しないことを指摘した[100]。それからさらに 20 年以上の歳月が過ぎたが、現在も Brooks が指摘したようなソフトウェア開発における多くの問題は解決できていない。

ソフトウェア産業が、なぜ現在のような分業構造を取るようになったのか、その要因の一つに、ソフトウェアがコンピューターを含むハードウェア産業から分離、独立してきた経緯が存在する。そこで、今日の受託ソフトウェア産業における開発プロセスの問題がいかなるものであるのか、なぜそうした問題が生じるに至ったのかを理解するため、本節ではまずこのハードウェアとソフトウェアの関係を整理するとともに、今日のソフトウェアがどのように発展してきたのか、歴史的背景を確認する。

---

[99] IBM の SYSTEM/360 の開発者で科学者。

[100] Brooks は、その著書『人月の神話』（Brooks, 1986, 1995）において、中世ヨーロッパで銀が魔除けの効果があると信じられ、狼男や吸血鬼、悪魔などを撃退できる武器として銀の弾丸や銀のナイフが作られた信仰になぞらえ、ソフトウェアの開発のさまざまな問題を解決するような「銀の弾丸」は存在しないと説いた。この主張は、ソフトウェア工学のコミュニティで多くの反響を呼び、ソフトウェア工学の古典として現在も読み続けられている。

①　ハードウェアからの独立

ソフトウェア産業は、ハードウェアであるコンピューターの発展とともに成長してきた。特にソフトウェアは、一時期世界のハードウェアの 70%以上を占めた IBM の影響が大きい（今井・安藤・白井・辻・久保・玉置・浜田, 1989; Cusumano, 1991, 2004; 杉山, 2011）。このような IBM を中心としたコンピューターの発展やソフトウェアの歴史については、情報処理学会歴史特別委員会（2010）や一般社団法人情報サービス産業協会編（2013）がまとめているほか、情報産業の価値がハードウェアからソフトウェアにシフトする研究として今井・安藤他（1989）や杉山（2007, 2008, 2009, 2011）、朴・藤本（2016）が詳しい。

これら先行研究によると、コンピューターの歴史は、1939 年にアメリカで ABC[101]と呼ばれる世界初のコンピューターの登場から始まった。1946 年にアメリカ陸軍の弾道計算用に開発された ENIAC[102]は、その後水爆の実験などに利用されており、当時のコンピューターは第二次世界大戦などにおける軍事利用が中心であった。

1949 年になると、現在のコンピューターの基礎となったノイマン型コンピューターと呼ばれる EDSAC[103]が世界で初めて開発された。しかし、当時のコンピューターは、現在のパソコンのように持ち運びが可能な程小型で性能が良いものではなく、大型で、かつ高価なものであり、まだ一般企業に普及するようなものではなかった。

第二次世界大戦を経て、コンピューターが商用として利用され始めたのは、1950 年頃といわれている。

アメリカでコンピューターが商業的に利用され始めた 1950 年頃には、コンピューターは科学技術計算を目的としており、企業の給与計算や売上集計など日常的業務のデータ処理のほか、トランザクション処理[104]として利用されていた。企業の業務は、コンピューターに処理させるために標準化が進められ、OS により制御を自動化することで成果を上げていった。

このようなコンピューター産業の発展に対し、ソフトウェア産業は、1964 年にアメリカで IBM のメインフレームが登場してから産業の体をなしたとされる。IBM は、1911 年に創業したが、コンピューター事業に参入する以前は、パンチカード関連事業に注力し、独

---

[101] Atanasoff-Berry Computer。ABC は一部未完成で稼働実績もなかった。また、当時のコンピューターは真空管の配列、配線の物理的な変更により計算を行っていたため、汎用性に欠けていた。

[102] Electronic Numerical Integrator and Computer。ABC のような専門計算機とは異なる汎用計算機。長らく ENIAC が世界最初のコンピューターとされてきたが、その特許を巡る裁判により、ABC が先に存在することが認められた。しかし、この ENIAC が、現在のコンピューターの基礎となるプログラム内蔵型の EDSAC の開発に繋がっていった。

[103] Electronic Delay Storage Automatic Calculator。プログラム内蔵方式として現在のコンピューターの基礎となったノイマン型と呼ばれるコンピューター。

[104] TPS（Transaction Processing System）。関連した複数の処理を一つにまとめて処理する方式。

占的地位を築いていた。1950 年代当時はまだハードウェアが IT ビジネスの中心であり、ソフトウェアは付属品であった。当時の IBM は、創業して以来、パンチカードの機器を販売せず、賃貸し、その使い方や保守を指導することでレンタル料金を得ていた。IBM はこのビジネスモデルをコンピューターにも適用し、IBM を中心としたアメリカのコンピューターメーカーは、ハードウェアや OS、ミドルソフトウェアを提供し、顧客固有の業務システムについては顧客自身が開発するという役割が分離した分業モデルが続いていた。

1960 年代になると、コンピューターの性能が上がるにつれて、大型のホストコンピューターの利用が進んでいった。さらに、コンピューターの稼働に対する信頼性が増すことで、オンラインを利用したシステムが実用化された。この頃、パンチカードシステムで独占的地位を確立していた IBM は、計算機などのコンピューターの開発にビジネスモデルの移行が成功し、独占的地位の確保へと繋がっていった[105]。ただし、IBM は当時大型コンピューター市場で多くの利益を上げていた一方、個人用コンピューター市場には参入していなかった。そのため、IBM は個人用コンピューター市場には遅れて参入することとなり、十分な製品開発期間を確保できず、くわえて、部品を新しく自社内部で開発、調達しようとすると高くついたため、アーキテクチャーのみを作成し、それ以外の CPU や OS などの開発はアウトソーシングせざるを得なかった。その結果、OS を Microsoft に、CPU を intel にアウトソースすることで、それら企業の成長に寄与するとともに、特定業界ごとの製品はそれぞれの専門メーカーが独占するようになっていったのである（Cusumano, 2004）。

1967 年時点で、世界のコンピューターの 70%以上が IBM のマシンに占められていた（杉山, 2011）。しかし、一つの企業による世界市場の寡占支配は競争を阻害し、技術の発達を遅らせる危険性がある。このような IBM による独占的支配の状況に対し、1969 年にアメリカの司法省により、独占禁止法違反の訴訟が起こされた。この訴訟に対し、IBM は直ちに和解を行い、ハードウェアとソフトウェア、教育サービス、エンジニアリングサービスを分離する価格分離政策を発表した。これにより、ソフトウェアは無償の物から、独立した有償な物、つまりソフトウェアはハードウェアとは独立した商品という認識ができあがったのである。

② 日本のソフトウェア産業の独立

日本でも 1970 年代から 1980 年代にかけて、ハードウェアとソフトウェアなどの価格分離が進展した。また、1980 年代半ばには、ユーザー企業の多くが、コンピューター利用の技術や経験を自社以外にも活かそうと、情報処理部門を子会社として独立させていった。しかしながら、1990 年代になると、コンピューターが高度に複雑化したことや、ユーザー

---

[105] IBM の SYSTEM/360 が商用的に成功した互換アーキテクチャーとなる。同一のアーキテクチャーを採用するコンピューター間で、同じソフトウェアを動作させることを可能にした（杉山, 2007, 2009）。

企業が自社の情報部門を子会社として切り離してしまったことにより社内での開発能力が大きく落ち込んだため、ユーザー企業は自前でシステム開発を進めることが難しくなった。その結果、ユーザー企業に代わって一からシステム開発を請け負う、システムインテグレーターと呼ばれる企業が出てくるようになった。

日本におけるコンピューターの商用利用は、1960 年頃稼働した旧国鉄の座席予約システム[106]や、同じく 1960 年代に稼働し始めた銀行の ATM などが代表的であり、これらシステムはリアルタイム・オンライン・システムと呼ばれた。1960 年頃から銀行などを中心に、メインフレームと呼ばれる大規模な基幹業務用のシステムが利用され始め、1980 年頃まで全盛期が続いた。特に、1960 年代末には、ソフトウェア開発の環境が整備されることで、ソフトウェア業務が拡大していった。しかし、その後オープン化[107]やダウンサイジング[108]により、メインフレームは現在のようなパーソナルコンピューターに代替されていった[109]。1990 年以降、コンピューターの性能は飛躍的に伸び、一般企業で使用する分には性能過多になっている。そのためハードウェアであるコンピューターより、システムを動かしその品質を左右するソフトウェアの重要性が増している。

現在も日本のカスタムソフトウェア開発に関わる企業は、主にメーカー系、ユーザー系（商社系）、独立系の 3 つに分類される。日本のソフトウェア企業の約 99%は従業員 500 人未満の中小・零細企業であるが、その大半がメーカーやユーザー系列とは資本関係を持たない独立系に分類される。

丸尾（2008）は、ソフトウェア企業に中小企業が多く、さらにその多くが下請けになっていることについて、最終的なアウトソーシング先が、個人や零細企業であることが多く、成果を個人の能力に依存している点が特徴的であると指摘している（図 6-1）。その上で、ソフトウェアの開発は製造業のような生産設備として巨大なものが必要でないため、既存資本ではない新興企業を中心に発展していることをその要因としている。

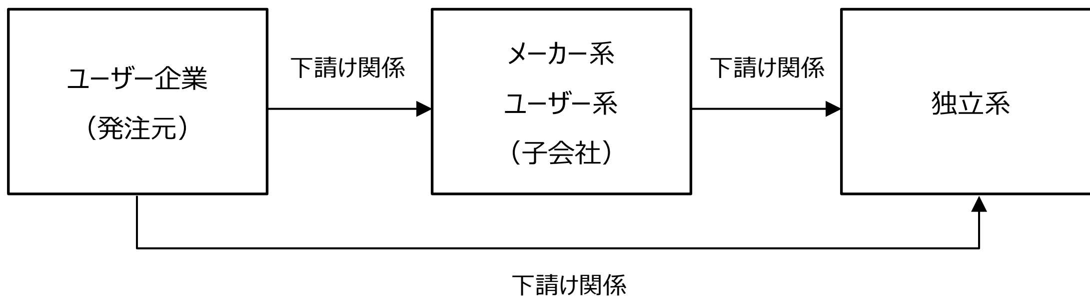
---

[106] この座席予約システムは、現在も「みどりの窓口」で利用されており、日立製作所が作成し、旅客販売総合システム「Multi Access Reservation System：MARS（マルス）」と呼ばれている。従来の鉄道の座席の予約は、人の手によって行われてきた。しかし、旅客数や路線、列車本数が増えるに従い、鉄道の座席の予約は人の手では限界が生じ、このマルスシステムにより膨大な数の座席をリアルタイムで予約できるようになったことで、大幅に業務が効率化された（鉄道情報システム株式会社, 2016）。

[107] 企業が、自社のソフトウェアの仕様やサーバーの接続方法などを公開することで、複数のメーカーが対応する製品を作れるようになること。これまでのシステムでは、互換性のない独自仕様が多く、一度開発を委託した業者にしかその後の保守を任せられなかった。しかし、オープン化により、柔軟なシステム構築が可能になり、特定のメーカーによる制限がなくなったことで他の業者も参入できるようになった（青木, 2004）。

[108] 大型のメインフレームなどの汎用コンピューターやシステムなどを、小型化、軽量化すること。技術の進歩により小型コンピューターの性能が上がったことで可能となった。

[109] メインフレームは現在も、銀行などの基幹系システムとして残っており、無くなったわけではない。



出所：丸尾（2008）をもとに、筆者作成

図 6-1 ソフトウェア開発の請負・下請け関係

ここまで、ソフトウェア産業の歴史を辿ってきたが、ソフトウェアはハードウェアを製造していたメーカーにより、ハードウェアに付随するもの、つまり無料のものとして作られてきた経緯が存在するのである。また、日本では、ユーザー企業のソフトウェア開発力が落ち込む代わりに、システム開発の元締めとなるシステムインテグレーターが登場し、そのような企業を中心とした中小 IT ベンダーの下請け構造が形成され、ソフトウェアをアウトソーシングする体制、慣習が構築されてきたのである。

(2) ソフトウェア産業の下請け構造

① 内製とアウトソーシング

カスタムソフトウェアの開発方法のひとつとして、内製するのではなく、外部の専門企業にアウトソーシングする方法があげられる。

ソフトウェアの開発に限らず多くの産業では、その複雑な作業工程を分割、分業することで成り立ってきた。複雑なシステムで構成されている作業は、組織の階層構造や分業制度によって負荷を削減し、複雑な作業をモジュール化し分業することで、個人での処理や外部の企業にアウトソーシングすることも可能となる。また、企業が特定の業務を専門の外部企業に任せることで、優れた技術の利用や時間の短縮、費用の節約が期待できる。特に、分業により自社はコアとなる業務の強化に集中できることが、アウトソーシングの利点とされ、ソフトウェア開発でアウトソーシングが行われる要因の一つとなっている。

ソフトウェア開発を製造業のように取り扱う企業では、要件定義や設計を分離し、下流工程に位置するプログラミング作成やテストを子会社やインドなどの海外の契約企業にアウトソースしてきた（Cusumano, 2004）。下流工程を外部に任せることで、自社で不足する技術者の確保やコストの削減を図ってきたのである。

日本のソフトウェア開発では、一企業に限らず、複数の企業と開発を行うための取引を行ったり、下請け企業として何層にもわたるような取引を行ったりするケースも存在する。

特に、ソフトウェア産業は、資本がほとんど必要なく参入障壁が低いことから新規参入がしやすい。そのため、競合企業が多く、競争が激しいことが特徴的である。さらに、開発の委託先として、中国やインドといった海外にアウトソーシングを行うこともあるが、そこでは文化の違いなどにより摩擦が生じており（大場, 2011）、その調整にかかわる費用も大きい。

一方で、より企業のビジネスに合わせた高度で複雑なソフトウェアが必要となれば、そのための専門家、専門企業と契約する必要があり、アウトソーシングしたとしてもその費用を削減できない可能性がある（今井他, 1989）。

このようなアウトソーシングによるソフトウェア開発の人材が必要とされる理由の多くは、人材確保や開発コストの削減であり、企業規模が大きいほどその利用率も高く、その需要は高まっている。例えば、ソフトウェアが完成してシステムが本番稼働した後も、細かな変更など機能の拡張のバージョンアップや保守作業のため、継続的に技術者を確保する必要がある。さらに、その保守作業も高度なプログラムで開発されたシステムの場合、その保守要員もそれに対応できる人材が必要となる。

しかしながら、日本では、企業の人材の流動性の低さから、優秀なソフトウェア開発の技術者であっても、海外ほど市場に流出していない。また、技術者の待遇も決して良いとはいえず､年収の面でもアメリカと比べると半分以下に抑えられており（経済産業省, 2017）[110]、このような状況の中、インドや中国などの新興国の IT ベンダーは、競争力を高め、規模を急速に拡大してきている。これまで発展途上国といわれていた多くの国々も、高収入が得られる海外向けのソフトウェア開発に積極的になってきており、なかには日本語教育を積極的に行うことで日本に対するサービス提供能力を高め、日本市場へ進出する企業も存在する。

ソフトウェアは当初はユーザー企業で内製されることが多かったが、現在のようにアウトソーシングが活発になった理由として､前述の 1980 年代半ばに企業のシステム部門が情報システム部として独立したことや、取引コスト（Coase, 1937; Williamson, 1975, 1985）の問題が考えられる。

今井他（1989）は、この当時のユーザー企業が内製することが多かった理由について、コストの視点から指摘している。今井他によると、ソフトウェアの市場が未発達でソフトウェア関連企業も未成熟なことにより、良質なソフトウェアの供給が乏しく、そのため取引コストが高くなり、内製するほうがマネジメントコストも低かったため、アウトソーシングされる割合は小さかったと述べている。しかしながら、ソフトウェア開発企業と市場

[110] 経済産業省（2017）の IT 関連産業の給与等に関する実態調査によると、単純な比較はできないものの、日本では 30 代の給与の平均が 526 万円、最大値が 1250 万円であるのに対し、米国では 30 代の給与の平均が 1,238 万円、最大値が 4,578 万円となっており、倍以上の開きがある。

規模の整備が進むことで、取引コストが低下し、さらに多様で高度なソフトウェアへのニーズが増大し、そのためのマネジメントコストが増大することで、ソフトウェアの開発も内部組織から市場へ移行していったとも述べている。

特に、製品開発のように、その内容や成果の見極めが難しく不確実性が高いものの場合、適切な相手企業を見つけて取引や行動を監視するよりも、その業務を内部化したほうが効率的となる（武石, 1999）。

一方で、今井他（1989）は、ソフトウェア開発のうちプログラム作成のような仕事は、企業の本業とは直接関わっておらず、他の部門と有機的に結びつかず、そのマネジメントコストも高くなるため、市場が積極的に活用されるとも述べている。

このように、プログラム作成のような下流工程は本業と直接関わらず、さらに上流工程からの指示に従って遂行される作業工程として標準化した単純労働に置き換えられ、その作業は代替可能な労働と位置付けられており、社内の技術者の人材不足やコスト削減のため、企業は積極的にアウトソーシングを進めているのである。

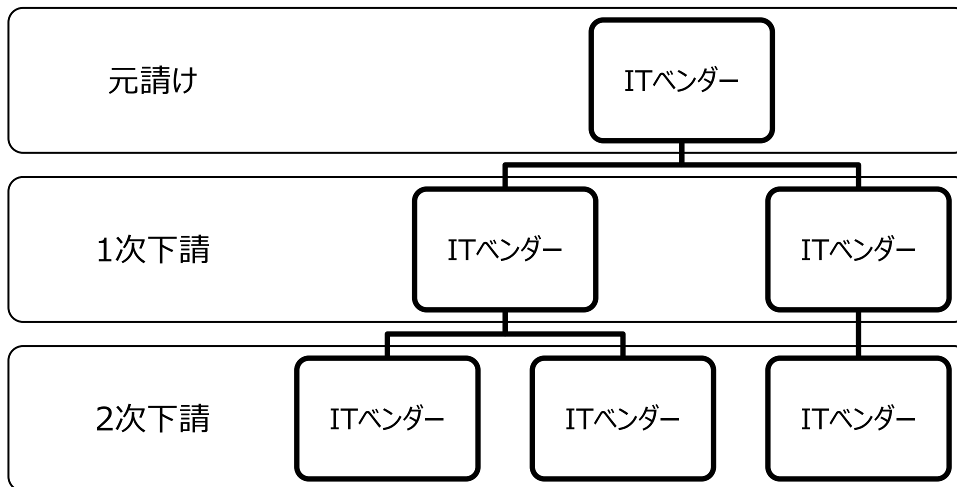
②　下請け構造

ソフトウェア開発のアウトソーシングについて述べてきたが、この受託を中心とした日本のソフトウェア開発の優れている点は、顧客をしっかり確保できていれば、きわめて安定的な企業経営ができる点にある（丸尾, 2008）。一方で、大幅な付加価値を付けた見積もりはできず、社内の改善活動や技術的な発展により多少利益率を上げることができたとしても、顧客の仕様に沿って開発をする限り、利益はほぼ一定となる。利益の絶対額を上げるには、技術者を増やし、顧客からの発注を増大させ、売り上げを上げるしか方法がない。

さらに、2001 年頃の IT バブルの崩壊以降、ユーザー企業は発注を抑え、システム開発の見積額も抑える傾向にある。ここに日本のソフトウェア開発の労働集約型の弱点がある。売り上げを増やすために技術者を雇用すれば、それだけ人件費がかさむことになる。そのためにも、余剰な人材は社内に抱え込むのではなく、下請け企業[111]を活用することとなる（図 6-2）。

一方で、ソフトウェアを含む IT 産業全般で先行しているアメリカでは、ソフトウェア産業の成り立ちで説明したように、ソフトウェアはハードウェアの付属品であったものの、価格分離政策によりソフトウェアとハードウェアの販売が分離されるとともに、ソフトウェアはハードウェアとは独立した商品であるという認識が存在している。そのため、数多くのプロフェッショナルサービス企業が種々のソフトウェアを開発、販売しており、日本のようなアウトソーシングを中心とした下請け構造を形成していないとされる。

[111] 下請けという言葉を直接使わず、協力会社やパートナー会社などといわれる場合もある。



出所：筆者作成

図 6-2 ソフトウェア産業の下請け構造

このような産業構造の違いについて、高木（2007）は、日本とアメリカやカナダといった北米の情報サービス産業に関する公的な統計調査の比較を行ない、北米では日本より産業の細分化が進んでおり、4 人以下のスモールサイズの企業が多いことを指摘している。また、北米では国外での売上が多いほか、日本の研究開発投資が約 1%なのに対し、北米では 57%の企業が総売上の 10%以上の研究開発投資を行っており、さらに 27%の企業が総売上の 20%以上の研究開発投資を行っていることを明らかにした。そのうえ、興味深いことに、カナダのエドモントンで行なったアンケートとインタビューによる実地調査では、下請けビジネスの存在が確認できなかったと述べている[112]。

しかしながら、日本の雇用制度では、正社員を解雇することは容易ではない[113]。ソフトウェア産業では、労働集約型の産業であることから設備投資とは異なり、社員の稼働率を低下させることで生産調整をするような方法が取りづらい。その結果、親企業、グループ企業間で技術者を融通し合うケースが多く、中小規模のベンダーが乱立する要因の一つとなっている。くわえて、ソフトウェアの開発工数や工程に応じた人材配置となり、企業規模に応じた人材格差が付きやすくなっている。そのため、同一のソフトウェアを設計する

[112] 高木（2007）によると、カナダのエドモントンでは、ソフトウェア開発に関して請負業務が事業の主体ではなく、そのため下請けというビジネス概念自体も一般的にはなっていないのではないかと述べている。

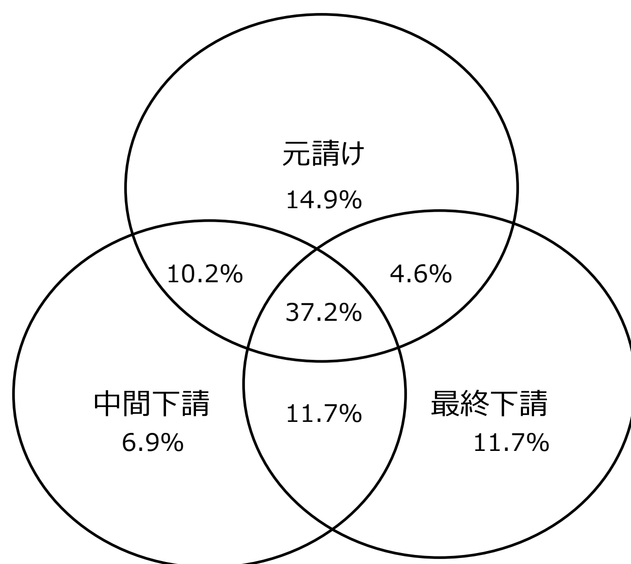
[113] このような日本の下請け構造に対し、アメリカの企業では社内に開発チームを所有した内製が中心となっている。日本とアメリカにおける労働環境の違いがあるが、アメリカのこのような社内の開発チームはプロジェクトごとに集められ、プロジェクトが終了すると解雇されるが、その分高い報酬が支払われており、高品質のソフトウェア開発が行われている点で日本と大きく異なっている（高橋, 2010）。特に、こうした他国に比べて契約を介さないソフトウェア開発が多いため、ソフトウェアの変更に対応しやすい、つまり計画を変更しやすいことが指摘される（情報処理推進機構, 2011）。

技術者であっても、所属企業の規模によって単価や賃金に差が出てしまう。そのうえ、日本の IT ベンダーは、1 人あたりのマージンが低いため、利益確保のためには社員の稼働率を高く維持する必要がある。

このようなソフトウェア産業の下請け構造は、その不透明な取引構造や産業構造が問題になっており、このような下請け構造からの脱却が、健全なソフトウェア産業の発展を進め、ユーザーの要望に応えるために重要なこととなっている。しかし、この構造は業界が数十年以上かけて解決できていない問題でもある。

このように、ソフトウェア産業は重層的な下請け構造で構成されているが、実際には、元請け、中間下請け、最終下請けのそれぞれ立場でのみ委託取引を行っている割合は少なく、一つの企業が複数の立場として業務を行っている場合が多い。

「情報サービス産業の委託取引等に関する調査研究報告書」(経済産業省商務情報政策局情報処理振興課, 2005)によると、調査企業の 37.2%が元請け、中間下請け、最終下請けのすべてを担当しており、最終下請けを行わずに元請けと中間下請けを行っている企業が 10.2%、逆に元請けを行わずに中間下請けと最終下請けを行っている企業が 11.7%となっているという(図 6-3)。



出所:「情報サービス産業の委託取引等に関する調査研究報告書」(経済産業省商務情報政策局情報処理振興課, 2005)p.17 をもとに筆者作成

図 6-3 ソフトウェア産業の取引構造

IT ベンダーは、企業の規模や開発するソフトウェアの規模との関わりで下請け構造の立ち位置は大きく異なる。中小 IT ベンダーでも、ユーザー企業が中堅や中小企業でその開発の規模も小さいような場合には、元請けの立場になることも多い。しかし、大手企業がユ

ーザーの場合は､開発規模も大きくなりやすく､そのため大手 IT ベンダーが元請けとなり、中小 IT ベンダーはその下請け企業になるケースが多いといえる。

③　人月による工数計算

ソフトウェア開発が Waterfall Model による工程別の契約が行われてきた背景には、人月と呼ばれる計算でその開発量と契約を結び付けている点が大きい。

ソフトウェア開発では、1 人が 1 か月で行うことのできる作業量を表す人月計算が蔓延しており、開発者を代替可能な作業者とみなすことを助長してきた。ソフトウェア開発に支払う費用は、本来であればソフトウェアの業務的な価値やその質、用いられた技術などに基づくべきだが、人月計算では、開発に要した人的コスト、つまり作業の量で見積もられてしまうのである（Brooks, 1975, 1995）[114]。ソフトウェアの価値やその品質の優劣は、その開発に要した時間や技術者といった人員コストと等しいとはいえない（ソフトウェア産業研究会, 2005）。

Brooks（1975, 1995）は、この人月計算による見積もりについて、コストを中心に算出されたものであり、労力と進捗を混同しており、人と月を置き換え可能なものとみなしてしまう危険な神話にすぎないと述べている。そして、ソフトウェア開発のスケジュールの遅れに対し、この人月計算に基づいて単純に人員を増やすことで対応しようとするため、その追加人員の配置や教育による作業の中断や、相互連絡の増加により、さらなるスケジュールの遅延を招いてしまうと､「ブルックスの法則」を定義している。

このブルックスの法則にある通り、ソフトウェア開発は部品間の調整や擦り合わせが必要であり、その構築の複雑さのため、システムエンジニアなどの技術者間のコミュニケーションが重要となるため、技術者を単に増やすような労働集約的な方法では生産性が低下してしまうのである（Brooks, 1975, 1995; 峰滝, 2004, 2005）。

一方で、このような人月計算によるソフトウェア開発の見積りが行われている理由として､ユーザー企業や IT ベンダーが人月計算による見積り手法に頼り切っていることが考えられる。

ユーザー企業にはソフトウェアの価値の算出が難しく､IT ベンダーの技術者への評価も難しい。その反面、この作業時間や人員コストによる見積りは一見わかりやすいため、IT

[114] あるソフトウェア開発の工数を 100 人月とした場合､1 人の技術者がそのソフトウェアを開発するために 100 ヶ月かかる計算であり、10 人であれば 10 ヶ月の期間を要することになる。一方で、この計算に基づけば、極端には 100 人月に対し、10 人で 10 か月かかる仕事が 100 人の技術者が確保できれば 1 ヶ月でそのソフトウェアの開発を終えられるという理論が成り立つ。しかし、ブルックスの指摘するように、労力と進捗を混同し、人と月を置き換え可能なものとみなしており、現実的には作業の段取りや工程手順などもあるため不可能である。また、この人月による工数の考え方は、技術者の技術レベルの高低などの能力も考慮されていない。

ベンダーからも説明しやすい。さらに、プロジェクトが予定より長期化した場合や作業量が増えた場合でも、その超過分の対価が計算しやすく、請求しやすいのである。

特に、ソフトウェア産業は比較的新しい産業のため新規参入がしやすく、多くの企業がひしめき合っているが、技術的な部分での差別化ができていない。そのうえ、一品一品を作り上げる受注型の開発であるため、発注側であるユーザー企業も価格を決めづらい。そのため、ユーザー企業が IT ベンダーの評価をすることができず、ソフトウェアも極端に高価格や低価格になる場合もあり、マーケットメカニズムによる最適な資源配分が実現していない。また、このような作業時間によりソフトウェアの価値が決まることで、IT ベンダー側はソフトウェア開発を短期に終了させるインセンティブが弱く、ユーザー企業は必要以上の費用が請求される恐れがある。このため、プロジェクトを短期で終了させられるような高レベルの技術者の投入が抑制され、代わりに技術レベルが低い技術者が投入される恐れがあり、完成するソフトウェアの質にも影響を及ぼすのである（ソフトウェア産業研究会, 2005）。

亀川・青淵編著（2009）によると、IT ベンダーは一度企業に納入したシステムの修理や運用・保守などが主力業務となり、顧客とその顧客企業内の課題を共有する機会が少ないのが現状であり、そのためデータウェアハウス[115]やビジネスインテリジェンス[116]といったシステムの提供ができない原因を指摘している。さらに、IT ベンダーは協力会社に対するプロジェクトマネジメントが中心となり、技術力や業種・業務ノウハウを磨く努力や機会を失ってしまうケースも見られ、その品質管理や進捗管理は未熟であり、今後改善の余地があるとしている。

一方、アメリカでは、1990 年代後半以降に創立された企業にとって、ソフトウェア開発はビジネスに密接に関わるものとなっており、ソフトウェア開発の形態も IT 部門によるプロジェクト単位の開発ではなく、ライン部門によるサービス含むプロダクト開発に移行しつつある。（独立行政法人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, 2011）。

工藤（2009）によると、日本のソフトウェア開発は、長期的観点で利益を出すことや、大手企業を顧客に持つ相乗効果により他社で利益を確保するという考え方が強くあり、個々のプロジェクトの採算を無視して受注に走るケースもあるという。

特に、日本のユーザー企業は、導入実績を重視し、他社の IT 導入状況や売上高等の企業の規模を参考にして IT ベンダーを選ぶ傾向が多い。こうした IT サービスの事業規模を競

[115] Data Warehouse（DWH）。さまざまな情報システムのデータを時系列に蓄積したデータベースやその運用システム（IT 用語辞典 e-Words）。

[116] Business Intelligence（BI）。企業に蓄積される膨大な業務データを経営者や現場のスタッフなど利用者自らが分析、加工し、業務や経営の意思決定に活用する手法（IT 用語辞典 e-Words）。

うあまり、売上高を重視している IT ベンダーもある。IT ベンダーが売上高を上げるためには、技術者の月額契約単価か、技術者の稼働人数を上げる方法しかない。しかしながら、技術者の単価は顧客の説得が必要であり、簡単には上昇させることはできない。むしろ、単価は下降する一方となっており、IT ベンダーは技術者の人数を増やし、企業規模を大きくすることで売上高を確保してきたといえる。しかし、技術者の人数を増やしすぎると、需要が減った際に人員に余剰が出て経営の負担となってしまうため、無尽蔵に増やすことはできない。また、日本の年功序列型の賃金制度から、技術者が高齢化するにつれて原価が上がり、生産性が下がってしまうリスクも存在する。

このような問題のため、IT ベンダーは自社の技術者の人数を増やすのではなく、重層的な下請け構造のもと、労働集約的なソフトウェア産業においてバッファー的に利用し（浅沼, 1997）、下請け企業の技術者を大量に活用することで、技術者の人数を確保するとともに売上を伸ばす施策が行われてきた。特にシステム構築・開発の全体を通して一括してサービスを提供するシステムインテグレーターは、社員 1 人あたりの売上高を上げるため、できるだけ多くの下請け会社の社員を使ってプロジェクトを推し進める必要がある。日本の受託ソフトウェア産業は、こうした下請け構造のもと労働集約型産業となっているのである。

