特集号
招待論文

楽天でのエンタープライズアジャイルとDevOps
—Dev/Test/Ops三位一体の自動化—

川口 恭伸[†1]　荻野 恒太郎[†1]　古川 貴朗[†1]

[†1]楽天（株）

本稿ではエンタープライズアジャイルの1つの実例として，部署やチームを越えたDevOpsプラクティスの導入の事例について紹介する．楽天では部門ごとに自動化が進展してきたが，エンタープライズ特有の課題として複数の担当，部署間の作業の受け渡しに改善の余地があると考えた．そこで従来の役割分担を超え，インフラ構築・デプロイ・テスト・稼働監視まで一貫した自動化を行った．その結果，全体工程で99.40%という大きな時間短縮を実現した．技術勉強会や自社カンファレンスなどの機会を通じた情報共有やコミュニティ構築により，部門横断的な取り組みが成功した．

1. はじめに

1.1 エンタープライズアジャイルの定義

本稿ではエンタープライズアジャイルの事例として，DevOpsプロジェクトの事例を述べる．具体的な事例を述べる前に，本稿における用語の定義，アジャイルおよびエンタープライズアジャイルについて説明する．

アジャイルという言葉が一連のソフトウェア開発手法のことを指すようになったのは，2001年の「アジャイルソフトウェア開発宣言」[1]においてである．エクストリームプログラミング（XP），スクラム，DSDM，Crystal Clear，Pragmaticといった開発手法やスタイルを提唱している人々が集まり，共通する価値観やプラクティスについて議論し，合意に至った[2]．この時点では，大企業でどのように適用するべきかという点については特に論じられていない．

2010年になると，アジャイル開発の主導的なカンファレンスであるAgile 2010において，エンタープライズの改善（Enterprise Improvement）というカテゴリが作られている[3]．それまでの10年間において，大企業特有の課題について知見が蓄積されてきた．開発を行っているチームや部門をアジャイル開発で効率化するだけでなく，その外側にある，業務部門をはじめ，インフラ，品質保証，人材，採用といった各種関連部門との効率的な協業や組織構成そのものの最適化について論じられるようになった．

エンタープライズアジャイルという言葉については，業界で一致した定義は存在せず，個別にフレームワークが提案されているのみである[4],[5],[6],[7],[8]．そこで，本稿においては，エンタープライズアジャイルとは，チームにおけるソフトウェア開発についての実践的なプラクティスを集めたアジャイル手法群を，チームを越えて適用する際に発生する諸問題に対応する取り組みすべてを指す総称と定義する．

エンタープライズアジャイルとは，アジャイル開発手法をチームを越えて適用する取り組みの総称である

1.2 大企業におけるアジャイルの課題

大企業（エンタープライズ）のソフトウェア開発部門では，専門分野ごとに組織が置かれる職能別組織を基本とし，その上で，特定のプロダクトや案件向けに職能を横断する一定期間のみのプロジェクトチームを編成するのが一般的である．

一方，アジャイルは少人数のクロスファンクショナルチームで持続的に開発することを基本としている．アジャイルソフトウェア開発宣言で「最良のアーキテクチャ・要求・設計は，自己組織的なチームから生み出されます」[9]とうたわれている通り，チーム自身に権限を委譲して，自分たちで決定できるようにすれば，日々の課題解決を通じてチームの成長を促すことが可能，という考えが前提にある．

しかし大企業においては，1チームでアジャイル開発がうまく機能したとしても，そのノウハウを持続的に維

持したり，組織全体に広げていくためにはさまざまな障壁が存在する．

A：チーム運営上の課題

- **A-1　プロジェクトチーム解散時の記憶喪失：**プロジェクト終了時にチームが解散し，チームに蓄積された多くのノウハウや暗黙知が失われ，「記憶喪失」[10]が発生する．
- **A-2　チームの熟成不足：**プロジェクトの全体工数見積りと予算確保の後に，開発チームが編成される場合，チームビルディングに時間がかかる．その場合，実際に作業にあたる開発チームの知見を利用した見積りを行っていないために計画精度が悪く，チームのコミットメントも低くなる．
- **A-3　スキルとプロセス合意の不足：**チーム内のスキルが不足していて短期間でリリースできない場合や，プロセスについての合意が不足すると，予定した成果物が出ないといった問題が生じる．

