特集

トヨタ生産方式の導入によるソフトウェア開発プロセスの革新

Innovation in Software Development Process by Introducting Toyota Production System

あらまし

富士通プライムソフトテクノロジ（PST，現在，富士通ソフトウェアテクノロジーズ）では2003年からトヨタ生産方式（TPS）の導入による生産性向上活動に取り組んでいる。TPSの基本コンセプトをITのソフトウェア開発分野で実践する手段としてアジャイル開発プロセスとストア管理手法を導入した。この二つの手法には，TPSの基本コンセプトである「ムダの徹底排除」，「平準化」，「自働化」，「目で見る管理」が具体的な実践手法（プラクティス）として組み込まれている。PSTでは，アジャイル開発プロセスをソフトウェア開発プロセスに，ストア管理手法をサポート保守プロセスに導入した。その結果，両プロセスの改善とともに組織風土においても大きな改善効果が得られた。

本稿では，まず，アジャイル開発プロセスで実践されているTPSのコンセプトを紹介するとともに，PSTでの実践効果を紹介する。つぎに，ストア管理手法における「平準化」の実践手法を紹介するとともにPSTでの実践効果を紹介する。

Abstract

Fujitsu Software Technologies [formerly Fujitsu Prime Software Technologies (PST)] has been conducting activities to improve productivity modeled on the Toyota Production System (TPS) since 2003. An agile development process and a store management method were introduced to implement the basic concepts of TPS in the IT software field. We included the basic concepts of TPS [the elimination of *Muda* (waste), *Heijunka* (leveled production), *Jidoka* (automatic detection of abnormal conditions)], and Visual Management in said agile development process and store management method as practical techniques. PST introduced this agile development process to its software development process and the store management method to support its maintenance process. As a result, PST achieved significant improvements in both processes and in its organizational climate. This paper introduces the TPS concepts employed in the agile development process, describes how *Heijunka* is used in store management, and examines the effects on PST.

古垣幸一
（ふるがき　こういち）

サーバシステム事業本部
所属
現在，基幹システム技術支援に従事。

高木　徹
（たかぎ　とおる）

富士通ソフトウェアテクノロジーズ 経営改革室
所属
現在，業務プロセス改善に従事。

坂田晶紀
（さかた　あきのり）

富士通ソフトウェアテクノロジーズ 経営改革室
所属
現在，業務プロセス改善に従事。

岡山大輔
（おかやま　だいすけ）

富士通ソフトウェアテクノロジーズ 経営改革室
所属
現在，業務プロセス改善に従事。

トヨタ生産方式の導入によるソフトウェア開発プロセスの革新

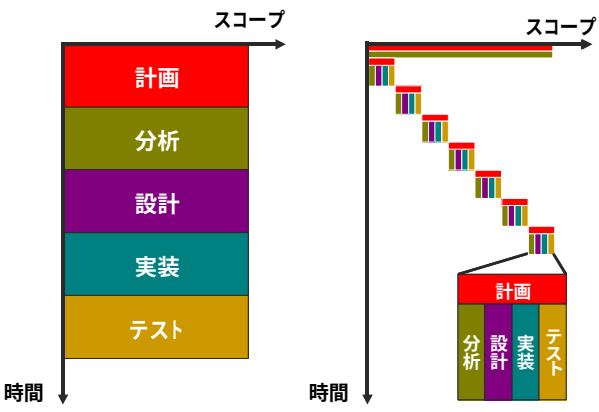
ま え が き

2000年のITバブル崩壊を契機に富士通プライムソフトテクノロジ（PST，現在，富士通ソフトウェアテクノロジーズ）ではソフトウェア開発の生産性向上に向け，UML（Unified Modeling Language）導入，プロジェクト管理の強化などに取り組んできたが，ITデフレーションを凌駕（りょうが）するまでの生産性向上には至らなかった。この状況を打開するため，すでにハードウェア製造分野で有効性が実証されていたトヨタ生産方式（以下，TPS）[1]をソフトウェア開発へ導入することを試みた。「ムダの徹底排除」，「平準化」，「自働化」，「目で見る管理」などのTPSのコンセプトのソフトウェア開発プロセスでの実践である。その手段としてTPSのコンセプトを具体的な実践手法（プラクティス）として持つアジャイル開発プロセス（以下，アジャイル開発）とストア管理手法（以下，ストア管理）を導入した。

本稿では，この二つの手法とその導入効果について紹介する。

アジャイル開発とプロセス改善

一般にアジャイル開発はウォータフォール型開発の対極として知られている。後者が計画・分析・設計・実装・テストの各工程を順に一度だけ実施するのに対し，前者はこれら一連の工程を繰り返すいわゆる繰返し型開発である（**図**-1）。



（a）ウォータフォール型開発　　（b）アジャイル開発

図-1　ウォータフォール型開発とアジャイル開発との比較
Fig.1-Comparison between waterfall development and agile development.

