Vol.2015-IS-133 No.1
2015/9/3

アジャイル開発=コンカレントエンジニアリング×小集団活動＋プログラムマネジメント

児玉公信[†1]

企業の基幹情報システムをアジャイルに再構築した経験を基に，アジャイルプロセスを再定義する．アジャイル開発への理解を混乱させている原因は，コンカレントエンジニアリングのとらえ方にあることを仮説し，統制された大規模アジャイルプロセスを再定義する．そこでは日本の製造業の設計プロセスで行われる“すり合わせ”がカギになる．

Agile Software Development = Concurrent Engineering × Small Group Movement + Program Management

KIMINOBU KODAMA[†1]

1. はじめに

1.1 「アジャイル」に対する混乱

Yourdon[1)]と Brooks[2)]は宣言する．“ウォータフォール”は誤りであったと．では，いわゆるアジャイルソフトウエア開発は正しいのか．筆者は，答えはすでに出ていると考えているが，現実にはアジャイルソフトウエア開発は浸透していない．その原因は，プラクティスに目を奪われて「アジャイル」に関する本当の理解が損なわれていることだろう．アジャイル開発が少人数で行われる事例を見て，大規模システムの開発に向かないと判断する人もいる．

今回，足かけ 6 年にわたる大規模基幹システムをドメイン駆動設計とアジャイルプロセスで開発した経験を基に，アジャイル開発の本質を問い直し，大規模アジャイル開発への道を考える．

1.2 本報告の目的

大規模アジャイル開発を通して多くのことを学ぶことができた．ここから，大規模システムを俊敏かつしなやかに設計・施工するための方法を再吟味する．

本報告では，アジャイルソフトウエア開発手法は，コンカレントエンジニアリングの手法に他ならないことを述べ，システムエンジニアリングとしてそれらを統合するためのアーキテクチャとコミュニケーション手段としてのモデル，成果物の変更管理，アーキテクチャを維持するためのレビューとプログラムマネジメントの重要性を主張する．混乱の元はアジャイルの小集団活動の側面だけが強調されることである．これらを分離して，手法としての質を高めることを狙いとする．

1.2.1 本報告の構成

本報告の構成は，第 2 章で，製造業の設計手法のコンカレントエンジニアリング，第 3 章で小集団活動，第 4 章でプロジェクト管理の要点をまとめ，第 5 章で現状のアジャイル手法について述べた後，第 6 章で大規模基幹システムをアジャイル開発する際の留意点について述べ，第 7 章で全体をまとめる．

2. コンカレントエンジニアリング

まず，製造業におけるコンカレントエンジニアリングについて概説する．

2.1 製造業におけるコンカレントエンジニアリング

有泉 [3)]は，製造業におけるコンカレントエンジニアリング誕生の歴史を以下のように概観している．

2.1.1 コンカレントエンジニアリングの誕生

1980 年代の始め，米国の製造業は，生産設備の老朽化と陳腐化，製造現場の質と士気の低下，ユニオンによる改革に対する阻害，融通の利かない職階制度，製品開発部門の技術革新力の低下などにより販売力を大きく低下させた．一方，新興の日本は，高品質，低価格，ニーズにマッチした製品を開発し，市場を席巻するに至った．そこで，米国の先進的な企業はさまざまなビジネス改革に乗り出した [4)].

そのビジネス改革のうちの一つが，それまでの成功モデルであった大量生産主義（Mass-Production）をリーン生産主義に変えること，もう一つが，製品開発のプロセスをシーケンシャルエンジニアリング（Sequential Engineering, 順次処理型設計）からコンカレントエンジニアリング[a]（Concurrent Engineering）に変えることであった．

†1 (株)情報システム総研
Information Systems Institute, Ltd.

a サイマルテニアスエンジニアリング（Simultaneous Engineering）ということもある．

2.1.2 米国流のコンカレントエンジニアリング

“エンジニアリング”という語は，製品の機能設計と生産工程を準備する製造設計を指し，生産活動（production）を含めないのが一般的である．エンジニアリングの工程は，概念設計，詳細設計，機能解析，生産準備，生産設備準備などに分けられる[8]．1980 年代初頭の米国では，これらの工程ごとに明確な職務規程があり，前工程から後工程への配慮がない，後工程は図面が来るまで何も分からない従属的な位置づけになる，前工程へのフィードバックがないため設計が改善されないなどから，独りよがりの設計，近視眼的な設計が横行していた．工程間に“壁”があり，全体的に非協調的で，それを補うために高品質のマニュアルが存在した．こうした結果として製品品質の低下，原価の増加，士気の低下のような弊害が目立つようになっていた．

このような弊害を解消するために，製品の設計に関わる工程と生産準備に関わる工程を同時並行的に進めることで，生産工程に配慮した設計を実現しようとした．これがコンカレントエンジニアリングである．

「コンカレントエンジニアリング（以下 CE と略す）」という語が公式に使われたのは，米国の Institute of Defense Analysis による 1988 年末の報告書，“The role of concurrent engineering in weapons system acquisition”，コード名 R-338 のようである[b]. 続いて 1989 年に，Nevins ら[5]が“Concurrent Design”という標題の本を出版する．これが CE の原典とされる．

