経済のグローバル化とアジャイル開発の必然性

大　場　　充

（受付　2017 年 10 月 31 日）

1. 序：議論の主題

著者は，「経時的に変化する製品の品質」［大場充，経時的に変化する製品の品質，2017］において，製品の品質に関する利用者の要求や評価が，時間の経過とともに変化することは本質的な傾向であることを議論した。これは，経済のグローバル化の進展によって，その変化の速度が速まっているため，変化する利用者の製品の品質に対する要求をどのように把握し，そして認識された新しい要求に応えられるように製品の開発・生産・保守のプロセスをどう再構築すべきかの課題が，21世紀の企業組織に対して提起されていることを意味する。

20世紀の最後の10年間で，世界は急速に不安定化した。それは，政治的にも経済的にもである。このことは，我々が社会の将来の姿はどのようになるのかを予測することを困難にさせている。この予測不可能性は，20世紀後半までの国内市場を中心として発展した産業化社会においてはなかったことである。つまり，人類にとって初めての経験である。この未経験の状況において，企業がどのような対応をとるべきかを決定するとき，その意思決定には大きな誤りリスクが伴う。

企業の経済活動においては，この意思決定に伴うリスクに対応するリスク管理の方法が必要になる。そのリスク管理の基本的な方法として，人類がローマ時代から実践しているものに「分割統治」がある。その分割統治の考え方をソフトウェア開発に応用した方法として，1970年代の終わりに提案された「段階的開発」がある［大場充，ソフトウェアの開発技術，1988］。この段階的開発の理論を現実のプロジェクトに適用する新しいソフトウェア開発の実践論に「アジャイル開発」があり，最近，注目されている。

本小論においては，このアジャイル開発の基礎となっている段階的開発法が，要求の経時変化のリスクが想定される状況において，合理的な選択であることを論じる。すなわち，アジャイル開発は，経済が急速にグローバル化している社会において，合理性があることを示す。そのために，本小論の議論においては，プロジェクトマネジメントで広く利用されている PERT をモデルとして利用し，従来型の開発法であるウォーターフォール開発と新しい開発法であるアジャイル開発とを比較し，その利点・欠点を論じる。

2. 思想的背景：プラグマティズム思想と進化パラダイム

英国で産業革命が進行する中，ダーウィンが提唱した「進化論」は，それまでの「絶対的な神が計画したように生物は生まれ，滅んでゆく」とするキリスト教の聖書の教えを基本としたパラダイムを打ち砕いた。生物界における新種の誕生は，偶然に起こる遺伝子変化としての突然変異によって誕生し，適者生存の原理に基づく自然淘汰が，新種生物の増加，競合する在来生物種の絶滅や減少を引き起こすとした［ダーウィン，1990］。

このことは，全知全能の神が創造した宇宙の中での予定調和に基づいた生物の誕生と消滅の自然観を否定した。つまり，我々が生きている「この世界を創造したのは全知全能の神ではない」とする，ある意味で反キリスト教的な新しい思想を生み出した。多くのキリスト教徒は，この新しい理論の誕生に困惑した。2千年近くの期間にわたり，ヨーロッパの人々が絶対であると信じてきたキリスト教的世界観・自然観が否定された[1)]。

ダーウィンが「種の起源」を出版する以前から，全ての脊椎動物の起源は同じではないかとする仮説が議論されていた。ダーウィンの祖父もそのような仮説を信じていたとされている。そのような知的環境の中で，ダーウィンは，古典的な遺伝学[2)] によって蓄積されつつあった新しい知識を吟味し，突然変異と自然淘汰の2つの理論に基づいた新しい進化論を構築した。

ダーウィンの進化論が提唱されて以降，19世紀の末から20世紀の初頭にかけ，米国の哲学者たちは全知全能の神の存在を前提としない新しい哲学[3)] の構築に着手した。つまり，科学的

1） その進化論は，歴史的には古代ギリシャのエンペドクレスに代表される哲学者も議論したことのある考え方である。古代ローマ以降，進化論はイスラムの哲学者によって研究されていた。ヨーロッパの中世が終わり，近代が始まる頃，ドイツの哲学者ライプニッツらも進化論を研究していた。しかし，ダーウィンのように理論と現実世界との対応を議論することはなかった。
古代ギリシャの哲学者であったエンペドクレスは，物質世界の根源を火，空気，土，水の4つの元素からなるとする4元素論を説いた。さらに，これら4つの元素に運動をもたらす原理として，複数の元素を結合させる力として作用する愛と，逆に物質を複数の元素に分解し，無秩序を生み出す力として作用する憎しみがあるとする自然観を提唱した。バートランド・ラッセルは，彼が進化論を提唱したとしている。これは，過去には様々な形態の動物が地球上に存在したが，自然淘汰によって現在，我々が知る動物たちだけが生き残ったとする考え方である。また，古代ギリシャのアナクシマンドロスも，生命の起源は海にあり，生物は海から陸上に移り住むようになったとする説を唱えていた。
古代ローマ帝国の滅亡とともに，古代ギリシャで生まれた進化論的な思想はイスラムの哲学者や科学者に受け継がれた。9世紀にアル・ジャヒズは，生物の生存環境が生物の存続に与える影響を論じ，自然淘汰に似た考え方を述べた。さらにイブン・ミスカワイは，無機物から植物，そして動物，さらに類人猿から人間への生物の発展の歴史について書き，イブン・アル・ハイサムは生物の進化を述べた書を書き残した。これらの人々の思想は，ルネッサンス期にラテン語に翻訳され，中世ヨーロッパに伝わったとされている。

2） メンデルらによって提唱された遺伝学を指す。

3） 米国におけるプラグマティズム思想の進展を言う。

な真理には，人間にはそれが普遍的な真理であるかどうかを判断できないものがあるとした。従って，人間は自分たちが生きている時間の範囲の中で，最も合理的な理論を見出し，それを当面の真理として受け入れざるをえないとした。このことは，新しい事実が見出されたときは，それまでの理論を変更して，より良い理論を作り出さざるをえないことを意味している。

このような哲学をプラグマティズム（pragmatism）と呼ぶ。プラグマティズム哲学は，人間の真理を知り得る完全な能力を否定する。そのため，現実をよく説明する理論を当面の真理として受け入れ，この当面の真理としている理論に適合しない現実の現象が発見された場合には，その新しい現実をも包み込んで説明できる新しい理論を構築することを主張する［デューイ，1968］[4)]。この手続きを繰り返すことで，人間の知識は少しずつ，普遍的な真理に接近できるとしている。

これは，経済理論で言えば，従来は技術革新によって，新しい技術を開発したり，新しい生産プロセスを開発したりして，経済を進展させるとするシュンペーターの「技術革新」理論［伊東光晴，根井雅弘，1993］が受け入れられていた。これに対して，日本的品質管理理論は「継続的改善」による生産性や品質の向上を重視する考えを提唱した［石川馨，1981］[5)]。

このことは，革新的（revolutionary）な変化ではなく，進化的（evolutional）な変化の積み重ねによって，結果的に大きな変化が起きるとする考え方である。革新的な変化の場合には，一般的にその結果の社会への影響も不連続的・離散的になる。これに対して，進化的な変化の連続であれば，その変化による社会への影響は小さく，予測可能で連続的な変化となる。

地球上の生物が，天体としての地球の変化，すなわち寒冷化，火山の噴火，小さな惑星等の天体の衝突による生存環境の変化など，に適応しながら長期間にわたり進化を継続し，現在に至るまで存続できていることは，進化が生物全体にとって，結果的に最適な生存戦略であったことを実証していると考える。このプラグマティズムの思想に基づく，新しいソフトウェア開発パラダイムとして，「アジャイル開発」が注目されている［マーチン，2008］。

3. 社会的背景：経済のグローバル化と時間

世界経済は，急速にグローバル化している[6)]。2016年の 6 月に英国において実施された国

4） デューイによる「哲学の改造」（清水幾多郎訳，岩波文庫1968年）を参照されたい。
5） 日本的品質管理では，製品，製品生産の技術，製品生産のプロセスなどを変化させ，従来の製品よりも良い製品，従来よりも安いコストでの生産をもたらす継続的改善に基づく企業経営を提唱している。
6） 経済のグローバル化については，拙著，「経時的に変化する製品の品質」において論じている［大場充，経時的に変化する製品の品質，2017］。

民投票では，このグローバル化によって発生している様々な問題，移民の流入による雇用の不安定化や，高収入を得ている人々と低収入に悩んでいる人々との格差の拡大［ファーロング，2009］など，に対する国民の問題意識から，反グローバル化を主張する EU 離脱派が勝利した[7)]。

このような一時的な揺り戻しはあっても，世界経済の発展を考えると，経済のグローバル化は止めることのできない潮流である［バーンスタイン，2006］。世界経済がこれからも持続的に発展してゆくためには，先進諸国の国民だけが有利になるような閉鎖的な国家経済システムの並立と，それらを統制するシステム間競争原理だけでは，非効率で不利だからである[8)]。経済のグローバル化は，これからも進展する。そして，その結果として企業間の競争は，国内市場に限定された競争から，地球規模の競争へと拡大する。

この経済のグローバル化において，我々の経済活動には大きな変化が生じている。それは，「時間」の要素が，従来の競争に比べて重要性を増していることである[9)]。このことは，経済のグローバル化が，先進諸国間のサービス経済競争を巻き起こしていることと関係している。サービス経済の特徴は，供給と消費が同時に起こることである。つまり，需要が発生しているときに，供給ができなければ，事業として成立しない。これは，従来からの「財」を主体とした市場での取引と，大きく異なる。

人間社会では，数千年の間，市場で財と財，財と貨幣を交換する経済を運営してきた。その財は，一定期間，財の価値を維持できる[10)] ことが原則である。自動車は，通常，数十年にわたり利用できる。食料品であっても，短い場合でも数日間にわたり美味しく食べることができる。すぐに食べられなくなる食料品は，たとえ収穫時の味が良くても，急速な品質劣化が原因で市場での売買は困難なため，経済財としては流通しなかった[11)]。

人間の行為を価値の源泉とするサービスは，そのサービスを提供する提供者が存在しなければ，供給が不可能である。そのため，従来は他国の市場におけるサービスの売買は成立しなかった[12)]。しかし，経済の発展とともに，一定範囲の限界の中で，外国市場での取引が可

7） 英国の EU 離脱を決定した国民投票と，EU 離脱の日系企業への影響に関する解説については，Jetro（日本貿易振興機構）において説明資料がまとめられている［坂口利彦，2016］。

8） 発展途上国も含めた開かれた市場における各国の経済システム間の自由な競争によって，世界規模での効率の良い経済運営が可能になる。これによって発展途上国の GDP のみならず，先進国を含めた世界の GDP も成長を維持できている。