(3) ソフトウェア・ファクトリー

Bean（2005）は、ソフトウェア開発のアウトソーシングが企業のイノベーションの能力と競争優位を失うことであると批判した。Bean によると、顧客ニーズの変化に柔軟にかつ迅速に対応し、革新的なソフトウェアを作成するという行為は、組み立てラインでできるようなことではなく、設計とデザインが重要であることを指摘しているが、日本のソフトウェア産業はそうした Bean の主張とは逆の方法を取り入れようとしてきた。

本節では、日本でかつて導入されていたソフトウェア・ファクトリーの貢献とその限界について、さらになぜ日本企業がこのソフトウェア・ファクトリーのような開発方法を採用しようとしてきたのかについて検討する。

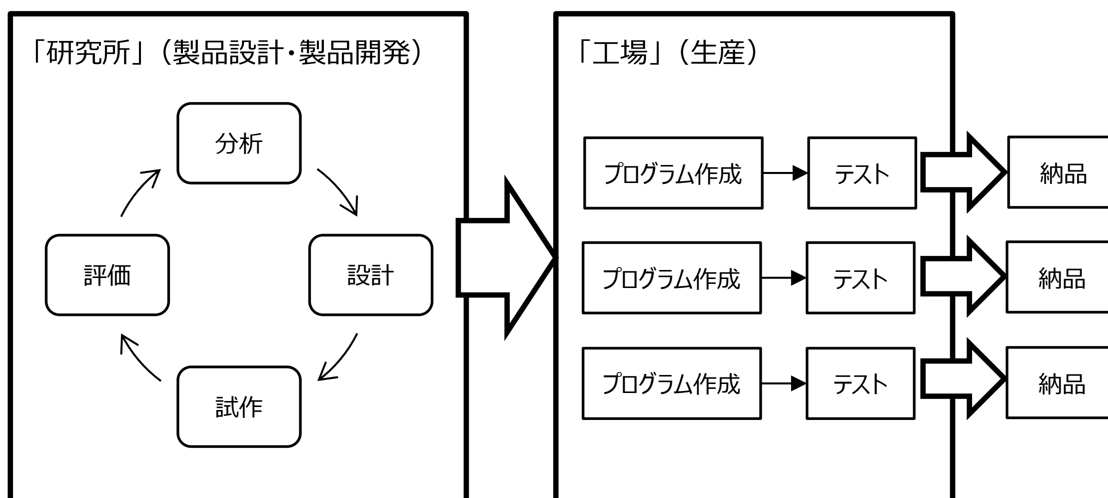
① ソフトウェア・ファクトリーによる成果

ソフトウェア・ファクトリーは、何千人というプログラマーやシステム技術者を一か所の施設に集め、「研究所」がソフトウェアの試作やその開発を行い、それを「工場」が生産する企業内における分業方式である（図 6-4）。

ソフトウェア開発を複数の工程に分離し、作業を定型化、開発プロセスを標準化することで品質を保証していくこの試みは、QC サークル活動[117]なども取り入れられた。さらに、

[117] Quality Control。品質管理の活動を職場内の小グループで自発的に行う活動。

このQCサークル活動は、日本のチーム型のソフトウェア開発に普及し、その高品質稼働が高く評価され、品質管理などの国際標準規格であるISO[118]や組織的能力の成熟度モデルであるCMM[119]へと発展し、現在にも受け継がれている（妹尾, 2001; 保田・大石・吉田・山田, 2001）。



出所：Cusumano（2004）などをもとに、筆者作成

図 6-4 ソフトウェア・ファクトリーのイメージ

日本のソフトウェアは、元々コンピューターなどのハードウェアメーカーによって作成されてきた経緯があり、1990年代まで、そうした高価なハードウェアに付随するおまけとしてソフトウェアも一緒に作成されてきた。そのため、ソフトウェアの開発は、ハードウェア、つまり製造業の品質管理や標準化などの開発手法がそのまま適用されてきた影響が大きい（高橋, 2010）。

ソフトウェア・ファクトリーはこのような組織形態のもと、プログラムの再利用なども取り入れられ、「安定した開発プラットフォーム、特定の開発言語、トレーニングされた自組織のソフトウェア開発者、画一的な開発プロセスという工場的アプローチの前提条件」[120]が揃った場合に非常に有効であった。

1960年代のアメリカのGEやAT&T、1970年代の日本のコンピューターメーカー、1980年代の Microsoft は、ソフトウェア開発をより扱いやすくなる方法を模索してきた

[118] International Organization for Standardization。品質管理などの国際標準規格。ソフトウェア関連としては、ISO 9126、ISO 25010などがある。
[119] Capability Maturity Model。品質・生産性の向上などのために、カーネギーメロン大学のソフトウェアエンジニアリング研究所が開発した組織的能力の成熟度モデル。ソフトウェア開発をアウトソーシングする際のツールとしても利用される。現在は、これをさらに統合、発展させたCapability Maturity Model Integration（CMMI）が存在する。
[120] 居駒（2011）p.15

（Cusumano, 2004）。その中で、日本の大手ソフトウェア企業は、コンピューターなどのハードウェアを製造していたメーカーやその子会社や関連会社が多く、ハードウェアで成功を収めた品質管理手法をソフトウェア開発に導入しようとしてきた（**表 6-1**）。

表 6-1 コンピューターメーカーのソフトウェア・ファクトリーへの取組み

年	ソフトウェア・ファクトリーへの取組み
1969 年	ソフトウェア工場（日立製作所）
1970 年	日立ソフトウェアエンジニアリング[121]　設立（日立グループ）
1976 年	ソフトウェア戦略プロジェクト（日本電気）
1977 年	府中ソフトウェア・ファクトリー（東芝）
1979 年	システム本部（富士通）

出所：大西（1998）をもとに筆者作成

1970 年から 1980 年代にかけて日立製作所や富士通、東芝、NEC などの大手コンピューターメーカーは、大規模システム開発の問題を解決すべく、工業製品などの製造業の開発技法や原則をソフトウェア開発に厳格に適用し、工場型の生産方式であるソフトウェア・ファクトリーを確立しようと試みていったのである（Cusumano, 1991, 2004）。

Cusumano（2004）は、1970～1980 年代当時の日本では、コンピューターや情報システムに関する大学教育が貧弱であり、それゆえ富士通や NEC といったソフトウェア企業が従業員の大部分を OJT として教育する必要性が生じたことを指摘している。その結果、ソフトウェア・ファクトリーによる標準化した開発手法やライブラリの再利用、コンピューターの支援ツール、社内の教育制度といったものがソフトウェア企業のニーズに適合したと述べている。

ソフトウェア・ファクトリーのこの手法は、プロジェクトからプロジェクトへ多くの共通性を引き継ぎながら開発が進んでいくため、大規模な業務用システムの構築に効果を発揮した。特に、標準化された設計パターンに基づいて、元の条件からほとんど変わらないようなカスタム、またはセミカスタム・アプリケーションを大量生産することに向いているため、類似したシステムを以前よりも少し安く、簡単に構築することに都合がよく、2000 年代に入っても活躍しているという（Cusumano, 2004）。

② ソフトウェア・ファクトリーの欠点

ソフトウェア・ファクトリーのような組織構造は、導入前と比較し、一定の基準においい

[121] 現日立ソリューションズ。2010 年に同じ日立グループのシステムインテグレーターの日立システムアンドサービスと合併して称号を変更した。

て非常に良い成果を発揮してきた。Cusumano（1991, 2004）も、日本のコンピューターメーカーは IBM に追従してメインフレームを開発してきたが、そういった企業にとって、1970～1980 年代の市場の変化や成長が比較的安定していた時代に、このファクトリー型アプローチがうまく機能したと述べている。一方で、パーソナルコンピューターなどの変化のテンポが速いものには適用が困難であり、より柔軟性の高い手法の考案が必要であるとも述べている。

しかしながら、このソフトウェア・ファクトリーは「工場」のソフトウェア技術者を製造工場の労働者に見立て、均一なスキルを前提としており（居駒, 2011）、開発といった頭脳はすべて「研究所」に集約された、Taylor（1911）の科学的管理法のような頭脳労働と手労働を分離する方法を採用しており、想像力豊かで才能のあるプログラマーにとってかえってその環境は順応しにくい。そのうえ、こうしたソフトウェア・ファクトリーで作られるソフトウェアの多くは、特定の顧客向けのものであり、設計上のイノベーションといえるようなものも少ない（Cusumano, 2004）。そのため、「工場」の技術者による改善やイノベーションは期待できず、さらにそこで従事する技術者は熟練を必要とされないことでその技術も一定のレベルに留まってしまう恐れが考えられる。

くわえて、2001 年頃の IT バブルの崩壊以降、顧客ニーズはコスト削減のシステム開発から、競争優位のための戦略的なシステム開発へと多様化している。そこでは要求される技術のレベルや変化のスピードが高まるとともに、多様なプラットフォームや言語、海外も含めた開発体制といった変化も起こり、ソフトウェア・ファクトリーによる開発プロセスの標準化を前提とした類似製品の大量生産のような手法は適用できなくなってきた（居駒, 2011）。

Bean（2005）も、ソフトウェア開発をコモディティ化させようとした取り組みとして、この日本のソフトウェア・ファクトリーを取り上げ、多くのプログラマーを投入したものの革新的なソフトウェアを作成することができず､失敗してしまったことを指摘している。

以上のように、ソフトウェア・ファクトリーの導入はハードウェアメーカーによって主導されてきたが、それまで無秩序であったソフトウェア開発を工程として分離し、定型化することで一定の標準化に成功した。しかし、ビジネスの変化に適応できず、現在はこのソフトウェア・ファクトリーよりも有効な手法が存在しており（Cusumano, 2004）、21 世紀に入ってからはその方面に力を入れている企業は少なくなってきていると考えられる[122]。

---

[122] 日本のソフトウェア・ファクトリーに関する研究は非常に少ない。学術情報検索データベースである CiNii で「ソフトウェアファクトリ」、もしくは「ソフトウェア工場」のキーワードで先行研究を調べたところ、論文タイトルや要約のみの検索となるが、2016 年 1 月末時点でその本数は 100 本未満であった。さらに、そのほとんどはソフトウェア・ファクトリーがもてはやされた 1990 年前後に集中しており、2000 年以降の論文は数本程度しか確認できなかった。

一方、ソフトウェアの「自動化」に関する研究は多く、国内論文を CiNii で検索したと

ソフトウェアの開発プロセスを製造業の方法から模倣しようとしたソフトウェア・ファクトリーは、類似製品の大量生産のような手法を目指したがために、21 世紀の魅力的な機能を持つ革新的なソフトウェアの開発を目的とした新たな流れに耐えられなくなったのである。

(4) 小括

本章では、日本のソフトウェア産業の下請け構造を確認するとともに、1970〜1990 年代にかけてソフトウェア開発手法の中心にあったソフトウェア・ファクトリーの貢献と限界を検討してきた。

日本のソフトウェア産業は、ソフトウェア・ファクトリーにより開発を複数の工程に分離し、作業を定型化、開発プロセスを標準化することで高い品質を保持してきた。市場の変化や成長が比較的安定していた時代には、この工場モデルは合致していたが、変化の速い市場に対応できず、さらに革新的なイノベーションを期待することができなかった。しかし、日本のソフトウェア産業は、下請け構造のもと労働集約型産業となっており、技術者の人数を確保するとともに売上を伸ばす施策が進められた。その結果、下流工程は上流工程と比べて知的な、創造的な能力が必要とされる労働として位置づけられず、プログラム作成は定型的な組立作業のように捉えられ、標準化した単純労働に置き換えられる要因の一つとなってきたのである。

次章では、本章で検討したソフトウェア･ファクトリーの貢献と限界から、そうした構造に基づいてプログラム開発を行うことがいかなる問題をもたらしうるのか、こうした問題の克服において従来の分業構造を再編する必要があるとすればそれはどのような再編であるのか、こうした論点に対する分析的視点および枠組みを検討する。

ころ、内容は精査していないが、単純に 500 本以上の論文などが確認できている。この自動化の研究は、工場的な生産としてではなく、自動化することによる技術者の開発における負荷の削減やそのサポート、生産性の向上の方面へシフトしていることが考えられる。

第7章 ソフトウェア開発プロセスにおける知識労働

本章では、ここまで述べられてきたソフトウェア開発で生じるさまざまな問題に対し、その解決を図ろうとしたソフトウェア工学について、その取り組みと限界を検討する。その上で、ソフトウェア・ファクトリーのような開発プロセスに基づいたソフトウェア開発がいかなる問題をもたらしうるのか、こうした問題の克服において従来の分業構造を再編する必要があるとすればそれはどのような再編であるのか、こうした論点に対する分析的視点および枠組みを検討する。

(1) ソフトウェア工学

ソフトウェアはコンピューターとともに発展してきており、その歴史は60年程だが、近代化とともに大規模・複雑化し、それに対し、製造業や建築業の知見や手法を取り入れながら発展してきた（表 7-1）。

特にハードウェア産業からの影響により、これまで手工業的で個々の技術者に任せられがちであったソフトウェア開発は、ある程度の秩序を持つようになり、製品の品質管理、生産性、再利用率が向上していった（妹尾, 2001; 中所, 2014）。

製造業などの手法が取り入れられた理由として、品質管理手法やソフトウェア開発の各工程を順に追っていく Waterfall Model に取り入れやすいことがあげられる。また、オブジェクト指向技術の導入とともに、建築家の著作[123]を例にとって、ソフトウェア・アーキテクチャー構築のデザインパターンの研究も進んでいる（妹尾, 2001）。

コンピューターが登場して間もない第二次世界大戦後の1950年代から60年代にかけて、ソフトウェア開発の主体は個人であった。特に初期の頃は、仕様書もないままプログラムを記述し、問題が発生したらその都度修正するような、統制や規律のない手工業的な開発が行われていた（妹尾, 2001; 中所, 2014）。その後、コンピューターの信頼性が増すにつれてオンラインを利用したシステムが実用化され、日本では銀行の ATM や旧国鉄の座席予約システムなどが開発された。しかし、当時はソフトウェアを利用できる企業はそれほど多くなく、その理由としてコンピューター本体が数億円近い高価格なものであったため、一部の限られた企業しか導入することができなかった。

1960 年代後半から 1970 年代に入ると、コンピューターの高性能化、大容量化に伴い、ソフトウェアも大規模化していった。このソフトウェアの大規模化に対して、バグなどに対する障害の防止や開発スケジュールの管理などがより求められるようになっていった。

[123] Alexander 他（1983）『パタン・ランゲージ—環境設計の手引』（邦訳）。建築分野の設計において、問題と解決方法を数百種類のパターンに識別し、再利用しようと試みた。

表 7-1 ハードウェア・ソフトウェア開発環境の変化

年代	ハードウェア・環境の変化	ソフトウェア開発への影響
1950年代〜	コンピューターの登場	ソフトウェア創世記。 ソフトウェアの開発の主体は個人。仕様書もないままプログラムを記述し、問題点は都度修正する、統制や規律のない手工業的な開発手法。
1970年代〜	コンピューターの高性能・大容量化	ソフトウェアの大規模化、複雑化。 個人の手工業的な開発手法では耐えられなくなり、品質（バグ等の障害の防止）やスケジュールを管理していく必要性が発生。
1990年代〜	コンピューターのダウンサイジング・高性能化（PC）・ネットワークの普及	PCやインターネットの普及により、ユーザーのソフトウェアの利用が活発化。 開発期間の短期化と外部との関係の緊密化が求められるようになった。
2010年代〜	インターネットの高速･大容量化、 クラウドサービスの普及 スマートフォンの普及	クラウドを利用したソフトウェアサービスの利用が増加。 業務と情報技術の一体化の組織的・体系的取り組みが求められるようになった。

出所：妹尾（2001），立川（2003），古殿（2006），野田（2006），島田・高原（2007）をもとに筆者作成。

このソフトウェアの開発過程を構造化して管理するフレームワークとして、ソフトウェア工学[124]が存在する。本節では、このソフトウェア工学の役割と、日本におけるソフトウェア工学の取組みについて検討する。

[124] Software Engineering。1968年にNATOがソフトウェア・エンジニアリング会議を開催した。会議ではコンピューターの高性能化とソフトウェアの複雑化からソフトウェア危機（software crisis）が叫ばれ、その対応としてソフトウェア開発を工学的な観点から方法論や開発論を整備することが必要だとされた。日本でソフトウェア工学の導入が強く求められたのは、それから10年以上経過した1980年頃となる（Tomayko and Hazzan, 2004; 中所, 2014）。

①　ソフトウェアの複雑化と危機

1968 年に NATO が開催したソフトウェア・エンジニアリング会議で、コンピューターの高性能化とソフトウェアが複雑化していくことの問題についてソフトウェア危機が提唱され、将来のソフトウェア開発に対する危機意識が高まり、ソフトウェア工学の重要性が認識された。ソフトウェア工学は、ソフトウェア開発の工程を細分化することによってそれぞれの工程を定量化しようとする試みである（野田, 2006）。1968 年当時のソフトウェアは急速に拡大、複雑化していたため、将来的に技術者の不足と従来の手工業的な開発手法では限界が来ることから、ソフトウェア工学の必要性が高まっていた。

ソフトウェア工学が誕生したことで、工程アーキテクチャーに基づいてソフトウェアの開発プロセスは工程ごとに分離して厳格に管理されるようになっていった。

1970 年頃のコンピューターメーカーは、ロックイン戦略を採用しており、一度あるコンピューターメーカーのハードウェアとそれに合うソフトウェアを導入すると、そのユーザーはそのソフトウェアを継続的に使用せざる得ない状況となっていた。これにより、IBM や日本の富士通、NEC、東芝といったコンピューターメーカーは、そのユーザーと独占的に取引することができ、安定的な利益の確保が期待されていた（高橋, 2010）。

しかし、このような特定のハードウェアのために開発されたソフトウェアは、他のメーカーのハードウェアで使用できないため、各メーカーのハードウェアに適合するソフトウェアをそれぞれ開発しなければならず、独自のソフトウェア製品の開発への障害ともなっていた。

1990 年代になると、コンピューターのダウンサイジング、パーソナルコンピューターの高性能化、ネットワークの普及が進んだ。特に 2000 年以降は、インターネットが爆発的に普及したことにより、ユーザーのソフトウェアの利用が活発化している。技術の進歩や市場の変化により、ソフトウェアがかつて以上に陳腐化し易くなり、開発期間が短期化されてきた。また、今までのようなシステム開発だけでなく、コンテンツの開発も同時に求められており、ユーザーなど外部との関係の緊密化が求められてきた。例えば、最小のシステムを開発し、一般ユーザーにサービスを利用してもらい、その反応を見て徐々にコンテンツを追加、変更し続けていく手法も登場した。特にインターネットを利用した Web 系システムは速さが重視されており、機能を絞ってリリースを最優先に行うことが多い（妹尾, 2001）。

一方で、ソフトウェア開発は製造業の品質管理などの工学手法を取り入れてきたが、下請け構造のもと属人的で労働集約型産業の特徴が非常に強い側面を残しており、ソフトウェア工学の導入はまだ多くの余地が残されている。

こうした状況に対し、ソフトウェア工学は社会科学的に成功事例を整理したものにすぎず、工学のベースとなる尺度が不十分のため、いまだ工学の体をなしていないとの批判がある（阿草, 2009）。

今井他（1989）も、ソフトウェア工学の貢献は既存の道具の改良にすぎず、生産性の向上に成果を上げていないとし、ソフトウェア開発は依然として職人技に留まっていることを指摘している。くわえて、ソフトウェアを分散して開発するようなことについても、1人で処理できない作業を人海戦術で行おうとしていることと大差ないと指摘している。そのうえで、ソフトウェア工学に対し、工学を名乗っているものの実際には定量的なものがなく、そのため基礎的な理論と実務への適用、さらにはその誤差の範囲といったものが明確にできておらず、工学でありたいとする願望の現れにすぎないと指摘している。

このような厳しい指摘があるとおり、半導体の設計や生産、バイオ・テクノロジーの開発などの他の分野では、製造や設計プロセス、精密に系統立てられた作業が成し遂げられてきたが、ソフトウェア産業ではまだその域に達していないといえる（Cusumano, 2004）。

② 日本におけるソフトウェア工学

ここまでソフトウェア工学の問題が指摘されたが、日本におけるソフトウェア工学の導入は特に遅れている。1968 年の NATO の会議におけるソフトウェア危機の宣言により、1970 代から 1980 年代にかけて世界的にソフトウェア・エンジニアリング研究がピークを迎えたが、日本国内では関心が示されなかった（有賀, 2008）。

情報通信産業のうち、コンピューター産業に関しては、国策産業として保護育成が行われてきた。一方で、ソフトウェアなどの情報サービス産業については、その歴史が浅いこともあり、コンピューター産業と比べその重要性への認識が遅れており、本格的な政策も遅れている（田中, 1988a)。そのため、日本のソフトウェア・エンジニアリングの研究は遅れ、学生などの人材を十分教育しなかったため、その後ソフトウェア業界は、急増するソフトウェア開発のニーズ対応することができず、専門的な教育を受けていない学生も技術者として採用することで、質よりも量で勝負する体質が浸透してしまった（有賀, 2008）。

さらに、日本のソフトウェア産業では、下請け企業をプロジェクト費用のコストという視点で見ており、ソフトウェア開発の実作業の下支えをしている下請け企業の技術者がレベルアップできるような施策を採ってこなかったことが IT サービス全体の生産性が向上しない要因の一つにもなっている（工藤, 2009）。その結果、情報サービス・ソフトウェア産業の従業者数は 2000 年代に急増したものの、質までは確保できていない状況となっている。

このような日本のソフトウェア工学の導入状況については、角埜・椿・鶴保（2007）が、ソフトウェア工学と企業の収益性の関係を調査し、人材育成や開発技術、プロセス改善などの取組みが事業収益の向上に有効であることを指摘している。また、伊藤（2009）は、静岡県内のソフトウェア開発企業を対象としたアンケートを実施し、多くの企業が生産管理などの重要性を認識しているものの、その生産性の測定や開発技法の導入までには至っておらず、現時点では結局のところ個人スキルに依存していることを明らかにしている。

ソフトウェア開発の個人差として、水野（1982）は標準的な個人差としても約 10 倍の差があり、下限まで含めると 25～30 倍の個人差があると述べている。また、Cusumano（2004）も、生産性の格差について、10～20 倍にもなると述べている。そのほか、多くの開発者が共同で作業を行う際に利用される統合開発環境[125]を利用したソフトウェアに含まれるバグの混入率を分析したところ、5 倍以上の個人差があり、多くの開発者によって変更が加えられたモジュールほどバグが混入されやすいという研究結果も出ている（松本・亀井・門田・松本, 2010）。

このように、ソフトウェア開発は労働集約型の特徴を強く持っており、製造業などを参考にした科学的な品質管理手法の導入が重要とされ、それにより日本のソフトウェア産業にも一定の成果を上げてきた。その一方で、日本のソフトウェアの開発現場には個人のスキルに強く依存した部分が依然として多く残っており、さらにこの品質管理手法なども、十分に浸透したとはいえず、未だ成熟していない。

高橋（2010）は、日本のソフトウェア産業について、他国と比べ製造業の手法に強い影響を受けたという点で、その特殊性を指摘している。高橋によると、一般に日本の製造業は、独創性よりも生産システムの改善によって国際競争力を確保してきたため、大手メーカーがソフトウェア開発に対して、製造業の手法を模倣したことは当然であり、その結果、創造的なソフトウェア開発の育成を阻害する要因となったと述べている。その上で、現在の日本のソフトウェア産業では、世界に通用するような創造的なソフトウェアの開発を行っている企業は少なく、日本のソフトウェア企業の売上高に占める研究開発費[126]の割合は、2000 年代では 1%以下と低い水準であったことを指摘している。

特に、IT 産業のような技術が重要視される産業では、高い技術力や専門能力を持つ人材を育成、確保しなければならない。しかし、技術革新の激しさや長期的な技術発展の予測の難しさ、景気変動による人材供給の調整などの経営環境の変化もあり、そういった高度な専門能力を持った人材を長期に育成、留保することが難しくなってきている（Cappelli, 1999; 若林, 2008）。

一方、水野（1975）は、ソフトウェアが人間を中心とした長期の組織活動によって開発されるということに対する認識の不十分さを指摘しており、金を出し、人を集めるような

---

[125] Integrated Development Environment。ソフトウェア開発に必要な多数のツールを一つにまとめ、画面操作などによって利用できるようにしたもの。これにより、大規模、複雑化したソフトウェア開発に伴う作業負荷を減らすことが可能となった。Microsoft の Visual Studio シリーズなどが代表的である。

[126] ソフトウェアは、購入や自社開発といった取得形態ではなく、その目的で 3 つに区分されている。自社利用や販売目的といった制作目的に応じて受注制作のソフトウェア、パッケージ等市場販売目的のソフトウェア、社内管理等自社利用のソフトウェアに区分され、それぞれの会計処理が定められている。研究開発費は、従来にはない製品やサービスに関する発想を導き出すための調査と探究、新しい知識の調査・探究の結果を受けて製品化または業務化などを行うための活動が含まれる（日本公認会計士協会, 2011）。

方法だけでは決して良いソフトウェアは生まれないと述べている。そのうえで、優れたソフトウェアの開発にはその構成要素に対する深い知識と経験が必要であり、単なる机上の知識だけでは使いやすく効率の良いソフトウェア開発はできないとし､職人的な面の下で、泥臭い作業の果てしない連続の中からソフトウェアが生まれてくると述べている。

ソフトウェアを開発するには、そのプログラミング言語の知識を習得することが必須である。しかし、これは最低限の条件であり、そのプログラミング言語を駆使してプログラムを作成するために、さまざまな技法が必要となる。そういった技法を会得するには、多くのプログラムを作成し、開発経験を積むと同時に、優れた開発者の指導や優れたプログラムに触れることが重要であり、そのような経験を経てソフトウェア開発能力を習得していくのである（竹田, 2005）。

ソフトウェア工学は、プログラミング作業で扱う領域に偏ってしまっているが、工学として本来求められるものは、プログラミングに先立つ問題整理などの理論化やその支援ツールを生み出すことである（今井他, 1989）。特に、ソフトウェア開発の問題整理に関わる作業は技術者の知的な部分に依存しており、そうした知的な部分、創造的な能力を強化することが重要となる。

以上より、ソフトウェア工学は、高性能化、複雑化、そして巨大化するソフトウェア開発の状況に対し、無秩序な手工業的な開発から脱却させ、科学的な知見に基づいた開発を行うことを可能にすることで、品質の向上などに一定の貢献を果たしてきた。一方で、ソフトウェア工学は、いまだ系統立てられた工学としてのレベルに達していないとの指摘もあり、さらにソフトウェアを低コストでスケジュール通り、かつ効率的に開発し、保守していく方法に重点が置かれたことで、ソフトウェア開発の製造業化を進めていったのである。

(2) ソフトウェア開発プロセスにおける知的側面

① ソフトウェア開発の価値と評価

前章でソフトウェア開発のアウトソーシングが進展していることを述べたが、多くの IT ベンダーやユーザー企業が、ソフトウェアを服や玩具、スニーカーのようなものと考え、そういったものが海外で製造されていることから、ソフトウェアも労働力の安い海外で作成するほうがコストも安く済むと考えることは想像に難くない（Bean, 2005）。

事実、単純なソフトウェアの場合、例えば画面に文字を表示するといった仕様や設計の擦り合わせが少ないものや、より定型化された作業を含むものであれば、機能ごとにモジュール化が行われておりアウトソーシングの対象となってきた。

Waterfall Model に代表されるように、ソフトウェア開発は各工程を順に追っていくため、上流工程と下流工程に分離する方法に適応しやすかった。また、開発プロセスを分離することで、下請け企業や海外のソフトウェア企業などへのアウトソーシングも行いやすくな

った（高橋, 2010）。一方で、こうした分業構造によって、下流工程が上流工程からの指示に従って遂行される作業工程として捉えられる傾向にあり、下流工程の作業は代替可能な労働と位置付けられる要因の一つともなってきた。

このようなアウトソーシングは、ソフトウェアの開発方法を工場の作業のように設計とプログラム作成に分離することになり、Waterfall Model と相性がいいが、試行錯誤を経て魅力的な機能を持つような革新的なソフトウェアを開発する方法としては最良ではない（Cusumano, 2004）。

また、アウトソーシング先として、以前利用したことがない企業やスタッフを利用することは、ソフトウェア開発の品質や納期についての不確実性を高めてしまう（佐野, 2001）。

Bean（2005）は、このようなソフトウェア開発におけるアウトソーシングを批判し、開発を組織から離れた場所に移すことについて、作業コストの削減や作業の高速化といったオペレーションの効率と、長期的な競争優位を作り出す戦略とを混同しており、その方法は過ちであると主張している。さらに、Bean は、ソフトウェア会社にとって競争優位となるものは革新的なソフトウェアを作成する能力であり、そのようなソフトウェアを作成するという行為は開発とデザインのスキルが必要とされ、組み立てラインでできるようなことではなく、アウトソーシングしてしまうことは企業のイノベーションの能力と競争優位を失うことになると指摘している[127]。

特に、日本のソフトウェア産業は、欧米と比較して無形資産としてのソフトウェアの価値が低いことが特徴的である。データでできているソフトウェアは、物理的な制約のあるハードウェアと異なる部分も多い。ソフトウェアは、製造業の工業製品とは異なり目で見ることができず、特にカスタムソフトウェアは 1 品 1 品が異なる受注生産であるため、評価が難しく、また、品質の差がわかりにくい。このため、ソフトウェアはハードウェアの付随物としての値引きの対象となり、結果的に無償、あるいは低価格で販売されることとなり、マーケットメカニズムによる最適な資源配分が実現していない。ソフトウェア開発を担う IT ベンダーの多くは中小企業であるが、参入障壁が低いことと、下請け構造において下流工程に位置することが多い（丸尾, 2008）。さらに、ソフトウェアはデータでできているため、品質や技術的な違いがわかりにくく、それゆえ他の企業との差別化がしにくい状況となっている。日本において、ソフトウェア開発は、低価格、もしくは無料のものとして扱われ、その価値が正当に評価されておらず、正当にその価格付けもされていないのである（Porter・竹内, 2000）。

高橋（2010）は、このようなソフトウェアがハードウェアの付随物として扱われていることにより、企業はソフトウェア開発のために人材や技術などの経営資源を投入すること

---

[127] ただし、Bean は、アウトソーシングによるイノベーションの喪失について、頻繁に、かつカジュアルなコミュニケーションが取れないことをその理由としており、詳しい説明にまでは至っていない。

は見合わないと考え、コストとしていかに安く開発できないかというソフトウェアによってもたらされる価値を軽視するような戦略の欠如により、創造的なソフトウェアが開発できなくなっていったと指摘している。

このように日本ではソフトウェアは、ハードウェアに付随する無償のものという考えもあり、その価値は低く見積もられるとともに価格競争に陥り、コスト削減の対象とされるような構造が形成されてきたのである。ソフトウェアの開発にとって、今までにない問題や新しい領域に取り組んでいくような革新的なソフトウェアを作る部分は、他の企業に対して差別化できる部分でもあり、競争優位を生み出す価値のある部分となりうる。しかしながら、日本のソフトウェア開発は、価格競争のもと、人材確保やコスト削減の目的を果たすため、そうした競争優位となりうるものをアウトソーシングしてしまっているのである。

ソフトウェア開発の上流工程を重視して下流工程を安易にアウトソーシングしてしまう方法では、ソフトウェアを作り込む作成工程を外部に任せてしまうため、どう作り込んだかの詳細を担当者が十分に把握できていないなどの危険性がある。さらに、ソフトウェアの作りの詳細がわからないため、自分たちによるメンテナンスも難しく、いざ障害が発生したときに自ら手直しすることも難しくなる。また、海外企業へ委託することで、コアな部分であるソフトウェア開発の設計技術が、海外へ移転してしまった事例も存在する（高橋, 2010）。

ソフトウェア開発では、工程を順番に追って開発を進めていくため、上流工程で決められた仕様の変更を防ぐことで下流工程は機械的に作業を行うことが望ましくなる。また、あらかじめ決められた手続きに沿うことで、マネジメントの点からも管理が容易になる。このことは、ソフトウェア製品を工業製品や建築物のアナロジーとして捉える製品観と、下流工程に従事する開発者を「知的作業をともなわない単純労働者と見なす開発者観が強化されてきた」[128]といえ（妹尾, 2001）、知的な、創造的な能力が必要であることが見過ごされてきたのである。