図 1 は，コンカレントエンジニアリングのフローである．

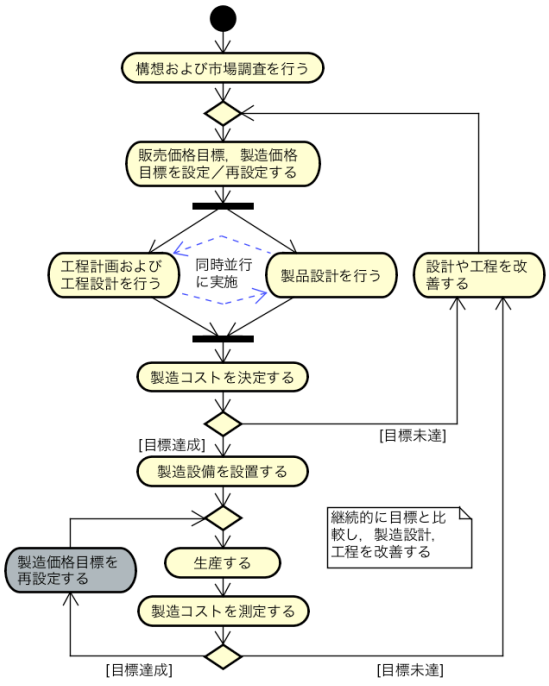

図 1 コンカレントエンジニアリングのフロー[3]

b 文献 6)では 1986 年の R-383 文書とされているが，これは誤りのようである．なお R-338 に，後述する Takeuchi & Nonaka9)への参照はない．

有泉が Nevins らの記述を整理し，筆者が UML の妥当なアクティビティ図に書き直したものである．この図で注目すべき点は，エンジニアリングの結果が目標に到達しなかった場合に，製品設計や工程設計の改善を行うだけでなく，生産段階に入った後も，目標未達の場合も同様に設計の改善を行うとしている点である．ただ，それだけでは設計の改善は終わってしまうので，生産段階で目標を徐々に上げていく必要がある．この分は筆者が書き足した（図ではグレーにしてある）．さらに，運用を含めた大きな設計改善サイクルがある．

実は，1950 年代でも，cross-functional team，製品設計における工程設計の重要性，time-to-market などは認識されていた．これらが CE の文脈で語られるようになったのは，Nonaka らのグループによる日本の製造業の分析が知られてからである[7]．

2.1.3 日本における CE

一方，日本のエンジニアリングの現場では，既に 1980 年代の初頭から CE に相当することを行っていた．有泉[3]から引用する（引用部分をカギ括弧で囲って示す）．

「お互いに，チーム全体の進捗状況や設計内容の把握をまず心がけていた．そしてときには，製図板の前に数人が集まり，自然発生的なミニ設計検討会が始まった．このようなチームプレーが当たり前のように行われていたのである」．また，「生産技術部署や製造部署，さらに検査やサービス部署などともたえず連絡を取り，意見を求めなら設計が行われていた」．

有泉は，こうした業務スタイルが実際に稼働した理由として，「自己完結できるだけの技術力」が現場に備わっていたこと，日本人の文化である協調を良しとする“和”の心が働いたこと，その当時，TQC（Total Quality Control，全社品質管理運動）が根付いていたことを挙げている．TQC は，1950 年代から 1970 年代にかけて日本の製造業の急激な躍進の原動力であった．

TQC では，真の品質とは何かが問われる．「その中では，設計部署のスタッフに対して，“マーケットイン”や“後工程への品質”などの考え」が要求された．こうして設計者たちは，自ずと「後工程を十分考慮した設計」や「絶えず周りとの連携をとりながら行う設計」を実践していた．「しかし，このプロセスには体系化されたマニュアルらしきものが存在していなかった．何らかの基準がないのにもかかわらず進化を続け継承されていた．恐らくそれは TQC という大きな“思想？”（ママ）がその後ろにあったからであろう」．

2.1.4 The New New Product Development Game

Takeuchi and Nonaka[9]（1986）は，製造業の新しい開発スタイルに注目して，主に日本の製造業を中心に 1970 年代の終わりに新製品を販売した事例について，CEO や現場のエンジニアへのインタビューを行い，その分析結果を示した．

これは，後述するソフトウエア開発の領域におけるアジャイルプロセスを導入するきっかけとなった重要な論文であるので，少し詳しく説明する．

成功した新製品の開発プロセスは，次に示す共通の特性を持っている．

- **内蔵された不安定性**（Built-in instability）：トップは明確な製品コンセプトを示さず，チームをあえて不安定（プリカオス）状態からスタートさせる
- **プロジェクトチームが自己組織化する**（Self-organizing project teams）：チームはまるで起業するかのように作られ，自律的に組織化し，卓越化していく
- **開発フェーズをオーバラップさせる**（Overlapping development phases）：複数の開発工程が重なるようにして設計を進める．従来のリレー競争（順次処理）型に対して，前後の工程を重ねる刺身型，さらに多くの工程を重ねるラグビー型がある
- **多重学習**（“Multilearning”）：組織階層に伴う管理視点の違いに基づく学びと，異なる業務機能に触れることによる学びとがある
- **絶妙な管理**（Subtle control）：押さえつける管理ではなく，自己管理，相互管理，“愛”を持った管理を行う
- **学習の組織的共有**（Organizational transfer of learning）：得られた知識をプロジェクトメンバ間だけでなく，会社レベルで共有しようとする