9） 時間の問題に関する議論は，前掲の拙著，「経時的に変化する製品の品質」の 3 章の議論を参照されたい［大場充，経時的に変化する製品の品質，2017］。

10） これは，物理的なモノとして市場に提供される財の場合，その基本が一定期間，出荷時点での財の品質を維持できることを言う。生鮮食品でも，生産から物流経路を経て，店頭に陳列され，消費者の購買と消費まで，その品質はほぼ維持できる。

11） 江戸時代にマグロのトロが食されず，捨てられていたのは，保存ができなかったことが原因で，商品として流通しなかったためである。

12） ハンバーガーチェーンのマクドナルドは，米国で始まった事業であった。この場合，売買される商 ↗

能なものが生まれた[13]。典型的なサービスが，金融サービスである。医療サービスや教育サービスなどの公的色彩の強いサービスも，インターネット技術を活用し，海外市場での提供が可能になりつつある。これらは，従来からサービスが提供可能であった市場の存在した先進国に集中していた。このため，これらのサービスは，従来から先進諸国において高度に発達していた。

これに対して，開発途上国の企業は，財を生産し，貿易によってそれを他国の市場へ輸出して供給し，その収入を得る。これにより開発途上国では，先進諸国よりも安いコストで生産し，先進諸国で生産される，似た財よりも少し安い価格で類似製品を販売することにより，結果として開発途上国へ大量の外貨流入をもたらした。この構図は，現在でも有効に機能している。2015年頃までの中国経済が潤ったのも，安い人民元で支払われる人件費を武器に，低価格で財を生産し，米国や日本，ヨーロッパ諸国の市場へ輸出することで，外貨を稼ぐことができたからであった。

このような背景から，経済がグローバル化する過程で，先進諸国は，自分たちの強みであるサービスを高い価格で輸出し，財を安い価格で輸入すると言う経済運営に転換した。このとき，新しい問題が発生する。サービスは，それを提供できる人材の育成が重要で，人材が確保できなければ競争に勝てない。このことは，開発途上国でも先進諸国と同様に，一定の割合で人材を輩出することが可能なので，先行企業が同じ内容のサービスを長期間にわたり提供し続けていると，結局は，「サービス分野においても先進諸国は開発途上国とのコスト競争に負ける」という問題を発生させる[14]。

↘ 品はハンバーガーであるものの，最も重要な要素は原料とレシピである。マクドナルドは，米国市場における全国展開のために，レシピを詳細に規定し，マニュアル化した。また，原料の買い付けを一元化した。このことが，サービスとしてのハンバーガーチェーンのブランドと経営システムを輸出することを成功させた。

13） これを可能にするため，米国政府は1980年代初頭から，GATT ウルグアイラウンドの締結に努力し，相互認証システムの導入にこぎつけた。これによって，相互認証に関する協定が成立している 2 国間においては，一方の国で取得した営業免許は，他方の国においても有効となる。例えば，銀行業務の営業許可については，米国で営業許可を得た金融機関は，基本的に日本市場でも営業を許される。この相互認証システムの導入により，サービスの国際取引に対する門戸は大きく開かれたと言える。

14） このため，先進諸国でサービス提供に従事している専門家は，提供するサービスの内容を時間の経過とともに変化させ，競争相手が追いつくことを許さないようにしなければならない。従って，このサービス内容を変化させるまでの時間をいかに短くし，新しいサービスの提供を待つ顧客たちの要望に適時的に応えてゆく能力を維持することが，勝ち残りのための条件になる［フロリダ，2008］。従来型の財の生産では，新しい財の開発のために，調査研究を行い，試作を繰り返し，最終的に量産する財の設計を決めて，生産ラインを整備し，生産ラインで働く作業者を教育してから，財の大量生産を開始する。しかし，無形なサービスの場合，そのような準備はほとんど必要ない。従って，変化の準備に必要な時間の長さは極度に短くなる。フロリダは，米国社会において知的業務の遂行を主たる職業とする知的階層に属する人々が，2000年の時点で全労働者の30パーセントを超えていることを指摘し，その新しい社会階層を「クリエイティブ階層」と呼び，そのような人々の思想や，価値観，生き方の特徴について調査結果をまとめた。

このサービスの特徴が，市場におけるサービス開発競争における時間の重要性を際立たせる。供給者側には，可能な限り早く，新サービスを市場へ投入することが，開発投資資金の回収を早め，さらに高い収益を得るために重要になる。また，そのようにして市場に投入した新サービスについても，それをいつまで継続し，つぎにどのような新サービスを開発するのかを計画し，その開発を管理することが成功継続の鍵となる。新サービスも，一定の時間が経過すれば，類似サービスを競合他社が供給できるようになるからである。

もう一つ，時間に関して重要なことは，市場の時間的変化も無視できなくなっている。財の市場への投入では，前述したようにいくつかの段階を経なければならない。研究開発にも生産設備の準備にも多大な資金の投入が必要になる特徴がある。このことは，同一市場への他企業の新規参入の障壁が高いことを意味する。新規参入しても，市場で勝てると言う保証はないので，リスクが潜在する。従って，財の経済において類似製品を生産し，市場へ投入することには，労働コストに大きな差がある場合を除き，大きなリスクを伴う。

一般の財のような高い参入障壁のないサービスにおいては，サービス提供ができる人材を集めることができれば，類似サービスの提供準備に必要な時間は，財に比較して著しく短縮できる。このことは，市場にも影響し，市場におけるニーズの変化も財の場合と違って，時間とともに急速に変化する傾向を生み出す。つまり，少し前まで市場で高い評価を受けていたサービスが，別のサービスの出現や，社会情勢の変化によって，全く市場に受け入れられなくなるような状況も発生しうる。ポピュラー音楽の分野で，ヒット曲が目まぐるしく変わるのもこれが原因と言える。サービスに対する需要は，財に対する需要よりも簡単に変化する傾向がある。

経済のグローバル化と，先進諸国における経済のサービス化により，特に先進諸国の市場において，時間の重要性は従来の財を中心とした市場に比較して，著しく高まっている。このことは，サービス取引における需要と供給の時間的一致の原則によって，それを供給する企業の時間に対する対応を厳しくする。このことが，ソフトウェア開発における最近のアジャイル開発方式の要請の根底にある。しかし，このことは本質的には，ソフトウェア開発に限定されるものではなく，財の開発・生産を含めた，幅広い分野にも適用される普遍的な性質である。

4. 組織的解決策：変化への対応

人類は，古代ローマ帝国の時代に大規模な組織を構成し，その大規模な組織を利用して広大な地域を支配し，そこから得られる収穫物を集め，主都ローマに運び，ローマの市民やそこで働く奴隷たちの生活を維持するための物資を効率的に分配する方法を生み出した。世界

4大文明が発祥して，わずか3,000年後のことである。古代ローマ人が考案した基本的な方法は，今日，「分割統治」（divide and conquer）と呼ばれている方法である[15)]。

分割統治とは，巨大な組織を構成するために，十人から数十人で構成される最小単位の組織をつくる。この最小単位の組織を統率するために，その最小単位の組織を複数束ねることで，最小単位組織の数倍から10倍程度の規模の小組織を構成する。さらに，この小組織をいくつか束ねて中組織を構成し，その中組織をいくつか束ねて大組織を構成する。大組織になると小さなものでも数千人規模の組織が作られ，大きなものになると数万人規模の組織になる[16)]。

この分割統治の手法は，5世紀に古代ローマ帝国（西ローマ帝国）が滅亡した後，ローマに設立されたカトリック教会（ローマ教会）によって踏襲された。5世紀から15世紀までの約1,000年間，ヨーロッパ世界を実質的に支配したローマ教会は，地域内のすべての村に建設した村の教会，それをまとめた教区（地域）の教会，地方の教会，国の教会，それを取りまとめる宗派の教会，そしてそれらを支配するローマ教会と言うピラミッド構造の組織を作り上げた。

このシステムでは，各地域を支配する貴族や，それを取りまとめている国家を支配する国王も，ローマ教会からの承認がなければ貴族や国王の地位に就くことはできなかった。国王を公式に決定する戴冠式では，教会を代表する司教やローマ教皇が国王に冠を授けた。このことは，世俗的な政治組織である国の上に，宗教上の組織であるローマ教会が存在することを示していた[17)]。

16世紀頃に北ヨーロッパの国々で始まった宗教改革によって，北ヨーロッパにおけるローマ教会の権威は徐々に低下し始め，17世紀になるとローマ教会はこの地域における実質的な権威を失った。このことは，各国の人民を支配するのは，ローマ教会ではなく，その国の国家権力を代表する国王や共和国の執政官になった。国家は，国民に税金を課し，国民を統治

15） 今日，「分割統治」の方法は，数学的な問題解決の方法論としてよく知られている。これは，複雑なシステムの設計や大規模なプロジェクトの計画策定などにも応用されている。基本的には，複雑な対象を，より複雑性の度合いの低い部分問題に分解する。この手順を何回も繰り返すことで，最終的には数多くの単純な問題にまで分解し，その単純な問題を一つ一つ解決することで，対象となっている問題を解決する方法である。この方法は，分解の段数によって問題解決全体の手間が決定するので，人間が考え出した方法の中で，最も効率の良い問題解決の方法である。

16） そのような大組織をさらにいくつか束ねて，数十万人単位の組織を作ることも可能である。これを繰り返すことで，地中海世界全体をまとめ，全体で数千万人の人々が暮らす地域（ローマ帝国）を作ることに成功した。古代ローマの軍隊は，この地域を支配するための軍事的な組織であると同時に，行政のための組織でもあった。古代ローマ軍は，戦うためだけの組織ではなく，戦い，人々を支配し，地域の都市基盤を構築するための組織でもあった。

17） 中世ヨーロッパで広く信じられていた王権神授説は，国家の主権者としての王は，神から王権を授かり，その国を支配しているとされていた。その神の代理として国王を認める機関としての役割を担っているのが，ローマ教会であり，それを代表するのが教皇であるとされていた。

するための政治コストを賄うようになった。この新しい統治組織である国家も，ローマ教会の組織を真似た行政組織構造をもつようになった。

北ヨーロッパ諸国では，行政のための組織と，対外的な戦争を実施するための国民軍を整備した。それらもローマ教会が古代ローマ帝国から継承した巨大組織の構成法を軍事・行政組織の構成法として応用したものであり，巨大な組織を作るようになった。さらに，産業革命の時代が終わって，産業化社会（Industrial Society）時代に入り，企業の資本集約が進むと，企業組織の構成に，古代ローマ帝国で開発され，中世のローマ教会によって発展し，近代国家や軍隊の組織構成と運用の方法として確立した分割統治が応用されるようになった。