② ソフトウェア開発プロセスの見直し

このようなソフトウェア開発手法の変化に伴い、これまで製造業や建築業など他の産業の開発手法を利用するために矯正してきた製品観や開発者観も見直されるようになってきた。

妹尾（2001）は、工業製品の製品観と単純労働者という開発者観は、ソフトウェア開発の実態には適用できないことを指摘している。

ソフトウェア産業は、ハードウェアなどの製造業の手法を模してきたが、その産業上の

[128] 妹尾（2001）p.72

特性から、ソフトウェアに適用できない部分が存在する。工業製品とソフトウェアの違いの一つとしては、工業製品は作り直すために多大なコストがかかることがあげられる。原材料の再調達、金型の再作成、部品の再作成などが発生し、場合によっては作り直し自体が不可能な場合もある。

また、ソフトウェアは無形性の特徴があり、有形物である工業製品に対し、デジタルデータという無形物であるソフトウェア製品は、容易に開発途中の仕様変更が行われる傾向にある。そしてソフトウェアの名のとおり、物理法則などに支配されず柔軟な部分が大きい。そのため、ソフトウェアの開発は、必ずしも決まった方法が存在するわけではなく、資源、環境、手順、プログラム構成で無数の開発方法が存在する。

ただし、ソフトウェアがデジタルのために変更が容易であるとはいえ、実際にはソフトウェアの既存の仕様にも関係してくるため、変更による影響が甚大となるケースが多い。さらに、製品として完成したものが目に見えないものであるため、完成品の確認が最後までできない。ソフトウェアは無形物であることから、ほぼ完成に近い状態までできあがって実際にソフトとして利用してみないことには、ユーザーが要望していたソフトウェアとしてできあがっているかどうか、十分に理解することができず、ユーザーは新しいソフトウェアを使用することを通して、そのシステムを理解するのである（Brooks, 1975, 1995）。カスタムソフトウェアの市場は、IT ベンダーとユーザーの間に、極端な不完全情報が存在するといえる（田中, 2009）。

ハードウェアは、モジュール化を進めることで外部から製品や部品を容易に調達できるようにし、それにより劇的に生産性を上げ、成功を収めてきたが（峰滝, 2004）、そのハードウェアの成功モデルを他の産業であるソフトウェア産業に機械的に適用することはできないのである。

さらに、ソフトウェアは品質が向上していくものの、プログラムにも必ず潜在的なバグを含んでおり、完璧なソフトウェアは存在しない[129]。このような潜在的なバグがソフトウ

---

[129] ソフトウェアのバグによるシステム障害としては、2002 年 4 月に発生したみずほフィナンシャルグループの大規模なシステム障害があげられる。これは、旧第一勧業銀行、旧富士銀行、旧日本興業銀行のシステム統合によるバグである。この障害発生時には、1 ヶ月以上に渡って ATM が使用できず、料金の引き落としや口座振替などができない事態となり、ソフトウェアの障害が社会に広く大きな影響を与えることが明らかになった。

このようなバグは、ソフトウェアに必ず含まれている。例えば、何万字もの文章に誤字脱字が必ず含まれているように、システムは数万から数十万行のプログラムで作成されている。さらに、そのシステムがサブシステムとして繋ぎ合い、一つの大きなシステムを構成していることが多く、全体で数百万行のプログラム規模になる場合もあり、バグを完全に除去することは難しい。

ただし、ソフトウェアにバグが含まれていても、必ずしもこのようなシステム障害に繋がるわけではない。また、必ずソフトウェアの中にはバグが含まれていても、そのバグのレベルも大小の差があり、例えば、文字のフォントサイズが異なるといった許容できるレベルのバグであれば、修正コストを考慮に入れ、運用でカバーされることも多い。

ェアに含まれている理由の一つとして、ソフトウェアの開発が個人に依存した属人的なものであることがあげられる。この点に関して、梅澤（2000）は労働者の生産性に着目し、「ソフトウェア開発の工程の一部は必ずしも集団的一律作業を必要とせず、個人作業の側面を多くもっている」[130]と述べている。つまり、開発者個人のノウハウや経験、技能、思考に依存する部分が多く、そのために頭脳労働の時間が長くなり、ルーチンワークや標準化の比率が低いといえる。

事実、日本国内のソフトウェア開発企業に対する調査によると、不具合発生の最も多い工程が、このソフトウェアの作成工程という結果が出ている（独立行政法人情報処理推進機構編, 2012)。

アプリケーションやソフトウェアの開発、導入プロジェクトの失敗要因を研究しているCHAOS Research(Standish Group HP)（SD Times Software Development News, 2009）の調査結果によると、ソフトウェアの開発プロジェクトは、予算内、かつ予定通りに完了し、ユーザーの要件を満たして成功した割合は、1994 年に 16.2%だったものが、2007 年には倍以上に上昇したものの、依然として 35%に留まっているという。一方で、プロジェクトの 46%が遅延や予算をオーバーし、顧客のニーズを満たせず、19%は完全に失敗しており、半分以上が何かしらの問題を抱えていることを指摘している[131]。

このようなソフトウェア開発の失敗は、要求された仕様を満たしていないことや、技術的に問題を抱えていること、ユーザーがニーズをうまく連携できないこと、仕様が曖昧なこと、そして情報システムの稼働後になって顧客のニーズが判明するなどの不確実な部分が多いことが原因として考えられる。このような不確実性が低くすることができれば、製品の設計はパラメーターの選択にすぎず,設計の問題は解決しやすいといえる（竹村, 2001)。

ここで重要なことは、Waterfall Model の下流工程に位置するプログラム記述作業やテスト作業は、工業製品などの製造作業に相当するものではないと考えられることである。製造業における製造工程のように、ソフトウェア開発の「製造」に位置するものは、記述したプログラムをコンピューター言語に変換すること[132]や、マスタープログラムを CD-ROM やシステムに登録する作業こそが、ソフトウェア開発の「製造」とみなされるべきといえる（Bean, 2005)。従来の開発手法が主張するプログラム記述作業を代替可能な製造とみなすことは難しく、ソフトウェアのプログラムを作成するということは、デザインの問題（**図 7-1**）であり、試行錯誤を繰り返して現れる問題を解決していく点で、製造業の製品

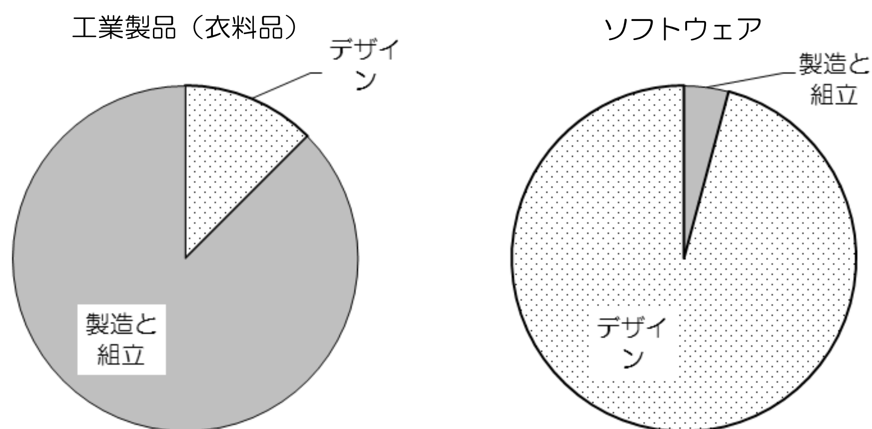
---

[130] 梅澤（2000）p.46

[131] Cusumano（2004）も、製品開発プロジェクトの 75～80%が予算超過となっていること、さらにプロジェクトの 20%を時間通りに成し遂げると「ベスト・プラクティス」とみなされていることを指摘している。

[132] さまざまなプログラム言語が存在するが、そのままではソフトウェアは動かすことはできない。コンピューターが理解できる言語に変換することで初めてソフトウェアは動くことが可能となる。

設計などに位置すると考えられる（妹尾, 2001）。



出所：Bean（2005）p.14 をもとに筆者作成

図 7-1 製造業とソフトウェアのデザインと製造・組立

Cusumano（2004）も、ソフトウェア開発が製造活動ではなく、むしろ製品設計であるとし、設計こそが製品であり、ソフトウェアを複製すること自体はたいしたことではないと指摘している。その上で、設計プロセスには、いくつものユニークな側面が存在し、なん百万ステップものソースコードの作成や、アルゴリズムの考案、プログラムを動かすための数学的手法のため、芸術や科学、工学、管理スキルといったさまざまな組み合わせが必要となり、それゆえソフトウェア開発は相当に困難であると述べている。

この点に関して、戸塚・中村・梅沢（1990）は、ソフトウェア産業が短期間で成長してきた歴史の浅い産業であることを指摘し、製造業の開発思想やその手法を模してきたが、この産業にはなじまないのではないかと、労務管理上の問題を指摘している。

このように、ソフトウェア開発の管理方法については、工学の登場により数十年前よりも遥かに進歩しており、より優れたプログラミング言語や開発ツールも登場している。そして、そこから作成されるプログラムは、非常に高度で洗練されてきている。一方で、プログラムのアルゴリズムを作成するという行為は、単なるルーチンワークではなく、問題解決や試行錯誤を伴い、創意工夫や無から有を作り出す力が必要であり（Cusumano, 2004）、そのような点を含む限り、難しい作業であり続けると考えられる。

(3) 小括

本章では、ソフトウェア工学の発達と日本のソフトウェア産業の分業構造を中心に、ソフトウェア・ファクトリーのような開発プロセスに基づいたソフトウェア開発がいかなる問題をもたらしうるのか、こうした問題の克服において従来の分業構造を再編する必要が

あるとすればそれはどのような再編であるのか、こうした論点に対する分析的視点および枠組みを検討してきた。

1968 年の NATO のエンジニアリング会議より、ソフトウェア工学の研究が進んだが、日本ではその導入は遅れており、さらにその分業構造は下請けを中心としたものであった。特に開発プロセスを分離し、各工程を順に追っていくことで、下請け企業や海外ソフトウェア企業などのアウトソーシングも行いやすかったのである。さらに、Waterfall Model に代表されるように、このような下請けを中心とした上流工程と下流工程に分離する方法にも適応しやすく、こうした方法は、前章で説明したソフトウェア・ファクトリーとも相性が良い。

しかし、ソフトウェア開発の問題整理に関わる作業は技術者の知的な、創造的な能力に依存しており、そうした創造力を強化することが重要となる。ソフトウェア開発において、設計とプログラム作成を分離し、アウトソーシング先の企業に任せてしまうことの問題は、その複雑な設計機能の意志疎通が困難なところにある。設計段階ですべての問題が解決していることはなく、プログラムを作成しつつ問題解決を行っていくため、設計とプログラム作成の担当者の綿密な意思疎通が必要なのである（Tomayko and Hazzan, 2004）。

特に、ソフトウェアを開発するには、専門性を持った人たちがチームを組んで行う必要があり、作業や工程を分割し、協業していく部分が多く発生する。そのようなチームや組織では、特定の工程に対する専門性だけでは不十分であり、全体の流れがつかめるような、企業の境界を超えた幅広い関連知識を持っている必要がある（伊丹・松島・橘川編, 1998）。

ソフトウェア開発の一連のプロセスを分割し、下流工程を標準化した単純労働に置き換えてしまうような方法は、工程間の緊密なコミュニケーションを困難にし、下流工程における試行錯誤や創造的な問題解決といった活動を阻害することとなるため、魅力的な機能を持つ革新的なソフトウェアを開発する方法としては最良ではないと考えられる。

以上のことを踏まえ、次章では、実際のソフトウェア開発で、どのような分業と協業が行われ、さらに開発の場で行われた作業プロセス間の連携や開発担当者の知的活動について、事例分析を行う。

第8章 ソフトウェア開発における知識労働と分業の問題－開発事例研究－

本章では、前章までの議論を踏まえて、ソフトウェアの開発プロジェクトの事例を対象として、開発プロセスにおける工程間分業ならびに協業の編成が、開発のパフォーマンス、特にコストや品質、納期、メンバー構成、プロジェクト運営といった点について、どのように関係しているのかを明らかにする。ここでは、こうした開発パフォーマンスの観点から、成功あるいは失敗であったと評価される開発事例を取り上げ、そこでの工程間分業および協業の編成がプロジェクトの成否に対してどのように影響を及ぼしたのか、に焦点を当て分析を行う。

(1) 調査概要

① 調査目的・意義

ここまでの議論を通じて、日本のカスタムソフトウェア開発の多くが、Waterfall Model に基づいて要件定義や設計などの上流工程からプログラム作成やテストなどの下流工程へと進む開発手法を取り入れてきたことが確認された。一方で、この管理を中心とし、工程を分割して開発を進める方法には、変更に弱い側面や下流工程の製造工場化に伴う作業者の知的活動の阻害などさまざまな問題があることも明らかにされた。

ソフトウェアを開発するには、効率性を高めるという観点から作業や工程を分割するとともに、高度な専門性を持った技術者がチームを組んで取り組む必要が存在し、そのため、分業と同時に工程間の協業や連携を如何に編成するかという点が重要となるが、これまでの研究ではそういった開発現場の分業と協業の実態まで踏み込んだ研究は少ない。

そこで、本研究では、実際のソフトウェア開発の現場において、開発作業やその工程をどのように分割され、さらにそれら工程間の連携がどのように編成されているのか、その事例を分析する。ソフトウェア開発プロジェクトの工程や知的部分の結合において、特に問題が発生した事例を対象とし、そうした問題発生の背景にどのような要因があり、さらに開発の現場においてそうした問題に対し開発従事者がどのように対応しようとしたのかを分析する。

② 分析の枠組み

ソフトウェアの開発は、開発プロジェクトによってその方法も大きく異なる。開発の規模や進行方法、ユーザー企業との関係など、条件が異なればその方法も変わってくる。

ソフトウェア開発の個々のプロジェクトの目的やゴールはさまざまであり、似たような要求仕様であっても、プロジェクトによってその要求の背景や制約条件、メンバーが異なっている。そのため、類似プロジェクトは存在しても、まったく同じプロジェクトは存在せず、個々の要求を分析してもその関係性が正しいとは限らない（須賀・牧野・加藤, 2014）。

一般に、製品開発や生産の現場の競争力を測る標準的な指標として QCD（品質、コスト、納期）が存在するが（藤本, 2003）、ソフトウェア開発についてもその評価について QCD を基準とすることが多い（浦塚, 2011; 神原, 2012; 長田, 2013; 桑原, 2016）。

そこで本研究で、分析の対象とする 3 つの事例にそれぞれについて、a)プロジェクト概要と目的、b)メンバーとその分業構成、c)開発の特徴を検討するとともに、またプロジェクトの結果について、単なる成否にとどまらず、d)品質（Quality）、e)コスト（Cost）、f)納期（Delivery）ならびに、g)人的・技術的サポート（Service）の 4 点を分析の焦点に置くことにより、プロジェクトとしていかなる帰結を迎えたのかを分析する。その上で、h)プロジェクトの事後的評価、についても検討を加える。特に、プロジェクトに対する社内での評価や顧客からの評価、その後の発注状況への影響、さらにソフトウェア開発プロセスにおける知的活動がどのように展開されたのかに焦点を当て、プロジェクトとしてどのような結果、成果が導き出されたのかを分析し、そうした結果、成果をもたらすに至った要因を明らかにする（**表 8-1**）。

表 8-1 開発事例の分析のフレームワーク（焦点）

プロジェクトの特徴	
a)	プロジェクト概要と目的
b)	メンバーとその分業構成
c)	開発の特徴
プロジェクト結果の分析	
d)	品質（Quality）
e)	コスト（Cost）
f)	納期（Delivery）
g)	人的・技術的サポート（Service）
h)	プロジェクトの事後的評価

③ 対象の選定

本研究では、ソフトウェアの開発プロジェクト事例として 3 つの事例を選定した。

いずれの事例も、開発プロジェクトとして完遂されたものである。しかしながら、その成否という点では必ずしもすべてが成功した事例ではない。分析の焦点となる QCD といった開発パフォーマンスの個別の評価点については、成功と評価されるものもあれば、失敗と評価せざるを得ないプロジェクトも存在する。

こうした事例を選定した背景には、北米など海外のソフトウェア開発では途中で開発プ

ロジェクト自体を中止してしまうこともあるが、日本のソフトウェア開発ではそのプロジェクトが QCD のすべてを十分に満たせなくても最後まで完遂させることが多いという事情がある[133]。また、リスクがあっても受託会社は開発を請け負うことも多く、顧客との継続的な取引関係の維持という点ではコストの面で赤字になろうとも、それが必ずしも失敗プロジェクトといえない場合もある。

例えば、一度開発したソフトウェアはその開発企業が続けて保守作業や改善に向けた次期プロジェクトを受注することが多く、そうした同じ顧客からの継続した開発案件の獲得のために採算ラインぎりぎりで受注する場合がある。また、技術的に難しくかなりのコストが見込まれるものの、多くのユーザー企業が発注先の企業を選定する際にそれまでの開発実績を重視することから､そういった開発実績や経験を得るために受注することもある。

本研究で取り上げる事例企業に限らず、ソフトウェアの開発は一企業の中でも多くのプロジェクトが稼働しており､それらプロジェクトによって開発プロセスも大きく異なる（東京大学社会科学研究所, 1989）[134]。特に、開発プロセスは企業や各種関係部署、プロジェクトのリーダーの考え方に少なからず左右され、さらに開発の規模や難易度、メンバーの熟練度などにも強く影響される。そのため、各開発プロジェクトのプロセスの実態やそこでの問題を知るためには、プロジェクト・リーダーやメンバーに対する聞き取りやプロジェクト報告書などの書面による調査により、各プロジェクトを精査していく必要がある。

しかしながらその一方で､このようなソフトウェア開発プロジェクトの多様な性質から、その実態を反映するような定量的調査を通じてその実態に迫るデータを収集することは難しい。さらにほとんどの企業は、守秘義務や顧客企業との関係を考慮することから、プロジェクトの詳細まで公開することはほとんどないため、しばしばインタビューによる調査自体も困難を伴う。

こうした調査に伴う困難について、中尾（2009）は、ユーザー企業や IT ベンダーがソフトウェア開発に失敗した場合でもその事故原因を公表しないことを指摘している。さらに中尾は、たとえソフトウェアの開発が失敗したとしても、自社の損失に結びつくようなその事実自体が企業より公表されることは少なく、公表されたとしてもその詳細まで明かさ

---

133 日経コンピュータ（2008）が実施した 8,800 社（有効回答数 814 社）へのソフトウェア開発の調査によると、QCD をすべて満たしたプロジェクトは 31.1%であり、残りのプロジェクトは何かしらの問題を抱えていることが指摘されている。ただし、日経コンピュータの調査によると、ソフトウェア開発のプロジェクトの成功や失敗をどこで判断するかといった明確な定義が存在しないことを指摘しており、QCD をすべて満たせなかった場合、完全な成功ではないもののそれがすなわちソフトウェア開発プロジェクトが失敗したわけではないといえる。

134 古い調査であるが、東京大学社会科学研究所（1989）によると、71.5%の企業がプロジェクトチーム方式をとっており、企業の部・課による開発方式は 24.5%だという。さらに、技術者は幾つものプロジェクトチームに参加することも多く、そのうえプロジェクトによって担当する工程も異なることが多いことを指摘している。

れるような事例はほとんど存在せず、そのために正確な事例が存在しないと述べている。当然ながら、これまで述べてきたようなソフトウェア開発の失敗要因に関する先行研究は数多くあるが、その多くも守秘義務によりその公開や内容に制限がかけられていることが多い。こうした調査上の問題から、本研究でも、各事例の分析にあたって企業や個人が特定できないよう処理が施されるとともに、対象企業において調査、開示可能な範囲での分析とならざるを得なかった。

本研究が検討する事例は、プロジェクトの規模が 50 人月未満の中小規模のものを対象としている。これは、独立行政法人情報処理推進機構編（2014）の調査結果にあるとおり、ソフトウェア開発は小規模化しつつあり、特に 10 人月未満の開発が 5 割以上を占めていることが明らかとなっており、さらに今後もその傾向が増すことが考えられるからである。ただし、10 人月未満の開発になると、1 人から 3 人程度のメンバーで構成されることが多く、人数も極端に少ないことから工程ごとに作業の分割を行うのではなく、全工程を協働でこなすようなこと多い。このため、本研究では、10 人月より少し規模の大きい、20 人月から 30 人月程度の開発事例も含めた 50 人月未満の開発規模のプロジェクトを対象としている。

④　事例に関する調査の方法

対象となる事例は、国内中規模の IT ベンダーの A 社の開発プロジェクトであり、ユーザー企業による内製ではなく、すべて IT ベンダーである A 社が担当することとなった開発プロジェクトとなる。

事例の調査方法として、プロジェクトの開始報告書、および終了報告書による書面調査を中心に行った。また、実際に開発に関わった A 社のプロジェクト・リーダーやメンバーに対し、質問票をもとに聞き取りを行った。

調査時期は、2015 年から 2016 年にかけて行っており、対象となるプロジェクトは、2012 年から 2015 年にかけて遂行されたものである。

前述の対象の選定でも述べたように、A 社においても守秘義務契約や顧客企業との関係からプロジェクトの詳細までを明らかにすることができなかった。各事例の分析にあたっては、開発費用等の数値は非公開とし、また対象となるプロジェクトやその正確な実施時期、参画したメンバー、企業を特定できないよう処理を施すとともに、対象企業において調査、開示可能な範囲で分析を行っている。

⑤　事例の概要

事例の概要は、次の表に示されるとおりである（**表** 8-2）。

本事例のソフトウェア開発プロジェクトの受託企業である A 社は、1960 年代に創業した独立系のソフトウェア開発会社であり、従業員規模は約 300 人、売上高は 27.3 億円（2015

年 7 月期）である。受託によるソフトウェア開発のほか、自社開発のパッケージソフトウェアの販売・サポートなども行っている。受託ソフトウェア開発では、銀行や生命保険会社などの金融機関から、印刷会社、書店、商社、大学などの教育機関、飲食店など、幅広いユーザー企業を相手に、事業を行っている。

表 8-2 ソフトウェア開発事例のまとめ

	事例 1	事例 2	事例 3
顧客企業	X 社	Y 社	Z 社
業種	事務用機器メーカー	金融業	金融業
開発種類	新規開発	保守（改修）	保守（改修）
開発手法	Waterfall Model	Waterfall Model	Waterfall Model
分業体制	企業内分業（A 社内）	企業間分業（A 社/B 社）	企業内分業（A 社内）
分割工程	要件定義／設計／プログラム作成	上流工程（概要設計まで）／下流工程（詳細設計以降）	要件定義／設計以降
開発規模	全体で約 50 人月（事例のチームでは約 12 人月）	約 30 人月	約 20 人月
開発期間	約 4 ヶ月	約 1 年	約 6 ヶ月
技術者数	15～20 人（事例のチームでは 4～5 名）	3～7 名	3～4 名
品質	次の工程で多くの障害	本番稼働後に障害	問題なし（設計～テストで検証）
納期	残業などで遵守	問題なし	問題なし
コスト	予算を大幅にオーバー	上流工程は予定通り 下流工程は予算オーバー	設計～プログラム作成で若干オーバー
論点	要件、設計、プログラム作成で分業された。専門性の高い技術者が社内より集められた	上流工程と下流工程を、企業間で分断され、設計とプログラム作成間が繋がっていない	設計以降は分業していない。設計とプログラム作成で繰り返しの作業が発生

出所：筆者作成

対象となる事例では、これまで本研究で述べてきたような工程間の分業を中心に、ソフトウェア開発が遂行された。具体的には、ソフトウェア開発のプロセスが複数の企業もしくはメンバーで分割して行われたプロジェクトであり、この複数間の関係には、外部の組織、つまりアウトソーシング先との分割に限らず、社内での分業も対象としており、同じ

チームとして最初から一緒に作業をしてきた技術者以外との開発作業も含まれている。

事例 1 は、上流工程と下流工程の分割の事例となる。特に、上流工程である要件定義や設計と、下流工程であるソフトウェア作成とテストが、A 社の内部、つまり 1 企業内におけるソフトウェア開発でありながら、工程ごとに作業とメンバーを分割して編成されたケースである。開発メンバーも、社内から選りすぐりの専門性の高い技術者が集められ、チームを組んで開発を行った。事例 1 では、この工程ごとの分割が、ソフトウェア開発にどのように影響を及ぼしているのかを明らかにする。さらに、上流工程の中の要件定義と設計も分割され、開発全体に影響を及ぼしており、このことはソフトウェア開発プロセスの説明で述べたユーザー企業から要件を含む仕様を明確にできないため、そのような明文化されない部分を IT ベンダー側が業務知識を含む過去の開発経験知識やコミュニケーションで補う必要がある（神岡・細谷・張, 2006）といった要件定義の重要性にも関連する部分となる。

事例 2 も、上流工程と下流工程の分割の事例となる。事例 1 と同様に、ここでも開発プロセスを上流工程と下流工程とで分割しているが、事例 2 の特徴として、A 社単独による開発ではなく上流工程と下流工程とでそれぞれ異なる企業が開発作業を受け持っている。このような企業間のやりとりは、社内での開発と比べてその調整のため想定外のコストがかかるが（伊丹・松島・橘川編, 1998; 武石, 1999）、ここでは特にソフトウェア開発の設計とプログラム作成という部分を企業間で分割することがどのような影響をもたらすのかを明らかにする。

事例 3 では、単純な上流工程と下流工程の分離ではなく、ここまで重要とされてきた上流工程の要件定義と、それ以降の設計などとの分離が行われた事例となる。事例 1 と同様に、A 社内での開発であり、さらに要件定義と設計以降の担当者が異なっている。一方、上流工程から下流工程にかけての設計とプログラム作成などが分離されず工程間の連携が維持されており、この点において事例 1 とは大きく異なる。ここでは設計とプログラム作成が密接に結びつくだけでなく、その間で試行錯誤することの重要性を明らかにする。

(2) 事例 1：要件定義、設計、プログラム作成の分業

最初の事例は、対象となるユーザー企業にとってもまったく新しい試みとして開発するソフトウェアであり、他の事例よりも革新的なソフトウェアの開発となる。この開発プロジェクトの特徴として、特に工程ごとに分業を行っており、本研究の主軸となる、設計とプログラム作成は分離すべきではないという代表的な事例のひとつである。

① プロジェクトの内容

a) プロジェクト概要と目的

事例のプロジェクトは、事務用機器でグローバルに展開している企業 X 社が発注元で、

X 社の顧客である企業や事務所などのエンドユーザーの下で稼働している業務用機器のデータ収集システムの新規開発である。事務用機器は、日本だけでなく世界中で稼働しており、その使用率や印刷の傾向、インクの消費具合などのデータを集計、分析することで、機器のメンテナンスに役立てるとともに、新しい機器への取り換えなど営業活動や販売促進へと繋げようという 2010 年以降に盛んになってきている IoT としての試みである（図 8-1）。

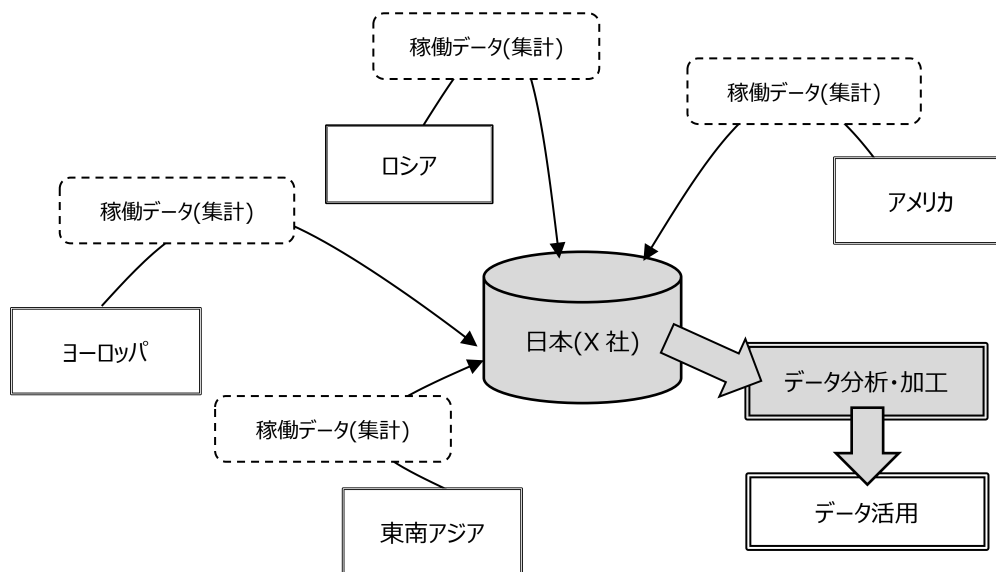


図 8-1 事例 1：開発システムのイメージ

このプロジェクトは、本研究で述べた Waterfall Model を原則として採用したが、新規開発であったため、X 社の中でも要件は定まっておらず、Agile のようにその仕様は打ち合わせの度に、より最新のビジネスの実態に合うように修正されていくようなものでもあった。また、該当システムはクラウド・コンピューティングによるサービスを利用しており、クラウド上で、A 社のほか、X 社自身によるメンテナンス作業のため、それぞれの社内からソフトウェアを開発していた。さらに、対象となるシステムの一部は、海外でも開発しており、同じくクラウド上でシステムを構築していた。

b） 分業構成

IT ベンダーA 社の開発メンバーは入社 20 年近いプロジェクト・マネジャーを中心に、20 人近い開発メンバーが集められた。メンバーは入社 1〜2 年の新人から、5〜10 年の若手技術者、10 年以上のベテラン技術者までが集められた。一方、納期は 4 か月程度と短く、要件定義から単体テスト工程までを対象とした短期決戦型のプロジェクトであった。

要件定義や大まかな設計方針など主要な部分は、それまで顧客と打ち合せを続けてきた

コアメンバーが担当することになったが、プロジェクトの規模に対し圧倒的に開発技術者が不足していたため、設計工程やプログラム作成工程で次々と社内の手が空いている技術者を追加していった。

プロジェクトでは、データの加工のほか、加工したデータの画面や帳票への出力など、多くの機能の開発が行われたが、本事例では、このうち業務用機器からデータを取り出して、画面や帳票で利用できるように集計や加工する処理を担当したチームを取り上げる。

このデータ作成チームでは、開発プロセスを要件定義と設計、それ以降のプログラム作成及びテストの工程として大きく 3 分割されていた（図 8-2）

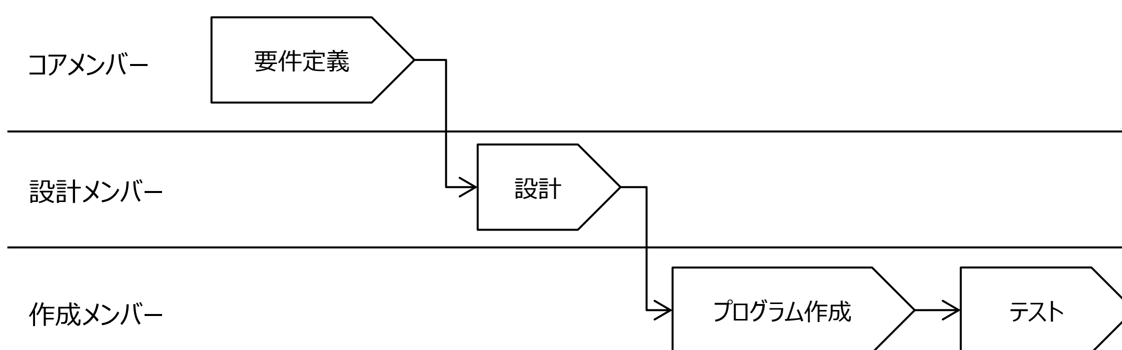


図 8-2 事例 1：工程と分業構造（担当）

前述の開発技術者の不足のため、社内から設計とプログラム作成でそれぞれ新規のメンバーを採用し、それぞれ 2～3 名程度でチームを編成した。ただし、全員がそれぞれの工程の作業に専属したメンバーではなく、他チームからの一時的なサポートメンバーが追加されたほか、怪我の入院治療のための長期離脱といった異動があった。作業の分業方法は、コアメンバーが仕様を決め、設計チームへと連携し、さらに設計メンバーが書き上げた設計書を元に、プログラム作成メンバーがプログラムを作り上げる構造となった（図 8-3）。

c) 開発の特徴

設計担当者は、入社 10 年以上のベテラン技術者で、この設計担当者がチームリーダーを担った。しかし、本プロジェクトで利用されるクラウドのような技術的知識には乏しく、そのため細かい部分を除いたプログラムの基本的な設計を行った。この設計者を起用した理由は、これまで幾つかのプロジェクトで上流工程から下流工程まですべてを担当してきた経験があり、技術的知識が不足したとしても、これまでの経験から基本的な開発要件の能力を満たしていると判断されたからである。