繰返し型のアジャイル開発に比べ，ウォータフォール型開発には，以下の課題がある。

（1） 計画の途中での変更は前提としておらず，変更が発生した場合の手戻り工数が大きい。
（2） バグは，工程の区切りでのレビューと下流工程におけるテストで検出されるが，バグが検出された場合，手戻り工数が大きい。
（3） 設計・実装の品質が悪い場合には，テスト（デバッグ）工数が膨れ上がる。
（4） 最終工程が完了するまで，動作するソフトウェアを手にすることができない。

アジャイル開発は，これらウォータフォール型開発の課題を補う以下のメリットがある。

（1） 小単位で分析・設計・実装・テストを繰り返すため，計画（要件）の変更・追加への耐性が強い。
（2） 要件ごとに実装・テストを完了させるため，実装機能のスコープは小さいものの常に動作するソフトウェアを手にすることができる。
（3） 小単位の実装ごとに，回帰テストを繰り返すことで早期に障害を検出できる。

一般的にソフトウェア開発では，工業製品のように同一の製品を繰り返し製造し続けることはない。また，開発プロセスにおいてもウォータフォール型開発では，一連の工程をそれぞれ1回実施するのみである。したがって，開発プロセスで経験した成功体験や失敗から得た教訓を後続の開発プロセスで生かすチャンスが少なく，PDCA（Plan, Do, Check, Action）サイクルを回すことが難しい。

それに対し，アジャイル開発は一連のソフトウェアを繰返し型で開発するため，全く同じではないが，同一のメンバが，同一の資源を用い，同一の環境で開発を繰り返すことになり，開発プロセスで得た学習を後続の開発プロセスで生かす機会が存在する。すなわち，PDCAサイクルを回す機会を得ることができる。このことがアジャイル開発の真のメリットである。

アジャイル開発とTPSのコンセプト

アジャイル開発には数多くのTPSのコンセプトが内包されている。以下に例を挙げる。

（1） 後工程引き取り

「後工程引き取り」のコンセプトは，アジャイル

開発の受入れテストにおいて体現されている。お客様の要求する機能の完了条件となる受入れテストによって実装が駆動される。上流から到着した設計書によってプッシュ型で駆動される従来型の開発とは大きく異なる。

(2) ジャストインタイム

「ジャストインタイム」のコンセプトは，お客様によって優先度付けされた機能を順次実装していくことがこれに当たる。またYAGNI（You Are't Going to Need It）とも表現される。

(3) 目で見る管理

目で見る管理は，アジャイル開発においてはXP（eXtreme Programming）のプラクティスである「ミラー」が，これを体現している。プロジェクトの状態をアナログな媒体で一目瞭然（りょう）にし，より早いフィードバックを行動レベルで促す。

(4) 平準化・多能工化

作業者が複数の作業スキルを備える「多能工化」と作業者ごとの作業時間をならす「平準化」は，アジャイル開発を進める上で原則となっている。

アジャイル開発の手順

アジャイル開発では，開発期間をイテレーションと呼ばれる単位に分割する。イテレーションとは繰返しのことであり，ある機能範囲を開発する期間でもあり，かつ，プロセスでもある。

開発開始とともにイテレーションが開始される。このときお客様要求が優先順位とともに開発チームに明示される。開発チームは明示された要求を優先順位に従って実装する。また，イテレーションは開始時に，その終了日が決定される。期間は通常1週間から1箇月程度が設定される。要求の実装がすべて完了しない場合でも終了日が優先される。実装しきれなかった（優先順位の低い）要求は先送りされる。または終了日以前にすべての要求を実装した場合には，新たなお客様要求の実装を行う。このため，予定より機能が足りないことは発生しても，提供が遅れることはない。すなわち納期が機能スコープよりも優先される。