1980年代末に，米国において産業化社会の時代が終焉し，巨大企業の時代が終わりを迎えた。それに代わって，徐々にではあるが，小規模なベンチャー企業によるインターネットを活用した事業における技術革新の事例が増え，巨大組織の時代も終焉した。例えば，インターネット書店のアマゾンが，従来の書店とは全く異なるやり方で事業を展開し，大成功をおさめ，従来型の大規模書店を経営危機に陥れた［米国商務省，1999］[18)]。

知的な労働を基本とするソフトウェア開発においては，個々の作業者は専門的な仕事を担当するので，肉体労働を基本とする建設現場や製造現場の作業とは本質的に違う。より大規模な成果を得るために，単純に作業者の数を増やし，それを大規模な組織で管理することで，目標を達成しようとする方法が，成功を約束するわけではない。ソフトウェアは，大規模になればなるほど複雑になり，作業者の数を増やすだけでは成功できない［ブルックス，1982］。

一般的には，ソフトウェア開発の場合，優秀な専門家を集めて，少人数で十分な時間をかけて作業する方が成功の確率を高める[19)]。開発期間が短く，その制約が厳しい場合，納期を守るために作業者の数を増やし，分業の度合いを高める方法を採用することがある。この方法は，ソフトウェア開発のような高度に知的な労働を必要とする分野では，組織管理の問題を複雑化させる結果となり，失敗のリスクを増大させる。

ソフトウェア開発における失敗は，既に完了したとされる工程での作業のし直しを原因とする例が多い。その結果として多大な追加作業を発生させることがある。そのような追加作

18） この小規模なインターネット書店が，大規模な従来型の書店との競争に勝ち，従来型の書店の経営を危機に陥れたことによって，古代ローマ帝国で開発された分割統治の原理を利用した大規模な組織が，少人数で構成された小規模組織に負けたことになった。書店のような従来型の事業分野においても，規模が大きな組織ほど有利であるとする従来型の規模の経済の考え方は否定されたのである。

19） ソフトウェア工学の黎明期に活躍したブルックスは，その著書「ソフトウェア開発の神話」で，開発組織を大規模にすることが，大規模で複雑なソフトウェアの開発に成功することを保証しないことを自らの経験に基づいて実証した。さらに，1987年に発表した論文「銀の弾丸はない（no silver bullet）」［Brooks, 1987］において，ソフトウェア開発の難しさの原因に，①規模による複雑性，②実体のないソフトウェアの不可視性，③論理構造の変化のし易さ，変え易さ，そして④人為的な構築物であるソフトウェアが人間の自由な決まり事にしか縛られない 4 つの性質があることを説明した。

業が増えてくると，結果としてソフトウェア開発は失敗する。約束した納期までに開発を終わらせることができないからである。つまり，大規模な組織によるソフトウェア開発には，欠点が多いにもかかわらず，利点が少ない[20)]。

アジャイル開発では，1つの開発チームを数名の専門技術者で構成する方法を採用する。小さな規模の場合には4名程度から，大きな規模でも8名程度で構成するようにする。チームに参画する専門家としては，ユーザの代表，要求仕様をまとめる技術者，設計と実現に責任を持つ技術者1名から数名，機能の確認や作成したプログラムの品質を確認する技術者1から2名とするのが，一般的である[21)]。

このように，開発チームの規模を小さくし，それに合わせて比較的小規模なソフトウェアで実装できる機能にシステムの機能を分割し，小さなチームで小さな機能のプログラム開発を実施することで，失敗のリスクを低減するとともに，開発効率を高めることが可能になる[22)]。この方法は，オブジェクト指向技術[23)]が導入される前から，米国のソフトウェア開発組織で試みられていた機能項目別段階的開発[24)]の方法を踏襲したものである。

この方法の問題点としては，類似した機能の開発であっても，上位機能が異なる場合は，類似した2つのプログラムが重複して開発されること。個々の機能項目単位でプログラムが設計・実現・試験され，完成するとそれまでに開発されている部分と統合し，試験が実施されるため，煩雑なソフトウェア統合とテストの作業の繰り返しの労力が増大することなどが

20） このようなことは，以前からソフトウェア工学の研究者の間では，よく理解されていた。しかし，実践を行っている企業の技術者の間では，十分に理解されていたとは言えなかった。さらに最近では，ソフトウェア開発の現場だけでなく，様々な従来型製品の開発現場でも，大規模組織の問題が少しずつ理解されるようになりつつある。つまり，分割統治の原理は踏襲しつつも，一つ一つの開発チームの規模を小さくするようになってきている。

21） このような開発組織の構成は，1980年代後半のNASAにおけるスペースシャトルの組込みソフトウェア開発でも実践されていた。

22） NASAのスペースシャトル組込みソフトウェア開発でも，宇宙飛行士から請求される要求項目は，プログラム記述量で5行から50行で実現できる程度の内容に限定されていた。その場合，要求項目は1つの単文（英語）で表現できる程度のものであった。

23） オブジェクト指向技術（object-oriented technology）は，ソフトウェアの実現に用いるプログラミング言語として，Javaなどのオブジェクト指向言語を利用する開発である。その特徴は，ソフトウェアの対象となる世界の要素をクラスと呼ぶ抽象的な概念のまとまりとして考え，そのクラスにおける他のクラスとの複数の処理や操作をメソッドとして定義し，クラスとそのメソッド群を一塊のものとして設計する。これは，人間の言葉で説明される名詞と動詞に対応させやすく，対象となっている世界の，プログラミング言語による自然な記述となることが知られている。

24） 1970年代の後半，IBMのFederal Systems Divisionに所属していたミルズは，従来のソフトウェア開発で一般的に実践されていたウォーターフォール・プロセスによる開発は，各工程での人間の作業を完全な方法で実践できるとしている。その仮定は誤っているとして，最も重要な機能の試作・実現から初めて，少しずつ開発を積み重ねてゆき，それを目的とするソフトウェアが完成するまで繰り返す段階的開発（incremental development）を提案した［大場充，ソフトウェアの開発技術，1988］。ブルックスは，IBMを退職し，ノースカロライナ大学において，この方法を実験し，良い成果を得たと発表した。「銀の弾丸」の論文でも，この方法が有望であることを述べている。

あげられていた[25]。

ただし，最近のソフトウェア開発環境構築技術の進歩によって，ソフトウェアの統合作業や回帰テスト[26]の自動化などが可能になったため，実際に人間の肉体労働を必要とする部分は縮小しつつあり，作業の効率化と生産性の向上は，進みつつある。このため，重要な問題はシステム全体の機能を，分割統治法を適用して粒度の細かい小さな機能に分解し，個々の小さな部分の開発を個別のチームに割り当てることに集約される。つまり，知的な作業の重要性が増してきている。この機能分割には，高度に専門的な知識と経験が要求されるため，高度な専門家人材の育成が重要な問題になっている。

5. 技術的解決策：アジャイル開発とその合理性

ソフトウェア技術の革新で，アジャイル開発の実践に最も寄与しているものは，オブジェクト指向設計法[27]とそのプログラミング環境である。このプログラミング環境には，プログラミング言語とコンパイラ等の言語処理系だけでなく，汎用ライブラリ[28]の整備なども含まれる。特に，プログラミング言語においては，Java 系言語の普及が大きく寄与している。

25） この問題は，プログラムの実現においては，プログラミング言語でプログラムを記述する作業よりも，複数のプログラムを統合して 1 つのサブシステムに組み上げる統合という処理に時間がかかっていたからである。これは，各プログラムで使われている変数の名前の中に，複数のプログラムで使われる変数を同一のものと認識し，それに同一の記憶域を割り当てなければならないためである。これをリンカと呼ばれる特殊なプログラムを使って実施する。このリンカの処理に必要な時間が長かったのである。最近では，プログラムで参照される変数を完全に分析せずに，一時的な変数名の一覧を作成し，そのプログラムの実行時に本当にその変数が参照される時点で，その変数が複数のプログラムで共有される変数であるかどうかを調べ，既に値が代入されている変数であれば，そのアドレスを同定して，そこから値を読み出せるようにする動的リンキングと言う方法が考え出され，リンカの処理時間が大幅に短縮されたため，問題にはならなくなっている。ただし，これは開発過程におけるリンカの処理時間は短縮できるが，プログラムの実行時にリンク処理の後半部分を実行するので，プログラムの実行時間は遅くなると言う別の問題は発生する。

26） 回帰テスト（regression test）とは，それまでの開発作業で既に実施済みのテストを，何らかの理由で変更を加えたソフトウェアを対象として再実行し，変更によって新たな問題が発生していないことを確認する作業を言う。

27） オブジェクト指向設計法とは，クラス設計のための抽象データ型，情報隠ぺいまたはカプセル化，さらに一般的には上位のクラスの性質をそこから生成されるオブジェクトの性質として引き継ぐ継承（インヘリタンス）の 3 つの方法を適用して，ソフトウェアの部品であるクラスを設計する方法を言う。これによって，一部の部品のプログラミングを変更しても，他の部分のプログラミングに悪影響を与えない保守の容易なソフトウェア設計が可能となる。

28） オブシェクト指向技術を適用する場合，汎用クラスの集合を予め定義しているクラスライブラリが重要であると言われている。これは，開発するソフトウェアが解決すべき問題の領域において一般的に使われる名詞と動詞の組合せを整理し，それらの語彙に対応したプログラムの要素を部品として提供するものである。例えば三角形を問題にする領域では，三角形，底辺，高さなどのクラス定義と，面積を求めるメソッドなどの定義を一般化して定義したクラスライブラリの要素が考えられる。

Prologのような論理型プログラミング言語や，MLやErlangのような関数型プログラミング言語[29]などの非手続き型言語も提案されているが，実用的なソフトウェアの開発では，応用例は多くない。従って，非手続き型言語系ではプログラミング環境も，特にライブラリの整備は，C++やJavaなどのJava系の言語ほどは進んではいない。

Java系の言語を使ってソフトウェアを開発する場合，オブジェクト指向開発の特徴であるクラスライブラリ[30]の選択や準備が重要になる。ソフトウェアの各応用分野別に準備されているクラスライブラリを適切に選択することで，ソフトウェア開発を著しく効率化できる。また，応用分野が特別な問題分野になると，その応用分野に特有なクラスライブラリを準備することが重要になる[31]。そのようなクラスライブラリの開発には，優秀な技術者集団が必要になる。

クラスライブラリの設計は，オブジェクト指向設計では，プログラミングのための基本的なアーキテクチャ（基本構造）を決めることになるので，長期間にわたり利用するソフトウェアの拡張性や変更のし易さに大きく影響を与える[32]。最近のソフトウェアでは，開発期間に

29） 関数型プログラミングとは，ソフトウェアで一つのまとまった機能を実現する「関数」を，手続き型プログラミングでは許されている「副作用」を用いず，あたかも数学の関数のように，変数への値の代入をしない記述法で作成できるようにするプログラミング法や言語を言う。これによって，i=i+1のような表現がなくなるため，プログラムは読みやすく，誤りも少なくなる。