「開発フェーズをオーバラップさせる」特性はまさに CE を意味するが，前述のとおり，この時点ではその語は使われていない．残りの特性は，次章で述べる小集団活動の特性に対応している．

2.1.5 その後の日本の状況

日本経済は，1980 年代後半から急激に膨張し，製品開発競争が激しくなると，革新的な開発組織において，TQC は画一的な発想しか許さないもの，開発期間の短縮を妨げるもの，過去のデータのみを重視した“科学的”で形式を重視した品質管理手法ととらえられるようになる．さらに，賞取りビジネス化する推進組織の姿勢などから，TQC 活動は疎まれるようになっていった．

「バブル期の異常とも言える製品開発競争は，極端な設計の分業化を生んだ」．また，バブル崩壊に伴う技術者の「採用減は，設計者間の世代断絶や技術継承の断絶を招いた．優秀な理工系新卒の減少も技術継承の断絶に拍車をかけた」．こうして，「後工程を十分考慮した設計」や「絶えず周りとの連携をとりながら行う設計」が途絶えてしまう．

そこにちょうど，米国から CE のアイデアがもたらされる．自らの危機を感じていた製造業は，かつては自らのものであったこのアイデアを積極的に取り入れようとしたc．1990 年代初頭のことである．このときの CE には，すでに CAD や PDM（Product Data Management）などのツールが備わっていた．PDM は CAD などで書いた設計図面間の改版管理を行うツールである．CE には，変更追跡と版管理の機能が必須である．ツールについては 2.2.2 でも触れる．

2.2 CE の実現

2.2.1 実現の形態

佐田 [8]によると，CE の実現形態は次のように分類される．

- **フロントローディング**d：概念設計の早期段階で次工程以降に予備的情報を流して，各工程で考えられる準備を始めておく
- **アーリーソーシング**：次工程が開始できそうな程度まで検討が進んだ状態で，早めに設計情報を渡す
- **パラレルアプローチ**：前後の工程間でフィードバックを繰り返しながら，それぞれの設計を並行的に進める
- **期間短縮**：システム化を進めることで，個々の工程にかかる時間を短縮する
- **グループ間共同作業**：多くのあるいはすべての工程間の情報交流を促進する．場合によっては全工程から代表者を出してグループを結成する

これらを Takeuchi & Nonaka[9]に照らすと，パラレルアプローチは刺身型に，グループ間共同作業はラグビー型に対応すると言える．

2.2.2 情報技術と CE

藤本 [10]は，実際にフロントローディングを行うには，三次元 CAD による構造設計，CAE による機能検証や製造可能性検証など，設計開発を支援する情報技術は不可欠であると述べている．しかし，日本の企業においては，こうした情報技術の導入が進んでいないにもかかわらず，これを「うまく行う組織能力が高い傾向にあった」．

「日本企業がフロントローディングを欧米企業以上にうまく推進できていたのは，三次元 CAD をより早期から使用していたからではない．組織的に自動車企業と部品企業の関連技術者が共同で効果的に問題解決に取り組む統合型の組織能力こそがより重要だ．IT は必要条件ではあっても，十分条件ではない」．CE を実現するためには，技術者の高度な設計能力だけでなく，統合的組織能力，有泉のいう“和”の心も必要とされる．

3. 小集団活動

ここでは，小集団活動とは，「①対面接触する少人数でなり立ち，②成員間で相互依存関係が見られ，③一定期間存続する集団」[11]が行うさまざまな活動をいう．中でも，1962 年に発足した QC サークルは小集団活動の主体である．以来，日本の製造現場において，生産性の低さや製品不良の

c この状況は，1985 年ごろの MIT などの日本の製造業に関する研究によって，リーン生産方式（Lean Production System）が定義されたことによって，のちにトヨタが，自らの業務スタイルをシステム（Toyota Production System）としてあらためて認識したことと並行している．

d 藤本ら 10)は，「問題解決の“前倒し”を開発初期に集中する」ことで，「できるだけ早期に問題発見・問題解決の質と量を増やす」と述べている．

発生する状況に分析手法を駆使して原因を究明し，生産性や不良発生率を改善するなどのボトムアップな QC 活動を担ってきた．生産現場に限られていた QC 活動は，やがて設計部門，事務部門も含めた全社活動とすることで，真の品質改善が可能となるとする TQC 活動へと発展していった．これは，2.1.3 で述べたように，日本における実質的な CE 実践の母体となった．さらに製造業にとどまらず，サービス業や医療の現場でも取組みの事例が増えている [12]．

3.1 知的創造システムとしての小集団活動

近年，QC サークルは，企業を取り巻く環境の変化に合わせて，柔軟性が求められるようになり，“進化した QC サークル活動（e-QCC）”が提唱されている．ここでは①“個”の価値を高め感動を共有する活動，②業務一体の活動の中で自己実現を図る活動，③形式にとらわれない幅広い部門で活用される活動を標榜している．

また，現場力の向上や知識創造のための，構成員間の相互啓発も強調されている．とくに，PDCA を回すことによる得られる小集団内部での知識の創造と蓄積は，いまや重要な経営資源と見なされるようになっている [12]．