イテレーションが終了すると，すべてのお客様要求が実装されてはいないものの，優先順位が高い機能は実装され，動作可能なソフトウェアが出力される。

イテレーション終了後，直ちに次のイテレーションが開始される。このときの入力は，初回のイテレーションと同様に優先順位付きのお客様要求と初回のイテレーションの出力である動作するソフトウェアが入力となる。このイテレーションの期間は初回と同様の期間が設定されることが多い。これは，同一の期間を繰り返すことで，チーム内に期間内の開発リズムを体得させる効果がある。

イテレーション終了時には，初回と同様に動作するソフトウェアが出力される。初回の機能スコープを含み，さらに今回の機能が実装された動作するソフトウェアである。

このようにイテレーションを繰り返し，実装スコープを拡張しながらソフトウェアを開発していく。

なお，お客様に対するソフトウェアのリリースは，イテレーション終了後にいつでも可能である。

各イテレーション内の開発手順を**図**-2に示す。イテレーション開始時に入力となるお客様要求は，ストーリと呼ばれる簡潔な単機能に整理される。ストーリはお客様が自ら作成することが理想であるが，お客様プロキシと呼ばれる役割を開発チーム内に設定することが多い。お客様プロキシとは，お客様要求を代表するチーム内の代理人である。開発メンバ内で，よりお客様の立場を理解しているメンバがこの役割を担う。各ストーリには優先順位が必ず設定される。

作成されたストーリは，計画会議（または計画ゲーム）と呼ばれる全員参加のミーティングによってタスクと呼ばれる単位に分割される。ストーリがお客様から見た機能単位であるのに対し，タスクは開発者から見た実装単位である。すなわち計画会議

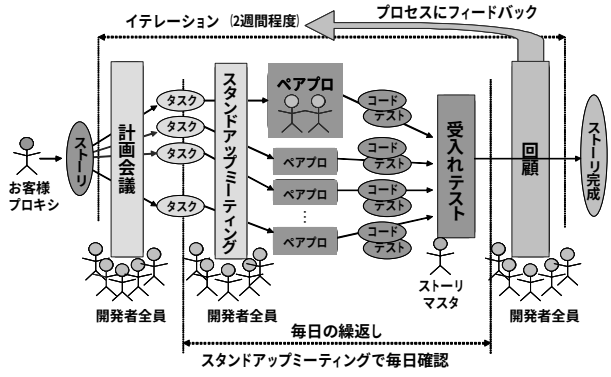


図-2 アジャイル開発手順
Fig.2-Agile development procedure.

は設計作業であると言える。この計画会議はイテレーションの開始時に実施される。

計画会議によって分割されたタスクは，毎朝実施されるスタンドアップミーティング（全員参加で実施される短時間のミーティング）で，開発者に割り当てられる。タスクの割当ては，管理者によるトップダウンでなく，開発者自らの志願による場合が多い。全体の開発状況を全開発者が常に把握し，今自分が何をすべきかを自律的に判断し行動することが求められる。

また，ペアプログラミングを実施する場合には，スタンドアップミーティング時にその日のペアを決定する。ペアは毎日交代することが望ましい。ペア交代の連鎖によって，知識の伝播が促進され，開発チームにより高い学習効果を与えることができる。また，ソースコードの共同所有（プログラムコードやコンポーネントを特定の担当者に割り当てないこと）と併せて実施することにより，開発対象のソフトウェアに対する「知識の偏り」を防止する効果がある。

ペアによってプログラムコードが開発されると同時に，テストプログラムコードも同時に作成される。開発されたプログラムは，このテストプログラムによりテストされ，タスクが消化される。ストーリを構成するタスクがすべて消化されたとき，ストーリの完了がテストされる。このテストは受入れテストと呼ばれる。受入れテストはお客様またはお客様プロキシによって作成されるテストプログラムである。別の見方をすれば，完了基準が明確なテスト手順である。受入れテストを通過することでストーリが完成する。タスクの実施から受入れテストによるストーリの完了が，イテレーション期間内で毎日繰り返される。

各イテレーションの最後には，開発者全員で回顧（ふりかえり）と呼ばれるミーティングを実施する。回顧では，このイテレーションで実施したすべての活動を振り返る。コーディングやテストはもちろん，ミーティングや電話応対さらには空調，換気，照明に至る文字どおりすべての活動について回顧する。そして，以下を実施する。

(1) つぎのイテレーションで継続する活動を全員で確認する。
(2) 発生した問題点を全員で共有し，解決策を立案し，つぎのイテレーションでの実施を確認する。
(3) つぎのイテレーションでの新たな取組みの活動を全員で合意する。