30） 従来型のC言語などに代表される手続き型プログラミングでは，1つのプログラムは，所定の機能の実現の全体を制御する主プログラムと，その主プログラムから呼び出され，特定の定型的な計算を実行する副プログラムや関数から構成される。オブジェクト指向言語のプログラミングでは，計算の手続きはメソッドと呼ばれる部分に記述されるが，そのメソッドは，特定のクラスを構成する要素として，他のメソッドと一緒に定義される。あるクラスにおいては，そのクラスで定義される全てのメソッドは，一連の共通的な（大域）変数を共有する。例えば，共有する変数に値を書き込むメソッドと，書き込まれている値を読み出すメソッドなどである。このメソッドを使うことで，他のクラスのメソッドは，その変数に対して直接，値の代入や参照をせずに，メソッドを使って間接的に値の代入や参照を実行する。そのような多くのプログラミング場面で利用される，汎用的なクラスとそのメソッド群をまとめて定義した，クラスの定義群をクラスライブラリと呼ぶ。これは，従来型の手続き型言語でのサブルーチンライブラリに似たものである。

31） これは，1991年に米国IBMで実施された，主力オペレーティングシステムに対するオブジェクト指向言語での書き換え実験によって判明した事実である。この実験では，その30年前に機械語で書かれた古いプログラムを，新しいプログラミング言語で，その機能とインタフェースを変えずに書きかえるというものであった。対象となったオペレーティングシステムの部分は，その最も古く，基礎的な機能を実現した，比較的小規模な部分であったが，新しい言語での記述量は，約2万行になった。そのほとんどは，クラスライブラリの記述である。このクラスライブラリは，何回か書き直されており，クラスライブラリが書き直されるたびに，それを使う機能の実現を行うプログラムの記述も大きく変更された。この実験の結果から，米国IBMは，自社内の技術者のみで，そのオペレーティングシステムの完全な書き換えを行うことは非現実的と判断した。このことは後に，クリントン政権が新しいソフトウェア開発環境構築を目標とした政策を立案する基礎となった。

32） ISO/IEC 9126の規格では，ソフトウェア品質を機能，信頼性，使用性，効率，保守，移植の6つの視点から評価することが定義されている。この6つの視点の最後の2つ，すなわち保守と移植の問題は，ソフトウェアの寿命を長く保つことが可能かどうかを保証するために必要な性質である。レーマンは，ソフトウェアにsタイプとeタイプがあるとし，eタイプのソフトウェアでは，この2つの性質（保守性，移植性）が重要になることを示した［Lehmann, 1980］。

比較して，運用期間中の移植，修正，機能追加，拡張等の作業に投入する工数の比率が大きくなる傾向がある。このため，ライブラリの設計においてしっかりと情報隠ぺい[33]がなされ，カプセル化されたクラス[34]を設計できるかどうかが重要になる。

Java系言語での開発の場合，既存の汎用開発環境が提供されている。特に，オープンソースで提供されている開発環境[35]もあり，それらの候補の中から，最も適切な開発環境を選定することが重要になる。また，形式手法の応用が要求される開発[36]では，仕様記述言語，仕様チェッカ（証明器を含む）などの選定とともに，図式による仕様記述の入力が可能な例もあり，どのような図式を利用するのかも，重要な問題になる可能性がある。

多数の小規模なチームによって同時並行的に開発作業が進むとき，全体の進捗管理は複雑になる傾向がある。さらに，個々のチームが担当する小さな機能の開発活動の間には，時として相互依存関係が存在する例も多く，あるチームの作業遅れが，数多くの他のチームの開発作業に直接的な影響を与えることも珍しくない。このような場合，開発中の機能間の関係などを明示して，その依存関係に基づいた進捗管理を実施するためのツールが開発環境で提供されることが望まれる。部分的には，そのような機能はすでに提供されている。

問題解決法としては，1960年からの半世紀間に，大きな進歩はなかった。ただし，コンピュータ科学分野の教育カリキュラムが整備されたため，多くの専門家が問題解決の基本的

33） 情報隠ぺいの概念は，1972年にパーナスが発表した論文で提案したものである［Parnas, 1972］。パーナスは，良いソフトウェアと言われているソフトウェアの特徴がそれらの「保守のし易さ」にあることを突き止め，それらの良いと言われているソフトウェアに共通したプログラミング上の性質があることを発見した。その性質をパーナスは「情報隠ぺい」と名付けた。その性質は，個々のプログラムが「どのように処理をしているか」が詳細に説明されていることよりも，「どんな処理をするためのものか」を分かり易く説明するため，どの変数が入力になっており，どの変数が出力になっているのかを明確にして，計算の詳細を可能な限り隠すことが重要であるとした。特に，他のプログラムにおいても参照され，値が代入される大域変数を使うことが危険であるとした。この情報隠ぺいの概念と，オブジェクト指向設計で重要とされるカプセル化の概念は，理論的には同じことを言っている。

34） 情報隠ぺいがしっかりと考えられているクラスの設計を，「カプセル化」されたクラスの設計と言う［Scott, 2006］。

35） オープンソースでは，ソフトウェアのソースコードを広く提供し，ライセンス規約によって，それを順守する全ての技術者に対してその改変や修正を許可する［Weber, 2005］。従って，ソースコードに対する独占的な著作権は主張しない。ライセンス規約によっては，改変したソフトウェアのソースコードもオープンソースとして公開することを求めている例もある。最も有名なライセンス規約にLinuxを提供しているGNUのGPLがある。オープンソースとして提供されているJava言語などによるプログラム開発を支援するプログラミング環境に，IBMが提供しているeclipseがある。

36） 形式手法とは，ソフトウェアの機能仕様を数学的に記述し，開発されたプログラムがその仕様を満足することの証明も可能とすることを言う［Abrial, 2010］。機能仕様は，VDLと呼ばれるプログラミング言語に似た特殊な言語，数学的な記法に似た方法を採用するZと呼ばれる言語などで記述される。車載組込みソフトウェアの開発で自動車の安全性に悪影響を与えないことを証明する方法として，ISO26262ではそのような形式手法を応用して組込みソフトウェアの設計を実施することを推奨している。

な方法を知るようになった[37]。1960年頃から1970年頃にかけて，ソフトウェア工学の黎明期に，ダイクストラが提唱し，ビルトによって確立された『構造化プログラミング』によって，トップダウンに問題を考える方法，さらに段階的に詳細化してゆく方法が一般的に教育されるようになった［大場充，ソフトウェアの開発技術，1988］。

トップダウン法は，ローマ人が考案した「分割統治」の方法を複雑な問題の解決に応用しようとする考え方で，古くから数学では利用されてきた方法の一つである。さらに，ジーコブスキーが一般意味論[38]で提唱した段階的詳細化の方法論を取り入れ［ハヤカワ，1985］，古典的な構造化プログラミングの方法が確立された。似たような方法が，プロジェクトマネジメントの分野でも，WBS（Work Breakdown Structure）法として，作業の詳細化のための方法として提案されている［Norman, 2010］。

アジャイル開発におけるシステムの機能項目の分解や，それに基づく小さなチームによる小さな機能の開発は，この分割統治の考え方を取り入れて，ソフトウェア開発の失敗のリスクを低減しようとしている。この小さな機能インクリメント[39]に分解して，リスクの小さな作業を繰り返すと言う方法は，1970年代の後半に，ミルズが「段階的開発法（incremental

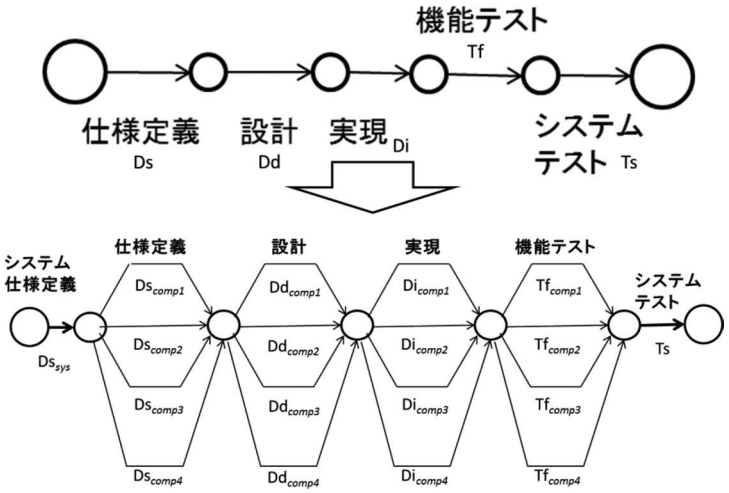

図 1．システムのウォーターフォール開発

37） 最新のカリキュラムについては，以下を参照されたい。https://www.acm.org/education/CS2013-final-report.pdf

38） ジーコブスキーは，彼が提唱した一般意味論において，自然言語によるコミュニケーションでは，問題記述の抽象化と，抽象的に表現された記述の段階的な詳細化の，対立する 2 つの方法を適宜，応用することが重要であるとした。このことによって，自然言語で伝えたい内容は，一般的で，抽象的ではあるが伝わりやすい表現に始まり，徐々に，個別的ではあるが，具体的で，詳細な複数の説明へと展開されてゆくことで，誤解なく伝わるのである。

39） 分割統治法を適用して，大きな機能から分解された，他の小さな機能の実現からは独立した，小さな機能を実現する一塊の小規模なプログラム要素集合を「機能インクリメント」と呼ぶ。

development)」として提案したものである［Brooks, 1987］[40)]。

例として，あるシステムの開発をウォーターフォール開発で実施しようとすると，仕様定義，設計，実現，機能テスト，システムテストの工程は，図 1 上段の PERT 図に示されるように計画され，図 1 下段の PERT 図のように分解され，実施されることになる［大場充，組込みソフトウェア工学ハンドブック，2014］。