3.2 小集団活動のプロセス

上田 [11]を基に，小集団活動の基本的なプロセスをまとめる．これらが，グループの自発的な知識創造活動につながっていることがわかる．

- **グループの編成**：活動のねらいをはっきりさせて，それにふさわしい人数を集める．メンバの交替も可能な開かれたグループにしておく
- **リーダを選ぶ**：リーダのテーマに関する体験が，グループの活動水準を決める．リーダの交替が可能となるように育成に心がける
- **職制による支援**：職制はグループとの関係を明確にし，信頼に足る支援者としてグループの自発的活動を支える
- **テーマの選定**：できるだけ具体的で自分たちの問題，グループで解決でき長期にわたらずに達成できそうな問題をとりあげる
- **討議**：2，3 分でも時間を決めて，できるだけ多くのミーティングを設定する．のびのびと発言し，語り合える雰囲気を作る．検討のまとめはみんなで確認する．統計分析など（「QC 七つ道具」「新 QC 七つ道具」が有名）で客観的な検討を行って問題の発見，対策の決定に役立てる．グループワークで知恵を出し合って問題を発見する
- **問題解決**：問題を決める→解決の準備（計画を立て，役割，評価基準を決める）→解決策の実施とデータの収集→結果を評価基準に照らして検討する
- **活動のチェック**：進捗をチェックし，方向がずれているときはみんなで討議し対策を施す
- **報告と見直し**：期間内の活動結果を報告する．社内外で発表するのもよい．活動全体を評価し見直す．記録を残して，次の活動に活かす
- **活動水準を高める**：より自律的で，洗練され，魅力的で，挑戦的な活動になるように，活動水準を高めるよう努力する

3.3 小集団活動を活性化する

小集団活動は，慣れてくると惰性で“だれて”くる．さらに士気が低下し，パフォーマンスが低下する．小集団活動を，組織の中で安定した活動とし，参加を促し，メンバにとっても魅力的な活動となるように，活動全体を活性化する工夫が必要である．

まず，推進組織を作って，活動のシンボルマークを決めること，グループがテーマ選択に困ったときのために，テーマの候補をためておく（テーマバンキング）ことや，活動マニュアルを作成することなどを行う．

ほかにも，たとえば，バッチ，帽子，T シャツなどの小道具を用いることもグループの一体化の促進に役立つことがある．活動発表のための機関誌，サークル誌の発行も活性化に役立つ．グループ間の交流や研究会活動を通して，活動のコツ，楽しさ，成果の出し方を学ぶことも意味がある．

4. プログラムマネジメント

基幹システムは企業の経営方針と密接に関わる．その再構築は，大規模なビジネスイノベーションを意味する．このプログラムを遂行するために，プロジェクトを多数のプロジェクトに分割し，それらを統合的にマネジメントすることになるが，その全体と細部に及ぶ不確実性によって，マネジメントの難度はますます高くなる．

4.1 プログラムマネジメント

清水 [13]は，プログラムを「組織にとって重要な目的を達成するために複数のプロジェクトを有機的に組み合わせた活動」と定義している．さらに吉田 [14]は，社会的分業が進んだ時代にあって「成熟社会の難題を取り扱う活動」ととらえている．プログラムマネジメントでは，個々のプロジェクトのミッションを達成するだけでなく，プログラム全体がシステムとして十全に機能することが要請される．そこには，プロジェクトマネジメントとは異なる統合的マネジメントの活動が必要である [14]という．

- **プロファイリング**：プログラムの初期段階で，施主のミッションを理解し，作業レベルに詳細化する
- **プログラム戦略**：外部環境，とくに他社との競争に勝つための実現性を高める
- **アーキテクチャ**：プロジェクト分割とその間の関係を規定する
- **プラットフォーム**：協調作業の場とプロトコルを設計し運用する
- **ライフサイクル**：プログラムの価値を最大に保つように，日々発生する状況変化やインシデントに対応する
- **価値指標**：プログラムの価値指標を定め，計測と評価を

行う

4.2 プロジェクトマネジメント

プロジェクトとは「特定使命を受けて」,「特定期間内に実施する将来に向けた価値創造事業」である[15]. プロジェクトマネジメントとは,「有期のチームを編成し,プロジェクトマネジメントの専門職能を駆使してプロジェクトを公正な手段で効率的･効果的に遂行し,確実な成果を獲得する実践的能力をプロジェクトに適用することである」.

清水[13]はプロジェクトマネジメントに3種類のマネジメント活動があるという. すなわち,「もの」「こと」「場」を作ることである.

- **成果物システムズ**：プロジェクトの目標であるシステムを構想し，設計し，施工する最終製品および中間生成物の体系を示す
- **プロジェクト遂行**：製品および成果物を生成する作業を計画し，手順化し，その作業を実施し，モニタし，コントロールする
- **現場アドミニストレーション**：人，資源，資材，基盤を調達し，使用可能にする

5. アジャイルソフトウエア開発

アジャイルソフトウエア開発（以下，単に「アジャイル開発」という）とは，端的に言えば，ソフトウエアの開発を CE で行うことである. ただし，そのコンカレンシーはチームメンバの多能工化で吸収できるほどの規模であったために，初期のアジャイル手法の提唱者たちは，上述したような CE を意識していないと思われる.