一方、プログラム作成メンバーは、入社 4～7 年前後のメンバーで構成されたが、プロジェクトで使用されるプログラム技術にはある程度精通しており、最新のクラウドといった技術にも慣れていたため、設計書をもとにプログラムを作成していった。特に、今回はク

ラウド技術が中心となっており、これまでの開発プロジェクトの経験から採用されたといえる。

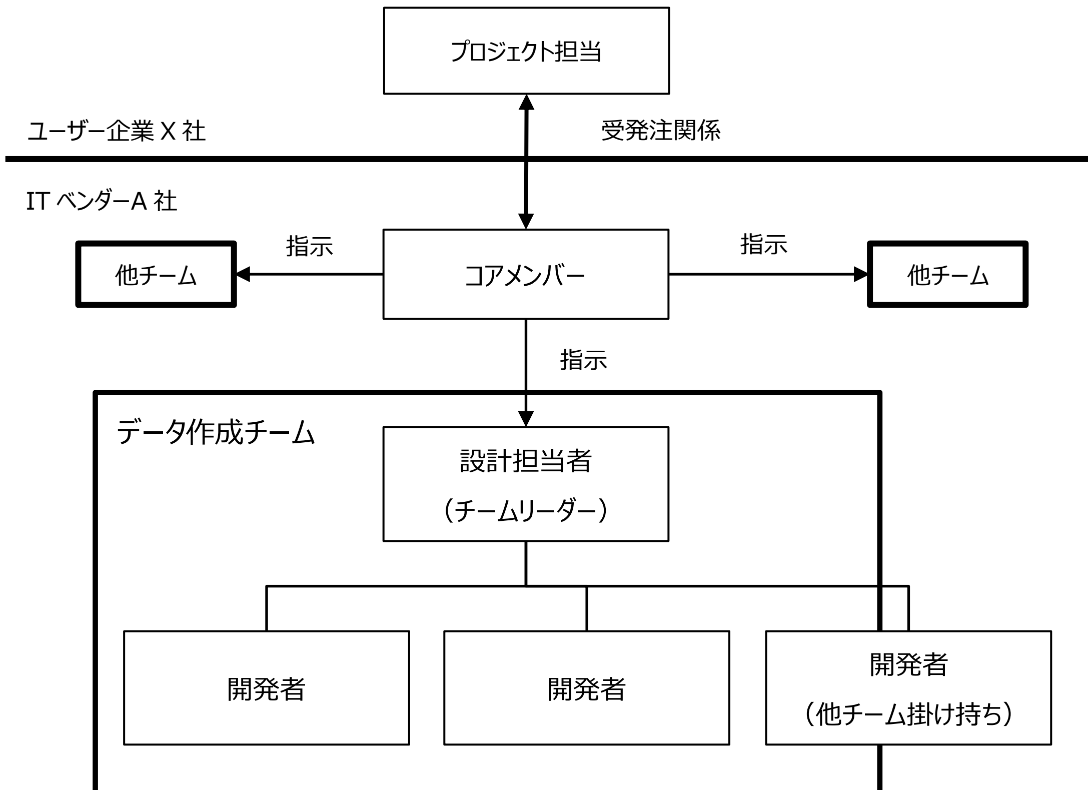


図 8-3 事例 1：メンバー構成

この設計者とプログラム作成者の連携方法は、設計書と口頭による連携であり、作業もすぐ近くで背中合わせの作業を行っていた。ただし、設計とプログラム作成のメンバー同士は初顔合わせであり、それぞれ設計のスペシャリスト、プログラム作成のスペシャリストとして紹介されたが、お互いの具体的なシステムへの理解や技術レベルまでは把握できていなかった。

② プロジェクトの結果

d) 品質

品質については、新規開発のため、開発途中で仕様や設計の確認や変更が何度も発生しており、単体テスト工程で多くのバグや問題が発生した。A 社と X 社の間の仕様などの確認は、メールや対面での打ち合わせのほか、Skype などのオンラインコミュニケーションツール[135]を利用し、仕様や設計の齟齬がなるべく出ないように作業を進めていった。

[135] 音声だけでなく映像もやりとりできる、インターネットを介した電話サービス。

度重なる仕様の変更や発生したバグなどのさまざまな問題により、開発のスケジュールは遅延し、そのため、メンバーによる連日深夜までの残業と休日出勤などで何とか対応を行うこととなった。その結果、納品時点では重大な問題となるようなバグは直っており、品質も多少の問題はあるものの、検品の結果、ある程度の水準は確保されたと判断され、次の工程へと進むこととなった。

e) コスト

コストの点については、連日の残業や土日の出社、さらに仕様の調整などによる追加作業の発生により、A 社にとっては想定以上の負荷となっていた。このため、スケジュールの遅延の解消や作業者の負荷を減らそうと、社内の手が空いている技術者を次々と投入したり、部分的にサポートを利用したりしていった。その結果、予定していた作業工数を超過してしまい、コストは悪化した。また、発注側である X 社も想定外の作業の発生、仕様の検討、そしてそれによる追加の発注をすることとなり、同じくスケジュールの遅延やコストの悪化を招いてしまった。

f) 納期

納期については､最終的に予定日までに単体テストを終え､納品することができている。しかしながら、新規開発ということもあり、納期が短いにもかかわらずなかなか要件が定まらず、仕様の変更も多く発生していた。

特に、このプロジェクトの進め方は、第 5 章でも述べた Agile に近い形で、その仕様は打ち合わせをする度に変化していった。そして、それに合わせて設計、そして開発内容も変えていかなければならず、そのことが作業負荷を大きくさせ、開発スケジュールは遅延していき、納期も危ぶまれていた。このために、毎日の残業や土日の出社などで開発を進めることで、スケジュールの遅延を解消しようとした。

g) 人的・技術的サポート

コアメンバーは、進捗の管理や全体の整合性を取るとともに、データ作成チームを含むすべてのチームに対してサポートを行った。しかしながら、全体の開発期間が短く、各チームとも十分な擦り合わせができておらず、そのサポートも限定的であり、各チームともなかなかコアメンバーに十分な説明をもらうことができなかった。また、データ作成チームでもコアメンバーと設計メンバー、プログラム作成メンバーの間で十分な意思疎通ができておらず、それによりコアメンバーのサポートが設計メンバーを通じて、プログラム作成メンバーにまでソフトウェア開発に関わる情報を十分に伝えることが難しかった。

h） プロジェクトの事後的評価

このプロジェクトの結果は、プロジェクト終了時点で成功とされた。

コストの点で問題視されたものの、このプロジェクトは挑戦的なものであり、納期が短いにも関わらず納品することができたため､開発企業側であるA社の社内評価は高かった。また、ユーザー企業側であるX社も、それまでのやりとりを含め、A社の納期に間に合わせた努力やそのサポートを高く評価した。その結果、A社は、X社とこのプロジェクトの単体テスト工程までの契約であったが、その後、結合テストから本番稼働までの後に続く工程についても新たな契約を結ぶことができ、何人かのメンバーが引き続きプロジェクトに従事した。

しかしながら、次の工程の結合テストで、プログラムのミスの発覚や、仕様の勘違い、大幅なレスポンスの低下など、初歩的なミスから、実際にある程度稼働させなければわからないようなミスまで多発してしまい、顧客の信用の低下を招くとともに、結合テスト以降を担当したメンバーに大きな負担を残す結果となってしまった。

③ プロジェクトの成功や失敗の要因

本事例のプロジェクトの成否を左右する諸要因として、次のようなことが考えられる。

まず、工程間で分業されており、特にコアメンバーから設計メンバーへ、そしてプログラム作成メンバーへの連携が不足していたことである。このプロジェクトでは、いくつものチームが同時に動いていたため、取りまとめ役として、要件定義をそれまでユーザー企業とやり取りを行ってきたコアメンバーが担当し、それを設計メンバーに伝え、さらにその設計メンバーがプログラム作成メンバーに実装内容を伝えるという、伝言ゲームに近い形をとっていた。コアメンバーは顧客との折衝のほか、データ作成以外のチームの要件も行っているため、設計工程には関与していたものの、実際のプログラムができあがるプログラム作成工程にはほとんど関わることができなかった（図 8-4)。

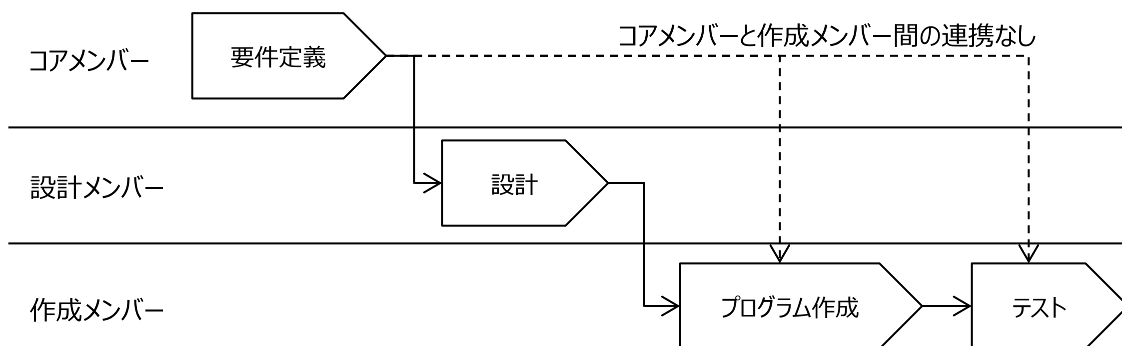


図 8-4 事例 1：工程間の連携

そのため、コアメンバーと設計メンバーの間でやりとりは多く行われたものの、コアメンバーとプログラム作成メンバー間のやりとりはほとんどなく、プログラム作成メンバーは設計メンバーと意思疎通を行うしかなかった。また、設計メンバーとプログラム作成メンバーは、日々背中合わせの距離でお互いに話し合いながら作業を進めていたが、設計の細かい部分までは決まっていないことも多く、その都度、設計メンバーがコアメンバーに確認しに行き、設計書に書き起こしていた。日々変わっていく設計をその都度伝える形をとっていたため、コアメンバーの意図していたものが必ずしも最後のプログラムに反映できてはいなかった。

④ 事例 1 のまとめ

事例 1 からいえることは、高い専門性を持った技術者がチームを組んだとしても、分業間の問題として、工程間で切り離してしまうことで連携がうまくいかなくなることである。特に、設計で決められたことがそのまま開発できることはなく、設計とプログラム作成の作業者を分離してしまうことの危険性が指摘される。さらに、設計とプログラム作成作業を分離してしまうことで、ソフトウェアを設計さえできていれば、それを元にプログラムを作成することができる単純作業とみなしてしまった部分が見受けられる。本事例は、企業間の分業ではなく、社内のメンバーのみの開発プロジェクトではあったが、その連携部分に問題が発生していたのである。

事例 1 のプロジェクトでは、設計メンバーがプロジェクトで利用されるクラウドなどの技術やそのプログラムを理解していなかったため、特定のプログラム言語にあまり依存しない汎用的な設計となっていた。そのため、プログラム作成メンバーはその設計をもとに自身の技術力と照らし合わせ、プログラムを試行錯誤しつつ創り出していった。しかし、そのプログラム作成メンバーの中の試行錯誤の結果は、設計には正確にフィードバックされることはなく、さらに要件が日々変更される不安定さもあり、設計の十分な見直しができなかったのである。

設計者とプログラム開発者は安易に分業するのではなく、また、仮に分業せざるを得ない場合でも、密接に連携しなければ顧客の要望に応えられるソフトウェアの開発は難しい。コアメンバーから要件を聞き、設計をしつつ、実際にプログラムにもその設計デザインを反映し、問題があれば設計を都度調整していくといった作業を繰り返していく必要があるのである。

(3) 事例 2：上流工程と下流工程の企業間分業

事例 2 は、事例 1 の工程ごとの分業とは異なり、上流工程と下流工程間での分業事例である。プロジェクト内容は、既存のシステムを自動化しようとするものであり、要件も明確であり、開発はそれほど困難なものではなかった。ただし、事例 1 の社内開発での分業

とは異なり、本事例では企業間で分業が行われた事例となる。

① プロジェクトの内容

a) プロジェクト概要、目的

プロジェクトは、国内の金融機関 Y 社の社内システムの保守で、システム改修に伴う開発である。これまで稼働していた保守システムに新たな機能を追加するものであった（図 8-5）。

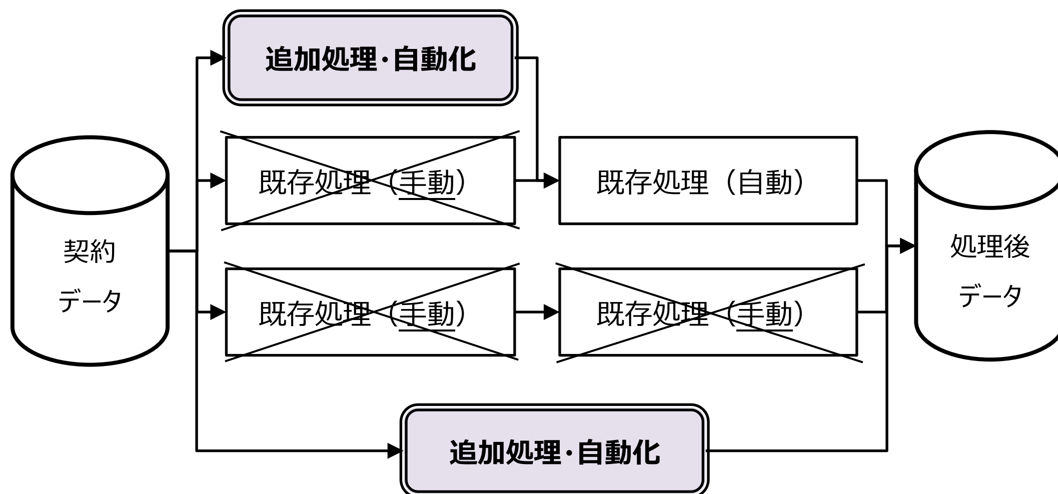


図 8-5 事例 2：開発システムのイメージ

社内システムの契約データを整備し、さらに社内イントラネット上で処理できるようにすることで、今まで手動で行ってきた画面のデータ入力や処理の選択、事務手続きなどを自動化し、効率化をはかるとともに、人の手による作業ミスを減らすことを目的としていた。

開発方法は Waterfall Model を採用し、特に工程の後戻りもできない厳格な管理体制であった。

b) 分業構成

これまで保守を行ってきた A 社が設計までの上流工程を担当し、新規に参画した B 社が続く実際にプログラムを作成する部分である下流工程から担当することとなった（図 8-6）。

この理由として、A 社は長年保守を担当しており、その技術者の契約単価も高いものとなっていたためである。A 社の技術者が開発の全工程を担当してしまうと、開発コストが高くなってしまうことになり、Y 社のシステム開発プロジェクトの担当者は、業務内容やシステムに精通している A 社に上流工程を担当してもらい、下流工程については少し技術者の単価が安い B 社に担当させることで、開発コストの削減を図ったのである。

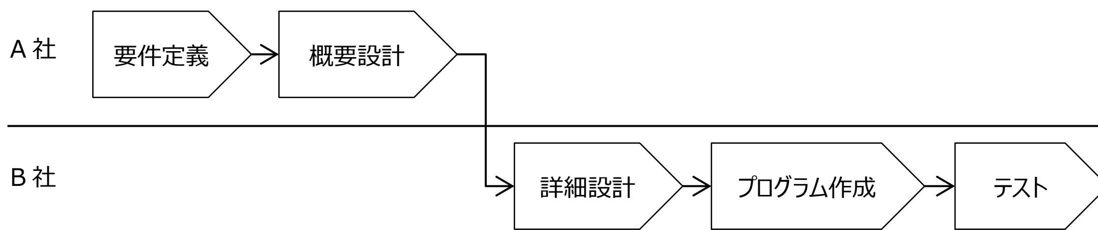


図 8-6 事例 2：工程と分業構造（担当）

プロジェクト全体では、A 社と B 社を含めて、工程に応じて 3〜7 人程度が開発を行った（図 8-7）。

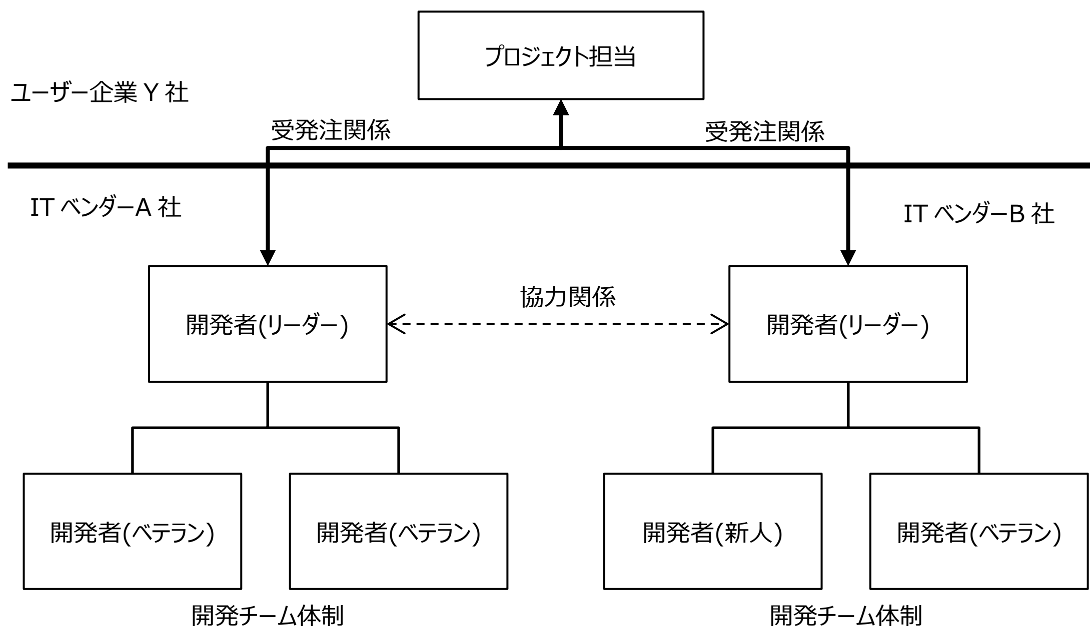


図 8-7 事例 2：メンバー構成

事例 1 の新規開発とは異なり既存システムの改修であり、上流工程の A 社の開発技術者も 10 年近く保守作業を行い、システムに精通したベテラン数人で構成された。一方で、下流のプログラム作成を担当した B 社は、取りまとめ役のチームリーダーを含む、ベテランと新入社員の混成チームであった。A 社と B 社のベテラン技術者は、年齢も近いこともあり、その関係も良好であった。

c) 開発の特徴

当然ながら、下流工程を担当した B 社は該当システムに触れるのは初めてあり、既存のシステムへの追加改修のため、設計書などは充実しているものの、プログラムについては上流工程の A 社の癖の強さなど、システムに慣れるのに苦労した。一方、A 社側は、長年

にわたり保守作業を進めてきたことで、A 社内の技術者間で、作業手順の違いや得意分野、プログラムの癖など、多くの連携方法に精通していた。ただし、設計書は彼らが作成したものの、開発業務の多忙などを理由に最新化できていないものも多かった。

このプロジェクトでは、工程の完了ごとにユーザー企業である Y 社とのレビューによる確認が実施され、仕様とズレがないかなどをチェックされた。Y 社の担当は数年間同じシステムを担当してきた社員であり、システムの全体やシステムが担う業務内容についても精通していた。

② プロジェクトの結果

d) 品質

上流工程を担当した A 社が、メンバー内のレビューやより上位者によるレビュー、ユーザーである Y 社とのレビューを実施することで設計の品質はかなりの保証を取っていた。その結果、B 社が開発を終えて納品した時点で仕様のミスなどはなく、さらに Y 社の確認も経て、品質上も問題ないと判断された。ただし、下流工程の作業は B 社の契約とそのために確保された作業工数であり、そのため、A 社は B 社の作成したプログラムの精査までは行うことはできず、B 社からの問い合わせへの対応や、テスト検証の方向性などの確認のみを行っていた。

e) コスト

コストについては、下流工程を担当した B 社がシステムに不慣れなことにより、プログラム作成やテストにおいて、作業工数の大幅な増加が発生した。発注元の Y 社にとって、B 社の超過作業分の請求に関する問題はあったものの、それ以外はコストの点で概ね問題はなかった。上流工程を担当した A 社は、予定通りの納期、品質、コストで納めており、どちらかといえば、安い金額で開発を請け負った B 社が、開発メンバーの育成と作業の実績を得たものの超過作業分の損失をこうむった。

f) 納期

開発期間は十分にあり、A 社が丁寧に設計を行い、B 社への引継ぎの期間も準備していた。下流工程を担当した B 社は、初めてのシステムなこともあり開発にかなり苦労を伴ったが、A 社の担当した要件定義から設計、B 社の担当したプログラム作成からテスト、納品まで、すべての工程で大きな問題は発生せず、納期に間に合わせることができた。

g) 人的・技術的サポート

A 社のメンバーはシステムのことを隅々まで理解していたが、当然ながら B 社のメンバーにはわからない部分が多かった。その解消のためにも、本来であれば設計とプログラム

作成の間で打ち合わせや相互のチェックなどの多くの意思疎通が必要であるが、企業間で工程を分けてしまったことで、その調整のためのコストが多くかかってしまい、意思疎通は思うようにできなかった。また、当初そういった調整のためのコストはほとんど見積もられておらず、作業工数を圧迫することとなった。

h） プロジェクトの事後的評価

成功裏に終わるかに見えたプロジェクトだが、納品し、稼働した当日に障害が発生してしまった。障害の内容は、改修した部分は問題なかったものの、それ以外の部分にその改修の影響が出てしまい、想定とは違った計算結果が出力されてしまっていた。そのままでは業務に支障が出るため、Y 社により次の日の営業開始までの応急の修正が命じられ、プログラム作成を担当した B 社が徹夜で対応すると同時に、A 社の有志も業務内容の確認やプログラム修正のサポートとして協力し、一緒に夜を明かすこととなった。奮闘の結果、どうにかプログラムの修正、テストまで終えることができ、次の日の営業開始までにシステムは正常に稼働するようになった。

最終的に、B 社のメンバーは、障害の対応を終えるとともに Y 社のプロジェクトから撤退し、その後の保守作業については、再び A 社が担当することとなった。Y 社の担当者にとっても、下流工程を安易に他の企業に任せることの危険性を認識することとなった。一方、A 社は再びシステムの保守全体を請け負うことができることとなったが、プログラムの改修を直接担当したわけではないため、B 社による細かいプログラム記述指針がわからず、既存のプログラムとの標準化ができなくなっていた。さらに、担当した B 社のプログラマーの癖や、プログラムをどのような意図のもとにロジックを組み立てたのか、詳細な部分まで把握ができなかった。プログラム設計書も B 社が最終的に納品したが、プログラムの細かい部分すべてが記述されているわけではなく、A 社はその保守作業に後々多大な負担を負うこととなった。

③ プロジェクトの成功や失敗の要因

本事例のプロジェクトの成否を左右する諸要因として、次のようなことが考えられる。

最も大きな要因は、上流工程と下流工程を完全に分離し、かつそれぞれ別の企業にアウトソーシングしてしまったことであろう。

当然ながら、既存システムの保守を担当してきた A 社は、システムの全体を見渡すことができている。改修による変更箇所以外の部分についても把握しており、どのようなプログラムの設定をすればよいのか、そして、どのようなプログラムにしてしまうと問題が発生するのか、深く精通している。A 社は上流工程として概要設計までの担当であったが、実際には B 社のことを考慮し、プログラムに近い設計書、つまり、どの部分をどう直すべきか、さらにどのようなモジュールを呼び出すのかを網羅した資料を作成し、B 社の開発

者に渡していた。ただし、あくまで今回のプロジェクトの改修のためにどこをどのように作成するかといった指示が中心であり、逆にどのように作成してはいけない、もしくはこの部分を変更してしまうと同様の処理を行っている他の部分にも影響が発生してしまうといった情報までは盛り込んでいなかった。

A 社はシステムの全体を見渡すことができていたが、暗黙的な部分までは設計書には表されておらず、企業間の分業により、そういった部分を密に連携をすることができなかったのである。この結果、B 社による開発は、A 社の設計書の指示通りではあったが、プログラムを作成する際に他のプログラムに影響が出るような形となってしまったのである。

④　事例 2 のまとめ

事例 2 からいえることは、ソフトウェア開発における企業間の分業の難しさである。特に、この事例では上流工程と下流工程を大きく分断してしまっており、そこでは設計書を中心とした連携が行われていたが、そこには関連するすべての情報が含まれているわけではなく、それだけではソフトウェアを開発することは困難であった。

たとえ、上流工程を完璧にこなし、十分な設計書を作成したとしても、下流工程の実際に動くソフトウェアを作成する部分を軽視してしまうと、最終的にソフトウェアの品質にも影響が出てしまう。開発者は設計書の指示通りにプログラムを作成することになるが、その設計書はプロジェクトで開発すべきことはすべて書かれていても、システム全体の知識がないと読み解くことは困難である。さらに、ソフトウェア開発は、現実世界の業務や習慣をプログラム化しようとするため、プロジェクトの改修とは直接関係ない部分や、開発してはいけないことなどもある程度は記載されるものの、例外や禁止事項はあげればキリがなく、それらすべてが設計書に記載されるわけではない。設計書は、要件の必要条件は満たしているものの十分なものではないのである（図 8-8）。

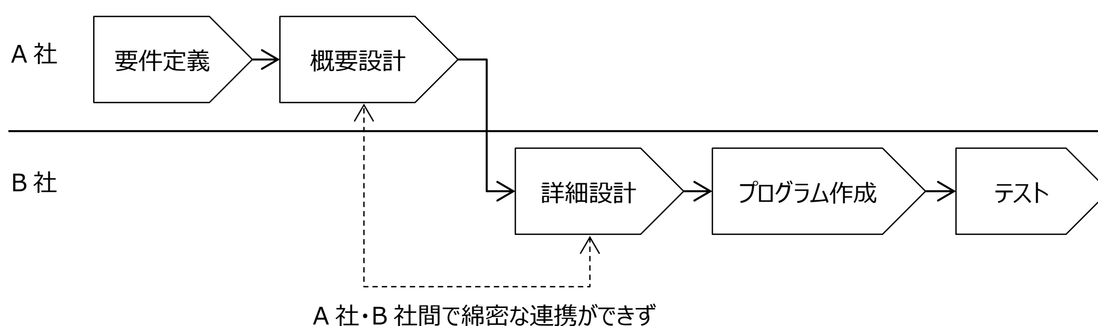


図 8-8 事例 2：工程間の連携

ソフトウェアの設計には実際にプログラムとほぼ同等のものも存在する。例えば、プログラムをそのまま設計書にコピーして貼り付けたものや、プログラムを強引に日本語に置

き換えたようなものも見かける。しかし、これらのほとんどは、システムができあがってから後追いでプログラムを作成していることが多く、そのうえ、プログラムをそのままコピーしてしまうような設計書は可読性が極めて低く、そのような設計書を読み解くのであれば、プログラムのソースコードを直接見て解析したほうが作業を進めやすいことが多い。

また、設計書にプログラムレベルまで記述されたとしても、そこにはプログラムとして動かしてみないとわからないような問題が含まれているのである。さらに、そういった完璧な設計書を作るのであれば、実際にプログラムを作成し、設計に含まれると考えられるすべての問題を解決してから作る方が確実となってしまうのである。

ソフトウェア開発は、下流の工程に位置するプログラム作成は機械的に作業することは難しく、設計に含まれた問題を見抜く力が必要となるのである。

(4) 事例 3：要件定義と設計以降の分業

事例 1 と事例 2 では、途中の工程を分業してしまうことによりプロジェクトが結果的にうまくいかなかった部分が発生した事例であった。次の事例 3 は、これまでの事例とは異なり、要件定義と設計以降とで分業した事例である。事例 2 と同様、既存のシステムの改修であるが、古いプログラム言語を使用しており、システムとシステムを連携させる、より難しく、かつ複雑な開発であった。

① プロジェクトの内容

a) プロジェクト概要と目的

プロジェクトは、国内中規模の金融業 Z 社の社内システムの保守で、事例 2 と同様、システム改修に伴う開発である。このプロジェクトは、既存の保守システムのいくつかの画面を変更し、他のシステムと機能を連携し、データの同期をとることで、業務の効率化を図るとともに最新の情報を反映することを目的としていた（**図** 8-9）。

ただし、事例 2 とは異なり、A 社単独の開発である。

これまで稼働していた保守システムに新たな機能を追加するものであるが、COBOL と呼ばれる 1960 年代に主流であった古い事務処理用のプログラム言語で作られているシステムであった。設計書はプログラムレベルまで落とし込んで記載されていたものの、長年のメンテナンスでシステムもそれを記載した設計書も複雑になっており、更新されていない部分など、誤りや記載の漏れもあった。

連携対象の他システムも同じように開発が進められており、本プロジェクトでは、他のシステムと並行、連携して開発を行うため、作業工程が合わせやすい Waterfall Model を採用し、基本的に作業の後戻りを許さない厳格なものであった。ただし、プロジェクトのマスタースケジュールを越えない範囲であれば、多少の作業の手戻りなどは認められていた。

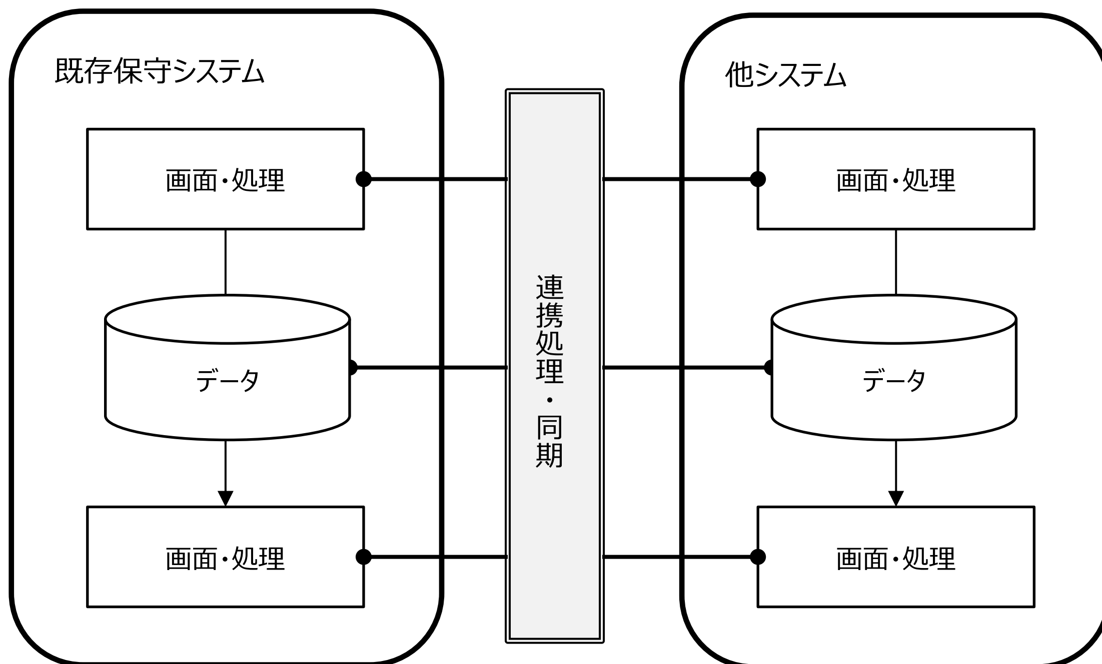


図 8-9 事例 3：開発システムのイメージ

b) 分業構成

保守対象のシステムの改修のため、担当する技術者も長年にわたり保守作業を行ってきたベテランメンバーが中心となった。

プロジェクトの開始にあたり、長年保守を続けてきた A 社のプロジェクト・マネジャーとベテランメンバーが、Z 社と要件定義を行ってきたが、開発作業の量が多いことと、このプロジェクト以外の保守作業への対応も必要なことから、設計工程より新しくメンバーを追加する形となった。設計工程から参加した新メンバーは同じ A 社の社員であるが、該当システムの開発は初めてであり、使用される COBOL 技術については多少の知識があったもののシステム特有の構造などもあったため、技術的に理解が難しい部分があった。