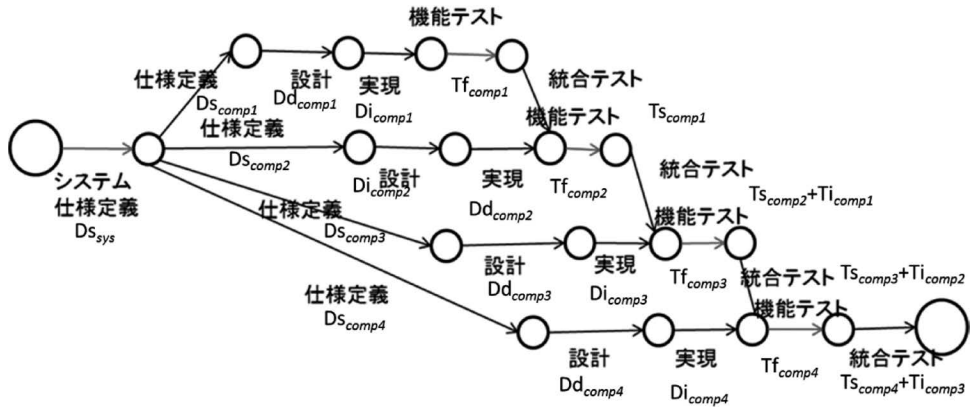

図 **2**．システムの段階的開発

この図 1 のような開発は，仕様定義の開始から実現の完了まで，途中の段階で仕様の正しさや設計・実現の正しさをレビュー以外の方法で検証することは困難である。このことから，機能テストの開始まで，開発プロジェクトが本当に成功しそうかどうかは，開発チーム外部の人間には全く分からない。つまり，リスクは隠された状態にある。

ミルズの提唱した段階的開発では，このウォーターフォール開発を細かく分割することによって，図 2 の PERT 図に示されるような複数の小工程に分解する。図 2 においては，図 1 で仕様定義とされた工程が，大きくシステム仕様定義の工程と，4 つの分解された機能項目別の仕様定義の工程に分解されている。さらにそれに続く設計，実現，機能テスト，統合テストも，機能項目別の作業に分解されている[41)]。

ここで，図 1 上段の仕様定義を Ds と表記し，図 1 下段と図 2 のシステム仕様定義を Ds_{sys}，分割された個々の要素の仕様定義を Ds_{comp1} から Ds_{comp4} までの 4 つの表記で表す。ここで，

40）ミルズは，これによってウォーターフォール開発では避けることの不可能な巨大なリスクを，管理できる大きさのリスクに縮小しようとした。

41）このように作業を細分化することで，個々の作業を小さなチームで実施することが可能になり，管理を容易にするとともに，失敗の影響を小さくすることが可能になる。図 2 の PERT 図では，ひとつひとつの矢印で示される作業に必要な工数は，図 1 上段の PERT 図の矢印の作業に必要な工数に比較して，4 分の 1 程度に小さくなっている。

任意の作業または工程 x の工数を与える関数を $w(x)$ で表現する。このとき，以下の関係が成立する。

$$w(Ds) \cong w(Ds_{sys}) + w(Ds_{comp1}) + w(Ds_{comp2}) + w(Ds_{comp3}) + w(Ds_{comp4}) \quad (1)$$

これは，作業として見たとき，Ds と，Ds_{sys} および Ds_{comp1} から Ds_{comp4} までの全ての作業の内容が，ほぼ等価なものであることを仮定している。

さらに，図 1 上段の設計を Dd と表記し，図 1 下段と図 2 の分割された個々の要素の設計を Dd_{comp1} から Dd_{comp4} までの表記で表す。このとき，任意の作業または工程 x の工数を与える関数 $w(x)$ を用いれば，以下の関係が成立する。

$$w(Dd) \cong w(Dd_{comp1}) + w(Dd_{comp2}) + w(Dd_{comp3}) + w(Dd_{comp4}) \quad (2)$$

ここでも，作業として見たとき，Dd と Dd_{comp1} から Dd_{comp4} までの全ての作業の内容が，ほぼ等価なものであることを仮定している。

同様にして，図 1 上段の実現を Di と表記し，図 1 下段と図 2 の分割された個々の要素の実現を Di_{comp1} から Di_{comp4} までの表記で表す。また，図 1 上段の機能テストを Tf と表記し，図 1 下段と図 2 の分割された個々の要素の機能テストを Tf_{comp1} から Tf_{comp4} までの表記で表す。このとき，任意の作業または工程 x の工数を与える関数 $w(x)$ を用いれば，式（3）および式（4）の関係が成立する。

$$w(Di) \cong w(Di_{comp1}) + w(Di_{comp2}) + w(Di_{comp3}) + w(Di_{comp4}) \quad (3)$$

$$w(Tf) \cong w(Tf_{comp1}) + w(Tf_{comp2}) + w(Tf_{comp3}) + w(Tf_{comp4}) \quad (4)$$

ここでも，作業として見たとき，Di と Di_{comp1} から Di_{comp4} までの全ての作業，および Tf と Tf_{comp1} から Tf_{comp4} までの全ての作業の内容が，ほぼ等価なものであることを仮定している。

現実を考えると，式（4）は，成り立たない。それは，機能テストで問題が発見された場合，その修正作業が実施されるからである。厳密に考えると，式（4）は，式（5）のようになる。

$$w(Tf) \geq w(Tf_{comp1}) + w(Tf_{comp2}) + w(Tf_{comp3}) + w(Tf_{comp4}) \quad (5)$$

ここでは，式（4）の左辺と右辺の差は，無視できる程度であると仮定する。この差は，図 1 のウォーターフォール開発の場合，システムが 1 つの完結したものとして，設計，実現されているため，機能テストで誤りまたは設計から修正すべき問題が発見されると，その修正がシステム全体に波及する例があり，テストで問題が発見された処理だけでなく，他の部分

の設計変更，プログラムの修正も必要になる例があるからである[42)]。

最後に，図 1 上段のシステムテストを Ts と表記し，図 1 下段と図 2 の分割された個々の要素の統合テストを Ti_{comp1}から Ti_{comp4}までの表記で表す。このとき，任意の作業または工程 x の工数を与える関数 $\mathrm{w}(x)$ を用いれば，式（6）の関係が成立する。

$$\mathrm{w(Ts)} < \mathrm{w(Ti}_{comp1}\mathrm{)} + \mathrm{w(Ti}_{comp2}\mathrm{)} + \mathrm{w(Ti}_{comp3}\mathrm{)} + \mathrm{w(Ti}_{comp4}\mathrm{)} \tag{6}$$

ここでは，作業として見たとき，Ts と Ti_{comp1}から Ti_{comp4}までの全ての作業の内容について，以下の関係が成立していることを仮定している。

$$\mathrm{w(Ti}_{comp1}\mathrm{)} = \mathrm{w(Ts}_{comp1}\mathrm{)} \tag{7}$$

$$\mathrm{w(Ti}_{comp2}\mathrm{)} = \mathrm{w(Ts}_{comp2}\mathrm{)} + \mathrm{w(Ti}_{comp1}\mathrm{)} \tag{8}$$

$$\mathrm{w(Ti}_{comp3}\mathrm{)} = \mathrm{w(Ts}_{comp3}\mathrm{)} + \mathrm{w(Ti}_{comp2}\mathrm{)} \tag{9}$$

$$\mathrm{w(Ti}_{comp4}\mathrm{)} = \mathrm{w(Ts}_{comp4}\mathrm{)} + \mathrm{w(Ti}_{comp3}\mathrm{)} \tag{10}$$

ただし，Ts_{comp1}は，Ts のうちの *comp*1の要素に関係するテストだけを選択したものであるとする。Ts_{comp2}から Ts_{comp4}までも同様とする。従って，システム統合を Ti と表現すると，式（11）が成立する。

$$\mathrm{w(Ts)} \cong \mathrm{w(Ts}_{comp1}\mathrm{)} + \mathrm{w(Ts}_{comp2}\mathrm{)} + \mathrm{w(Ts}_{comp3}\mathrm{)} + \mathrm{w(Ts}_{comp4}\mathrm{)} + \mathrm{w(Ti)} \tag{11}$$

このことから，アジャイル開発における統合テスト Ast の総工数に関して，式（12）が得られる。また，Ti について式（13）が成り立つとする。

$$\mathrm{w(Ast)} \cong 4\mathrm{w(Ts}_{comp1}\mathrm{)} + 3\mathrm{w(Ts}_{comp2}\mathrm{)} + 2\mathrm{w(Ts}_{comp3}\mathrm{)} + \mathrm{w(Ts}_{comp4}\mathrm{)} \tag{12}$$

$$\begin{aligned}\mathrm{w(Ti)} &\cong \mathrm{w(Ti}_{comp1} + \mathrm{Ti}_{comp2} + \mathrm{Ti}_{comp3} + \mathrm{Ti}_{comp4}\mathrm{)} \\ &\leq \mathrm{w(Ti}_{comp1}\mathrm{)} + \mathrm{w(Ti}_{comp2}\mathrm{)} + \mathrm{w(Ti}_{comp3}\mathrm{)} + \mathrm{w(Ti}_{comp4}\mathrm{)}\end{aligned} \tag{13}$$

[43)]

これより，

$$\mathrm{w(Ast)} \geq \mathrm{w(Ts)} \tag{14}$$

42） この問題は，ウォーターフォール型の開発においては，設計の良さに依存することが知られている。「設計の良さ」とは，パーナスが提唱した情報隠ぺいが十分に考慮された設計であるかどうかを言う。しっかりと情報隠ぺいが考慮されている設計では，一部の機能の実現が変更されても，それが他の部分に影響する確率は小さくなる。

43） これは，*comp*1から *comp*4までの要素を統合する作業について，全体を 1 つのシステムに統合する作業と，*comp*1と *comp*2を統合し，それに *comp*3を統合し，最後にそれに *comp*4を統合する作業を別々に実施するのとでは，前者の方が工数が少なくなることを意味している。

が導かれる。テストの工数から見れば，アジャイル開発は，ウォーターフォール開発より工数が増加する傾向がある。

アジャイル開発では，仮に一部の機能項目別の開発工程で失敗が生じても，作業のやり直しは，その部分の開発だけに限定でき，これによる工数の増大を最小限にすることができる。また，各機能項目別開発工程の最後の統合テストでは，それまでの開発で実現できているソフトウェアが統合され，テストされるので，その実現品質を第三者が確認することが可能になる。これは，開発中のソフトウェアの品質の可視化になると言う利点になる。

ただし，図 1 と図 2 を比較すれば分かるように，図 2 の工程の方が，重複が多く，冗長であり，結果として投入する工数は，上述したように増大する結果になる。つまり，リスクをしっかりと管理でき，失敗がなければ，式（13）の関係からウォーターフォール開発の方が，効率は勝っている[44)]。ではなぜ，それでもアジャイル開発なのか。その理由は，前述した経済のサービス化による変化の速さが問題だからである。つまり，開発の途中であっても社会の様々な変化によって，システムへの要求が変化する可能性があるからである。

分解されたシステムの機能項目のうちのいくつかは，その実現が完了するまでに仕様そのものを変える必要性が生じる。そのため，仕様定義から作業をやり直さざるをえない状況になる。このとき，ウォーターフォールでは，全ての開発を最初から見直し，必要なやり直しを実施することが必要になる。段階的開発を採用しているアジャイル開発であれば，このやり直しの対象範囲を，やり直しが必要な機能項目だけに限定することができる。この歩留まりの良さが，現代のソフトウェア開発では決定的に重要である[45)]。