やや後発だった Poppendiek たち[20]のリーンソフトウエア開発では，コンカレントソフトウエア開発（Concurrent Software Development）eを強く勧めており，次のように言う.「コンカレント開発によって広さ優先の手法がとれるようになり，費用のかかる重大な問題を，手遅れになる前に発見できる」.

5.1 CE としてのアジャイル

ソフトウエアの開発に限らず，シーケンシャルエンジニアリングの弊害については，いまさら述べる必要もないだろう[2]. 不確実で複雑な要求に対して完全な設計成果物を次プロセスに渡すことは不可能である. 結果として，ゴミを次プロセスの壁の向こうに投げることになり，最終的に出来上がるソフトウエアは，施主の最初の要求からかけ離れたものになる. 開発途中で要求も変わる. よいソフトウエアを提供しようと思えば，次プロセスからの逆流や変更を受け入れて，各プロセスが協調することで，短期間で施主にとってよい設計できるようにする. これが CE の本質であった. ソフトウエア開発で CE ができれば，これまでの多くの問題が解決する.

5.2 アジャイルの手法

アジャイル開発のアイデアが具体的な形になったのは，1990 年代の後半以降である. さまざまな“手法”があり[16]，スクラム[17][18]，XP（eXtreme Programming）[19]，リーンソフトウエア開発（Lean Software Development）[20]などの名前が付いている. ともに，短期間開発のために，スコープを小さくとって（軽量），比較的短時間（タイムボックス）で設計と施工を繰り返す（イテレーション）手法をとる. この手法は, 1980 年代後半にすでに提唱されていた RAD（Rapid Application Development）[21]からの多くを引き継いでいる. そしてその源流は 1980 年代前半の Rapid Prototyping にある（図 2 参照）.

スクラムは, Ken Schwaber と Jeff Sutherland が Takeuch & Nonaka[9]のラグビー型の手法に触発されて整備した手法である[17][18]. スクラムと RAD の本質的な違いはない. 違いはスクラムのほうが方法論としての制度化が進んでいる点にあり，プロジェクトとチームのマネジメントに関わる制度が定められている.

XP は，Kent Beck が提唱した手法で，要求の変更を受け入れる（Embrace change）ための 5 つの価値観と多くの技術的プラクティスからなる.

5.3 アジャイルの制度とプラクティス

実際のアジャイル開発の現場では，スクラムの制度の下で，XP のプラクティスを必要に応じて取り入れることが多い[17]. スクラムを中心にアジャイル開発のプラクティスを以下にまとめる.

5.3.1 スクラムチーム

プロダクトオーナ，開発チーム，スクラムマスタからなる. プロダクトオーナは開発チームにプロダクトバックログ（要求リスト）を提示する. 開発チームは多能工の集まりで，分析，設計，実装，テストなどを担当する. スクラムマスタは，チームが十全に機能できるような場を維持する.

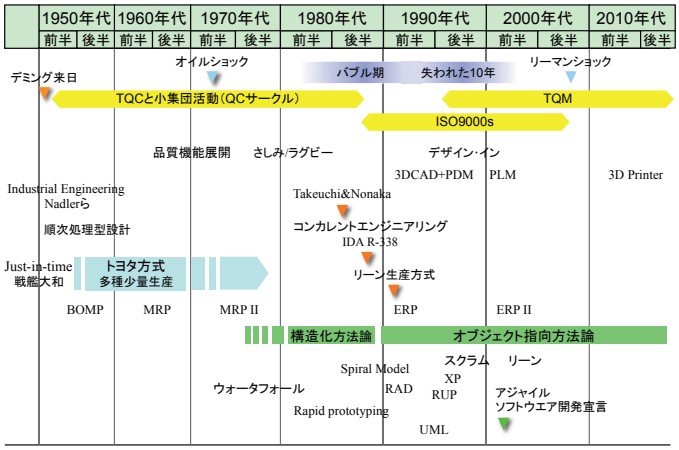

図 2 アジャイルソフトウエア開発の系譜

e ほかに，TQC への言及もあり，製造業についての理解がある.

5.3.2 プラクティス

アジャイルのプラクティスについては，さまざまな媒体で述べられている [16)17)18)19)20)]ので，ここでは簡単に触れるだけとする．

スクラムではタイムボックスをスプリントと呼ぶ．開発プロジェクトに対する要求をプロダクトバックログと呼び，スプリントで取り組む要求をプロダクトバックログからチームの開発容量（ベロシティ）に収まるように，メンバの協議（スプリント計画）によって選択する．スプリントの成果物をインクリメント（増分）と呼び，それを関係者にデモンストレーションすることをスプリントレビューと呼ぶ．スプリントの終わりに振り返りを行って，次のスプリントに向けての改善点を整理する．

バックログはユーザストーリ（ユーザ要求）のリストで表現される．計画ゲームはプラニングポーカーとも呼ばれ，メンバが集まって相対的な開発量を見積もる行為である．デイリースクラムは一日の始めにその日やることをメンバと共有する 15 分程度のミーティングで，立って行われることが多い．タスクカンバンは，壁やボードを todo，doing，done の領域に分けて，付箋に記入されたタスクを貼ることで，日々の進捗をメンバが共有することをいう．タスクは，ユーザストーリを実現するために必要な作業を，1 日程度の作業量に分解したものである．バーンダウンチャートは，タスクの消化速度を見るための図である．