プロジェクトの期間は約 6 ヶ月であり、プロジェクトチームは 3〜4 人の体制であった（図 8-10）。

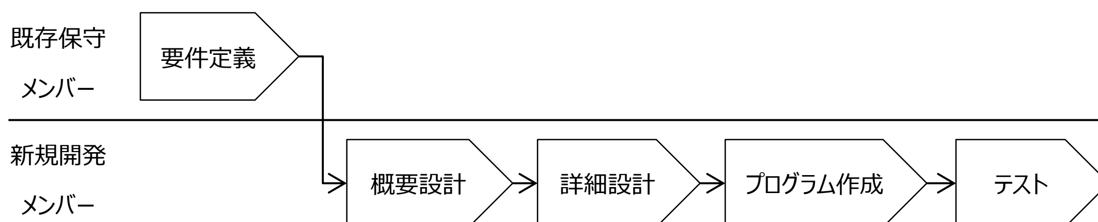


図 8-10 事例 3：工程と分業構造（担当）

c)　開発の特徴

本プロジェクトは既存の保守システムの改修であるが、この既存保守システムは開発時点でも稼働しており、保守を続ける必要があった（図 8-11）。

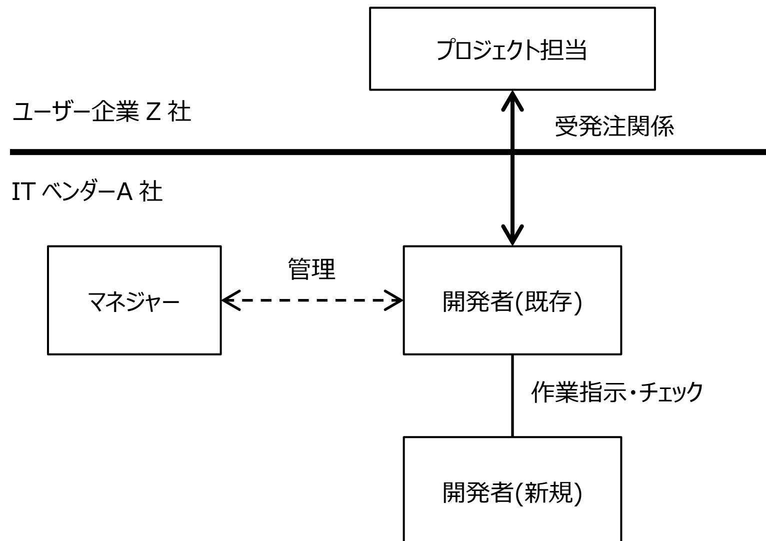


図 8-11 事例 3：メンバー構成

そのため、ベテランメンバーは、本プロジェクト以外の通常の保守作業なども多く担っており、改修作業は新規メンバーが中心になって進めることとなった。

プロジェクトの進め方は、新規メンバーが設計やプログラム作成、テストなどの大部分の工程を担当し、ベテランメンバーは Z 社側の担当者との打ち合わせを中心に、新規メンバーの作業のサポートやその内容のチェックを行った。

②　プロジェクトの結果

d)　品質

品質については、新規メンバーがシステムに慣れていないこともあり、設計工程とプログラム作成工程を何度も繰り返していた。例えば、データの受け渡しの際のデータの配列や、システムに沿った記述の仕方、画面やプログラム内の変数設定のミスなど、多くの問題が発生した。また、要件を十分に理解できておらず、データの編集要領を間違えてしまったこともあった。

しかし、ベテランメンバーが都度品質チェックを行うことで、十分な結果となった。特に、納品前に対面でのレビューを含む細かいチェックを行っており、最終的な品質の向上を図っていた。

e)　コスト

コストについては、新規メンバーがシステムに慣れていないこともあり、全体のスケジュールは守られていたものの、予定していた作業工数をオーバーしてしまった。設計作業はベテランメンバーの指導のもとで行ったが、本事例でも長年保守していたメンバーのプログラムの癖が強く、それに合わせた設計書の修正が何度も発生した。

長年の保守システムのため設計書は充実していたが、最新化されていない設計書もあった。さらに、プログラム自体もなかなか想定した動きにならず、実装したプログラムのソースコードが既存のプログラムと書き方が異なっていたため、標準化させるための修正が何度も発生してしまった。

f)　納期

開発期間は、ベテランメンバーが要件定義を少ない工数で終えることで作業工数の余力を残したこともあり、十分に余裕があった。また、新規メンバーに対するベテランメンバーの手厚いサポートもあり、予定されたスケジュール通りに開発作業は進み、最終的な納期についても問題は発生しなかった。

g)　人的・技術的サポート

ベテランメンバーと新規メンバーの間で、技術的なレベルやシステムへの理解といった点で大きな差があった。そのため、ベテランメンバーが進捗に合わせて細かいチェックを行うとともに、継続的に意思疎通を行った。特にベテランメンバーはその知識からシステムの全体を見渡すことができ、新規メンバーが陥りやすいミスをあらかじめ予防するとともに、設計書に記載されていない暗黙的な部分のサポートを行った。

また、設計やプログラムの修正する箇所について、ベテランメンバーが目途をつけていたため、それにより新規メンバーも作業が滞ることなく進めることができるとともに、ベテランメンバーの知識を学び、徐々に作業のフィードバックの回数も減っていった。くわえて、工程間の連携が維持できたことにより、仕様の齟齬や知識の伝達ミスを減らすことができた。

h)　プロジェクトの事後的評価

納期には間に合い、品質も問題なかったものの、そこに至るまでにかなりのプログラムの修正と設計のやり直しが発生した。しかしながら、納品し、本番稼働した後もシステムは安定し、障害などは発生していない。このため、Z 社の満足も高く、新規メンバーに対する信頼も得たことから、新規メンバーを含む A 社は引き続きシステムの改修について打診された。

③　プロジェクトの成功や失敗の要因

本事例のプロジェクトの成否を左右する諸要因として、次のようなことが考えられる。

まず、これまでの事例とは異なり、新規メンバーは設計工程からプロジェクトに参加しており、設計工程とプログラム作成工程の分断が起きていない。新規メンバーによる設計のため、不十分な部分が多く、その結果プログラム作成で問題を発見し、それを解決するとともに設計工程にフィードバックした。そして、設計書を修正し、再度プログラムに反映するという、設計とプログラム作成間で試行錯誤が繰り返された。

プロジェクト全体としては、Waterfall Model を採用していたものの、実際には内部で反復型の開発を行っており、このことにより工程間における反復活動、つまり試行錯誤が行われていたのである。

また、要件定義を担当したベテラン保守メンバーと、それ以降を担当した新規メンバーとの間の連携も分断されていなかった。ベテランメンバーは、頻繁に新規メンバーの作業状況を確認しており、Z 社へのレビューの際には必ず同席していた。ベテランメンバーと新規メンバーは、設計工程だけに限らず、プログラム作成工程、テスト工程、本番稼働を含むすべての工程で連携が取れていたことになる。

④　事例 3 のまとめ

事例 3 からいえることは、工程間を跨いだ反復活動がソフトウェア開発を進めるうえで有効であったということである。

要件定義と設計以降の工程は大きく区切られていたが､設計やプログラム作成､そして、テストまでの工程が連携されていたことで、情報の分断がされなかった。

さらに、実際の開発現場では厳格な Waterfall Model ではなく、反復型開発のように設計工程とプログラム作成工程を何度も繰り返す問題発見と問題解決、そのフィードバックという試行錯誤が行われ、それにより、仕様の反映ミスを無くすとともに品質の向上が図られた。また、要件定義を担当したベテランメンバーが、それ以降の工程にも何らかの形で携わっていたことで、部分的な知識しか持っていない新規メンバーに対し、システム全体の知識の伝達が行われた（図 8-12）。

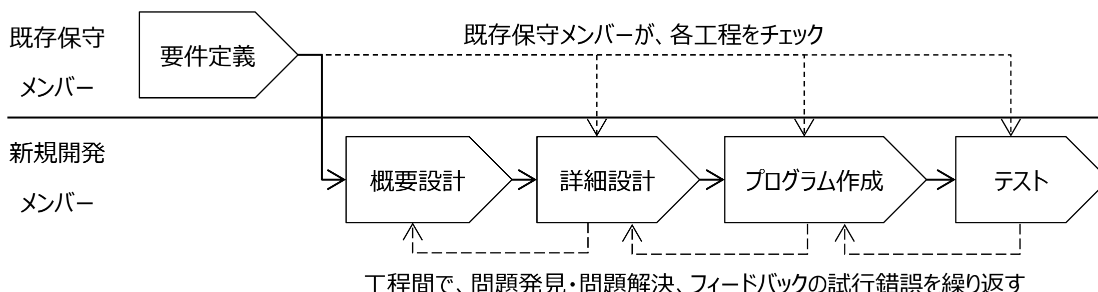


図 8-12 事例 3：工程間の連携

これまでの事例から述べられたように、設計工程で作成される設計書は必ずしも十分なものではない。そのため、事例 3 では A 社単独の開発であったことも工程間の分断が抑えられた要因の一つと考えられるが、なかでも設計とプログラム作成とがセットで、かつその間に反復活動が行われることが重要なのであり、要件定義、設計、プログラム作成を分離してしまった事例1や､企業間で工程を分割してしまった事例2との大きな違いとなる。

(5) 小括

本章では事例より、工程間の分業、特に上流工程と下流工程の分業によって引き起こされる問題を明らかにし､その中で設計とプログラム作成を繋げることの重要性を確認した。これら事例から確認されたことは、ソフトウェア開発では分業が行われているが、分割すべきところを誤ると､品質やコストなどに多大な影響が発生してしまうということである。納期については、契約の問題もあり、すべてのプロジェクトで遵守されているが、その内容も開発メンバーの過度な残業や土日の出社など、その努力や人海戦術により成り立っているのであり、その代償はコストの超過やメンバーの疲弊、不満の蓄積、退職などで払っているといえる。

まず、事例 1 から示されたことは、工程間をそれぞれ分業してしまうことが、作業の流れの分断をも招いてしまい、その内容も伝言ゲームとなってしまう恐れがあるということである。熟練した技術者を揃えても､隣同士の工程間で連携が十分でなかったり､さらに､その前後の工程と連携を取ることができなかったりすると、それが品質やコストに影響を及ぼしてしまうのである。特に、ソフトウェアを実際に作りあげる設計とプログラム作成間の連携がうまくいかないと、最終的に品質に大きな影響が発生してしまうのである。また、事例 1 は新規開発だったことも災いし、変化が激しいプロジェクトであった。そのため、仕様がなかなか決まらなかったり、決まった仕様の度重なる変更などに見舞われたりしたことで、工程間の連携もより複雑さを増していたのである。

次に、事例 2 から示されたことは、企業間の分業の難しさである。本研究では、下流工程のアウトソーシングを開発プロセスの分業の問題の 1 つとして検討してきたが、事例 2 はまさにその典型例といえる。企業間の分業により、そこでの情報連携や意思疎通、調整などの難しさが浮き彫りとなった。上流工程で決められた設計書を元に、下流工程は作業者として開発を行った結果、表面上うまくいったように見えたものの、実際に動かしてみることでその品質に大きな問題を残してしまった。特に、ソフトウェアは目に見えないため、設計書やテスト結果といった検証物で確認することには限界があり、実際に稼働して初めてわかることが多い。A 社はシステムの全体を見渡すことができていたが、設計書には暗黙的な部分までは表されておらず、上流工程と下流工程がこのような企業間の分業だったことにより、その連携も密接に行うことができず、細かいシステムに関する知識を B 社に連携することができなかったのである。

最後に、事例 3 から示されたことは、設計工程とプログラム作成工程が結びつくことの重要性である。設計以降を担当したメンバーは新規にもかかわらず、開発コストがオーバーしたものの、無事に納品、本番稼働を果たしている。要件定義を担当したベテランメンバーによって、各工程にも細かいサポートとチェックが行われた。また、新規メンバーは設計以降のすべての工程を担当し、設計に含まれていた不備をプログラム作成で発見し、その問題を解決するとともに、設計工程にフィードバックするという反復作業を繰り返し、品質を上げていったのである。

次章では、事例分析から得られた結果をもとに、工程間の分業がどのように開発プロセスや知識の適用、知的能力の発揮を困難にさせているのか、試行錯誤を繰り返す「創造的な問題解決過程」と、それに携わる技術者の「知識マネジメント」の視点から考察を行う。

第9章 ソフトウェア開発プロセスにおける知識労働と分業の問題－考察－

事例の分析からソフトウェア開発プロセスにおいて、従来の Waterfall Model ではしばしば工程間の連携が分割されることが確認されるとともに、そうした工程間の連携が確保されることがソフトウェア開発には重要であるということが明らかにされた。

本章では、前章で示された開発の現場で行われた作業プロセス間の連携や開発担当者の知的活動とプロジェクトの成否との関係に関する分析を中心として、ソフトウェア開発プロジェクトの成否を左右する諸要因を考察する。また、ソフトウェア開発プロジェクトの成否に影響を及ぼす諸要因が、なぜ、またどのようにそうした開発プロジェクトの有効性の発揮の有無と関係しているのかについて、有効なソフトウェア開発プロセスが試行錯誤を繰り返して問題発見、解決を図っていく知識労働を伴うものであること、さらにそれに携わる技術者が創造的作業者としてプロジェクトに従事することが必要であることを知識マネジメントの視点から論じる。

(1) 開発事例から明らかになった問題

前章の事例分析では、3 件のソフトウェア開発プロジェクトを対象に、ソフトウェアの開発プロセスにおける分業関係の編成によってどのような問題が生じるのか、さらにそうした問題が開発プロジェクトの成否にいかに関係しているのかを明らかにした。事例分析による発見事実は以下のように要約される。

① 事例 1

事例 1 からは、工程間の分業で生じる問題のほか、頻繁な設計変更が発生する問題、そして熟練技術者でもその工程間が分断されてしまうと連携が難しくなってしまう問題が明らかにされた。事例 1 は同じ企業の中におけるメンバー間の分業を対象としたものであったが、もう 1 つの特徴として、新規開発のため日々変更していく仕様や設計に悩まされていたという点がある。これまで述べられてきたように、ソフトウェア開発では要件定義工程や設計工程で仕様が確定することは少なく、そのため曖昧なまま工程を進める過程で、徐々に擦り合わせを行っていくという特徴がある。さらに、頻繁な仕様変更が発生するため、そういった仕様や設計の変更に対応できることが求められる。

また、事例 1 では、長年システム開発を行ってきた高い専門性を持つ技術者がメンバーとして採用されたが、そのメンバー間の連携が順調にいかなかった。ソフトウェア開発に高い専門性が必要なことはすでに述べてきたが、ソフトウェアは技術者がチームを組んで開発するため、そういった専門性の高い技術者が隣り合ったメンバーや工程の内容を理解しなければ、その作業も順調にいかないといえる。特に、事例 1 は新規開発であったため、仕様や設計を一から決めなければならない部分が多く、こうした作業における知識労働を

分割してしまったことで、工程間を通したイノベーションが困難になったことが考えられる。

② 事例 2

事例 2 からは、企業間の分業の難しさと上流工程で決められた要件や設計が必ずしも十分ではないことが明らかにされた。事例 2 では、上流工程と下流工程で企業間の連携が行われ、そこでは上流工程を担当した企業が作成した仕様や設計を元に、下流工程を担当した企業がソフトウェアを作成する分業が行われていた。しかし、ソフトウェアの設計書には、設計に関わる情報がすべて記載されているわけではなく、ソフトウェア開発に必要となるものは満たしていても、それ以上の十分なものまでは書かれていない。また、そのような設計書に書かれたものは、必ずしもプログラムとして実際に作成して動かしたものではなく、稼働させてみて初めて分かるような多くの問題を抱えている。そのため、設計に含まれた問題を見抜く力が必要となるのである。

さらに、事例 2 では、企業間の分業が行われているが、そこでの情報連携の難しさも浮き彫りとなった。社内でのやり取りとは異なり、企業間でやり取りを行う場合、分業をどのように進めていくのか、リスクと成果をどのように分担するか、そして意思疎通や発生する問題をどのように解決していくのかを決めなければならず（武石, 1999）、さまざまな費用が発生することになる。

③ 事例 3

事例 3 からは、設計工程とプログラム作成工程を交互に繰り返す試行錯誤の取り組みがその特徴として明らかにされた。ソフトウェア開発の過程で、設計工程に含まれていた不備をプログラム作成工程の中で発見し、その問題を解決するとともに設計工程にフィードバックするという一連の作業を何度も繰り返し、品質を上げていった。このようなソフトウェアの設計とプログラム作成の工程は、そこで生じる複雑な問題に対して試行錯誤を繰り返し、解決を図っていくという点において、製造業の製品設計のようなものとして捉えることができる（妹尾, 2001; Cusumano, 2004）。こうした複雑な問題を解決していくには、専門化や細分化することが重要となるが、前章の事例で示されたように、特定の工程に対する専門性だけでは不十分であり、専門性を持った技術者間や工程間のやりとりにソフトウェア開発の難しさが存在するのである。

以上の発見事実を踏まえて、次節では、ソフトウェア産業の分業構造およびソフトウェア開発プロセスにおける知識労働の重要性、さらに情報技術を含む知識を使いこなして問題解決に結びつけたり、設計変更に対応したりする組織能力について検討する。また、ソフトウェアの設計の複雑さに対し、専門化と熟練、細分化がどのように影響するのか、そ

こに含まれる問題を発見し、その解決のための試行錯誤をいかに実現するかを検討するとともに、開発に従事する作業者の問題解決能力の根幹に関わる経験やそれに基づく類推といった知的活動の重要性について論じる。

(2) ソフトウェア開発の分業構造と問題解決

① 専門性の問題

前章の事例 1 では、長年システム開発を行ってきた高い専門性を持つ技術者がメンバーとして採用された事例を検討した。

ソフトウェア産業では、製造業や建築業の上流・下流工程の分離と、それに影響された Waterfall Model のような工程の前戻りを許さないといった開発手法を援用してきたが、製造業や建築業ではこの工程の分離により、上流工程と下流工程で設計技術者や製造技術者、建築家や大工などの熟練、専門分化が進んできた。

ソフトウェア産業でも、工程を分離することにより、技術者をシステム分析・コンサルタント、設計者、コーダー、テスター、保守要員などの分業、専門化が進んでいった[136]。

こういった専門性について、守島（2002）は、人材マネジメントの観点から 20 世紀前半を分業と専門化による大量生産の時代とし、そこでは熟練による効率性の達成のため、組織は固定、かつ安定的に分業、専門化され、従業員も一つの職種に専門・固定化されたとしている。そして、この専門化と固定化により、同じ専門知識や行動様式を身に付けたメンバーが、迅速、かつ確実に業務を遂行し、ヒエラルキー型のマネジメントがうまく機能していたと説明している。

このような専門性の細分化、深化について、高橋（2012）は、キャリア形成の大きな流れとして、多くの分野で高い専門性が必要となり、さらにその専門性を持った人たちがチームを組まなければ成果をあげることができないような仕事が出てきており、理論体系的な専門性を持ったプロフェッショナルが求められていると述べている。

一方で、このような専門化の度がすぎると、辺境的な思考パターンの原因となってしまうため、専門能力とまとめる能力を共存・バランスさせるべきという指摘もある（藤本, 2001b）。

特に、藤本（2001a）は、管理層やインダストリアル・エンジニアリングの専門家による意思決定と現場の作業者との間に生じる垂直的な分断の流れについて、20 世紀後半のアメリカの製造システムで競争力低下の一因となった過度の分業化や管理層の現場からの遊離であることを指摘している。

藤本（2001b）は、個々の技術者の作業割りあて範囲が広く、専門化の程度が低いほど、

[136] 第 1 章の用語の定義で説明したように、日本ではこれら職種を総じてシステムエンジニア、もしくはその一部をプログラマーのように、まとめて呼ぶことが多く、その職務上の定義は非常に曖昧となっている。

プロジェクトはより早くかつ効率的になるとし、過剰な専門化はパフォーマンスに悪影響を及ぼすと述べている。さらに、社内調整型のプロジェクト・リーダーが強いほど、プロジェクトはより早く、より効率的になる傾向にあるとしている。

また､専門性の高い人材を集めるだけではプロジェクトはうまくいかない可能性があり、プロジェクト・リーダーによる、プロジェクト全体とその開発ステージごとの進捗状況に応じたミッションをメンバー間で共有化させるといった境界マネジメント能力が知識の組織的創造に決定的意味を有してくる（林, 2008）。

以上のことは、前章の事例でそれぞれの工程や作業のプロフェッショナルが集まったにもかかわらず、プロジェクトが順調にいかなかったことからも確認される。

こうした高度な専門能力を持った人材は長期に育成、留保することが難しく、事例 1 では新人からベテランまで社内から技術者をかき集めるより他がなく、さらに知識の共有化を困難にしていたのである。

Tomayko and Hazzan（2004）は、ソフトウェア開発の人間的側面に注目し、ソフトウェア技術者同士の協力がソフトウェア開発のプロセスにおいて必須であることを指摘している。さらに、ソフトウェアはまったく新しく見えるものでも、以前のプロジェクトと共通する点が多く、そのため単純な経験データが重要であることを指摘している。

また、同じ空間、課題を共有したとしても、共通の知識ベースがなければ、理解が成立しないという可能性もある（伊丹・松島・橘川編, 1998）。顧客の要求内容は日々変更していくため、それに柔軟に対応することが求められており、特定の工程に対する専門性だけでは不十分となる。そのため、全体の流れがつかめるような、企業の境界を超えた幅広い関連知識を持っていることが重要となるのである。

②　企業間の分業

前章の事例 2 では、ソフトウェア開発における企業間の分業の難しさと上流工程で決められた要件や設計が必ずしも十分ではないことが明らかにされた。また、ソフトウェア開発では、要件定義や設計などの上流工程で決められたことが、下流工程で変更されてしまうことが多いことはこれまで述べられてきたが、その原因として上流工程で作られた要件や設計自体が不十分なこともあげられてきた。

ソフトウェア開発では、専門性を持った人たちがチームを組み、作業や工程を分割し、協業していく部分が多く発生するが、このような分業が不可欠であるならば、どの部分までを社内で行い、どの部分までを外部に任せるのか、そしてどのような相手にその作業を委託すればよいのかを考えなければならない。つまり、特定の業務を外部の企業に任せる場合、どこで内と外との境界を設定するかという取引の管理の問題を検討する必要が生じる（武石, 1999）。企業の経営戦略にとって、相手とどのような関係を築きあげ協業を進めていくのかということは、重大な課題である（武石, 2003）

企業間で、開発や生産の連携を行うには、互いの知識や資源、目的などに合わせ、企業間の取引関係を適切に設定していかなければならない。企業の分業については、Smith（1776）が、ピン工場を事例として、個人が複数の仕事を行うよりそれぞれ専門的な仕事を協働するほうが全体の生産性があがることを述べている。一方で、こういった分業がより細分化され、そして専門化が進むことでその調整のための費用も複雑となり、増加していく。

そうした分業や協業に関わる費用は、金銭的なものだけに限らない。むしろ、金銭的なものよりも、分業関係者が調整のためにかける手間の時間やエネルギーといった表に出てこないものが多く存在し、こうした分業間の調整に多くの時間やエネルギーが取られてしまうのであれば、その分業自体が経済的に見合わないことになる（伊丹・松島・橘川編, 1998）。

非定型的な作業、規格化されない情報や知識、作業者の行為や経験を離れてしまうことで容易に機能しないような知識の移転は、取引コストを著しく高めることになる。企業が境界を設定したとしても、その取引関係や分業プロセスのあり方によって、その費用や便益は異なってくる（武石, 1999）。こうした分業によって生じる不確実性を抑えるためには、頻繁なコミュニケーションをとり、相互の情報の精度を高めることが重要となる。

ソフトウェア開発においても、その工程間を分業してしまうことが、知識の適用や知的能力の発揮を困難にしてしまうことが考えられ、分業間の調整や、情報、知識の移転のために取引コストが高くなる。例えば、事例 2 で示されたように、特に設計書に含まれない暗黙知的な部分は連携が難しく、工程間を分断してしまったことで、知識の移転が阻害され、本番稼働後に障害を引き起こしてしまった。

取引関係や分業プロセスについて、伊丹・松島・橘川編（1998）は、産業集積という観点から同じ分業の方法でも、シリコンバレーと日本やイタリアとの違いについて述べている。それによると、シリコンバレーでは同一業種の類似企業が集まっており水平分業が中心であるが、日本やイタリアでは水平分業だけでなく垂直分業の程度も高く分業が繋がり合っているという。そして、シリコンバレーでは、類似の企業が競い合って適者生存を決める大きな市場であり、そういった企業の参入と退出の競争により活性化を図っているとしている。

武石（1999）は、このような境界設定と取引関係が相互に関連している点が重要であり、製品開発のように、その内容や成果の見極めが難しく不確実性が高いものの場合、適切な相手企業を見つけて取引や行動を監視するより、その業務を内部化したほうが効率的であることを指摘している。一方で、そのような内部化によって生じるコストも考慮に入れなければならず、そういった取引費用を抑制できれば、外部化を通じて開発費用やリスクの軽減も可能となるのである。

③ 情報技術と問題解決能力

第 4 章の製品・工程アーキテクチャーの議論において、インテグラル型の製品は多くの

部品が機能的、かつ構造的に複雑な相互依存関係にあるため、部門間での協調的な知識創造活動や企業間での連携を行う必要が指摘された（中川, 2011)。また、前章の事例より、設計変更やソフトウェアの検証といった問題解決サイクルを何度も繰り返す必要性が指摘された（藤本, 2007)。

このような問題解決方法について、最新の技術や手法を導入していくことが有効のように考えられるが、藤本（2001a; 2001b）は、製造業の生産マネジメントを例に、これまで多くの企業が組織やプロセスの準備をすることなく新しい情報技術を過信し、その結果、最新の技術や手法を導入することで失敗を繰り返してきたことを指摘した。特に最先端の情報技術を持つことが、開発期間の短縮競争など企業の競争優位を築くための必要条件であっても十分条件ではないことを指摘している。また、藤本は、組織的な問題解決能力なくして目標が達成されることはないとし、この組織的な問題解決能力を構築するためには、自社の組織能力の正確な現状把握が必要であり、問題解決に対する深い知識と高い組織能力を持つ企業のみが、パフォーマンス向上に繋げることができると述べている。

このような最新の技術や手法を導入せず、これまでと同じような技術や設計図を使い続けることは、管理や見積りを容易にするほか、システム導入の観点で安定性がある。しかし、ソフトウェアを含む IT 技術の進歩の速度は速く、より新しいサービスや顧客の要望の実現などで従来の技術や手法では劣ってしまう可能性があり、そのためソフトウェアを開発する際にあえて新しい技術や手法、ソフトウェアの最新バージョンといったものを積極的に採用する面がある（中尾, 2009)。

こうした情報技術と組織の同化について、松田（1990）と白石（2010）が論じている。松田（1990）は、情報技術の組織への同化が不十分であることを指摘している。その上で情報技術の現在の組織への役割に関する研究は多いが、未来の組織への貢献についての研究がされていないことを指摘し、技術の意義を現在と未踏の未来への橋渡しであると述べている。

一方、白石（2010）は、知識ベースビューの企業理論から、組織と情報技術の同化が不完全なだけでなく、さらにコンピューターを使いこなせていない原因を組織パラダイムの欠如にあると述べ、情報システムや技術ではなく組織能力が知識の共有共用を促進し、組織内に知的触発を発生させるとしている。また、組織のコミュニケーション特性や意思決定、構造も知識の共有共用へ作用すると述べ、知識の共有共用のためには、新しい情報技術を使いこなす体制と組織能力の形成が必要だとしている。

この 2 人の研究から、情報技術は組織的な能力に強く影響されることが指摘されるが、このような情報技術について、藤本（2001b）は、製品開発を例として、情報技術などを使いこなし問題解決に結びつける組織能力の重要性を説いている。藤本は、自動車産業などの製造業の製品開発を取り上げ、その製品開発のプロセスを「多分に繰り返し性のある組

織的な問題発見・問題解決活動である」[137]と指摘し、顧客のニーズの把握から製品コンセプトや仕様の決定、そして詳細設計へと続く、情報ストックの連鎖的翻訳のプロセスや問題解決サイクルの束とみなしている。

この翻訳プロセスとは、製品の仕様を設計に翻訳する製品エンジニアリングのことである。製品開発は、製品全体に関連した試作と実験の問題解決サイクルであり、要求仕様や原価目標を満たすまで、設計変更を繰り返していく。このためには、詳細設計などに関する組織的な問題解決能力を高めることが重要である（藤本, 2001b）。

ソフトウェア開発においても、システム稼働後の不具合は設計工程の部分、特にプログラム作成工程と密接にある詳細設計に原因があることが多く、設計工程における問題を発見する能力と、その問題を解決する能力が重要であることが指摘されている（独立行政法人情報処理推進機構, 2012）。

さらに、藤本（2001b）は、この問題解決のサイクルの束という観点から、設計に必要な能力について、アイデアのレパートリーや技術知識の幅の広さと深さ、ノウハウなど、「所与の問題群に対して、迅速かつ正確に設計代替案をサーチ（探索）する、組織的な知識と能力にほかならない」[138]と述べている。

事例でも示されたように、ソフトウェア開発は複数人のチームで作り上げる協働作業であり、開発プロセスで発生した問題を発見、解決していくサイクルを何度も繰り返していくことになる。そのためには、チーム内だけでなく、チーム間や企業間を含めた協調的な知識創造活動や連携が必要となり、そのためには、情報技術などを含む知識を使いこなし、そういった問題解決に結びつける能力こそが、ソフトウェア開発に重要となるのである。

(3) ソフトウェア開発プロセスにおける知識創造活動

①　設計の複雑性

先行研究においてソフトウェア開発は製品設計に近いことが指摘されたが（妹尾, 2001; Cusumano, 2004）、大日方（1971）は、プログラム開発について研究開発的な性格が強いことを指摘している。大日方は、一般的に研究開発は、改良を重ねて試作を経る必要があり、その成果がそのまま実用になることは少なく、アイデアを製品化するためには試行錯誤を繰り返しながら、アイデアを取捨選択して具体化する必要性があることを指摘している。一方で、ソフトウェアのプログラム開発は、研究開発的な性格にも関わらず、試作品段階のものが直ちに実用品として使用されてしまうため、その実用化の工程には多くの無理が含まれており、その結果、実用化されたプログラムには多くの不備や潜在的なバグが含まれていることが考えられると述べている。

特に、本研究が対象とするカスタムソフトウェアは企業固有の業務に合わせた一品受注

[137] 藤本（2001b）p.277
[138] 藤本（2001b）p.263

生産の特徴が強く、毎回ほぼ新しく作成することが多いため、プロトタイプとしてシステムを作成するようなことはその規模や予算、スケジュールなどにより難しい面がある[139]。

藤田（2013）は、ソフトウェアの造り込みが十分ではない理由として、電機産業を例にとり、日本がかつてハードウェアの技術力で世界をリードしていたため、社内のソフトウェア部門の力が弱いことが多いと指摘している。

藤本（2001b）は、このような試行錯誤や試作品を作成するうえで、設計を変更するような場合に組織能力が必要であると述べている。ソフトウェアや製造業の設計には多くの変更が発生することはすでに述べたが、藤本によると、製造業の設計作業で発生する変更に対して、「設計改善の提案力」と「設計変更の混乱収拾力」の 2 つの組織能力が重要であると指摘している。