この回顧による開発プロセスへのフィードバックを続けていくことが改善の駆動力となる。

アジャイル開発の実践

アジャイル開発の実践事例[2]を以下に紹介する。

実践したのは開発期間が約2箇月間の製造業向けのプロトタイプシステム開発である。基盤技術にサーバサイドJava（サーブレット）を用いたWebアプリケーションである。開発メンバは9名である。各メンバの役割を以下に示す。カッコ内にソフトウェア開発経験（左）とオブジェクト指向開発経験（右）を示す。

X氏：トラッカー・お客様プロキシ（9年，8年）
Y氏：マネージャ・コーチ（5年，3年）
Z氏：コーチ（3年，2年）
A氏：開発者（3年，1年）
B氏：開発者（3年，1年）
C氏：開発者（0年，0年）～新人
D氏：開発者（2年，1年）
E氏：開発者（4年，1年）
F氏：開発者（14年，7年）

E氏，F氏は7月下旬から途中参加した。またアジャイル開発は全員が未経験であった。導入したXPのプラクティスを以下に示す。

(1) イテレーション（反復開発）

初めにトライアルとして5日間の第0イテレーションと2週間×3回のイテレーションを設定した。

(2) 回顧
(3) 目で見る管理（ミラー）

ストーリーカード・タスクカード・バーンダウンチャート（バックログ数のグラフ）を壁にはり，進捗状態を開発者がリアルタイムに見られる進捗管理をした。

(4) ペアプログラミング
(5) ソースコードの共同所有
(6) コーディング標準
(7) スタンドアップミーティング
(8) 継続的統合
(9) ユニットテスト

（10）受入れテスト
（11）お客様プロキシ
（12）リファクタリング

テスト駆動開発は実施していない。これは，全員が初めてのアジャイル開発であり，技術的なリスクを回避するためである。

アジャイル開発の評価

実践したアジャイル開発の評価を以下に示す。

評価の観点は，生産性，品質，コスト，納期および開発者の感想である。

● 生産性

各イテレーションの生産性および全体を通しての生産性を生産性指標(注)で以下に示す。

イテレーション0：0.760
イテレーション1：0.949
イテレーション2：0.911
イテレーション3：1.962
通期：1.268

PSTの標準生産性と比較して開発初期の段階では24%低い生産性であったが，最終段階では向上し，通期では26.8%の向上を示した。この生産性向上の理由を以下に考察する。

（1） 優先順位に従って実装したため不要な機能の実装がなかった。また，ストーリをタスクへ分割し，一個ずつ（「一個流し」）開発したため，未テストのプログラムが滞留するなどのムダが排除された。さらに，お客様プロキシの存在によって，要求されていないドキュメントなどの中間生産物は作成しなかった。
（2） ペアプログラミングにより，開発者はいつでも必要な情報を聞ける状態にあった。また，ソースコードの共同所有によって，開発者の誰もが任意のプログラムの改修が可能であり，従来型の開発で発生しがちな他担当者の改修待ちが発生しなかった。
（3） 毎朝のスタンドアップミーティングで，日々の進捗が確認され，遅れに対するよりタイムリな対処が可能となり，タスクスケジュールが最適化された。
（4） イテレーションごとに回顧を実施することで，開発チーム内で問題点を共有し，その対処を次のイテレーションにフィードバックするというPDCAサイクルが開発プロセス内で実施された。

上記のプラクティスが生産性向上につながったと考えられる。

● 品質

アジャイル開発では，ウォータフォール型開発のように，計画・分析・設計・実装・テストといった工程の区切りがなく，各工程のバグ発生のトラッキングによる品質検証ができない。しかし，品質の良さを開発者は実感している。

（1） ペアプログラミングによって，疑問点・不明点はいつでも聞ける状態であり，また，常時レビュー状態にあることから，従来の方式と比較してバグの作り込みが大幅に回避される。
（2） 自動化された回帰テストによって，最長1日でバグは検出される。また，プログラムコードは，同時に作成されるテストプログラムによって，即座にテストされバグは除去される。

● コスト

各イテレーションにおける開発者の残業時間の推移を**図**-3に示す。

各イテレーションにおける残業時間は，イテレーションが進むにつれて開発者間でばらつきが少なくなっている。作業時間の平準化が進んでいる。

作業時間の平準化には，開発者の「多能工化」が前提となる。発生した作業にどの開発者も携わることができれば，作業時間を平準化できる。多能工化状況を構成管理ツールであるCVS（Concurrent Versions System）のコミットログで測定した結果