仮に，図 2 に示された例の第 3 番目の要素 *comp*3に対する要求が，社会の変化によって大きく変化したと仮定する。この場合，ウォーターフォール開発では，新しいシステムの開発をやり直すことになる。つまり，全体の工数 w(*T*w) は，式（15）で与えられる。

$$\begin{aligned} w(Tw) = {} & w(Ds) + w(Dd) + w(Di) + w(Tf) + w(Ts) \\ & + w(Ds') + w(Dd') + w(Di') + w(Tf') + w(Ts') \end{aligned} \tag{15}$$

44） このことは，現場で開発を行っている技術者達には知られていない。現場で開発を行っている場合，アジャイル開発の効率は作業者達の感覚によって判断される。各インクリメントの開発は規模が小さくなり，短期間での開発ができるようになり，ユーザからのフィードバックも得られやすくなるため，感覚的には効率が改善したように感じられるのであろう。

45） 全ての機能項目の開発をやり直しなく，成功裏に終えられる確率はかなり小さいはずである。例えば，ユーザがソフトウェアの機能に関する要求を最初から明確に述べられる確率は小さい。さらに，開発を担当する技術者がユーザが意図したことを正確に理解できずに，間違って理解する確率も高いはずである。だとすれば，プログラムとして実現されたソフトウェアがユーザの意図した機能を実現したものである確率は，小さいはずである。ただし，アジャイル開発では，テストの時点でユーザは，自分の意図したものが作られていないことを認知できるので，修正を依頼することができる。そのような意味で，開発の歩留まり率は改善されると言える。

これに対して，アジャイル開発では，*comp*3の修正だけで解決できるため，全体の工数 w(*T*a) は，式（16）で与えられる。

$$\begin{aligned} w(Ta) = {} & w(Ds_{sys}) + w(Ds_{comp1}) + w(Ds_{comp2}) + w(Ds_{comp3}) + w(Ds_{comp4}) \\ & + w(Dd_{comp1}) + w(Dd_{comp2}) + w(Dd_{comp3}) + w(Dd_{comp4}) \\ & + w(Di_{comp1}) + w(Di_{comp2}) + w(Di_{comp3}) + w(Di_{comp4}) \\ & + w(Tf_{comp1}) + w(Tf_{comp2}) + w(Tf_{comp3}) + w(Tf_{comp4}) \\ & + w(Ti_{comp1}) + w(Ti_{comp2}) + w(Ti_{comp3}) \\ & + w(Ds_{comp3'}) + w(Dd_{comp3'}) + w(Di_{comp3'}) + w(Tf_{comp3'}) \\ & + w(Ti_{comp3'}) + w(Ti_{comp4}) \end{aligned} \tag{16}$$

さらに，式（15）と式（16）の間には，以下の関係が仮定できる。すなわち，

$$w(Ds_{comp3'}) \ll w(Ds') \tag{17}$$

$$w(Dd_{comp3'}) \ll w(Dd') \tag{18}$$

$$w(Di_{comp3'}) \ll w(Di') \tag{19}$$

$$w(Tf_{comp3'}) \ll w(Tf') \tag{20}$$

$$w(Ti_{comp3'}) + w(Ti_{comp4}) \ll w(Ts') \tag{21}$$

である。このことから，開発中にシステムの要求に変更が発生するリスクが一定程度存在すると考えられる場合，ウォーターフォール開発よりも，アジャイル開発の方が，総工数の期待値は小さくなる。つまり，開発効率が向上する。同様の議論は，開発期間についても可能である[46)]。

6. 社会的課題：日本社会が直面している課題

アジャイル開発の時代背景として，経済のグローバル化に伴った先進国経済のサービス化は，それまでの資本主義における「時間」とは違った時間観を我々に要求している［大場充，経時的に変化する製品の品質，2017］。従来の資本主義，特に産業化社会時代（Industrial Society Age）の時間は，一人の人間の一生をかけてゆっくりと進む程度の進み方であった。変化はゆっくりとしており，1980年代の社会は，1970年代・1960年代の社会と似ていた。

46） 工数の議論では，作業の工数を与える関数を定義したが，時間を議論する場合には，作業完了までの時間を与える関数 t を定義すればよい。ただし，その場合，並行的に実施できる作業については，より時間を必要とする最大時間をとる関数となる。

しかし，1990年以降の時代になると，社会の様相は，1960年代のものとは大きく様変わりした。1980年代中頃までの時代は，時間の進み方がゆっくりとしていて，我々は社会の変化を実感できるほどの著しい変化を感じなかった[47)]。

このような変化が次々と起こる時代の世界においては，我々には数年先に起きそうなこと，10年先の社会がどのような社会であるのか，10年先の人々が何を必要とするのかなどを予想することが難しい。つまり，産業化社会の時代であれば，10年先の社会がどのような社会であるのかを予想し，予測できたが，現代の社会ではいくつかの根源的な変化を除き，将来の変化を予測することができない。予測が可能な例としては，各地域の人口の変化，各地域に居住する人々の最終学歴の変化程度である。10年後の我々の住む国家の GDP の予測値ですら，予測は困難になっている[48)]。

このように変化の激しい時代，ソフトウェアでどのような機能を実現すべきか，どのようなサービスを開発すべきかなどの問題を考えることは，難しい課題の範疇に入る。仮に，特定の目的を達成するシステムを計画したとしても，そのシステムの主要な機能がどのようなものであるべきかについては，それを具体化してゆく過程で，初期の段階では全く想定されていなかった機能を，組み込まざるをえない状況に陥ることは頻繁に起こるであろう。

このような当初想定していなかった要求が，開発活動の進展に伴って発生する例は，従来の開発でもまれに経験された［ブルックス，1982］。しかし，それはプロジェクトの初期段階における検討不足が原因で，しっかりと検討しておけば起こらなかった機能の追加や修正が明確になったものであった。つまり，その新しい機能の追加や既存の機能への修正は，不可避的なものではなく，人為的な現場作業での失敗が原因で発生したものなのであった。これに対して，これからの時代に我々が直面する問題は，どんなに注意深く検討しても事前には予測不可能な事態が現実となり，要求や機能仕様の変更をせざるをえない状況が発生するというものである。

日本人は，国民性として変化を嫌う傾向がある［リンカーン，2004］。そして，変化や変

47） しかし，今日，我々が生きている社会における時間の進み方は，どんどん早くなっている。例えば，50年前には，普通の企業にもたくさんの電話機があったが，携帯電話はなく，小規模の企業にはコンピュータはなかった。コンピュータがほとんどの企業に入り，会計処理や在庫管理などに利用され始めた30年前，携帯電話を持っている人はほとんどいなかった。個人の家にパーソナルコンピュータはなかった。そして小学校にもパーソナルコンピュータが入り始めた20年前，コンピュータはインターネットに繋がっていなかった。アジアやアフリカの社会では変化はもっと顕著である。これまで，社会に電話網がなく，必要な時でも電話を利用することができなかった人々が，今ではスマートフォン携帯電話を持ち歩き，いつでもどこでも，家族と話をし，世界の最新の情報を調べ，メールの交換をすることができるようになっている。

48） 米国 IBM の基礎研究部門では，1970年代から「10年後の計算機環境」（10 Year Outlook）に関する報告書を，毎年同社経営陣に提出している。1990年代中頃からは，その将来予測を大きく超えた事態が起き始めていた。

更が不可避と認識する状況に追い込まれても，可能な限り従来からのきまりごとに沿って物事を決めることを好む。しかし，そのような態度は，問題の原因が本質的な変化である場合，問題への対応が表面的になり，実質的な問題の解決にならない結果に終ることがある。このような表面的な取り繕いを重ねていると，いつか社会全体としては，全く時代の潮流に適合しないものとなり，社会全体が危機に陥ることになりかねない[49)]。

特に，経済のグローバル化が進むと，他国の法律と日本の法律との不一致や矛盾が問題になる。それは，日本企業が外国で業務を遂行する場合にも，逆に，海外の企業が日本国内で業務を実施する場合にも，ある企業が事業を実施する国において守るべき法律が本国と異なると，業務実施上の制約が国によって異なる結果となる。2016年当時，日本政府や米国政府と環太平洋諸国の政府が協議を行い，条約締結にこぎつけようとしていた環太平洋パートナーシッププログラム（TPP）は，そのようなこれまでは国別に決められていた法律を，国境を越えて共有しようとする努力であった[50)]。グローバルな経済の運営を効率的に実施しようとするとき，このような調整が必要になる。

日本社会が世界の変化に対する即応性が低い理由の一つに，日本社会における根源的な規範である「既得権」重視の態度がある。社会の制度を変更するとき，できるだけ，その変更によって不利益を得る人が出ないようにするという考え方である[51)]。新しいシステムや制度の導入により，サービスの対象となる人々の中で，誰がどの程度の利益を受け，誰がどのような不利益を被るのかを総合的に評価して，導入を決定することが一般的な世界的規範である。しかし，日本の社会では既得権を持つ人々が被る不利益を過大評価する傾向がある。

このことが，日本社会における迅速で適切な変革を困難にし，日本社会を硬直化させる。このような傾向は，ヨーロッパ社会の一部にも見られることであるが，ヨーロッパ社会には日本社会ほどの硬直性はなく，時代の変わり目においては大きな変化を受け入れる社会規範が確立している。これは，両社会の有史以来の社会における歴史の蓄積の差が影響している

49） 例えば，現在の日本の民法は，その基礎を江戸時代の社会規範に置いた明治時代の民法を踏襲している。第二次世界大戦に敗北し，憲法を帝国憲法から平和憲法に変えても，民法のほとんどの条項は戦前からの民法を引き継いだ。この民法の条文は，男尊女卑の思想など，平和憲法の根本的な思想と矛盾する例もあり，本来であれば戦後，平和憲法の制定に続いて，新しく書きかえられるべきものであった。しかし，国会議員も法務省もその努力を惜しみ，最低限の表面的な修正のみにとどめた。このため，婚外子と嫡出子の法的地位の差など，最高裁判所の審判によって解釈を変えるなどの，表面的な対応を迫られている。

50） 2016年11月の米国大統領選挙で当選したトランプ候補は，選挙戦においてTPP協定が米国で生産される製品の海外への輸出を阻害する要因になり，結果的に米国労働者の失業率を高めるとして，TPP協定からの脱退を表明していた。2017年1月にトランプ氏が大統領に就任した直後，同氏は米国がTPP協定から脱退する大統領令に署名し，米国は日本や中国などと個別に2国間の貿易協定を締結する方針に転換した

51） コンピュータを活用した処理を導入するとき，その変化によって，従来は何も問題を持っていなかった人々にとって，何か不利益になるような事態が発生してはならないという考え方である。