6. 大規模アジャイルに向けての総合的検討

大規模アジャイルの事例を紹介し，大規模アジャイルのためのプロセスについて検討する．

6.1 プログラムの事例

大規模な基幹システムの再構築プログラムにおいて，筆者は全体のアーキテクトとして参加した．このプログラムは，初期段階からコンカレントに進められた．まず，この実践を振り返る．

6.1.1 体制

プログラムの体制は，時間の経過で大きく変わったが，おおむね，施主（プログラムオーナ），プログラムリーダ，PMO，アーキテクトチーム，業務設計チーム，実装チーム，テストチームからなった．上記アーキテクトチームまでが本部機能であり，業務設計チーム以下は現業機能である．実装チームには専属のアーキテクトを置くと同時に，アーキテクトチームを兼務させることで，アーキテクトの意思が実装チームに伝わるようにした．

6.1.2 マイルストンの設定

ISO/IEC 15288（JIS X 0170）に準拠し，利害関係者要求定義（業務構想書，Concept of Operation），要求分析（業務設計），方式設計（論理アーキテクチャ），機能設計（概念モデルとユースケース記述），詳細設計（実装モデルとシーケンス図），実装（コード，単体テスト），結合テスト，受入テスト，移行（環境設定，トレーニング，切替リハーサル）のプロセスからなる．プロセスどうしはかなりの部分をオーバラップさせた．

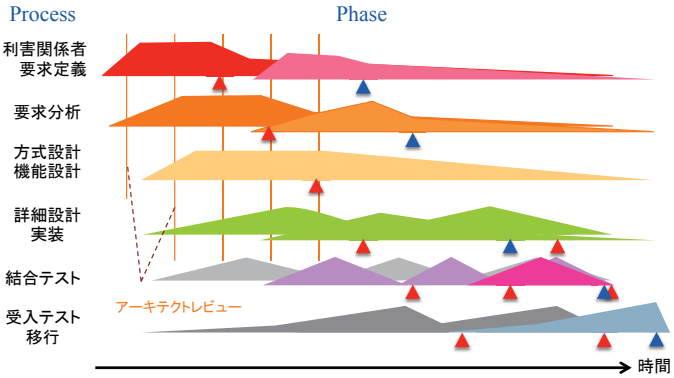

図 3 情報システムのコンカレントエンジニアリング

図 3 はそれを模式的に表現した図である．文献 [24)25)]などに倣って，縦軸は各プロセスfのワークロードのイメージである．一つの作業分野に異なる色の山は，異なるプロジェクトを示し，プロジェクト内だけでなく，プロジェクト間でもコンカレントに進むことを表す．CE なので，RUP のようにフェーズを明確に設定せず，代わりにマイルストンを設けた．図 3 では小さな▲で表している．

いわゆるアジャイルのプラクティスを適用したプロセスは詳細設計と実装であるが，それは初期段階にとどまる．後に述べるが，実装段階から参加したベンダは，実質的にウォータフォールで実施した．

6.1.3 フロントローディングと変更管理

基幹システムの再構築は業務改革を伴う．その理想のシステムをあらかじめ完全に設計することはできないため，各プロセスでは仮説的に設計を行い，段階的にプロトタイプを作って設計を確定するようにした．

とくに業務上重要と思われる機能については早期に先行して機能設計を行い，実装して仕様の確認，動作の検証を行った．このために採用したベンダは高度に抽象的なドメインモデルを理解し，実装能力も高かったので，アーキテクトとは密に連携して，先行実装からのフィードバックによって設計全体を揺さぶっていった．設計変更のための議論は主にモデル図に基づいて行った．

6.1.4 新規ベンダはウォータフォール

重要機能についての検証が終わり，いよいよ詳細設計と施工プロセスを拡大するに当たって新しい開発ベンダを採用した．先行実装の段階から参加していたベンダはモデルを理解し，コンカレントなプロセス運営を理解し，アーキテクチャを共有していた．しかし，新たに参加したベンダは，業務内容，プログラムの目的，モデルを含めたドキュメント，アーキテクチャ，コンカレントなプロセスなどの

f RUP (Rational Unified Process)では，プロセスを Activity と呼び，その大きなくくりを Discipline と呼ぶ．

キャッチアップに相当時間がかかった．

新規ベンダは，ウォータフォールで実装を行い，アーキテクトレビューが思うようにできなかった．ベンダは準委任契約であり，製造物に対する責任はなかったが，請負契約と同様の納品主義をとり，設計文書はプロジェクトで共有されなかった．上流で要件が変動する中でのウォータフォールgは容易ではなかったと想像されるが，ドメイン駆動，モデル主導，コンカレントプロセスの趣旨は徹底できなかった．

6.1.5 多重環境の維持

正式稼働が間近になると，開発環境，結合テスト環境，受入環境，トレーニング環境，本番環境と，少なくとも 5 つの環境が，それぞれに異なるバージョンのプログラムとマスタデータを持つようになる．不具合が発見され，対応されても，他の環境に移行するタイミングをはからなければならない．これらの環境はクラウド上に構築されたが，そのコントロールを適正に保つことは非常に難しい．