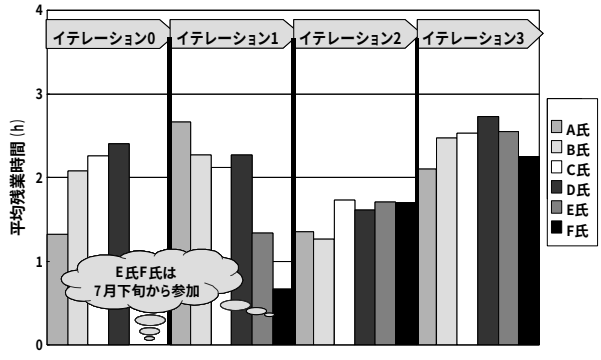


図-3 各イテレーションにおける開発者の残業時間の推移
Fig.3-Transition of developer's overworking-hours in each iteration.

（注） 1人月あたりの開発ソースコード行数に関するPSTの標準値との比。

を**図**-4に示す。各プログラムへのコミット回数は，各開発者に分散されており，「多能工化」が進んでいる。

従来方式のように，コンポーネントごとまたはプログラムごとに開発者が専任化されていると各プログラムはそれぞれ特定の開発者のマークに塗りつぶされることになる。

多能工化の実現には，ソースコードの共同所有，およびペアプログラミングが寄与している。担当を専任化せず，毎日，担当するプログラムが替わり，また，毎日変わるペアの相手に経験者がノウハウを伝播することが，多能工化を促進する仕組みとなった。

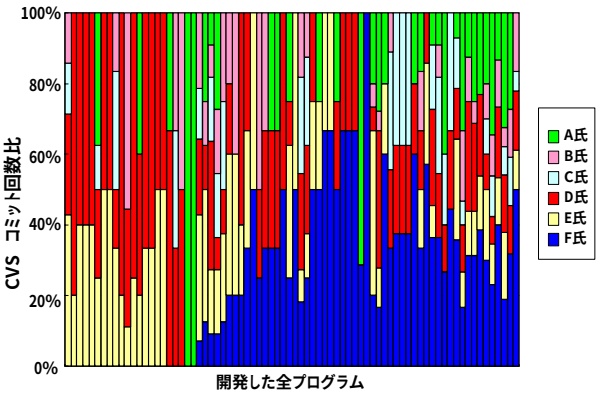


図-4　開発者による各プログラムへのCVSコミット回数比率
Fig.4-CVS commitment ratio for each program by developer.

今回のアジャイル開発では，同一システムの前回の開発（ウォータフォール型）と比較して原価率が16%低下（コストダウン）する結果となった。

● **納期**

今回のアジャイル開発におけるお客様要件の発生件数と実装件数の推移を**図**-5に示す。

開発当初から明確になっていたお客様要求は全体の約50%であり，残りは開発工程期間中に散発的に到着している。また，消滅する要求も多い。しかし，随時到着する要求を順次実装し，リリース日には全要求の実装が完了した。各要求の到着から実装までの時間は平均1週間であった。つまり，要求到着の遅延を許容し，なおかつ一つの要求あたりの実装も1週間という短時間で実現した。

これは，お客様プロキシ，イテレーション，ストーリのプラクティスによるものである。要求の優先度付けがなされ，一個流しの実装により，納期の短縮が実現され，継続的統合の環境下で回帰テストが実施され，リファクタリングと合わせて，品質を確保しつつ追加要求に耐え得る開発プロセスが実現されたと考えている。