と言える[52)]。

日本の社会は，中国式律令国家が成立した貴族社会，鎌倉幕府が成立して以降，江戸幕府が滅びるまでの武家を中心とした封建社会，明治維新以降の第二次世界大戦に敗戦するまでの官僚と軍人を中心とした天皇制の社会，敗戦後の平和憲法に基づく国民を中心とした民主制の社会へと，変化してきた。この間，行政の主体は変化してきたものの，行政の主たる方法については，戦後の民主制に移行するまで基本的な変化はなかった[53)]。

経済のグローバル化は，これまでに人類が経験した封建社会から資本主義社会への変革，巨大国家の帝国主義社会から民主主義国家の産業化社会への変革などと同じ水準か，またはそれ以上の大変化が予想される地球規模での社会秩序の変革である。日本が今後も国際社会においてこれまでと同じような貢献を維持しようとするのであれば，この大きな社会的変革にどう対応すべきかの戦略に基づいた制度改革，さらに国民の意識改革を進める必要がある。我々は，その覚悟を明確に持ち，この国家的な改革事業，すなわち社会制度と国民意識の改革に立ち向かわなければならない。

7. 教育的課題：技術者はどう育成すべきか

経済のグローバル化が進み，先進国における経済のサービス化が加速度的に進展する社会において，ソフトウェアだけでなく，全ての製品開発がアジャイル開発化するのは自然な傾向のようである。それは，ポスト資本主義の経済における「時間」の重要性が，これまでの産業化社会における資本主義経済の時間とは，全く様相を異にするためである[54)]。

そのアジャイル的な方法が成功する基本的条件の中で，最も重要なものが，作業に参画する技術者の技術的な知識の質の高さと，彼ら彼女らのコンピテンシ［スペンサー，2011］の高さである。前者は，作業を確実に実施するために必要な知識を確実に持っているかどうか，そしてその知識の質が作業の遂行に要求される水準を上回っているかどうかを保証するための必要条件である。後者は，そのような仕事に従事する技術者たちの知識の量と質が十分で

52） 古代ローマ帝国の滅亡以来のヨーロッパの歴史を見ても，ローマ教会を中心とした十字軍までの中世封建制度の社会，ルネッサンス期を経て宗教改革と30年戦争を経験した社会，科学と知識が発展し産業革命が進展するとともに資本主義が誕生した市民革命時代の社会，資本主義と帝国主義が一段と進み民主化・産業化が進展した社会，そして資本主義が成熟して産業化社会から経済のグローバル化が始まった社会と，大きな変化を受け入れながら進展してきている。

53） そして，この民主制への移行も，日本人がその意志で主体的に変化を選択したものではなく，日本社会を軍国主義社会から民主主義社会へ転換しようとする米国政府の影響を強く受けたものであったと言える。

54） 時間の進み方が早くなり，社会の時間変化はめまぐるしくなりつつある。そのような環境下で，合理的にリスクを管理し，利益を上げられるソフトウェア開発や製品開発をしようとすれば，今の時代の我々にできることは，アジャイル開発になる。

あったとしても，その技術者たちにその仕事に真剣に取り組み，自分の能力の全てを発揮して問題を解決しようとする姿勢がなければ，使命を達成することはできない。そのため，一人の技術者としての仕事に向き合う姿勢を保証する十分条件である。

技術的な知識の基本的なものは，全て高等教育で学ぶことができる。もちろん，高等教育機関で教育を受けた者は，社会が必要であると認識している全ての知識を保持しているかどうかを判定し，その資格を認定しなければならない。大学では，履修単位の認定がこれに相当する。さらに，一般的には知識を保持していることは単に必要条件を満足していることを意味しており，十分条件を満足しているわけではない。実務経験によって教育の過程で取得した知識を適確に応用し，現実の問題解決を実践の場でやれるかどうかの応用能力を判断することが問題になる[55)]。

技術者の「仕事に向き合う姿勢」は，高等教育で学ばせることは容易でない。それは，この生きる姿勢が人間の人生の早い段階で脳に組み込まれ，初等教育以降の知識獲得を中心とした教育では，修得させることに困難性があるパーソナリティや性格に近いものだからである。この「生きる姿勢」は，人間が誕生から幼児期にかけて，特に母親や周囲の家族から，日常生活を通して学ぶ基本的価値観で，ものの善悪の判断や好き嫌いの判断を形成する過程で獲得されるものである[56)]。

子供の自制心の強さとその後の社会的成功との因果関係が，1960年代の後半から1970年代前半にかけて米国のスタンフォード大学で実施された「マシュマロテスト（マシュマロ実験）」[57)] で確認された。それは，自制心の強い子供ほどその後の生涯で成功する確率が高い傾向にあることを示した［ミシェル，2017］。マシュマロテストの実験の後も，その被験者の追跡調査が続けられた。その結果，高等教育まで進んだ子供は，マシュマロをすぐに食べるのを我慢した子供たちに多く，さらに社会的に成功した人々も，マシュマロをすぐに食べるのを我慢した子供たちから相対的に高い確率で出た。

マシュマロテストが意味していると考えられていることは，「自制心」と言う人生の早い段階で形成される行動の特性，または獲得される生きるための姿勢や規律が，人間の人生における成功と深く結び付いているのではないかとする仮説である。ただし，この実験で自制心を示した子供たちは，相対的に裕福な家庭の子供が多く，別の観点から見ると，親の社会階

55） インターンシップにおける就業体験や過去の実務経験が問題にされるのは，このためである。

56） これらの価値観は，人間が言語能力を獲得する過程とほぼ同時に学習するものなので，言語能力を獲得した後に学習する形式的な知識よりも先に人格の一部として確立される傾向にある。

57） この実験では，200人程度の入学前の児童に対して，実験者はマシュマロを 1 つ見せ，「これを，直ぐに食べても良い」と言う指示を出した後，「しかし，私が部屋を出て戻るまでの少しの間，食べるのを我慢して食べなかったら，戻った時にもう一つマシュマロをあげます」と約束して，被験者である児童を部屋に残して部屋を出た後，被験者がどのような行動を取るかを観察した。

級格差が表出しているとも考えられる。いずれにしろ，高い自制心をもった人ほど，その生涯を通して成功する可能性は高いと言える[58)]。

専門家として社会へ出て，技術者として仕事に従事している人々について述べると，技術者として働いているからと言って，社会的に成功しているとは言えない。技術者として社会的に成功するためには，特定分野の専門家として社会的に認知される必要がある。それは，同じ専門分野で仕事をする他の専門家と比較して，他の専門家が持たない特殊な知識をもっていたり，他の専門家では解決できない問題を解決できる能力を持っていたりすることが必要である。

全ての技術者が，高等教育を終えたばかりで，実社会へ出たばかりの時からそのような専門家として社会的に認知された人になっているわけではない。数多くの技術者集団の中で，優秀な技術者として認知されるためには，技術者としての経験を少しずつ積み重ねることで，少しずつ成長をし続け，最終的にその人しか分からないこと，その人しか解決できない問題があるという状態に到達する［フロリダ，2008］。

ある段階まで到達した技術者が学ぶことを怠り，それまでに獲得している知識や経験だけに基づいてのみ仕事をし続け，自分の仕事の仕方を改善する努力をしなくなれば，その技術者はその段階で成長がとまり，それ以上の段階に進むことができなくなる。これがその人の限界と言うことになる。このことは，組織論で「ピーターの法則」としてよく知られた理論である［ピーター，1970］[59)]。

このことから専門家としての技術者は，専門家として生きてゆこうとする限り，「学び続ける姿勢」を維持しなければならない［猪飼美恵子，2006］。新しい形式的な知識を学び，また，自分の実務経験から学んで，問題解決のための知識を増やしてゆかなければならない。これは，大学病院などの医師が，医師として勤務し続けるために，自分の専門分野における新しい理論や，新しい病気の知識，新しい治療法，新しい薬について学び続けることを要求されているのと同じである。技術者も専門家として生きる限り，その生き方に違いはない。

しかし，この学び続ける姿勢を維持することは，決して容易なことではない。技術者がこの学び続ける姿勢を放棄してしまうと，その技術者が実社会で実践する問題解決は適切なものではなくなり，時として社会的な問題を引き起す原因になるかも知れない。もしそのようなことが起これば，社会的な損失が生まれる。例えば，企業が生産活動の結果として排出す

58） ここで重要なことは，小学校以降の教育で獲得した形式的知識だけでは，人間は成功できないと示したことである。幼児期からの家庭教育によって，しっかりとした人格が形成され，さらに小学校教育から高等教育までで獲得する形式的知識の蓄積に成功した専門家や技術者でなければ，実社会での実践的な問題解決に成功し，生き残ってゆくことはできない。

59） ピーターの法則では，「人間は，無能であることが分かるまで，組織の中で昇進し続ける」と表現している。つまり，能力があると判断されている間は，組織の階段を上り，知識や経験が不足して能力が発揮できなくなると，出世が止まることを意味している。これは，専門家としての技術者も同じである。

る公害によって地域社会に損失が出るような問題が生じるかもしれないと考えられる。

技術者が適用しようとしている技術にそのような副作用があることを調べ，問題が発生する可能性があれば，技術者は専門家としてそのような解決策を採用することを回避しなければならない。「そのような副作用があることは知らなかった」と言う言い訳は，専門家の言葉として社会的には受け入れられない。専門家としての技術者にはそれだけの社会的責任が背負わされている［大場充，ソフトウェア技術者：プロの精神と職業倫理，2014］[60)]。

重要なことは，問題解決のための技術の選択権限を専門家である技術者に委譲した一般の人々は，技術者の説明に納得して，その選択判断を受け入れなければならない。それが技術者と一般の人々との間の，暗黙の契約である[61)]。専門家としての技術者は，一般の人々に対して，選択する技術の利点と欠点について説明し，自分の選択が合理的であることを説得し，一般の人々の理解を得る責任がある。これは，技術者と一般の人々とのコミュニケーションを通して実施されなければならない。技術者は，この一般の人々とのコミュニケーションを怠ってはならない[62)]。

このような専門家としての社会的責任を全うするためには，技術者たちがその社会的責任を明確に認識し，その社会的責任を果たすためのコミュニケーションの方法に関する知識をもち，そのコミュニケーションの実践を学んでおく必要がある。社会的責任の認識については，技術者として生きる姿勢と倫理観に関する問題なので，高等教育で学ばせることは困難である。これに対して，コミュニケーションの方法論や実践能力については，教育することが可能である。このような高等教育で教育訓練が可能な知識や実践能力は，一般的にジェネリックスキル（generic skills）と呼ばれる［PROG 白書プロジェクト，2014］。

60） それは，科学技術の進歩によって，一般の人々には，利用する技術の利点・欠点を理解する科学的知識を持たないため，技術選択の権限を技術者に移譲しているからである。技術者は，一般の人々に代わって，技術の選択権限を行使している。