6.2 プログラムマネジメントの重要性

大規模なプログラムを支えたカギは，結果的にアーキテクトミーティングであった．これは 4.1 で述べたプログラムマネジメントにおける統合マネジメントの活動，「アーキテクチャ」と「プラットフォーム」のマネジメントに対応する．また，全プロセスがコンカレントに進むことにより，動的に変化する「プロファイリング」のマネジメントに留意した．

6.2.1 アーキテクチャマネジメント

プログラムを適正なプロジェクトに分割するカギは，システムのアーキテクチャ，すなわち適正なドメインのくくり出しと，その間のインタフェース定義である．各ドメインは，それぞれの関心対象のライフサイクルを管理するようにモデル化され，変更容易性を保つように実装される．ドメイン間のインタフェースは，一方向でセマンティクスを強要しない疎な参照関係を保つように設計され，高い信頼性を実現するように実装される [27)28)]．

このように独立性の高いドメインを設定することで，要求分析，機能設計，実装がコンカレントに実施できる．ただし，そのようなドメインが一挙に定まるわけではない．一定の枠組みから始めて，機能配置を調整していく過程で，ドメインモデルは変化する．アーキテクチャが安定すれば，以降はこれを維持するようにプロジェクトを統制する．

6.2.2 プラットフォームマネジメント

コンカレントにプロセスを進めるには，成果物の共有，変更の共有，知識の共有が必須であり，共有の場「プラットフォーム」が必要である．フォーマルには，アーキテクトミーティングとアーキテクトレビューが設定され，実務的には Redmine などのチケット管理ツールが用いられる．しかし，日本ではそれ以上に，統合的組織能力，言い換えると“和”が必要とされる．

前述のとおり，日本では 1950-70 年代にすでに，それと意識されないまま，コンカレントなプロセスが実施されていたし，小集団活動によって，現場のプラットフォームは確立していた．

アジャイルのプラクティスの中には，小集団活動の維持や向上に関わるものが多い．たとえば，朝会（デイリースクラム），振り返り（KPT），リズムなどである．ほかにも，技術的なプラクティスであるが，計画ゲーム（プラニングポーカー），ソースの共同所有，コーディング基準，ペアプログラミング，課題の共有としてのタスクカンバンなども小集団活動を強化する側面を持つ．

アジャイルのプラクティスもプラットフォームマネジメントの観点で整理しなおすとよい．

6.2.3 プロファイリングの動的変更と変更同期

多くのプロセスがコンカレントに進むということは，さまざまな条件に起因する変更がダイナミックに発生するということでもある．変更が新たな変更を生む連鎖反応もある．適時に変更に対応することに加えて，変更の連鎖を断つようなアーキテクチャを考える必要がある．また，工事で一度掘り返して埋めた道路を，すぐにまた掘り返すようなことがないように，変更実施に対する統制も必要である．実際，製造業においては，設計変更（Engineering Change）自体と設計変更実施（生産工程への適用）とは厳密に区分される．

変更管理の主体はアーキテクトチームのようなアーキテクトマネジメントを行うチームがよい．ライブラリアンのような役割を設け，アーキテクトチームの統制の下，すべての成果物を一元的に管理する（プロアイリング）．これは，製造業における PDM に相当する．

Poppendiek たち [20)]は，アジャイル開発のコンカレント性について意識していたが，それを支える変更管理については触れていない．

6.2.4 プログラムマネジメントの主体

大規模アジャイルのプラクティスに，Scrum of scrums ミーティングと呼ばれるものがある．これは，Scrum チームの代表者が集まって行うデイリースクラムのミーティングとされる [30)]．しかし，上で見たようにボトムアップでのプログラムマネジメントには限界がある．実際，Paasivaara ら [31)]は，その困難さを訴えている．

大規模プログラムをコンカレントに行うためには，短時間のミーティングを積み上げるのではなく，上で述べたような統合マネジメントの能動的な活動が必須である．その主体は，Program Management Office が負うべきである．PMO は，アーキテクトや設計チーム，実装チームが対話するプラットフォームを維持する．このようなプログラムを“大規模 CE プロセス”と呼ぶことにする．

g ウォータ・スクラム・フォール 26)などと揶揄されるプロセス．

現状のアジャイル開発は，大規模 CE プロセスの軽量版と言えないだろうか．経験的に，軽量版でうまく行くプロジェクトは多いだろう．一方，軽量版を単に拡大しても大規模プログラムに対応できるとは思えない．CE を効果的に行うプロセスが必要となる．

6.3 大規模 CE プロセスの定義

ここで，標題に掲げた「アジャイル開発=コンカレントエンジニアリング×小集団活動＋プログラムマネジメント」の意味を述べる．

アジャイル開発と呼ばれた開発手法の本質は，CE である．そのプロセスは ISO/IEC 15288 のプロセス [22]であり，INCOSE のプロセス [23]である．プロセスの実行主体は小集団である．小集団はそのパフォーマンスを持続的に最大化するように，学習的にその活動を定める．その活動が全体のプロセスを制約する．この全体のスケールと並行性を律するようにプログラムマネジメントを行う．

7. おわりに

アジャイル開発の源流は，日本の製造業の，1950 年代から続く顧客価値を高める TQC にあった．それは，製造現場の小集団の地道な改善活動の積み重ねであり，その根底には“和”の心があった．