● **現場の声～開発メンバから～**

アジャイル開発の実践で開発メンバがどのような感想を持ったのか。以下に現場の声を挙げる。

(1)　「以前は過負荷になると精神的に会社に来たくなくなった。しかし今回は過負荷になっても出社がおっくうではなかった。」

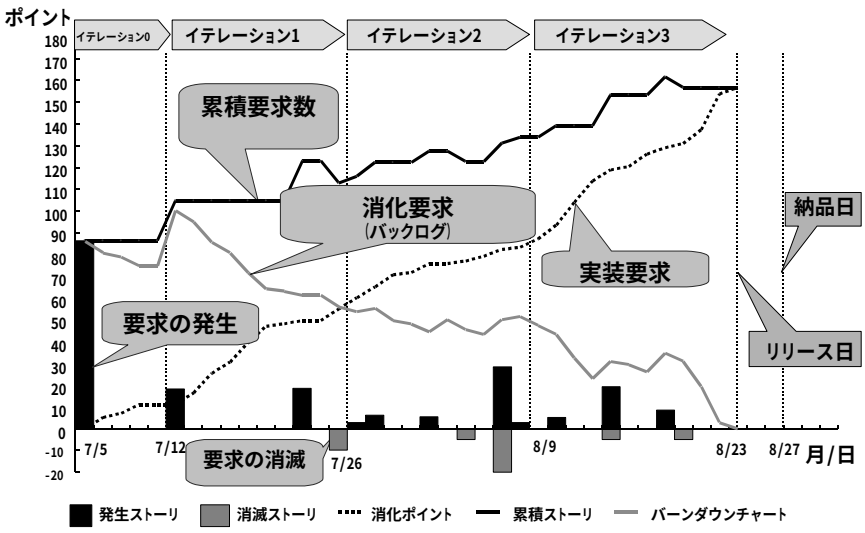


図-5　お客様要求の発生件数と実装件数の推移
Fig.5-Transition of customer requirements and requirements implemented.

(2) 「1日の作業量が，スタンドアップミーティングのタスク配分によって分かりやすくてよい。」
(3) 「コーチという存在がいて，疑問/質問が随時しやすく聞きやすい。」
(4) 「チーム全体の雰囲気やメンバのモチベーションは最後まで良い状態に保たれた。」
(5) 「ペアプロをしていると，困って作業が止まることがなくなった。」
(6) 「ペアプロは，お互いに知っていることや，知らないことを話しながら進められるので勉強になる。」
(7) 「ペアプロによってチームのスキルアップが短期間でできた。」
(8) 「ペアプロによって間違いを減らし，効率的に作業ができた。」

このように，全開発期間を通して，モチベーションを維持し，安定した精神状態で開発が実施できた。

アジャイル開発の適用拡大に向けて

今回の報告した事例は，プロトタイプシステム開発であり，比較的小規模な開発である。アジャイル開発の中でもXPをベースに実施しており，一般的に言われるXPの適性範囲（生命にかかわらないソフトウェア，かつチーム人数は20人未満）内に適合するものであった。

アジャイル開発を数十人，数百人を超える大規模プロジェクト，またはミッションクリティカルなシステム開発へ適用するには，プラクティスの実践手法，品質検証など，まだ解決すべき課題は多い。しかし，アジャイル開発のプラクティス内には，目で見る管理や，開発プロセス内でPDCAサイクルを実現するイテレーション・回顧，開発メンバの負荷と能力を平準化するペアプログラミングなど，ウォータフォール型開発においても有効なものが多く存在すると考えている。今後，ウォータフォール型開発など，従来のソフトウェア開発プロセスでのアジャイル開発プラクティスの実践を段階的に拡大していく予定である。

ストア管理によるプロセス改善

アジャイル開発がソフトウェア開発プロセスを対象としているのに対し，ストア管理はより幅広い業務プロセスを対象とし，TPSのコンセプトである「平準化」を実践することでプロセス改善を実現する手法である。

図-6に示す作業モデルを用いてストア管理を紹介する。

作業時間および必要な能力が異なる3種類の業務作業（a，b，c）がランダムに発生し，各業務担当が順次作業を処理していくという作業モデルである。

一般的に業務プロセスで発生する作業は，各業務ごとに担当者が決められて処理される。しかし，この形態で作業を続けると担当者間で処理能力と作業負荷の偏在（ばらつき）が発生し，全体として実処理能力が低下してくる。ストア管理では作業の指示管理を集約し，担当者間の作業負荷と処理能力のばらつきを最小化（「平準化」）することで全体の処理能力を高めることをねらったプロセス改善手法である。その考え方を**図**-7に示す。

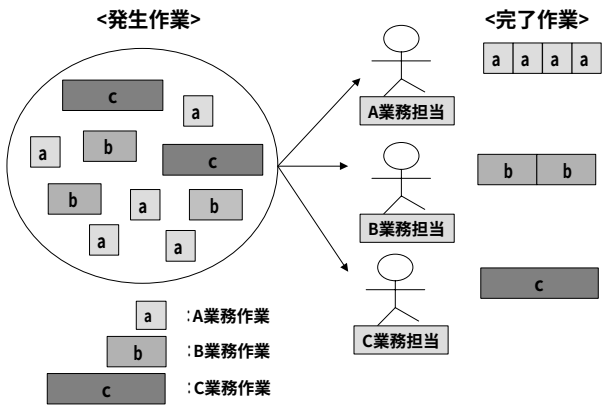


図-6　ストア管理の作業モデル
Fig.6-Working model of store management.