61） これは，医師が病気の治癒を目的として，手術方法の選択などについて患者から全ての権限を委譲され，その選択を行い，患者に選択した手術方法の説明を実施して了解を得たうえで，手術を実施することと同じである。仮に手術に成功しなかったとしても，患者やその家族は医師による説明に納得して，医師に手術法選択の権限を委譲したわけなので，それで問題が発生するわけではない。失敗の責任は，権限を委譲した患者やその家族に帰属すると考えられる。これと同じことが，技術者の専門分野での問題解決にも適用される。

62） 例えば，自動車のブレーキでセンサからの情報を，組込みコンピュータ上で稼働するソフトウェアで処理し，必要な時，車輪の回転を止めるブレーキ機構を搭載することで，自動車の安全性を向上させる技術がある。一般の人々は，このような技術を応用すれば，自動車をコンピュータで完全に制御することも可能であろうと考える。しかし，この考え方には大きな危険性が潜んでいる。それは，コンピュータを動かしているソフトウェアは技術者によって記述されるものであり，人間が記述する限りにおいては，誤りが混入することを完全に排除することはできない事実である。もし，重要な部分にそのような誤りが混入し，テストで発見することができなければ，大事故が起きる可能性が残ることとなる。テストも人間によって実施されるからである。自動車開発に携わる技術者達は，そのような現実の技術の限界を，一般の人々に分かり易く説明する責任がある。「売れるから」と言う理由で，むやみにそのような自動車を実現し，販売することは，無責任である。

専門的な技術者の養成を目的とする高等教育では，専門分野の形式的な知識を獲得させるための専門教育と，専門に限定されないジェネリックスキルの両方を教育訓練することが必要である。しかし，これまでの日本の大学教育では，このジェネリックスキルに関する教育が軽視されていた。技術が高度化し，経済のグローバル化が進展するこれからの社会では，日本の大学教育も世界的な流れに従い，ジェネリックスキルの教育にも力を入れる必要がある。そうしなければ，日本で新しい世界が必要とする人材を育成できなくなる。

さらに，これからの日本の大学に必要なことは，専門家としての技術者が必要とする最新の技術に関する形式的知識を，社会で活躍している技術者をも対象に教授する社会人教育プログラムを準備し，実施することである［稲垣諭，長沼貴美，2012］。この社会人技術者教育プログラムでは，必要な知識を獲得した人材に対して，その知識の水準を保証する認定制度を確立し，技術者個人や，技術者を雇用する企業組織は雇用する技術者を，そのような教育プログラムへ送り込むことをし易くする制度の確立が重要になる。技術者は，自分の責任で，大学と教育プログラムを選択し，自分自身の自己啓発のために大学で学ばなければならない。

日本の大学は，社会が要求している人材を育成するため，高校から入学する学部学生の教育プログラムも見直す必要がある。大学で専門を学び，専門分野で技術者として認められる人材を育成するために，従来の日本の大学における教育プログラムよりも実践的な知識を教育する教育プログラムが必要になる。大学の学部を卒業した人材は，専門家として働くために必要な最低限の技術的知識を持つことを保証する必要がある[63)]。

日本社会において，大学教育が上述したような形態に変わると，企業における従来からの入社後の新入社員教育に相当する社内教育は不要になる。このことは，企業の人材育成に対する財政負担を軽減させ，短期的な日本企業の国際競争力を向上させる。しかし，長期的に企業側は，企業がどのような知識を獲得した人材を必要としているかの人材に対する要求項目を明示して求人活動を実施することが必要になる[64)]。このことは，大学が教育プログラムを設計するための参考になり，産業界と大学との連携による専門家教育プログラムの開発や改善が可能になる。

8. 結び：議論のまとめ

本小論においては，ソフトウェアの本質的な品質に影響を与える実現機能に対する要求が，

63） そのために，大学の技術者教育プログラムが社会的な要請に適合していることを第三者機関が審査し，認証することが必要になる。これは，米国等におけるアクレディテーション制度（accreditation system）に相当するものである［前田早苗，2003］。

64） 一般的には，職務記述書（job description）とよばれる文書である。

経済のグローバル化が進展する社会においては，開発プロジェクトの進行中，経時的に変化する傾向があることを示した。そしてそれは，従来の産業化社会のような動的な変化が少なく，安定した社会におけるソフトウェア開発とは異なる特徴であることを議論した。特に，開発の途中段階における新しい要求の出現や，開発の当初には必要とされていた機能に対する要求が，社会の変化に従って不必要とされる状況など，プロジェクトの成否を左右する不測の事態を発生させる原因となる。

そのような開発すべきソフトウェアに対する要求事項が動的に変化する傾向がある環境においては，従来型の，要求事項がほとんど経時変化しない，安定した工業化社会に適した開発方法であったウォーターフォール開発は，要求の変化に対する即座の対応が困難であると言う根本的な欠陥をもつ。逆に，要求事項が変化しない安定した社会においては，開発の効率に問題のあった段階的開発法は，新しい社会環境では，要求変化のリスクを吸収する方法として，合理的な方法であると考えられることを本小論において示した。この段階的開発法を現代的に翻訳しなおし，新しい名前を冠したものが「アジャイル開発」である。

そのような意味で，アジャイル開発は，経済のグローバル化が急速に進展する21世紀初頭の世界で，経時的にソフトウェア機能や品質に対する要求が変化する環境においては，開発組織や，実際の開発作業に関わる技術者達にとって，合理的な選択となる。このような理論的背景から，アジャイル開発は，利用者の要求に適合するソフトウェアを，適切な開発予算の制約の中で，さらに開発計画の時間的制約条件を満足するプロジェクトとしての実践を可能とする。これは，プロジェクトマネジメントにおいて，プロジェクトの重要な制約と言われる，品質（quality），経費（cost），そして納期（delivery）を同時に達成する方法であると言える。

ソフトウェア開発が置かれている状況は，他の一般の製品が置かれている状況と本質的に差異がない。すなわち，アジャイル開発は，ソフトウェア開発のみの方法論ではなく，一般の製品開発のための新しい方法論とも言える。マツダの電装系システムの開発者によれば，現在の複雑で高度に統合された自動車の電装系システム開発は，単に個々の部品を独立した部品として設計するだけでは十分でなく，複数の新しい部品の試作を作成して，それを統合した試作車を組み立て，実際に試験をして問題を見出し，それを改良するような設計に変更することが必要になっているとのことである[65)]。このやり方は，アジャイル開発に似ている。

現実は，我々が考えているよりも急速に変化しているようである。本小論で議論した内容が，実際に新製品開発の現場で，日々，問題解決に奮闘している多くの技術者達にも，何ら

65）これは，マツダの技術者であり，かつて留学生として著者の研究室において博士前期課程の学生として学び，モバイル無線 IP 技術の研究を行っていた，白雪峰氏との懇談で聴いた説明を簡略化して記述したものである。

かの参考，指針になることを期待する。

謝辞

本小論を執筆するのに際し，40年来の知人で，ソフトウェア業界の大先輩でもあり，ソフトウェア技術者協会設立発起人である岸田孝一氏より，「最近のアジャイル開発に関する議論はソフトウェア工学の本質から逸脱している。」との示唆に富んだご高察を拝聴し，参考とした。それが本小論作成の主たる動機となったことを記し，謝意を表する次第である。

参 考 文 献

Abrial U. & Glaesser, J-R.ed.（2010）. Rigorous Methods for Software Construction and Analysis. Springer.
Brooks F. P.（1987）. No silver bullet. IEEE Computer, vol. 20, No. 4.
Lehmann M. M.（1980）. Programs, lifecycle, and laws of software evolution. Proceedings of IEEE, vol. 68, issue9.
Norman S. et. al.（2010）. Work Breakdown Structures. Wiley.
Parnas D. L.（1972）. On the criteria to be used in decomposing systems into modules. Communications of ACM, vol. 15.
PROG 白書プロジェクト．（2014）．PROG 白書2015．学事出版.
Scott M. L.（2006）. Programming Language Pragmatics. Morgan Kaufmann.
Weber S.（2005）. The Success of Open Source. Harvard University Press.
スペンサー，M.（2011）. コンピテンシー・マネジメントの展開．生産性出版.
ダーウィン C.（1990）．種の起源．岩波文庫.
デューイ J.（1968）．哲学の改造．岩波文庫.
バーンスタイン W.（2006）．豊かさの誕生．日本経済新聞社.
ハヤカワ I.S.（1985）．思考と行動における言語．岩波書店.
ピーター J.L.（1970）. ピーターの法則．ダイヤモンド社.
ファーロング & カートメル，F.A.（2009）．若者と社会変容．大月書店.
ブルックス F.P.（1982）．ソフトウェア開発の神話．企画センター.
フロリダ R.（2008）．クリエィティブ資本論．ダイヤモンド社.
マーチン C.R.（2008）．アジャイルソフトウェア開発の奥義．SB クリエイティブ.
ミシェル W.（2017）．マシュマロ・テスト．早川書房.
リンカーン E.（2004）．それでも日本は変われない．日本評論社.
伊東光晴，根井雅弘．（1993）．シュンペーター．岩波新書.
稲垣　諭，長沼貴美．（2012）．社会人大学院生のススメ．オクムラ書店.
坂口利彦．（2016）．英国の EU 離脱と日本企業への影響．ロンドン：日本貿易振興機構.
松田　毅．（2015）．ライプニッツの“evolutio”概念について．第 9 回生物学基礎論研究会．北海道網走市.
石川　肇．（1981）．日本的品質管理．日科技連出版社.
前田早苗．（2003）．アメリカの大学基準成立史研究．東信堂.
大場　充．（1988）．ソフトウェアの開発技術．オーム社.
大場　充．（2014）．ソフトウェア技術者：プロの精神と職業倫理．日科技連.
大場　充．（2014）．組込みソフトウェア工学ハンドブック．日科技連出版社.
大場　充．（2017）．経時的に変化する製品の品質．経済科学研究第 20 巻第 2 号.
猪飼美恵子．（2006）．成人の発達と学習．学文社.
米国商務省．（1999）．デジタル・エコノミー．東洋経済新報社.

Abstract

Agile development: a natural consequence of world economy globalization

Mitsuru Ohba

This paper discusses that the agile software development is a natural consequence of expanding globalization of the world economy today. The root cause of an organizational need of the agile software development is continuously changing requirements which are results of changes happening all the time in our society, which reflect the cutting-edge competitions among organizations in the marketplace. The agile method is a way to manage risks of software developments by means of applying the divide-and-conquer policy for properly handling unpredictable incidents during software development projects. We show this in the paper using the PERT diagrams of the conventional waterfall and the agile developments, then compare the workload differences between them. The social and engineers' ethics issues are also discussed for reforming the Japanese society for the new age.