企業戦略に関わる現代の情報システム開発では，設計から施工，移行，運用までをコンカレントに行いながら，要求や仕様をすり合わせる必要がある．アジャイル開発の本質は，制度やプラクティスにあるのではなく，このすり合わせにある．大規模な情報システムの開発プログラムに，意図的にすり合わせのプロセスを組み込みたい．

参考文献

1) Yourdon, E. “Yourdon Report,” http://www.yourdonreport.com/index.php/2007/05/29/icse-peoplware-panel-session/（2015/7/24）
2) Brooks Jr., F. P., “The Mythical man-month: essays on software engineering,” Anniversary edition, Addison-Wesley (1995), 滝沢 徹ほか（訳），「人月の神話−狼人間を打つ銀の弾はない」原著発行 20 周年記念増訂版，アジソン・ウエスレイ・パブリッシャーズ・ジャパン，pp. 256-258（1996）
3) 有泉 徹,「コンカレントエンジニアリングによる設計の改革術」，日刊工業新聞社（2000）
4) Hammer, M. and Champy, J., “Reengineering the Corporation: A Manifesto for Business Revolution,” HarperBusiness (2006), 野中郁次郎（監訳），「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」，日経（1993）
5) Nevins, J. L. and Whitney, D. E., “Concurrent Design of Products and Processes: A Strategy for the Next Generation in Manufacturing,” Mcgraw-Hill (1989)
6) 斉藤 実,「実践コンカレントエンジニアリング」，工業調査会（1993）
7) Loch, C. and Terwiesch, C., “Product Development and Concurrent Engineering,” in “Innovation in Competitive Manufacturing,” Springer, pp.263-273 (2000)
8) 佐田登志夫,「コンカレント エンジニアリングについて」，小特集：コンカレント エンジニアリング，電気学会誌，Vol.113, pp.178-182（1993）
9) Takeuchi, H. and Nonaka, I., “The new new product development game,” Harvard Business Review (1986)
10) 藤本隆宏ほか著，田村明比古（訳），「製品開発力［増補版］」，ダイヤモンド社（2009）
11) 上田利男，「小集団活動の手引き」，日本経済新聞社（1980）
12) 長田 洋，「小集団活動自己評価方法」，日本規格協会（2010）
13) 清水基夫，「実践 プロジェクト＆プログラムマネジメント」，JMAM（2010）
14) 吉田邦夫hほか，「実践 プログラムマネジメント」，日刊工業新聞社（2014）
15) Ohara, S. (Representative Author), “A Guidebook of Project and Program Management for Enterprise Innovation,” PMAJ (2005)
16) Boehm, B. and Turner, R., “Balancing Agility and Discipline,” Pearson Education (2004) 河野正幸ほか（監訳），「アジャイルと規律」，日経 BP（2004）
17) 平鍋健児，野中郁次郎，「アジャイル開発とスクラム」，翔泳社（2013）
18) Schwaber, K. and Beedle, M., “Agile Software Development with Scrum,” Prentice Hall (2002), 長瀬嘉秀ほか（監修）「アジャイルソフトウエア開発スクラム」，ピアソンエデュケーション（2003）
19) Beck, K. and Cynthia Andres, C.著，角 征典（訳），「エクストリームプログラミング」，オーム社（2015）
20) Poppendieck, M. and Poppendieck, T.著，平鍋健児ほか（訳），「リーンソフトウエア開発～アジャイル開発を実践する 22 の方法～」，日経 BP（2004）
21) Martin, J. “Rapid Application Development,” Macmillan (1991), 芦沢真佐子ほか（訳），「ラピッド・アプリケーション・ディベロプメント」，リックテレコム（1994）
22) ISO/IEC 15288-2008 “System life cycle processes”
23) INCOSE, “Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities Ed.4,” Wiley（2015）
24) Jacobson, I., et al, “The Unified Software Development Process,” Addison (1999), 藤井 拓（監訳），「UML による統一ソフトウエア開発プロセス」，翔泳社（2000）
25) Ambler, S. et al, “The Enterprise Unified Process: Extending the Rational Unified Process,” Pearson Education (2005)
26) Info Q, “Have the Pragmatists Won? Water-Scrum-Fall Is the Norm,” http://www.infoq.com/news/2011/12/water-scrum-fall-is-the-norm (2015.08.07)
27) 児玉公信,「計画･実行システムの一般モデル ―生産管理システムと金融業務システムの共通性―」，情報システムと社会環境研究会，Vol.2009-IS-109, No.4（2009）
28) 児玉公信,「企業情報システムのための早期アーキテクティングの一方法」，情報システムと社会環境研究会，Vol.2012-IS-120, No.3（2012）
29) Schiei, J., “Enterprise-Scale Agile Software Development,” CRC Press (2009)
30) Faria, L. “Scrum of Scrums: Running Agile on Large Projects,” https://www.scrumalliance.org/community/articles/2013/june/scrum-of-scrums-running-agile-on-large-projects (2013)
31) Paasivaara, M. et al, “Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work? ,” Proc. of the ACM-IEEE international symposium on Empirical software engineering and measurement, pp.235-238 (2012)

h 吉は土に口