製造業において、部品などの設計変更は他の部分に大きな影響を与えるが、藤本は、それら部品間の複雑な相互依存関係のなか、設計変更による影響を予期する技術知識やシミュレーション能力を重視することが重要であると指摘している。そして、そのような状況では、部門間の緊密なコミュニケーションや情報共有が解決のために重要であるとも述べている。

こうした製造業の設計について、Ferguson（1992）は、設計者自身の心に浮かんだイメージを図面や仕様書に落とし込むものとし、その過程で設計者は明確になっていない問題を解決していくと述べている。そのうえで、この問題について、唯一の正しい回答は存在しないとし、「心の中の構想を明確にし、どうすれば不明瞭な要素をはっきりさせられるかを求めて奮闘するのが設計の過程だが、技術者はこの過程を通して多くのことを学ぶ」[140]と指摘している。

さらに、Ferguson（1992）は、設計書に表された図面は一見正確に見えるが、その背後には「多くの公式に則らない選択や言葉で表せない判断、直観の動き、そしてあらゆるものの作動の仕方についての過程が隠されている」[141]ことを指摘し、アイデアを人工物へと変える過程は複雑であり、科学より芸術に近いと述べている。

以上のような製造業の設計に含まれる特徴やそれによって生じる問題は、ソフトウェア産業にも強くあてはまる。

ソフトウェアを含めたコンピューターが複雑な理由として、Dreyfus and Dreyfus（1986）は、コンピューターである機械が人間と知的対話を行うには、コンピューターに人間という生き物を理解させる必要があると述べている。Dreyfus and Dreyfus によると、コンピュ

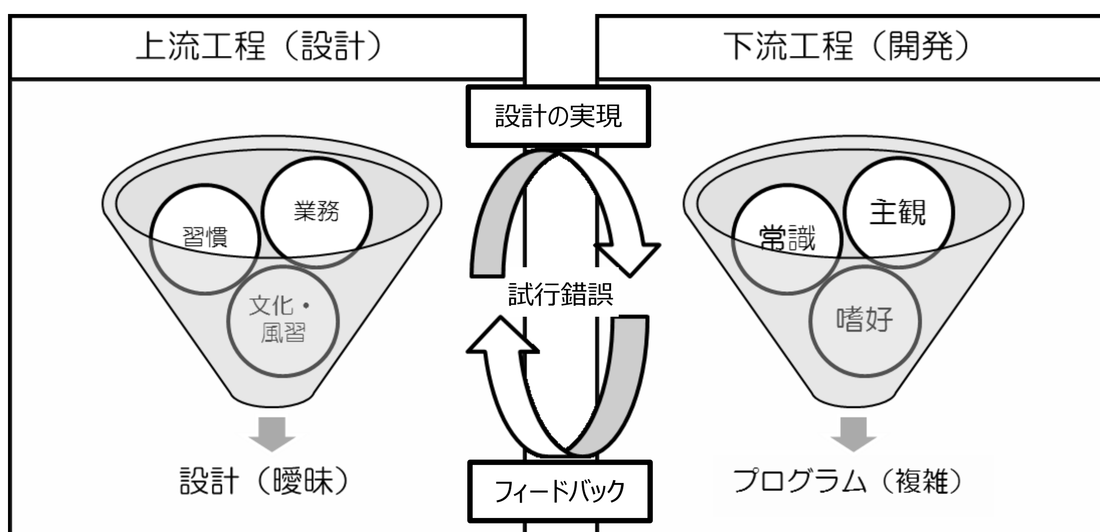
[139] ソフトウェア開発は、デザイン思考による製品開発のように、開発の過程で試行錯誤を行ってプロトタイプを作成していくこともあるが、カスタムソフトウェアは 1 品受注生産のため、ある程度作ってから壊してしまうようなプロトタイプは敬遠される傾向にある。

[140] Ferguson（1992）p.16

[141] Ferguson（1992）p.19

ーターを動かすには、人間にとってはあたり前のことをデータと規則という形でコンピューターにインプットさせなくてはならない。つまり「コンピュータに人間の信念の体系をインプットする必要がある」[142]のである。さらに、この規則には幾つもの例外が存在し、その例外のための規則を作成する必要がある。こうして例外のための例外のように、ネズミ算式に規則の数が増えていくため、あらゆる状況を想定したプログラムを作成することは不可能であることを指摘している。

ソフトウェアは、この上流工程で作成された設計書を元にプログラムを作成することになるが、設計書が曖昧で不十分なもののため、ソフトウェアの作成に大きな負荷となっている。くわえて、作成するプログラマーの主観や常識、嗜好といったものが加わってしまうため、プログラムにより複雑になってしまうのである（図 9-1）。



出所：Dreyfus and Dreyfus(1986)、Ferguson（1992）を元に筆者作成

図 9-1 ソフトウェア開発の困難性と試行錯誤

特にソフトウェアは目に見えないデータであり、現実世界の業務、習慣、文化、風習までをもコンピュータープログラムとしてシステム化しようとしてきた。このような設計作業は、不明確な問題を解決していく過程であるが、Ferguson（1992）が述べたように唯一の正しい回答が存在せず、設計者個人の判断や直観といった言葉で表せないものを含んだ複雑なものなのである。

ソフトウェアプログラム作成工程は Cusumano（2004）も指摘するように、製造活動というよりは製品設計のようなものとして捉えることが可能であり、そこでは業務、習慣、文化、風習といった個々の企業、人間で異なるものを対象としており、不確実で複雑な問

[142] Dreyfus and Dreyfus（1986）p.132

題に対して、試行錯誤を繰り返して問題の解決を図る知識労働としての側面を持っているのである。

② 経験と熟練

事例 1 では、それぞれの分野で高い専門性を持つ、熟練した技術者がチームを組んで開発を行った。ソフトウェア開発に限らず、多くの分野で高い専門性が必要となり、さらにその専門性を持った人たちがチームを組むことの重要性を述べたが、これまでの日本の製造業では、工業化社会を前提とし、熟練技術者の暗黙知のもと安価で高品質な製品を生産してきた。

企業の人材の中心はこのような熟練工に置かれ、工場内の QC サークル活動やカイゼン活動を通じて人材育成がおこなわれ（齋藤, 2009）、そこから生まれた暗黙知などのナレッジが企業の中核となる資産となり、競争優位の源泉となっていった。こうしたモノづくり中心の経営では、いかに製品を効率的に生産していくか、そのプロセスに焦点があてられていった（谷内, 2008）。近代的な工場生産の方法が確立されることによって、機械化と分業の高度化、そして職務の細分化の方向で合理化が進められてきた（神田, 1981）[143]。タスクを個人が処理できるレベルまで分割することで、個々の労働者に期待されることは、事前計画通りの産物を低コストで算出することであり、与えられた業務を素早く、確実に処理することとなる（妹尾, 2009）。

ソフトウェア産業において、このような作業を手順化し、計画通りに開発を進める管理主体の方法として、1970 年代の日本のコンピューターメーカーがソフトウェア・ファクトリーを採用しようとしたことに繋がり、その結果、失敗したことはすでに述べた。競争環境が大きく変化してスピードが速まり、多様な問題が発生するような状況では、上司の指示に従って効率よくするこれまで重視されてきた方法は通用しにくい（守島, 2016）。さらに、過剰な分業により、作業を細かく専門化しすぎると、生産システムが硬直化し、そのための調整コストやムダが発生してしまう可能性もある（藤本, 2003）。

一方、そのような多様な問題が発生する場では、その問題の性質に応じて解決に必要となる行為も異なる。特に、発生した問題が変異性に富むような場合は、多様な例外を処理する必要がある。こうした例外の処理には十分な知識が必要であるが、そうした知識が不足する場合には、個人の直観や経験、推測といったものに依存した問題解決活動が必要となる（Perrow, 1967; 加護野, 1980）。

[143] 神田（1981）は、社会・技術システム論に関連して、この工場生産方法の確立により、労働者は極端に細分化された単調な反復作業を行うこととなり、かえって労働者に作業に対する不満を抱かせると述べている。そして、その結果、本来意図された生産システムの効率の向上を阻害する事態に繋がり、企業的にも重要な問題となっていることを指摘している。

このような、変異性に富んだ異常事態への対応能力として、「知的熟練」（小池, 1977）があげられる。小池（1977）は、ふだんと違った作業を処理する能力やスキル、つまり問題や変化といった異常事態に対応する能力を「知的熟練」と呼び、その「知的熟練」の程度が人材の価値を決定するとしている。

ソフトウェア開発の設計者が、設計の時に多様な問題を事前に予測し、その問題に対する適切な解決ノウハウが判明しているのであれば、コンピュータープログラムに組み込むことが可能となるが、前もってすべての問題を見通せるわけではなく（小池, 2001）、あらゆる状況を想定したプログラムを作成することは不可能である（Dreyfus and Dreyfus, 1986）。小池（2001）は、このような設計の問題に対する技能について、最新の知識のほか、多様な問題を解決してきた経験から導き出される予測する能力を指摘している。

経験に裏打ちされた能力、すなわち熟練は単に器用でうまく加工できるだけのようなものではなく、仕事の根本にある原理を理解し、多様な注文に対応できる能力である（伊丹・松島・橘川編, 1998）。例えば、会計や経理に関するソフトウェアの場合、技術的な知識だけでは開発することは難しく、要件定義ですべての仕様を洗い出すことは困難である。このような場合、Agile 開発のようにさまざまな要望に応えるために仕様変更を前提とし、仕様を擦り合わせつつ部分的に機能を完成させていく方法が考えられる。生じる多様な問題を解決するため、ソフトウェア技術者は、情報技術の専門知識だけでなく、関連する知識や文脈的知識を駆使しており、開発するシステムの業務に関する基礎知識や業務フローを理解することで有効なシステムが開発できるのである（三輪, 2001, 2010）。

こうした経験について、Collis and Montgomery（1998）は、工学技術の進歩や技術的改善といった要因によっても経験効果をもたらすが、そのような経験効果は、個人の実体験を通して獲得される機能であるため、ある程度の時間を要するものであり、企業内であっても移転することが困難であることを指摘している。

経験から熟練を形成するには、単に経験があるだけではなく、なぜそのような結果になったのかを問う能力と、幅広い関連知識を持つことが重要となるのである（伊丹・松島・橘川編, 1998）。

③ タスク処理と類推

ここまで専門性と熟練について述べてきたが、妹尾（2009）は、労働者が技能を発揮する場面に着目し、製造業が低コスト戦略を遂行するうえで、「生産システムの一要素である工場労働者の道具操作能力向上が技能熟達として重視される傾向にあった」[144]と述べている。その上で、解決すべき問題が与えられているタスク処理型と解決すべき問題が与えられていない知識創造型の 2 つに分類している。

[144] 妹尾（2009）p.193

守島（2002, 2011）によると、人材マネジメント論では、労働者を情報やタスク処理のための人材とみなし、その人材の貢献として、与えられた目標や課題を処理することとみなされることが多かったとしている。そして、このような労働者をタスク処理のための人材として取り扱うことで、意思決定課題の設定やその処理方法のための知識創造活動はほとんど要求されていないと指摘している。

しかしながら、マニュアル化により、非熟練作業のみで完結してしまうような作業は、製品が多様化して変化の速さが高まるに従って減少する（藤本, 2001b）。企業の中で実際に価値を生み出しているのは、タスクを処理するだけの人材では不十分であり、知識創造的な部分をもった人材であり、不確実性や変化へ対応できる問題解決能力ももった人材である（守島, 2011）。

守島（2002）は、人材の貢献を「考える」という視点で見ることで、変化や不確実性への対応､知識創造を前提とした人材を支援する必要があるとも指摘している｡そのうえで、守島は、知識創造型の人材マネジメントの視点から、知識創造を無意味で不確実な情報からパターンを見出すことで意味を付与するプロセスとし、良質な経験を通じることで専門性と知的創造ためのスキルを獲得していくとしている。そして、こういった知識創造場面で労働者に求められるのは、チーム作業を通じて、問題発見と問題解決という一連の作業が終わった時に、また新たな問題を発見していることだと述べている。

こうした良質な経験は、他人の経験を共有することでも体験可能であり、企業のなかでの配置の問題だけではなく、組織の境界を越えた経験の積み重ねへと発展していくとされる（守島, 2001a）。

守島（2001a）は、人材の知的競争力として「単にタスクを実行する能力やスキルだけでは不十分で、不確実な事態に対応したときに、その原因を推測し、解決法を考える力が重要である」[145]とし、そうした問題解決能力の根幹の 1 つとして類推をあげている。この類推とは、目の前で発生している問題について、以前の経験をもとに解決策を考える能力である。つまり、まったく新しい状況への対応ではなく、これまでの似たような状況に基づいて対応を考える能力である。

この類推が組織の中で効果的に成立するためには、良質の経験と、新しい状況で類推に基づいたアクションを起こす勇気が必要であり、特に良質な経験によって、多くのクリティカルな場面での経験値が蓄積され、良質の類推が可能になる（守島, 2001a）。

ソフトウェア開発においても、その作業工程は細かく分割されるが、Waterfall Model のような上流工程や下流工程といった上下関係の扱いではなく、それぞれの工程の専門性が高まっており、例えば、ソフトウェアの設計、開発、運用、保守など、それぞれ異なった専門性が存在する（尾谷, 1997）。大規模、複雑化していくソフトウェアに対応するために

[145] 守島（2001a）p.99

は、単なる作業従事者ではなく、専門性に富んだ知識労働者が必要である。

ソフトウェア開発において、これまでの Waterfall Model に準拠した方法では、決められた要件や設計通りにソフトウェアを作成する必要があり、下流工程は与えられたタスクを処理することが目的となり、そこでは問題解決に関わる知識創造は必要とされてこなかった。

しかし、事例でも示された通り、現実ではそのようなタスク型のソフトウェア開発では予期せぬ例外など対応することは難しく、特に設計にはさまざまな問題が含まれている。ソフトウェアを開発していく過程において、そうした問題の発見と問題の解決に関する知識創造活動が必要となるのである。

問題発見の機会を有する下流工程の情報が設計作業に生かされることで、開発プロセスは価値創造のプロセスとなりうる。このように考えることで、ソフトウェアの開発は、顧客ニーズに対する理解、経験的に把握された問題解決に関する知識が要求されるマニュアル化できない高度な知識労働としての側面を備えていると捉えることができ、その分業の仕方も見直す必要が出てこよう。知識労働者の仕事は、その目的や内容が所与されるマニュアル・ワーカーとは異なり、提案力や問題解決力を必要とする（三輪, 2014）。そして、そのような知識労働者には、類推により過去の経験に基づいて対応する能力と、新しく困難な状況でも創造的な問題解決ができる能力が重要なのである。

(4) 小括

本章では、前章の事例分析から得られた結果の考察として、ソフトウェア開発プロジェクトの成否に影響を及ぼす諸要因が、なぜ、またどのようにそうした開発プロジェクトの有効性の発揮の有無と関係しているのか、ソフトウェア開発プロセスで見られた試行錯誤を繰り返す「創造的な問題解決過程」と、それに携わる技術者の「知識マネジメント」の視点から改めて検討してきた。

ソフトウェアの開発プロセスでは、プログラム作成工程と密接にある設計工程に問題の原因があることが多く、設計工程における問題を発見する能力と、それを解決する能力が重要となる。このようなソフトウェアの設計は多くの変更が発生するが、それに対して、設計改善の提案力や設計変更による混乱の収集力といった能力も重要となる。くわえて、設計工程で決められた情報は机上のものであり、不完全であることが多い。このため、開発工程で、設計工程までに解決できなかったそこに含まれるさまざまな問題を発見、解決していかなければならず、そうした問題発見や解決の機会を有する下流工程の情報が、設計作業に生かされることで、開発プロセスは価値創造のプロセスとなるのである。

このようなソフトウェアがもつ不確実性や複雑性に対し、技術者は試行錯誤しながら問題を解決していくとともに、経験を積んでいく。特に、良質な経験により、専門性と知的創造のためのスキルが備わり、そのようなソフトウェアの不確実性に対する類推が可能と

なるのである。

　また、技術者は経験を積むことで熟練していくが、そのような熟練には、定型的な作業に対する熟練と、非定型的な作業に対する熟練に分けることができる。特にソフトウェア・ファクトリーに代表されるものが定型的な作業といえ、そこでは熟練した工場型の作業員が必要とされた。しかし、事例から示されるように、ソフトウェアは開発しながら改良、変化が加えられていく。そのため、ソフトウェア開発者には、そのような非定型的な部分に対応することが求められるのであり、そのことがソフトウェア開発の困難さに繋がっている。

　さらに、こういったソフトウェア開発の非定型的な作業、つまり例外のような出来事を処理するには、単にタスクを実行する能力やスキルだけでは不十分であり、十分な知識や専門性といった高度で知的な作業が要求される。しかしながら、事例から示されたように、特定の工程や機能に対する狭い専門性の高い人材だけでは開発プロジェクトはうまくいかない可能性があり、メンバー間で共有化することが重要となる。そしてそのためには、ソフトウェア開発の一部分だけではなく、全体の流れがつかめるような、企業の境界を超えるような幅広い共通の知識を持っていることが重要となるのである。

第10章 結論と今後の課題

(1) 結論

① 本研究の要約

本研究では、日本の受託ソフトウェア開発を対象とし、その競争力が低いといわれる原因をソフトウェアの開発工程の分業構造の問題、とりわけ設計工程とプログラム作成工程を分離することで、プログラムの設計と作成の間の連携が分断され、その結果開発プロセスにおける試行錯誤を通じた創造的な問題解決が阻害されることにあるということを明らかにした。その上で、本研究は質の高い、革新的なプログラム開発には、こうした設計および作成工程間の連携が不可欠であり、そのためには開発プロセス、とくに作業工程であるとみなされてきた下流工程において、作業者の試行錯誤やそれを通じた創造的な問題解決が重要であること、それゆえ下流工程を外部調達の容易な代替可能な作業としてではなく、知識労働として捉える必要があることを指摘した。日本の受託開発を中心としたカスタムソフトウェア開発において、その下流工程を標準化した単純労働に置き換えてしまうこの分業構造は、技術者を代替可能な労働者として扱ってきたが、このことはソフトウェア開発の効率性を高めたものの、その一方で質の高い、革新的なソフトウェア開発につながる組織能力の蓄積と発揮を阻害することとなったと考えられるのである。

このような問題に対し、本研究では今日のソフトウェア産業がどのように位置づけられているのか、産業構造と歴史的経緯より分析を進めてきた。また、ソフトウェアの企業間の知識の取り扱いとして、部品の構成や相互関係のあり方を決めるアーキテクチャーから説明し、1970 年ごろから現代に至るまで、日本のソフトウェア開発で主流として機能してきた Waterfall Model が、時代の流れとしてどのように Agile に取って代わられつつあるのか、その開発プロセスの特徴と限界がいかなるところに見られるのかについて検討した。そのうえで日本のカスタムソフトウェア開発がなぜうまくいかなくなってきたのかについて、ソフトウェアの事例を分析にするとともに、ソフトウェア開発プロセスで見られた試行錯誤を繰り返す「創造的な問題解決過程」と、それに携わる技術者の「知識マネジメント」の視点から解明を試みてきた。

日本の受託ソフトウェア産業は、Waterfall Model に代表される順序立てた開発手法やソフトウェア・ファクトリーのような工場型の生産方式の導入により、無秩序であったソフトウェア開発を工程ごとに分離、管理することで、作業の定型化と開発プロセスの標準化を進めてきた。しかしながら、実際のソフトウェアの開発は分業が十分になされておらず、ゆるやかな関係を持っている。さらに技術の発展と顧客のニーズの変化といった環境変化により、その手法は適用することができなくなり、Agile のような顧客のビジネスや市場ニーズに合わせて臨機応変に仕様を変更して最速でビジネスを展開していくモデルに移行しつつある。

この上で、本研究では、ソフトウェア開発を製造業の製品設計や製品開発と関連付けて検討し、その開発プロセスにおいては作業者の保有する専門的な知識やノウハウ、創造的な問題解決といった知識労働としての側面が重要な役割を果たすことを明確にした。ここから明らかにされたことは、ソフトウェア開発の細かく分割される作業工程のうち、下流工程は、上流工程で決めた作業をこなすようなタスク処理型の労働あるいは、ライン生産方式に見られる工場における単純な製造作業に類する労働として捉えられるよりは、むしろ開発技術や企業の業務など幅広い知識と高い専門性が必要とされる労働としての性質を備えるということであった。

日本のソフトウェア開発では、上流工程や下流工程といった開発プロセスの単位で垂直的に企業間分業の境界を構築しており、分業の目的を労働力の確保やコスト削減のレベルに留めていることが多い。しかしながら、本研究の視点に立てば、より重要となるものは、こうした企業間でソフトウェア開発に関する知識の移転の重要性を踏まえ、その境界を設定するかという点であり、また、そういった境界を設定した後に、どのように知識の共有化を図っていくかという点である。

ソフトウェア開発は、現実世界の業務、習慣、文化、風習といった個々の企業、人間で異なるものを対象とするため、そこに現れてくる不確実で複雑な問題に対して、試行錯誤を繰り返していく知識労働としての側面を有する労働であり、そのような部分にソフトウェアを作成する醍醐味がある。そこには、製造業の製品設計や製品開発のように、度重なる変更と繰り返し性のある組織的な問題発見、問題解決活動が必要とされ、蓄積された知識や熟練といった良質な経験による類推に基づいて対応する能力と、新しく困難な状況でも創造的な問題解決ができる能力の双方が重要とされる。同時にこのような能力は、今までにない問題や新しい領域に取り組んでいくような革新的なソフトウェアの開発に繋がり、また、そうした機会を有する下流工程の情報が設計作業に生かされることで、ソフトウェア開発プロセスは価値創造のプロセスとなりうる。それゆえ、特定の作業工程や機能に対する狭い専門性の高い人材だけでは対応できず、チームメンバーや企業の境界を超えるような幅広い共通知識を備えた人材が必要となる。このような観点からも、代替可能な労働力ではなく創造的作業者としてのソフトウェア開発者が重要となるのである。

②　本研究の理論的貢献

本研究の理論的貢献について検討する。

第一に、ソフトウェア開発、特にその下流工程における開発作業が知識労働としての側面を備えていることを明らかにしたことである。本研究では、日本のカスタムソフトウェア開発について、その組織問題としてその分業構造について、とりわけ設計とプログラム作成を分離してしまうことの問題と、その知識労働という側面が見落とされていることに焦点をあててきた。このようなソフトウェア開発が標準化された単純労働ではない点につ

いては、Cusumano（1991, 2004）や妹尾（2001）、さらに Bean（2005）などにより指摘されてきたことである。本研究ではこうした主張に加え、今日の日本のソフトウェア開発が抱える問題をソフトウェア・ファクトリーなどの歴史を踏まえて明らかにするとともに、ソフトウェア開発業務の特性として現実世界の業務、習慣、文化、風習、嗜好といった複雑なものを作り込むという活動が存在することを指摘した。そのうえで、それゆえソフトウェア開発作業においては作業者の専門知識やノウハウ、経験、創造的な問題解決といった知識労働としての側面が重要な役割を果たすことになるのであり、質の高い、革新的なソフトウェア開発には、開発に従事するエンジニアの知的あるいは創造的な能力の活用を阻害しないような開発プロセスの編成を実現する必要があることを示した。

このような部分こそソフトウェアにとって本質的な価値につながる部分であり、わが国のような先進国の企業にとって競争優位となる部分である。こうした認識に立てば安易な国際分業はその競争優位の流出を招くものとして懸念されよう。

一方で、これまでの日本のソフトウェア開発に関する研究の多くは、最新の技術に関するものや、ソースコードの書き方といったプログラムを効率的に作成する方法やその自動化など、技術的な分野に特化しており、本研究のような視点をもった研究は非常に少ない。また、日本のソフトウェア開発の工程間分業が曖昧との指摘がいくつかの先行研究で見られるが、その内容を具体的に説明しようとした研究は少なく、その部分においても本研究の貢献の一つであると考えられる。

第二に、知識マネジメントの視点より、ソフトウェアの開発には開発担当者の問題解決能力と、隣接した工程に共通する幅広い知識が必要であることを示したことである。本研究では、事例を通して、工程間の分割により、情報連携や意思疎通、調整などが難しくなり、問題発見や解決といった試行錯誤の取り組みの障害となり、その結果革新的な問題解決に繋がるようなソフトウェア開発を困難にさせることを明らかにした。また、ソフトウェア開発が特定の工程や機能に対する狭い専門性だけでは、そのソフトウェア開発がうまくいかないことが明らかにされた。特にベテランとされるような熟練の技術者であっても、メンバー間での知識の共有ができなければ、作成されるソフトウェアの品質に影響を与えることが示された。ソフトウェアの開発は、機能や工程をモジュール化して行われるが、そういった一部分のみに対する専門性や、それを組み合わせていくだけの方法では、必ずしもうまくいくとは限らず、関連する部分を含めた幅広い共通の知識が必要となるという一定の解答を提示しえたと考えられる。

第三に、本研究ではこの上で、第一の貢献であるソフトウェア開発が知識労働としての側面を備えている点と、第二の貢献である知識マネジメントの視点との理論的結合を図ったことである。幾つかの研究において、ソフトウェア開発の困難性をあげ、その試行錯誤や問題解決活動の重要性が説かれてきた（藤本, 2001a, 2001b; 守島 2001a, 2001b, 2002; 妹尾, 2009 など)。しかし、ソフトウェア開発の知識労働としての側面を備えている点と、そ

ういった問題解決サイクルなどの知識マネジメントまでを関連付けた研究はなく、さらにそこに具体的な事例により、その関係性を明らかにしたところに、本研究の一つの貢献があると考えられる。

③　本研究の実務的貢献

次に、本研究の一連の結果から導かれる実務的貢献を述べる。

第一に、ソフトウェア開発業務の知的作業あるいは知識労働としての側面が、今日のソフトウェアの効果的な開発に必要とされていることを示したことである。ソフトウェア・ファクトリーに代表されるように、日本のソフトウェア開発は、製造業を手本とし、その製造工場化を進めようとしてきた。そこでは、上流工程と呼ばれる要件定義や設計工程が重視され、プログラムの作成といった下流工程はそれを元に作成するだけの単純な作業とみなされてしまう傾向にあり、アウトソーシングの対象となってきた。たしかにこの方法は、銀行の基幹システムなど、かつての安定した画一的なプロセスに基づいたソフトウェアには適していた。しかし、ビジネスが高度化し、顧客の求めるソフトウェアも多種多様になり、さらに複雑、かつ大規模になっていった。特に、ソフトウェアは IoT の推進やネットビジネスの登場により、一層迅速に対応することが求められ、従来のコスト削減だけではなく、企業のビジネスに貢献する付加価値までも求められるようになっている。

本研究では、このような状況において、上流工程と下流工程による垂直的な分業構造は、今までにない問題や新しい領域に取り組んでいくような革新的なソフトウェアの開発には適応できないことを主張し、ソフトウェアの開発が試行錯誤を繰り返しながら顧客にとってより付加価値のあるものを作り出す創作活動が必要となることを明らかにしてきた。このことは、先行研究の検討でも言及されているように、ソフトウェア開発業務が知識労働としての側面を備えており、今日のソフトウェアの効果的な開発に必要とされていることを改めて示すとともに、納期や費用という点でいかに効率的なものとして管理するかという点をもっぱら重視し、ソフトウェア開発を標準化した単純労働に置き換え、そこでの作業が軽視されがちな日本のソフトウェア産業に対して、今日におけるソフトウェア開発の付加価値の源泉がどこに存するのかという問いを改めて提起することになると考えられる。

第二に、本研究がソフトウェア開発の知識労働の問題を提示することで、同様の知識労働に関わる産業にも適用できる可能性を示したことである。本研究では、Agile などのイノベーションを目的としたソフトウェア開発の手法について述べてきた。これまでの日本のソフトウェア産業は、Waterfall Model に代表されるような工程間を分業する構造に合わせた開発手法を採用し、それゆえ変化の速い市場には対応できず、さらに革新的なイノベーションも期待することができなかった。一方で、Agile のようなイノベーションを生み出す取り組みは、顧客ニーズの変化に柔軟に、かつ迅速に対応し、今までにない問題や新しい領域に取り組んでいくようなソフトウェアを生み出しており、こうした革新的なソフトウ

ェアを作り出す開発プロセスについて述べてきた。こうした取組みはソフトウェア開発に限られたことではない。Rigby, Sutherland, and Takeuchi（2016）が、ソフトウェア開発で行われている Agile がラジオ番組の企画や、新しい機械の開発、戦闘機の生産、ワインの生産など幅広い産業で広く取り入れられていることを指摘しているように、多くの産業でも Agile 型の開発組織やその取り組みが行われており、このことは本研究が示した革新的なイノベーションを生み出す効果が期待できると考えられる。

(2) 残された研究課題

本研究では、ソフトウェアの分業構造を対象とし、今日の受託ソフトウェア開発にいかなる問題が存在しているのかを検討してきたが、本研究が取り上げたような分業構造とはまったく異なる開発アプローチをとる方法がある。特に、オープン・ソース・ソフトウェアによる無料で公開されたソフトウェアの開発や、サーバーなどの自前のインフラを必要とせずに必要な部分だけ利用が可能となるクラウド・コンピューティングを利用したソフトウェア開発に注目が集まっている。

また、本研究ではソフトウェア産業の開発プロセスの分業構造に焦点を当ててきたが、その分業構造を構成する要因の一つに日本のソフトウェア産業の多重下請け構造がある。この下請け構造は第 6 章でも説明したが、日本のソフトウェア産業の契約形態などとも強く結びついており、雇用体系やソフトウェアの価格など多くの要因と関わり合っている。

そのほか、ソフトウェア開発の自動化や AI[146]の導入も進んでいる。特に AI がより高度になり、本研究が述べてきたソフトウェア開発に関する問題発見や問題解決まで自律的に行えるようになれば、現在のソフトウェア開発の手法やその分業構造にも影響が出てくる可能性がある。

しかし、これらは本研究が扱うテーマを超えているとともに、現在も研究が進みつつある分野でもある。そこで、本研究の残された課題として、本研究とどのような関連性や違いがあるのかを述べておきたい。

① オープン・ソース・ソフトウェアによる不特定多数のネットワーク型開発

2000 年の IT バブルの崩壊以降、企業も IT への投資を厳しく見ており、大規模な開発を行うのではなく、要件を分解し、必要なものだけ開発を行うような小規模化してきたことは本研究でも度々述べてきた。開発規模が小さくなることにより、多重下請構造も見直されつつあり、元請けとしてのシステムインテグレーターではなく、中小ソフトウェア開発企業が直接、仕事を請け負うケースも珍しくなくなってきた（平井, 2012）。

[146] Artificial Intelligence。人工知能。

このようなコストに対する厳しい意識の中、2000 年頃から、Linux[147]などのオープンソース[148]といわれる無料公開されたソフトウェアに注目が集まっている（国領, 2004; 峰滝, 2004; 竹田, 2005; 谷花・野田, 2012, 2013; 野田・丹生・コークラン 2012, 2013; 野田・丹生, 2009, 2014; 神戸, 2014, 2015）。

このオープン・ソース・ソフトウェアでは、仕様書もないまま、インターネットを介して無数の開発者によって自由に開発が行われており、本研究で取り上げた Waterfall Model や Agile といった開発手法とも異なる。また、作成されたソフトウェアは、原則どのような企業、個人も無料で利用できるため、受託ソフトウェアやパッケージソフトウェアといった業態とも異なる。特徴的なことは、開発者が地理的に分散された環境の中で行われており、そのコミュニケーションはインターネットを利用し、メーリングリストなどを通じて行われることである。そして、そのメーリングリストのログを参照することで過去の開発プロセスを遡及的に体験できるため、メンバーが入れ替わっても、そのコミュニティの価値と文化を共有することができるのである（竹田, 2005）。このように、地理的に分散した環境の中で、ネットワークのみを通じたコミュニケーションによる開発が、企業が開発するソフトウェアに匹敵するような成功を収めていくことは注目に値する。