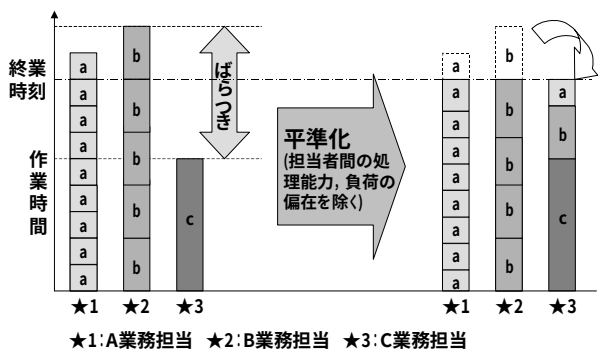


図-7　ストア管理による平準化改善の考え方
Fig.7-View of “Heijunka (leveling)” improvement by store management.

ストア管理では平準化を実現する手段として以下の三つのプラクティスを実践する。

(1) ストア引き取りによる作業
(2) 一列待ちの作業管理
(3) 作業負荷の「目で見る管理」による自律平準化

以下に各プラクティスを説明する。

(1) ストア引き取りによる作業

図-8に示すように，業務で発生するすべての作業は，ストア-INと呼ぶ作業指示ボックスで管理する。作業者は，自発的にストア-INから作業を一つ引き取り，作業を開始する。二つ以上の作業を並行して実施することは避ける。一つの作業を着実に完了させていく（「一個流し」する）ことで，作業者の頭の切替えによるムダの発生や各作業の終了時間のばらつきがなくなり，全体としてリードタイムの短縮につながる。

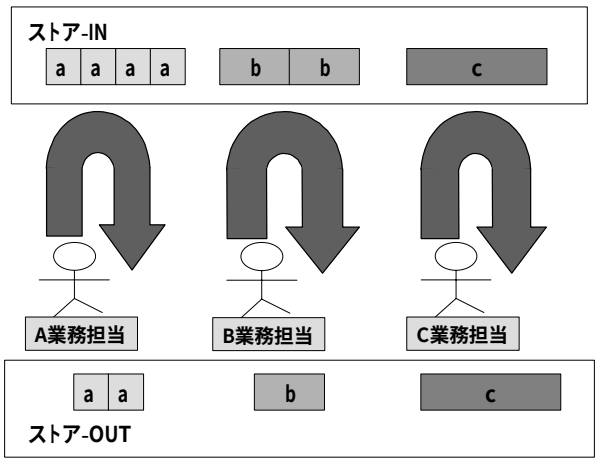


図-8　ストア引き取りによる作業
Fig.8-Work procedure by store pull system.

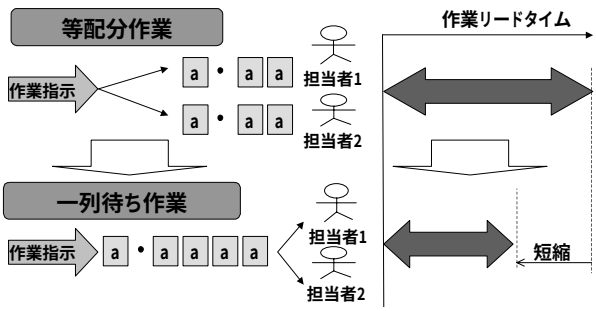


図-9　一列待ち作業によるリードタイム短縮
Fig.9-Shortening lead time by one-queueing workload management.

(2) 一列待ちの作業管理

図-9に示すように，作業は発生時点で作業者に配分せずにストア-IN内に一列待ちキューとして管理し，先頭から処理していく。ランダムに発生する作業を処理する場合，作業者に等配分して処理する形態と比べて，作業を一列待ちキューにして先頭から処理していく形態の方が，作業リードタイムがより短縮される。

(3) 作業負荷の「目で見る管理」による自律平準化

図-10に示すように，ストア-IN状況は，ボードなどを用いて業務別の処理待ち件数が常に作業者から目で見える状態（「目で見る管理」）にしておく。ボードの作業待ち件数の負荷状況を見て作業者は作業者の追加，担当変更などの対応の要否を判断し，作業負荷と処理能力のバランスを維持するように自律的に行動することで平準化が実現される。