オープン・ソース・ソフトウェアの開発は、インターネットを介した協働作業となるため、互いの情報を形式知としてやり取りするために共通の知識の土台が必要となる。そのため、開発コミュニティに入る条件として、非常に高いレベルの技術力が要求され、その技術力を土台にコミュニティ内で知識の共有化が図られている（峰滝, 2004）。

本研究が対象としてきたような企業におけるソフトウェア開発では、その開発コストを考慮に入れたメンバー編成とスケジュールが組まれてきた。一方で、オープン・ソース・ソフトウェアの開発は、全体として過大な人数の開発者がそれぞれ同じ部分の開発を進め、そこからプロジェクトに適したソースコードが選択するという冗長性の高い開発を行っており、それが結果的に高品質のソフトウェア開発に繋がっているとされる（竹田, 2005）。

ただし、こうした開発方法は、必ずしも成功するとは限らない。このようなオープン・ソース・ソフトウェアのプロジェクトでは、参加者は個人の知的好奇心や、自分のためのソフトウェアを 1 人では作ることが困難なためみんなで作るといった理由でプロジェクトに参加しており、決して社会貢献のためにソフトウェアを開発しているわけではない。そのため、参加者にとってあまり魅力的ではないソフトウェア開発プロジェクトは、技術者が集まらなかったり、失敗してしまったりすることも多いとされる（竹田, 2005）。

このような特徴を持つため、オープン・ソース・ソフトウェアの方法を企業組織におけ

---

[147] 日本でも官公庁や自治体で、オープン・ソース・ソフトウェアの活用として、Linux の採用が進められている。

[148] オープンソースとは、公然と利用可能になっている技術であり、ライセンス料を取られずに無料で使用できることを指すことが多い。

るソフトウェア開発に導入することは難しく、導入したとしても技術者本人に相応のインセンティブがない限り成功は困難であると考えられる。一方で、オープン・ソース・ソフトウェアから得られる知見は、企業におけるソフトウェア開発にも適応できる部分もあると考えられる。特に、ソフトウェア開発の技術者は、他の技術者と情報交換を熱心に行うことが多い。例えば、ハッカソン[149]と呼ばれるその場でソフトウェア開発を実践するイベントのほか、海外では技術に関するカンファレンスなども多く行われており、残念なことに、日本からの参加者は少なく、日本の受託ソフトウェア開発に携わる技術者の多くは、そういったコミュニティに入り込めていないと考えられる。

このようなオープン・ソース・ソフトウェアに代表されるようなコミュニティによる協働開発や、そこでの知的創造活動がどのように行われているのか、その分析や比較を行い、取り込んでいく必要があろう。

② クラウド・コンピューティングの利用とビジネスモデルへの影響

本研究では、2006 年以降注目されつつあるクラウド・コンピューティングについてあまり取り上げていない。その理由として、クラウド・コンピューティングとその開発方法は、多くの企業が導入を試みると同時に、未だ試行錯誤している状況であり、日本のソフトウェア開発の現場に標準的に採用されるようになるにはしばらく時間がかかることが予想されるためである。しかしながら、このクラウド・コンピューティングの浸透は、本研究が対象としてきたカスタムソフトウェアの開発、つまりこれまでソフトウェアを開発して販売するビジネスモデルに大きな影響を及ぼす可能性がある。

クラウド・コンピューティングの特徴は、これまで自前で揃えていたソフトウェアのアプリケーションやそれを実行するためのサーバーなどの環境を、インターネットを利用して必要な時に必要な量だけ利用できる仮想環境を土台とした点である。身近で代表的なものは、Google が提供する Gmail などであろう。自分のパソコンやスマートフォンにメールソフトをインストールする必要はなく、逆にどの端末からもネットワークを通じて、どこでも自由にメールソフトを利用することができる。さらに、メールボックスの容量が足りなければ、必要に応じて注ぎ足すことも可能である。企業のビジネスでは、これが流通管理システムや、販売管理システム、Web 販売サイトなどになる。

---

[149] Hackathon。プログラマーやデザイナーなどによる数名で構成されるグループごとに、1〜2 日の短期間でアプリケーションを作ることを競う大会。グループは異業種間のメンバーも含めて構成されることもあり、密度の高いコラボレーションが求められる。また、通常のソフトウェア開発とは異なり、1〜2 日間と極端に開発期間が短いため、24 時間すべてを使う必要があり、優れたアイデアのもと、革新的な成果物を生み出す必要がある（西・西本, 2015）。

海外のハッカソンでは、参加者は、大会の賞金を狙ったり、大会での実績をもとに企業に自信の売り込みを図ったり、そこで開発したもので企業による買収を狙ったりする目的で参加することもある（山根, 2014）。

このクラウド・コンピューティングにより、個々の企業がそれぞれ情報システムを開発し、自社の資源として所有していくようなこれまでのビジネスモデルは大きく変えられようとしている。このため、運用コストはこれまでよりも遥かに安価となり、企業は情報システム資源を「所有」することから、ネットワークを通じたサービスとして「利用」する方向へと変わりつつある（杉山, 2011）。

企業固有のビジネスに向けたアプリケーションについては、これまで通り開発が必要であろうが、幾つかの機能はすでに用意されているものをカスタマイズすることで可能となる。クラウド・コンピューティングにより、企業は社外の資源を利用することで、社内の情報機器資源への投資を減少させることができるようになるのである（杉山, 2009）。

このように、クラウド・コンピューティングによってソフトウェアをサービスとして提供することで利益を得るビジネスモデルへと変わりつつあり、本研究が対象とするようなIT ベンダーにとっても、利益の源泉をサービスで稼げるよう、その分業構造も見直す契機となりうる。

しかし、現在稼働しているシステムがすぐにこのクラウド・コンピューティングに変わるかどうかは不明である。インターネットを利用した環境のため、ネットワークの障害に弱いことや、サーバーが外国に設置されることで個人情報や企業の機密などの取り扱いが法的に難しいなどの問題もある。このクラウド・コンピューティングがソフトウェア産業にどのような影響を与えていくのか、今後注意を払っていく必要があろう。

③ ソフトウェア産業に根付く下請け構造との関連

本研究は、ソフトウェアの開発プロセスに焦点をあててきた。本文中にも述べたが、ソフトウェア産業の下請け構造に本研究が主張するような、ソフトウェア開発を標準化した単純労働に置き換え、そこでの作業が軽視されがちな日本のソフトウェア産業における根本的な問題があり、そうした問題が開発プロセスやIT ベンダーの契約方法などに影響を与えている。

ソフトウェア産業の下請け構造は、大規模プロジェクトが活発であった頃は、5 層や 6 層といった深い多重下請け構造が存在し、労働集約的な産業構造を形成してきた。こうした下請け構造は、バッファー的に利用される存在とされることが多い（浅沼, 1997）。IT バブルの崩壊以降、IT 投資が減少し、大人数の技術者が必要となるような大型の開発プロジェクトも小規模化した。さらに、親会社等による優越的地位の濫用、偽装請負、中間搾取の問題など、社会問題化や法整備の点から多重の下請け構造はある程度解消してきているものの、こうした下請け構造は下流に位置するIT ベンダーやその技術者の受注価格やキャリアに大きな影響を与えている。

このような下請け構造も、本研究が主張するソフトウェア開発の下流工程の作業を代替可能な労働と位置付ける要因の一つであるが、さまざまな要因が複雑に絡んでおり、本研

究の範囲を超えている。本研究は、ソフトウェアの開発プロセスを安易に分割してしまうこと、とりわけ開発プロセスにおいて設計工程とプログラム作成工程を分離することで下流工程の知識労働としての側面が看過されていることに焦点をあてており、産業構造そのものの問題については説明に留めている。このソフトウェア産業の下請け構造やそれに伴う価格、雇用関係といった多様な問題については、別途研究していく必要があろう。

④　ソフトウェア開発の自動化と AI の導入

本研究では、ソフトウェア開発の知識労働に焦点を絞っており、なかでもその製造工場化の問題を検討した。第 2 章の本研究の先行研究の検討において少し触れたが、筆者が調べた限り、このようなソフトウェア開発の製造工場化、すなわちソフトウェア・ファクトリーに対する研究は、その注目が集まった 1990 年前後に集中しており、2000 年以降の論文は数本程度しか確認できていない。

一方、ソフトウェア・ファクトリーに代わり、ソフトウェアの「自動化」に関する研究は多くなってきており、工場的な生産としてではなく、自動化による技術者の開発における負荷の削減やサポート、生産性の向上の方面へシフトしていることが考えられる。実際に、設計書上の記述からソフトウェアのソースコードを自動的に生成しようとする研究も進んできている。

また、企業の Web サイトであれば、営業企画部や人材開発部、総務部といったソフトウェア技術者ではない人間が作成することができるようなツールやソフトウェアはすでに存在しており、簡単なものであれば、IT ベンダーや専門の技術者を介することなく開発する環境が整いつつある。

さらに、このような自動化の究極的な方法として、AI を導入したソフトウェア開発が考えられる。AI を中心とした機械学習や深層学習は、すでにソフトウェア開発の一部やその利用に導入されており、例えば、利用者からの質問に対し、AI を利用した回答を用意するなどの使い方がされている。AI がより高度になり、問題発見や問題解決まで自立的に行えるようになれば、現在のソフトウェア開発の体制にも影響が出てくる可能性がある。

ただし、現時点ではソフトウェア開発の自動化や AI の導入は限定的であり、完全に代替できるようなものではなく、また、ソフトウェアがユーザーの曖昧なものや創造的なものを実現しようとする限り、AI を導入がどれほど効果的なのかは不明である。

これら自動化や AI の導入は、未来のより良いソフトウェア開発の発展のために、今日も研究が進められている分野であり、今後も多くの研究が期待される分野である。

謝辞

本研究を進める過程では、多くの方にご指導、ご助言を頂きました。それなくしてはこの論文を完成させることはできなかったと思います。深く感謝いたします。

指導教授である立教大学大学院ビジネスデザイン研究科教授・研究科委員長の亀川雅人先生には、博士課程後期課程の合同ゼミでの終始暖かい奨励、ご指導を頂き感謝しております。特に、合同ゼミでの活発な議論や指摘が、本研究を進めるうえで貴重な礎となりました。

同じく指導教授である立教大学大学院ビジネスデザイン研究科教授の山中伸彦先生には、博士課程前期課程から長きにわたりご指導いただき、深く感謝の意を申し上げます。何かと迷いがちだった私に多くの助言や指摘を頂きました。先生のご指導なければ、私が論文の完成に辿り着くことはなかったと思います。

立教大学大学院ビジネスデザイン研究科教授の品川啓介先生には、毎月の指導の他、審査員として、イノベーションやテクノジー・マネジメントの視点より細かい指摘を頂き、大変感謝しております。

立教大学大学院ビジネスデザイン研究科教授の木村剛先生にも、審査員として実務の観点より多くの助言を頂き、感謝しております。

立教大学大学院経営学研究科教授の鈴木秀一先生にも、博士課程前期課程より引き続きご指導いただき、感謝申し上げます。

そのほか、江口圭一先生、結城義晴先生をはじめ、立教大学大学院ビジネスデザイン研究科の先生方には、多くのご支援を頂きました。ここに感謝の意を表します。

麗澤大学経済学部経営学科・同大学院経済研究科教授の倍和博先生、麗澤大学経済学部経営学科准教授の吉田健一郎先生には、学会等で大変お世話になりました。感謝申し上げます。

ビジネスデザイン研究科博士課程期課程の馬場正実氏、濱中友美氏には、前期課程時代より同期として大変お世話になりました。長く厳しい後期課程の研究の中、お二人がいたことは大変心強く、精神的にも支えられるとともに切磋琢磨したなかで多くの気づきを頂きました。深く感謝いたします。

ビジネスデザイン研究科博士課程後期課程の院生のみなさまにもお礼申し上げます。みなさまからは常に刺激的な議論を行い、さまざまな指摘を頂き、そういった批判一つ一つが本研究の貴重な土台となりました。

そのほか、多くの方に貴重な意見を頂くとともに、あたたかな励ましを頂きました。感謝申し上げます。

最後に、長年の研究活動を見守ってくれた両親、そして 101 歳を超えた現在も応援してくれている祖母に感謝の意を表します。

参考文献

[1] Abernathy, William J., （1978）, *The productivity dilemma: roadblock to innovation in the automobile industry*, Baltimore: Johns Hopkins University Press.

[2] Alvesson, M., （2004）, *Knowledge Work and Knowledge-Intensive Firms*, Oxford University Press.

[3] Alexander, Christopher., Ishikawa, Sara., Silverstein, Murray., Jacobson, Max., Fiksdahl-King, Ingrid., and Angel, Shlomo., （1983）, *A pattern language: towns, buildings, construction*, Oxford University Press.（平田翰那訳（1984)『パタン・ランゲージ―環境設計の手引』鹿島出版会.）

[4] Babbage, Charles.,（1832）, *On the Economy of Machinery and Manufactures*, Charles Knight, Pall Mall East.

[5] Baldwin, Carliss Y. and Clark, Kim B., （2000）, *DESGIN RULES Vol.1: The Power of Modularity*, Massachusetts Institute of Technology.（安藤晴彦訳（2004)『デザイン・ルール―モジュール化パワー』東洋経済新報社.）

[6] Barney, Jay B.,（1986）,"Strategic Factor Market: Expectation, Luck, and Business Strategy," *Management Science*, Vol.32, No.10, pp.1231-1241.

[7] Barney, Jay B., （2002）, *Gaining and sustaining competitive advantage* (*2nd ed)*, Upper Saddle River, N.J.: Prentice Hall.（岡田正大訳（2003)『企業戦略論：競争優位の構築と持続』ダイヤモンド社.）

[8] Bean, Michael., （2005）, "The Pitfalls of Outsourcing Programmers: Why Some Software Companies Confuse the Box with the Chocolates," *The Best Software Writing I: Selected and Introduced by Joel Spolsky*, Edited by Joel Spolsky, Apress LP., pp.9-15.（青木靖訳（2008)「プログラマのアウトソーシングの落とし穴ーなぜソフトウェア会社はチョコレートと箱を取り違えるのか」『BEST SOFTWARE WRITING』翔泳社.）

[9] Bell, Thomas T. and Thayer, T. A., （1976）, "SOFTWARE REQUIREMENTS: ARE THEY REALLY A PROBLEM," Proceeding ICSE '76 Proceedings of the 2nd international conference on Software engineering, pp.61-68.

[10] Boehm, Barry W., （1988）, "A Spiral Model of Software Development and Enhancement," *Computer*, Vol.21, No.5, IEEE, 1988.

[11] Brooks, Frederick Phillips., （1975, 1995）, *The mythical man-month: essays on software engineering*, Addison-Wesley.（滝沢徹・牧野祐子・富澤昇訳（2014)『人月の神話【新装版】』丸善出版.）

[12] Cappelli, Peter., （1999）, *The new deal at work: managing the market-driven workforce*, Harvard Business School Press.（若山由美（2001)『雇用の未来』日本経済新聞社.）

[13] Chandler, A.D. Jr., （1962）, *Strategy and Structure: Chapter in the History of the American*

Industrial Enterprise, The M.I.T. Press.（有賀裕子訳（2004）『組織は戦略に従う』ダイヤモンド社.）

[14] Clark, Kim B. and Fujimoto, Takahiro.,（1991）, *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry*, Harvard Business School Press.

[15] Coase, Ronald H., （1937）, "The Nature of the Firm", *Economica, N.S.*, Vol.4, No.16.

[16] Collis, David J. and Montgomery Cynthia A., （1998）, *COPORATE STRATEGY: A Resource-Based Approach*, The McGraw-Hill Companies, Inc.（根来龍之・蛭田啓・久保亮一訳（2004）『資源ベースの経営戦略論』東洋経済新聞社.）

[17] Coplien, James O. and Harrison, Neil B., （2004）, *Organizational patterns of agile software development*, Prentice Hall.（和智右桂訳（2013）『組織パターン: チームの成長によりアジャイルソフトウェア開発の変革を促す』翔泳社.）

[18] Cusumano, Michael A.,（1991）, *Japan's software factories: a challenge to U.S. management*, New York; Tokyo: Oxford University Press.（富沢宏之・藤井留美訳（1993）『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.）

[19] Cusumano, Michael A., （2004）, *The business of software: what every manager, programmer and entrepreneur must know to thrive and survive in good times and bad*, Simon and Schuster.（サイコムインターナショナル訳（2004）『ソフトウエア企業の競争戦略』ダイヤモンド社.）

[20] Cusumano, Michael A., （2010）, *Staying power: six enduring principles for managing strategy and innovation in an uncertain world*, Oxford: Oxford University Press.（鬼澤忍訳（2012）『君臨する企業の 6 つの法則 : 戦略のベストプラクティスを求めて』日本経済新聞出版社.）

[21] Davenport, Thomas H., （2005）, *Thinking for a living: how to get better performance and results from knowledge workers*, Harvard Business Review Press （藤堂圭太訳（2006）『ナレッジワーカー』ランダムハウス講談社.）

[22] DeMarco, Tom. and Lister, Timothy R., （1987）, *Peopleware: productive projects and teams*, New York: Dorset House.（日立ソフトウェアエンジニアリング生産性研究会訳（1989）『ピープルウエア － ヤル気を潰す管理体制はこう変えよう！』日経 BP 社.）

[23] DeMarco, Tom. and Lister, Timothy R., （1999）, *Peopleware: productive projects and teams（2nd Edition）*, Dorset House Publishing Company, Incorporated.（松原友夫・山浦恒央訳（2001)『ピープルウエア 第 2 版 － ヤル気こそプロジェクト成功の鍵』日経 BP 社.）

[24] DeMarco, Tom. and Lister, Timothy R., （2013）, *Peopleware: productive projects and teams（3rd Edition）*, Addison-Wesley Professional.（松原友夫・山浦恒央・長尾高弘訳（2013）『ピープルウエア 第 3 版』日経 BP 社.）

[25] Dreyfus, Hubert. and Dreyfus, Stuart E., （1986）, *Mind over machine: the power of human*

intuition and expertise in the era of the computer, Free Press.（椋田直子訳（1987）『純粋人工知能批判―コンピュータは思考を獲得できるか』アスキー.）
[26] Ferguson, Eugene S.,（1992）, *Engineering and the Mind's Eye*, The MIT Press.（藤原良樹・砂田久吉訳（1995, 2009）『技術屋の心眼』平凡社.）
[27] Gawer, Annabelle. and Cusumano, Michael A.,（2002）, *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*, Harvard Business School Press.（小林敏男監訳（2005）『プラットフォーム・リーダーシップ：イノベーションを導く新しい経営戦略』有斐閣.）
[28] Glass, Robert L.,（2006）, *Software creativity 2.0*, Developer.* Books.（高嶋優子・徳弘太郎・森田創訳（2009）『ソフトウエア・クリエイティビティ: ソフトウエア開発に創造性はなぜ必要か』日経 BP 社.）
[29] Hammer, Michael. and Champy, James.,（1993）, *Reengineering the Corporation: A Manifesto for business revolution*, Harper Business.（野中郁次郎訳（1993）『リエンジニアリング革命―企業を根本から変える業務革新』日本経済新聞社.）
[30] Langlois, Richard N. and Robertson, Paul L.,（1995）, *Firms, Markets and Economic Change: A Dynamic Theory of Business Institutions*, Routledge, London.（谷口和弘訳（2004）『企業制度の理論―ケイパビリティ・取引費用・組織境界―』NTT 出版.）
[31] Larman, Craig.,（2003）, *AGILE AND ITERATIVE DEVELOPMENT: A Manager's Guide*, Addison-Wesley.（児高慎治郎・松田直樹監訳, 越智典子訳（2004）『初めてのアジャイル開発: スクラム、XP、UP、Evo で学ぶ反復型開発の進め方』日経 BP 社.）
[32] Penrose, E. T.,（1959）, *The Theory of the Growth of the Firm, 3rd Edition*, 1995, Oxford University Press.（日高千景訳（2010）『企業成長の理論【第 3 版】』ダイヤモンド社.）
[33] Perrow, Charles.,（1967）, "A Framework for the Comparative Analysis of Organizations," *American Sociological Review*, Vol.32, No.2, pp.194-208, American Sociological Association.
[34] Porter, Michael E. and 竹内弘高（2000）『日本の競争戦略』ダイヤモンド社.
[35] Prahalad, C.K. and Hamel, G.,（1990）, "The Core Competence of the Corporation," *Harvard Business Review*.
[36] Rigby, Darrell K., Sutherland, Jeff., and Takeuchi, Hirotaka.,（2016）, "Embracing Agile," *Harvard Business Review*, MAY 2016 ISSUE, Harvard Business School Publishing.（倉田幸信訳（2016）「アジャイル開発を経営に活かす 6 つの原則 臨機応変のマネジメントで生産性を劇的に高める」『ハーバード・ビジネス・レビュー』2016 年 9 月号, ダイヤモンド社.）
[37] Royce, Winston W.,（1970）, "Managing the Development of Large Software Systems," Proceedings of IEEE WESCON.
[38] Rumelt, Richard P.,（1984）, "Towards a strategic theory of the firm," *Competitive Strategic*

Management, Edited by R.B. Lamb, Prentice Hall, pp.556-570.

[39] Sako, Mari., （1991）, “The Role of 'Trust' in Japanese Buyer Supplier Relationships,” *Ricerche Economiche*, Vol.45, No.2-3, pp.449-474.

[40] Sako, Mari., （1992）, *Prices, Quality and Trust: Inter-farm Relations in Britain & Japan*, Cambridge University Press.

[41] Simon, Herbert A., （1957）, *Administrative Behavior (4ed)*, 1997, New York: Free Press.（二村敏子・桑田耕太郎・高尾義明・西脇暢子・高柳美香訳（2009）『新版 経営行動―経営組織における意思決定過程の研究』ダイヤモンド社.）

[42] Smith, Adam., （1776）, *An inquiry into the nature and causes of the wealth of nations*, Oxford University Press, 1976.（山岡洋訳（2007）『国富論 国の豊かさの本質と原因についての研究（上下)』日本経済新聞社出版局.）

[43] Takeuchi, Hirotaka. and Nonaka, Ikujiro., （1986）, “The New New Product Development Game,” *Harvard Business Review*, Harvard Business Publishing.

[44] Taylor, Frederick Winslow., （1911）, *The principles of scientific management*, Cosimo Classics, 2006.（有賀裕子訳（2009）『新訳 科学的管理法：マネジメントの原点』ダイヤモンド社.）

[45] Teece, David J., (1986). “Transaction Cost Economics and the Multinational Enterprise,” *Journal of Economic Behavior and Organization*, Vol.7, No.1, pp.21-45.

[46] Tomayko, James E. and Hazzan, Orit., （2004）, *Human Aspects of Software Engineering*, CHARLES RIVER MEDIA.（富野壽・荒木貞雄訳（2007）『ソフトウェア工学の人間的側面』共立出版.）

[47] Trist, E.L. and Bamforth, K.W., （1951）, “Some Social and Psychological Consequences of the Longwall Method of Coal-Getting : An Examination of the Psychological Situation and Defences of a Work Group in Relation to the Social Structure and Technological Content of the Work System,” *Human Relations*, Vol.4, No.3, pp.3-38, SAGE Publications.

[48] Ulrich, Karl., （1995）, “The role of product architecture in the manufacturing firm,” *Research Policy*, Volume 24, Issue 3, pp.419-440.

[49] Wakefield, Edward Gibbon., （1833）, *England and America: a comparison of the social and political state of both nations,* R. Bentley.（中野正訳（2010）『イギリスとアメリカ: 資本主義と近世植民地』Vol.1・2・3, 日本評論社.）

[50] Wernerfelt, Birger., （1984）, “A Resource-Based View of the Firm,” *Strategic Management Journal*, Vol. 5, Issue 2, pp.171-180.

[51] West, Dave. and Grant, Tom., （2010）, “Agile Development: Mainstream Adoption Has Changed Agility For Application Development & Program Management Professional,” January 20, 2010, FORRESTER REPORT.

[52] Williamson, O. E., （1975）, *Markets and Hierarchies: Analysis and Anti-Trust Implications*, New York: Free Press.（浅沼萬里・岩崎晃訳（1980）『市場と企業組織』日本評論社.）
[53] Williamson, O. E., （1985）, *The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting*, New York: Free Press.
[54] Wiseman, Charles., （1988）, *STRATEGIC INFORMATION SYSTEMS*, Irwin.（土屋守章・辻新六訳（1989）『戦略的情報システム: 競争戦略の武器としての情報技術』ダイヤモンド社.）
[55] 相原基大（2000）「企業の境界デザイン」『經濟學研究』Vol.49, No.4, pp.62-78, 北海道大学.
[56] 青木利晴（2004）『効率化から価値創造へ―IT プロフェッショナルからの提言』NTT 出版.
[57] 青島矢一編（2008）『企業の錯誤／教育の迷走－人材育成の「失われた 10 年」』東信堂.
[58] 青島矢一（2009）「戦略転換の遅延―デジタルカメラ産業における「性能幻想」の役割―」『研究技術計画』vol.24, No.1, pp.16-34, 研究・イノベーション学会.
[59] 青島矢一・武石彰・Cusumano A. Michael（2010）『メイド・イン・ジャパンは終わるのか―「奇跡」と「終焉」の先にあるもの』東洋経済新報社.
[60] 秋池篤（2012）「A-U モデルの誕生と変遷―経営学輪講 Abernathy and Utterback (1978)」『赤門マネジメント・レビュー』Vol.11, No.10, pp.665-680, 特定非営利活動法人グローバルビジネスリサーチセンター.
[61] 浅沼万里（1997）『日本の企業組織 革新的適応のメカニズム―長期取引関係の構造と機能』菊谷達弥編, 東洋経済新報社.
[62] 阿草清滋（2009）「"正しい"ソフトウェアの開発を」『コンピュータソフトウェア』Vol.26, No.2, p.109, 日本ソフトウェア科学会.
[63] 飫富順久・廣松毅・小林稔（2009）『現代社会の情報・通信マネジメント』中央経済社.
[64] 阿萬裕久・野中誠・水野修（2011）「ソフトウェアメトリクスとデータ分析の基礎」『コンピュータソフトウェア』No.28, Vol.3, pp.12-28, 日本ソフトウェア科学会.
[65] 新井雄一朗（2016）「定性的なソフトウェアプロジェクトデータに基づくプロダクト品質予測に関する研究」『法政大学大学院理工学・工学研究科紀要』Vol.57, 法政大学大学院理工学・工学研究科.
[66] 有賀貞一（2008）「情報産業の直面する労働問題 ―多面的な産業構造の問題に根差す―」『法とコンピュータ』No.26, pp.53-63, 法とコンピュータ学会.
[67] 居駒幹夫（2011）「ソフトウェア開発組織における生産技術に関する研究」博士論文, 静岡大学創造科学技術大学院情報科学専攻.
[68] 伊丹敬之（1984）『新・経営戦略の論理－見えざる資産のダイナミズム』日本経済新聞社.

[69] 伊丹敬之・松島茂・橘川武朗編（1998）『産業集積の本質 ： 柔軟な分業・集積の条件』有斐閣.
[70] 一條和生（2003）「知的資産活用の経営--狭義のナレッジ・マネジメントから広義のナレッジ・マネジメントへ」『一橋ビジネスレビュー』Vol.51, No.3, pp.24-35, 東洋経済新聞社.
[71] 伊藤暁人（2009）「ソフトウェア開発における工学的技法導入に関する考察―静岡県下ＳW開発企業へのアンケート調査結果」『全国研究発表大会要旨集』2009f(0), pp.62-62, 経営情報学会.
[72] 稲村雄大・渋谷隆（2013）「ネットワーク外部性の間接効果と戦略の柔軟性： NEC のパソコン事業およびサーバ事業についての事例研究」『SHIBAURA MOT DISCUSSION PAPER』No.2013-03, 芝浦工業大学大学院工学マネジメント研究科.
[73] 今井賢一・安藤博・白井豊・辻淳二・久保宏志・玉置彰宏・浜田淳司（1989）『ソフトウェア進化論』NTT 出版.
[74] 内布光（2005）「ソフトウェア開発委託契約紛争事例の研究(1)」『現代法学』Vol.10, pp.157-186, 東京経済大学現代法学会.
[75] 梅澤隆（1996）「情報サービス産業の分業とソフトウェア技術者のキャリア・職業意識」『三田商学研究』Vol.39, No.1, pp.63-80, 慶應義塾大学.
[76] 梅澤隆（2000）『情報サービス産業の人的資源管理』ミネルヴァ書房.
[77] 宇山通（2013）「自動車企業におけるモジュール化の新展開: 新興国市場急拡大とパワートレーン多様化のインパクト」『九州産業大学経営学会経営学論集』Vol.24, No.2, pp.27-47, 九州産業大学経営学会.
[78] 浦塚剛志（2011）「ソフトウェア開発における実践的プロジェクト管理手法（＜特集＞成功するプロジェクトのための仕組みと組織活動)」『プロジェクトマネジメント学会誌』Vol.13, No.6, pp.9-12, プロジェクトマネジメント学会.
[79] 大鹿隆・藤本隆宏（2006）「製品アーキテクチャ論と国際貿易論の実証分析」『RIETI Discussion Paper Series』06-J-015, 独立行政法人経済産業研究所.
[80] 大西勝明（1998）『大競争下の情報産業―アメリカ主導の世界標準に対抗する日本企業の選択』中央経済社.
[81] 大場允晶（2011）「中国のソフトウェア企業のオフショア開発を進めていくうえでの現状と課題」『日本大学経済学部経済科学研究所紀要』No.41, pp.61-69, 日本大学経済学部経済科学研究所.
[82] 大和田尚孝（2009）『システム統合の「正攻法」： 世界最大のプロジェクト三菱東京 UFJ 銀行「Day2」に学ぶ』日経 BP 社.
[83] 小川秀人（2011）「企業におけるソフトウェア開発とソフトウェア工学」『コンピュータソフトウェア』Vol.28, No. 3, pp.2-11, 日本ソフトウェア科学会.

[84] 小椋俊秀（2013）「ウォーターフォールモデルの起源に関する考察－ウォーターフォールに関する誤解を解く」『商学討究』Vol.64, No.1, pp.105-135, 小樽商科大.
[85] 尾埣博之（1997）「情報サービス産業の現状と将来」『経営システム』Vol.7, No.3, pp.157-162, 日本経営工学会.
[86] 大日方真（1971）「プログラム開発の工程管理（I）」『情報処理』Vol.12, No.10, pp.627-636, 一般社団法人情報処理学会.
[87] 加護野忠男（1980）『経営組織の環境適応』白桃書房.
[88] 角埜恭央・椿広計・鶴保征城（2007）「日本のエンタプライズ系ソフトウェア産業の実態・課題に関する考察」『横幹連合コンファレンス予稿集』p.80, 経営情報学会.
[89] 神岡太郎・細谷竜一・張嵐（2006）「日本における情報システム開発スタイルと中国オフショアリング」『経営情報学会 全国研究発表大会要旨集』2006s(0), p.41, 経営情報学会.
[90] 亀川雅人・青淵正行編著（2009）『創造的破壊-企業価値の阻害要因』学文社.
[91] 川崎千晶（2014）「組織間信頼の形成プロセス: 縁故に基づく信頼の場合」『日本経営学会誌』No.33, pp.40-49, 日本経営学会.
[92] 神田良（1981）「社会・技術システム論に関する一考察」『一橋研究』Vol.6, No.1, pp.1-16, 一橋大学.
[93] 上林憲雄（2001）『異文化の情報技術システム―技術の組織的利用パターンに関する日英比較』千倉書房.
[94] 神原典広（2012）「IT プロジェクトを成功させる為のスコープマネジメント」『プロジェクトマネジメント学会誌』Vol.14, No.3, pp.20-24, プロジェクトマネジメント学会.
[95] 神戸和雄（2014）「オープンソースソフトウェアの利用と企業情報システム開発」『三田商学研究』Vol.56, No.6, pp.81-86, 慶応義塾大学出版会.
[96] 神戸和雄（2015）「オープンソースソフトウェアと企業の関わり」『三田商学研究』Vol.58, No.2, pp.121-128, 慶応義塾大学商学会.
[97] 菊野一雄（2010）「新しい労働の人間化(ネオ QWL)運動としての「ディーセントワークの理念」の歴史的位置と意味」『跡見学園女子大学マネジメント学部紀要』Vol.9, pp.25-34, 跡見学園女子大学.
[98] 北村弘二（2009）「プロジェクトにおいて上流工程の品質を高めるための諸施策と効果について」『プロジェクトマネジメント学会研究発表大会予稿集』2009（春季）, pp.320-323, プロジェクトマネジメント学会.
[99] 楠見孝（2014）「ホワイトカラーの熟達化を支える実践知の獲得」『組織科学』Vol.48, No.2, pp.6-15, 組織学会.
[100] 工藤秀憲（2009）『米国流システム構築が日本企業を救う！～工数精算方式がユーザー、ベンダー双方に利益をもたらす～』星雲社.