上記のプラクティスを実践する過程で発生する課題を逐一解決していくことが結果的に業務プロセスの改善につながる。

ストア管理の導入効果

PSTでは，アジャイル開発が適合しないソフトウェア保守サポート業務にストア管理を導入し，業務プロセスの改善を実践した。従来業務の課題とストア管理導入効果を以下に紹介する。

従来，当該業務では以下のような課題があった。

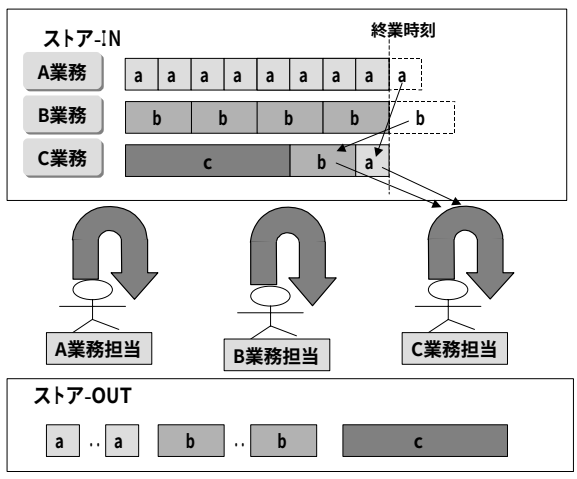


図-10　作業負荷の「目で見る管理」による自律平準化
Fig.10-“Visual management” of workload and autonomic “Heijunka (leveling)” control.

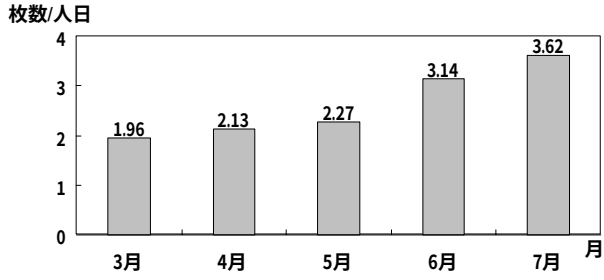


図-11　作業カード計測による改善進捗の定量化（1人日あたりの作業カード処理枚数）
Fig.11-Quantification of improvement-progress by counting work-card.

（1） 作業負荷が週後半，夕方に集中する傾向にあり，休日出勤，残業が常態化していた。
（2） メンバに能力のばらつきがあり，特定メンバに負荷が集中していた。

今回，ストア管理の導入・実践により以下のような効果が得られた。

（1） 作業負荷状況を「目で見る管理」することでチームメンバが常に作業負荷の分布状況を把握し，自律的に作業負荷の「平準化」をすることが可能になった。今回，実践したチームでは，ストア管理を導入以降，休日出勤がゼロとなっている。
（2） メンバ間の作業負荷の平準化が進み，残業時間の格差が約30%減少した。
（3） チーム作業処理能力が約20%向上した。

ソフトウェア保守サポート業務ではストア管理を導入する以前は作業量を計測する手段がなく，作業処理能力を定量的に把握することができなかった。しかし，ストア管理の導入後は，すべての作業は作業カードに記録され，実施した作業カードを分析集計することで作業処理能力の把握・評価が可能となる。この点もストア管理の導入の大きなメリットである。

事務管理部門でストア管理を実践し，作業カードの枚数で改善効果を計測した事例を**図**-11に示す。本ケースでは4箇月間で約80%の生産性向上を達成している。

む　す　び

今回のアジャイル開発，ストア管理のプラクティスによるTPSのコンセプトの実践を通してソフトウェア分野でも「ものづくり」の基本理念に立ち戻り，ソフトウェアの開発プロセスを改善することは非常に有効であることを実感した。

2003年に一部の部門で試行的に開始したTPSのコンセプトの導入は，実施部門が拡大し，2004年末にはPSTのすべての部門に広がっている。PST損益の下降も2004年9月期に止まり，以降，回復方向にある。

最後に今回のPSTのソフト開発プロセスの革新に向けてストア管理の導入指導をいただいたトーマツコンサルティング（株）　松井順一様に厚くお礼申し上げます。

参 考 文 献

（1） 大野耐一：トヨタ生産方式　－脱規模の経営をめざして－．ダイヤモンド社，1978.
（2） 坂田晶紀：アジャイル実践　アジャイルはトヨタ生産方式か？　ソフトウェア多能工化への試み．Developers Summit 2005発表資料，2005.