[101] 黒木英昭（2014）「内部化とアウトソーシングに関する意思決定ならびにマネジメントの研究」『横浜国際社会科学研究』Vol.19, No.1・2, pp.33-54, 横浜国際社会科学学会.

[102] 桑原希尽（2016）「ソフトウェア開発プロジェクトの工期と成功可否の関係の研究」『法政大学大学院理工学・工学研究科紀要』Vol.57,法政大学大学院理工学・工学研究科.

[103] 小池和男（1977）『職場の労働組合と参加 労使関係の日米比較』東洋経済新報社.

[104] 小池和男（2001）『もの造りの技能と競争力』『一橋ビジネスレビュー』Vol.49, No.1, pp.16-27, 東洋経済新聞社.

[105] 国領二郎（2004）『オープン・ソリューション社会の構想』日本経済新聞社.

[106] 古殿幸雄（2006）『入門ガイダンス 経営情報システム』中央経済社.

[107] 小林甲一（2008）「「労働の人間化」の展開と社会政策－労働をめぐるドイツ社会政策の構造転換」『名古屋学院大学論集. 社会科学篇』Vol.44, No.4, pp.1-32, 名古屋学院大学総合研究所.

[108] 権藤克彦・明石修・伊知地宏・岩崎英哉・河野健二・豊田正史・上田和紀（2009）「なぜソフトウェア論文を書くのは難しい（と感じる）のか」『コンピュータソフトウェア』Vol.26, No.4, pp.17-30, Japan Society for Software Science and Technology.

[109] 今野浩一郎・佐藤博樹（1987）「ソフトウエア産業における経営戦略と人材育成－人材育成体制とキャリア・パスの確立」『日本労働協会雑誌』Vol.29, No.7, pp.2-13, 日本労働協会.

[110] 今野浩一郎・佐藤博樹（1990）『ソフトウェア産業と経営―人材育成と開発戦略』東洋経済新報社.

[111] 斎藤昌義・後藤晃（2016）『システムインテグレーション再生の戦略 ～いま SIer は何を考え、どう行動すればいいのか 』技術評論社.

[112] 齋藤毅（2009）「生産システム論の再構成 : 労働研究と経営研究の統合に向けての試論」『評論・社会科学』Vol.88, pp.145-191, 同志社大学.

[113] 齊藤豊（2014）「多国籍企業における企業内国際労働力移動」『人間関係学研究 : 社会学社会心理学人間福祉学 : 大妻女子大学人間関係学部紀要』Vol.16, pp.1-16, 大妻女子大学.

[114] 佐野嘉秀（2001）「情報サービス業における外注化と社員の役割」佐藤博樹監修・電機連合総合研究センター編集『IT 時代の雇用システム』日本評論社.

[115] 設楽秀輔・中佐藤麻記子編（2012）『アジャイル概論』長瀬嘉秀監修, 東京電機大学出版局.

[116] 柴田友厚（2003）「製品システムとビジネス・システム－サブシステムへの分割という視点から」『研究技術計画』Vol.15, No.3/4, pp.227-240, 研究・イノベーション学会.

[117] 島田達巳・高原康彦（2007）『第三版 経営情報システム』日科技連.

[118] 白石弘幸（2010）「知識に関する組織能力と競争優位の研究」『経済学経営学系研究叢書』Vol.17, pp.1-302, 金沢大学人間社会研究域経済学経営学系.

[119] 須賀龍・牧野友祐・加藤和彦（2014）「相関ルールマイニングを用いたソフトウェア開発プロジェクトにおけるユーザ要求分析の提案」『プロジェクトマネジメント学会研究発表大会予稿集』2014（春季）, pp.350-353, プロジェクトマネジメント学会.

[120] 杉山克典（2007）「ビジネスコンピューティングの変移：汎用機世代の検証」『広島経済大学創立 40 周年記念論文集』pp.809-831, 広島経済大学.

[121] 杉山克典（2008）「日本のソフトウェア産業の現状分析」『広島経済大学経済学研究論集』第 31 巻, 第 3 号, pp.191-203, 広島経済大学経済学会.

[122] 杉山克典（2009）「企業情報システムにおけるクラウドコンピューティングの衝撃：クラウドコンピューティングへと向かう企業情報システムの歴史的検証」『広島経済大学経済学研究論集』Vol.32, No.2, pp.125-143, 広島経済大学経済学会.

[123] 杉山克典（2011）「プラットフォームとしてのクラウドコンピューティング」広島経済大学経済学研究論集』Vol.33, No.4, pp.55-63, 広島経済大学経済学会.

[124] 椙山泰生（2001）「グローバル化する製品開発の分析視角－知識の粘着性とその克服(特集 新製品開発研究の新潮流)」『組織科学』Vol.35, No.2, pp.81-94, 白桃書房.

[125] 鈴木潤（2009）「ソフトウェア・イノベーションの知識ベース」『RIETI Discussion Paper Series』09-J-019, 独立行政法人経済産業研究所.

[126] 妹尾大（2001）「ソフトウェア開発の新潮流－状況論的リーダーシップの胎動」『組織科学』Vol.35, pp.65-80, 組織学会.

[127] 妹尾大（2009）「知識創造企業に求められる技能熟達観」『経営システム』Vol.19, No.5, pp.192-196, 日本経営工学会.

[128] ソフトウェア産業研究会（2005）『ソフトウェアビジネスの競争力』中央経済社.

[129] 高木義和（2007）「日本と北米における情報サービス産業の構造比較」『新潟国際情報大学情報文化学部紀要』Vol.10, pp.119-130, 新潟国際情報大学.

[130] 高橋俊介（2012）『21 世紀のキャリア論』東洋経済新聞社.

[131] 高橋信弘（2010）「日本のソフトウェア産業の国際競争力に関する一考察:国際競争力欠如の諸要因とその因果関係」『經營研究』Vol.60, No.4, pp.151-167, 大阪市立大学.

[132] 高橋信弘（2013）「中国ソフトウェア企業の技術力向上とオフショア開発の変化」『經營研究』Vol.64, No.3, pp.1-23, 大阪市立大学.

[133] 武石彰（1999）「製品開発の戦略的アウトソーシング」『研究技術計画』Vol.14, No.2, pp.108-114, 研究・イノベーション学会.

[134] 武石彰（2003）『分業と競争―競争優位のアウトソーシング・マネジメント』有斐閣.

[135] 谷内篤博（2008）『日本的雇用システムの特質と変遷』泉文堂.

[136] 竹村正明（2001）「現代的な製品開発論の展開」『組織科学』Vol.35, No.2, pp.4-15, 組

織学会.
[137] 竹田昌弘（2005）「知識創造プロセスとしてのオープンソース・ソフトウェア開発」『立命館経営学』Vol.43, No.6, pp.19-33, 立命館大学.
[138] 立川丈夫（2003）『現代経営情報システム開発論』創成社.
[139] 立川丈夫（2005）『改訂版経営情報システム論－その歴史的展開と展望－』創成社.
[140] 伊達浩憲（2013）「大量生産システムの再検討 ―Ford の移動式組立ライン導入を中心に―」『社会科学研究年報』Vol.43, pp.1-19, 龍谷大学.
[141] 田中辰雄（2009）「カスタムソフト偏重は日本の問題点か?」『Economic review』Vol.13, No.1, pp.4-7, 富士通総研経済研究所.
[142] 田中辰雄（2010）「日本企業のソフトウエア選択と生産性 －カスタムソフトウエア対パッケージソフトウエア－」『RIETI Discussion Paper Series』10-J-027, 独立行政法人経済産業研究所.
[143] 田中美生（1988a）「情報サービス産業研究（I）－ソフトウェアの産業組織分析（上）－」『神戸学院経済学論集』Vol.19, No.4, pp.317-335, 神戸学院大学経済学会.
[144] 田中美生（1988b）「情報サービス産業研究（I）－ソフトウェアの産業組織分析（下）－」『神戸学院経済学論集』Vol.20, No.1, pp.75-92, 神戸学院大学経済学会.
[145] 谷花佳介・野田哲夫（2012）「オープンソース・ソフトウェアの市場価値と情報サービス産業の生産性に関する実証分析とその考察」『社会情報学会(SSI)学会大会研究発表論文集 2012』pp.203-208, 一般社団法人社会情報学会.
[146] 谷花佳介・野田哲夫（2013）「情報サービス産業における生産構造：OSS の市場価値と経済効果の観点から」『経済科学論集：島根大学法文学部紀要』Vol.39, pp.27-48, 島根大学法文学部.
[147] 土屋守章（1990）「情報技術の発達と組織の変化-問題提起-（情報技術と組織<特集>）」『組織科学』Vol.23, No.4, pp.2-6, 組織学会.
[148] 東京大学社会科学研究所（1989）『調査報告 第 22 集 情報サービス産業の経営と労働』.
[149] 床井直人・妹尾大（2010）「システム開発プロジェクトにおけるコンフリクトの早期顕在化に関する研究」『経営情報学会 全国研究発表大会要旨集』2010s(0), p.42, 経営情報学会.
[150] 戸塚秀夫・梅沢隆・中村圭介（1990）『日本のソフトウェア産業－経営と技術者』東京大学出版会.
[151] 豊田太郎（2012）「大量生産・大量消費の経済史 ―テイラー・システム,フォード・システム,大衆消費社会―」『札幌大学総合論叢』Vol.34, pp.73-85, 札幌大学.
[152] 中尾政之（2005）「創造設計と知識の活用（＜特集＞知識マネジメントの工学的アプローチ）」『品質』Vol.35, No.1, pp.11-17, 一般社団法人日本品質管理学会.

[153] 中尾政之（2009）「ソフトウェアの事故事例の分析」『横幹連合コンファレンス予稿集 』, 2009(0), p.117, 横断型基幹科学技術研究団体連合.

[154] 中川功一(2011)『技術革新のマネジメント－製品アーキテクチャによるアプローチ』有斐閣.

[155] 長田芙悠子 (2013)「ソフトウェア開発の失敗に関する会計処理案—ソフトウェアの仕損を会計ではどのように補足すればよいか—」『明治大学専門職大学院研究論集』Vol.5, pp.59-78, 明治大学専門職大学院.

[156] 中所武司（2014）『ソフトウェア工学第３版』朝倉書店.

[157] 西康太郎・西本一志（2015）「作業進捗状況と成果物イメージの共有によるグループハッカソンにおける協調活動支援」『情報処理学会研究報告』Vol.2015-HCI-162, No.17, 一般社団法人情報処理学会.

[158] 野田哲夫 (2006)「ソフトウェア生産のオープン化と地域の情報サービス産業 ： オープンソース・ソフトウェアによるソフトウェア生産のモジュール化と情報サービス産業の組織のモジュール化のマッチングの可能性」『経済科学論集』Vol.32, pp.77-118, 島根大学.

[159] 野田哲夫・丹生晃隆（2009）「オープンソース・ソフトウェアの開発モチベーションと労働時間に関する考察」『経済科学論集』Vol.35, pp.71-93, 島根大学.

[160] 野田哲夫・丹生晃隆・コークラン, シェーン（2012）「オープンソースライセンスによるビジネス戦略の展開」『経済科学論集：島根大学法文学部紀要』No.38, pp.1-34, 島根大学法文学部.

[161] 野田哲夫・丹生晃隆・コークラン, シェーン（2013）「日本の IT 企業におけるオープンソース・ソフトウェアの活用・開発貢献に関する研究」『経済科学論集：島根大学法文学部紀要』Vol.39, pp.59-72, 島根大学法文学部.

[162] 野田哲夫・丹生晃隆（2014）「オープンソース・ソフトウェアの活用と開発貢献における地域性の考察」『山陰研究』No.7, pp.35-51, 島根大学法文学部山陰研究センター.

[163] 延岡健太郎・藤本隆宏（2004）「製品開発の組織能力：日本自動車企業の国際競争力」『RIETI Discussion Paper Series』04-J-039, 独立行政法人経済産業研究所.

[164] 延岡健太郎（2006）「意味的価値の創造：コモディティ化を回避するものづくり」『国民経済雑誌』Vol.194, No.6, pp.1-14, 神戸大学経済経営学会.

[165] 萩森大介（2007）「ファンクションポイントを用いた要件定義変更管理の提案」『プロジェクトマネジメント学会研究発表大会予稿集』2007（秋季）, pp.132-134, プロジェクトマネジメント学会.

[166] 朴英元・藤本隆宏（2016）「製品アーキテクチャ視点からの日本企業の IT システムの進化プロセス：グローバル統合型ものづくり IT システムの提案」『第２０回進化経済学会東京大会２０１５報告論文』進化経済学会.

[167] 浜屋敏（2004）「組立業務の外部委託と製品・市場・企業特性－ビジネスモデルと製品アーキテクチャに関する実証研究」『Economic Review』Vol.8, No.2, pp.56-75, 富士通総研経済研究所.

[168] 林倬史（2008）「新製品開発プロセスにおける知識創造と異文化マネジメント－競争優位とプロジェクト・リーダー能力の視点から」『立教ビジネスレビュー１』pp.16-32, 立教大学.

[169] 平井直樹（2012）「中小ソフトウェア企業における下請け構造の実態と問題点」『経営会計研究』No.16, pp.41-54, 日本経営会計学会.

[170] 平鍋健児・野中郁次郎（2013）『アジャイル開発とスクラム』翔泳社.

[171] 福島利彦・山田茂（2006）「プロジェクトリスクを軽減するリスク・マネジメントと定量化分析」『プロジェクトマネジメント学会誌』Vol.8, No.4, pp.31-36, プロジェクトマネジメント学会.

[172] 藤田哲雄（2013）「わが国の電機産業の再生に向けて」『Japan Research Institute review』Vol.6, No.7, pp.57-81, 日本総合研究所.

[173] 藤本隆宏（2001a）『生産マネジメント入門 I 生産システム編』日本経済新聞社.

[174] 藤本隆宏（2001b）『生産マネジメント入門 II 生産資源・技術管理編』日本経済新聞社.

[175] 藤本隆宏・武石彰・青島矢一編（2001）『ビジネス・アーキテクチャ 製品・組織・プロセスの戦略的設計』有斐閣.

[176] 藤本隆宏（2002）「製品アーキテクチャの概念・測定・戦略に関するノート」『RIETI Discussion Paper Series』02-J-008, 独立行政法人経済産業研究所.

[177] 藤本隆宏（2003）『能力構築競争-日本の自動車産業はなぜ強いのか』中公新書.

[178] 藤本隆宏・延岡健太郎（2004）「日本の得意産業とは何か：アーキテクチャと組織能力の相性」『RIETI Discussion Paper Series』04-J-040, 独立行政法人経済産業研究所.

[179] 藤本隆宏（2004）『日本のもの造り哲学』日本経済新聞社.

[180] 藤本隆宏・東京大学 21 世紀 COE ものづくり経営研究センター（2007）『ものづくり経営学―製造業を超える生産思想』光文社.

[181] 藤本隆宏・桑島健一編（2009）『日本型プロセス産業：ものづくり経営学による競争力分析』有斐閣.

[182] 藤本隆弘・朴英元（2015）『ケースで解明　ＩＴを活かすものづくり』日本経済新聞出版社.

[183] 巫召鴻（2009）「システム開発の理論と実際」『研究報告ソフトウェア工学（SE）』Vol.2009, No.31(2009-SE-163), pp.241-248, 一般社団法人情報処理学会.

[184] 古田克利・藤本哲史・田中秀樹（2013）「ソフトウェア技術者の能力限界感の実態と要因に関する実証研究」『同志社政策科学研究』Vol.15, No.1, pp.29-43, 同志社大学.

[185] 保田勝通・大石晃裕・吉田直美・山田茂（2001）「グローバル時代のソフトウェア品質マネジメントシステムの提案と試行」『プロジェクトマネジメント学会誌』Vol.3, No.2, pp.21-26, プロジェクトマネジメント学会.

[186] 松田武彦（1990）「情報技術同化のための組織知能パラダイム（情報技術と組織<特集>）」『組織科学』Vol.23, No.4, pp.16-33, 組織学会.

[187] 松村知子・吉田誠・井手直子・森崎修司・戸田航史・松本健一（2011）「ソフトウェア開発の要件定義工程におけるユーザ・ベンダ間のコミュニケーション分析と活用方法」『プロジェクトマネジメント学会研究発表大会予稿集』2011（春季）, pp.427-432, プロジェクトマネジメント学会.

[188] 松本健一（1998）「ソフトウェアメトリクス」『日本ファジイ学会誌』No.10, Vol.5, pp.34-41, 日本知能情報ファジィ学会.

[189] 松本真佑・亀井靖高・門田暁人・松本健一（2010）「開発者メトリックスに基づくソフトウェア信頼性の分析」『電子情報通信学会論文誌』情報・システム J93-D(8), pp.1576-1589, 電子情報通信学会.

[190] 真鍋誠司（2002）「企業間信頼の構築：トヨタのケース」『Discussion Paper Series』J42, 神戸大学経済経営研究所.

[191] 真鍋誠司（2004）「企業間信頼の構築とサプライヤー・システム ： 日本自動車産業の分析」『横浜経営研究』Vol.25, No.2/3, pp.93-107, 横浜国立大学.

[192] 丸尾拓養（2008）「IT 産業の展開と従業者の法的地位～請負・派遣の区分をめぐって」『法とコンピュータ』No.26, pp.13-18, 法とコンピュータ学会.

[193] 水野幸男（1975）「情報処理産業におけるソフトウェアの重要性とその実際」『情報処理』Vol.16, No.1, p.1, 一般社団法人情報処理学会.

[194] 水野幸男（1982）「新しい工業技術製品―ソフトウェア製品の生産革新」『オペレーションズ・リサーチ ： 経営の科学』Vol.27, No.1, pp.2-3, 公益社団法人日本オペレーションズ・リサーチ学会.

[195] 港徹雄（2011）『日本のものづくり 競争力基盤の変遷』日本経済新聞出版社.

[196] 峰滝和典（2004）「情報技術革新と情報サービス産業」『Economic review』Vol.8, No.2, pp.24-55, 富士通総研経済研究所.

[197] 峰滝和典（2005）「日本の情報サービス産業の生産性分析：アウトソーシング・開発規模の経済性・参入退出構造・クラスターの視点」『SJC Discussion Paper』DP-2005-001-J, スタンフォード日本センター.

[198] 三輪卓己（2001）『ソフトウェア技術者のキャリア・ディベロップメント―成長プロセスの学習と行動』中央経済社.

[199] 三輪卓己（2009）「知識労働者のキャリア発達における多様性の分析－ソフトウェア技術者の組織内キャリアと組織を移るキャリア－」『日本労務学会誌』Vol.10, No.2,

pp.2-17, 日本労務学会.
[200] 三輪卓己（2010）「知識労働者のキャリア志向と組織間移動 : ソフトウェア技術者とコンサルタントの比較分析」『京都マネジメント・レビュー』Vol.17, pp.49-65, 京都産業大学.
[201] 三輪卓己（2012）「知識労働者の人的資源管理の論点と課題－先行研究と企業事例の検討から－」『京都マネジメント・レビュー』Vol.20, pp.73-91, 京都産業大学.
[202] 三輪卓己（2014）「IT 技術者の人的資源管理の事例分析 : 成果主義・市場志向の人的資源管理は一般的なのか」『京都産業大学論集. 社会科学系列』Vol.31, pp.29-56, 京都産業大学.
[203] 村田和博（2012）「古典派経済学における経営組織論の特質－分業と協働の観点から－」『経済学史学会第 76 回大会』経済学史学会.
[204] 目代武史（2012）「モジュール生産の工程アーキテクチャ分析」『赤門マネジメント・レビュー』Vol.11, No.10, 特定非営利活動法人グローバルビジネスリサーチセンター.
[205] 元橋一之（2005）『IT イノベーションの実証分析』東洋経済新報社.
[206] 守島基博（2001a）「総括 : 知的競争力としての人材」『一橋ビジネスレビュー』Vol.49, No.1, pp.98-103, 東洋経済新聞社.
[207] 守島基博（2001b）「内部労働市場に基づく 21 世紀型人材マネジメントモデルの概要」『組織科学』Vol.34, No.4, pp.39-52, 組織学会.
[208] 守島基博（2002）「知的創造と人材マネジメント」『組織科学』Vol.36, No.1, pp.41-50, 組織学会.
[209] 守島基博（2011）「知識創造を支える人材マネジメント」『一橋ビジネスレビュー』Vol.59, No.1, pp.24-38, 東洋経済新報社.
[210] 守島基博（2016）「求められる現場の組織開発 : 自律・分散・協働型組織を目指して」『ガバナンス』2016 年 6 月号, pp.23-25, ぎょうせい.
[211] 森田雅也（2010）「ソシオ・テクニカル・デザインの原則の展開と今日的意義」『関西大学社会学部紀要』Vol.38, No.2, pp.81-94, 関西大学.
[212] 八尋俊英・片山弘士（2008）「IT 産業界における労働問題と IT 人材育成施策」『法とコンピュータ』No.26, pp.19-24, 法とコンピュータ学会.
[213] 山沖義和（2011）「地域銀行によるシステム共同化の現状とその効果」『信州大学経済学論集』No.62, pp21-36, 信州大学経済学部.
[214] 山沖義和（2012）「地域銀行による共同システムの導入効果」『Staff Paper Series』11-02, 信州大学経済学部.
[215] 山沖義和（2014）「地域銀行によるシステム共同化のタイプ別経費削減効果等」『金融経済研究』No.36, pp.44-66, 日本金融学会.
[216] 山根淳平（2014）「ハッカソンを一過性のイベントで終わらせないために」『赤門マ

ネジメント・レビュー』Vol.13, No.102, pp.499-506, 特定非営利活動法人グローバルビジネスリサーチセンター.

[217] 湯野川恵美（2010）「IT 企業の持続的成長を支える企業戦略の P2M 的アプローチ」『国際プロジェクト・プログラムマネジメント学会誌』Vol.4, No.2, pp.17-27, 国際プロジェクト・プログラムマネジメント学会.

[218] 若田部昌澄（1991）「アダム・スミスの分業論－技術と内部組織－」『早稲田経済学研究』No.33, pp.97-109, 早稲田大学大学院経済学研究科経済学研究会.

[219] 若林直樹（2008）「専門能力を持つ人的資源の開発における企業間協力の考察－欧州製薬団体連合会の合同開発職研修の事例分析」『経済論叢』Vol.181, No.1, pp.104-122, 京都大学経済学会.

官公庁・社団法人データ・資料等

[1] 中小企業庁（2011）『中小企業白書 2011 年版』.

[2] 中小企業庁（2018）『2017 年版　中小企業白書』.

[3] 独立行政法人情報処理推進機構（2011）「グローバル化を支える IT 人材確保・育成施策に関する調査 調査報告書」平成 23 年 3 月.

[4] 独立行政法人情報処理推進機構 ソフトウェア・エンジニアリング・センター（2011）「非ウォーターフォール型開発 WG 活動報告書」平成 23 年 3 月 31 日.

[5] 独立行政法人情報処理推進機構 技術本部 ソフトウェア・エンジニアリング・センター（2012）「非ウォーターフォール型開発の普及要因と適用領域の拡大に関する調査～非ウォーターフォール型開発の普及要因の調査～　調査報告書」平成 24 年 6 月 11 日.

[6] 独立行政法人情報処理推進機構 技術本部 ソフトウェア・エンジニアリング・センター（2013）「2012 年度「ソフトウェア産業の実態把握に関する調査」 調査報告書」2013 年 4 月 26 日.

[7] 独立行政法人情報処理推進機構編（2012）『ソフトウェア開発データ白書 2012-2013』.

[8] 独立行政法人情報処理推進機構編（2014）『ソフトウェア開発データ白書 2014-2015』.

[9] 独立行政法人情報処理推進機構（2015）「IT 技術者の動向～ＩＴ人材白書から～」2015 年 3 月 25 日.

[10] 一般社団法人情報サービス産業協会（2008）『2008 年版基本統計調査報告書』.

[11] 一般社団法人情報サービス産業協会（2009）『2009 年版基本統計調査報告書』.

[12] 一般社団法人情報サービス産業協会（2010）『2010 年版基本統計調査報告書』.

[13] 一般社団法人情報サービス産業協会（2011）『2011 年版基本統計調査報告書』.

[14] 一般社団法人情報サービス産業協会（2012）『2012 年版基本統計調査報告書』.

[15] 一般社団法人情報サービス産業協会（2014）『2013 年版基本統計調査報告書』.

[16] 一般社団法人情報サービス産業協会（2015）『2014 年版基本統計調査報告書』.
[17] 一般社団法人情報サービス産業協会（2016）『2015 年版基本統計調査報告書』.
[18] 一般社団法人情報サービス産業協会編（2008）『情報サービス産業白書 2008』日経 BP 社.
[19] 一般社団法人情報サービス産業協会編（2010）『情報サービス産業白書 2010』日経 BP 社.
[20] 一般社団法人情報サービス産業協会編（2011）『情報サービス産業白書 2011-2012』日経 BP 社.
[21] 一般社団法人情報サービス産業協会編（2013）『我が国の情報サービス産業 2013』日経 BP 社.
[22] 一般社団法人情報サービス産業協会編（2014）『情報サービス産業白書 2014』日経 BP 社.
[23] 一般社団法人情報サービス産業協会編（2015）『情報サービス産業白書 2015』日経 BP 社.
[24] 一般社団法人 PMI 日本支部 アジャイルプロジェクトマネジメント研究会（2015）「アジャイル プロジェクト マネジメント 意識調査報告 2015」.
[25] 一般社団法人 PMI 日本支部 アジャイルプロジェクトマネジメント研究会（2016）「アジャイル プロジェクト マネジメント 意識調査報告 2016」.
[26] JISC 日本工業標準調査会「JISZ8301 規格票の様式及び作成方法」1951 年 10 月 31 日制定, 2011 年 1 月 20 日改定.
[27] 情報処理学会歴史特別委員会（2010）『日本のコンピュータ史』オーム社.
[28] 経済産業省「特定サービス産業動態統計調査」.
[29] 経済産業省経済産業政策局調査統計部サービス統計室「特定サービス産業実態調査（確報）」.
[30] 経済産業省（2006）「サービス産業の生産性向上に向けた検討の方向性について」.
[31] 経済産業省（2007）「サービス産業におけるイノベーションと生産性向上に向けて報告書」.
[32] 経済産業省商務情報政策局産業構造審議会情報経済分科会（2010）「情報経済革新戦略～情報通信コストの劇的低減を前提とした複合新産業の創出と社会システム構造の改革～」.
[33] 経済産業省（2017）「IT 関連産業の給与等に関する実態調査結果」.
[34] 日経パソコン編（2012）『日経パソコン デジタル・IT 用語事典』日経 BP 社.
[35] 日本経済団体連合会（2008）「サービス産業における中小企業の生産性向上の方向性」.
[36] 日本公認会計士協会（2011）「研究開発費及びソフトウェアの会計処理に関する実務指針」1999 年 3 月 31 日公表, 2011 年 3 月 29 日改定.

[37] 日本生産性本部（2010）「労働生産性の国際比較 2010 年版」.

[38] 日本生産性本部（2012）「労働生産性の国際比較 2011 年版」.

[39] 文部科学省（1991）「内閣告示第二号（1991 年 6 月 28 日）」.

Web サイト

[1] キリンビジネスシステム,
http://www.kirinbs.co.jp/ （2016 年 1 月 7 日閲覧）.

[2] ジェイアール東海情報システム,
http://www.jtis.co.jp/ （2016 年 1 月 7 日閲覧）.

[3] 鉄道情報システム株式会社「旅客販売総合システム「MARS（マルス）」」,
http://www.jrs.co.jp/article.php/business_mars （2016 年 1 月 7 日閲覧）.

[4] 首相官邸（2000）「ＩＴ基本戦略」，ＩＴ戦略会議・ＩＴ戦略本部合同会議（第６回），2000 年 11 月 27 日開催,
http://www.kantei.go.jp/jp/it/goudoukaigi/dai6/6siryou2.html （2016 年 3 月 21 日閲覧）.

[5] 首相官邸（2001）「ｅ－Ｊａｐａｎ戦略について」，高度情報通信ネットワーク社会推進戦略本部，ＩＴ戦略本部（第１回）, 2001 年 1 月 22 日開催,
http://www.kantei.go.jp/jp/singi/it2/dai1/1siryou05_2.html （2016 年 3 月 21 日閲覧）.

[6] 総務省,
http://www.soumu.go.jp/ （2015 年 12 月 7 日閲覧）.

[7] 総務省統計局（2013）「統計局ホームページ-日本標準産業分類」,
http://www.stat.go.jp/index/seido/sangyo/index.htm （2015 年 12 月 21 日閲覧）.

[8] 総務省 情報通信統計データベース「ICT の経済分析に関する調査」
（2016 年 8 月 21 日閲覧）.

[9] 日本経済新聞電子版「みずほ銀、システム統合再延期　動作テスト延長 運用 18 年以降」2016/11/12 2:00
http://www.nikkei.com/article/DGXLZO09463300R11C16A1EE8000/ （2017 年 2 月 11 日閲覧）.

[10] IBM「IBM Watson（ワトソン）」,
http://www.ibm.com/smarterplanet/jp/ja/ibmwatson/ （2016 年 3 月 13 日閲覧）.

[11] ITpro（2001）「「東京都庁商談の 750 円落札」にインテグレータ大手首脳が相次ぎ苦言」日経 BP 社,
http://itpro.nikkeibp.co.jp/free/NC/NEWS/20011007/3/ （2014 年 7 月 20 日閲覧）.

[12] 日経コンピュータ（2003）「プロジェクト成功率は26.7％ 2003年情報化実態調査」, 2003 年 11 月 17 日号（2003/11/13），日経 BP 社,
http://itpro.nikkeibp.co.jp/free/NC/TOKU1/20031111/1/ （2014 年 7 月 20 日閲覧）.

[13] 日経コンピュータ（2008）「成功率は 31.1％ 第 2 回 プロジェクト実態調査（対象 800 社）」2008 年 12 月 1 日号（2008/11/28），日経 BP 社,
http://itpro.nikkeibp.co.jp/article/NC/20081126/319990/ （2014 年 7 月 20 日閲覧）.

[14] IT 用語辞典 e-Words,
http://e-words.jp/ （2016 年 8 月 21 日, 2017 年 10 月 16 日閲覧）.

[15] ITpro Active（2016）「IT インフラ Summit 2016 レビュー プライベートクラウドのメリット最大化を目指す プライベートクラウド基盤編」日経 BP 社,
http://itpro.nikkeibp.co.jp/atclact/active/16/022300002/022300001/（2016 年 3 月 20 日閲覧）.

[16] ITpro（2016）「みずほ銀行のシステム統合、いつの間にか消えた“本当の”期限」2016/11/22, 日経 BP 社,
http://itpro.nikkeibp.co.jp/atcl/watcher/14/334361/111900724/ （2017 年 2 月 11 日閲覧）.

[17] James, Michael., （2010）, “Scrum Reference Card”,
http://scrumreferencecard.com/scrum-reference-card/ （2014 年 11 月 7 日閲覧）.

[18] Joel, Spolsky., （2001）, “In Defense of Not-Invented-Here Syndrome”
http://www.joelonsoftware.com/articles/fog0000000007.html （2016 年 3 月 20 日閲覧）.

[19] SD Times Software Development News, （2009）, “Standish Group Report: There’s Less Development Chaos Today”, March 25, 2009,
http://www.sdtimes.com/link/30247 （2012 年 12 月 15 日閲覧）.