B：チーム外との調整の課題

- **B-1　チーム外との調整負担：**職能別組織の場合，あるスキルを持つ人々が専門の部署にいて，開発チームと協働するために時間がかかる．一方，事業別組織では，調査などの予算や教育が部門ごとで，部署を越えた情報共有が起こりにくい．
- **B-2　承認プロセスが足を引っ張る：**チームが適切な権限を持っておらず，シニアマネジメントや委員会での承認や手続きが必要なことがある．過去に発生した障害の再発を防ぐ目的でプロセスが追加され，予算執行・ドキュメント・手順などの事前チェックによってスピードが落ちる．
- **B-3　予算管理や受発注契約と開発プロセスの不一致：**従来型のプロジェクト用の予算や会計管理方法がアジャイル開発に合致しない場合がある．期初に立てた開発計画は，スプリントごとの最新情報にあわせて変更されるため，予算執行管理，原価管理，受発注契約などが対応できるよう，規約や契約を見直さなければならない．

C：組織全体の課題

- **C-1　評価制度のミスマッチ：**個人の成績と連動する評価や報酬の仕組みを採用している場合に，チームを手助けするような業務を行っているメンバの評価やキャリアパス形成が難しいことがある．
- **C-2　指導者の不足：**アジャイル開発において必要となるスキルやプラクティスを教えられる経験者，研修を行うトレーナ，現場を指導するコーチが不足する．
- **C-3　情報共有不足と不信感：**社内でさまざまなチームが独自にアジャイルを進め，課題や組織依存のベストプラクティスが共有されないことがある．失敗事例が増えることで，賛同しない人々からの不信感が高まっていく．

分野によっては，すでに大企業よりスタートアップのほうが競争上優位になっている可能性がある

これらの課題は少人数のスタートアップ企業では発生しにくく，大企業特有のものである．スマートフォン向けのアプリなど，それほど財務体力や営業力を必要としない分野では，すでに大企業よりもスタートアップのほうが競争上優位になっている．エンタープライズアジャイルでは，こうした大企業であることが不利になる点について，対策を考える必要がある．

2. DevOps の潮流

大企業がスタートアップ企業のようにスピーディに振る舞うための考え方として，DevOpsがある．先に挙げた大企業におけるアジャイルの課題のうちいくつかを，担当間の協業と自動化によって克服しようとする流れである．

2.1　Flickr 社による提案

2009年にDevOpsの源流となるFlickr社の発表「10 Deoploys per a Day（1日に10回デプロイする）」が行われた[11]．この発表では，仮想化やクラウドといった基盤のソフトウェア化を背景に，開発者（Dev）と運用担当（Ops）が相互にスキルをオーバラップさせ，よりうまく協業することを主張している．道具立てとして挙げられているのは以下の6つである．

- 自動化されたインフラ
- バージョン管理の共通化
- ワンステップでのビルドとデプロイ
- フィーチャーフラグ
- メトリクスの共有
- IMやチャットによる運用

また，必要な文化として以下の4つを挙げている．

- 尊敬する
- 信頼する
- 失敗に対する健全な態度をとる

• 非難しない

ここで提案された原則やプラクティスのほとんどは，アジャイル開発において提案されたものか，それをより最新の技術動向やツールで具現化したものである．

開発チームがアジャイル開発に適応すると，サービスやアプリケーションを小刻みに素早くリリースしたいという要求が高まる．そうすると，インフラ部門や運用部門との軋轢やスピード感の違いが問題になる．スキルも違えば目的も違う各部門が連携して動くために，前述のDevOpsの4つの文化が必要であり，その上でリリースに必要なリードタイムを削減するために，従来の職域を越えて連携し，自動化に取り組もうということである．

2.2 大企業への DevOps 導入の事例

MicrosoftではVisual Studioプロダクトチームにアジャイルを導入した．パッケージ版ソフトウェアであったVisual Studio 2010では，リリース時点の不具合を劇的に減少させた[10]．次に，Web版のVisual Studio Onlineサービスを開始し，DevOpsのプラクティスを適用していった．たとえば，フィーチャーフラグを導入して稼働中のサービス上で利用者ごとに機能を出し分けるA/Bテストを簡単に実施できるようにした．A/Bテストを利用した新規アカウント作成フォームの改善を通じて，訪問者のうちアカウント作成に至る率が3%から20%へと大幅に改善した[12],[13].

大企業では，アジャイル開発がうまくいくようになると，リリースや品質保証の工程の改善がもっと必要になる

Yahoo!では，チームレベルのアジャイルの導入を進めたが，その結果，リリースと品質保証（QA）の工程がボトルネックとなっていた[14]．開発チームがリリース用のコードを準備した後，リリース担当がQA用の環境にデプロイして，QA担当がテストを行う，というリリース工程を踏んでいた．QA用の環境には関連するすべてのサービスがデプロイされ，全体を統合してシステムテストを行う．しかし，多くの担当者が作業するためミスも多く，さまざまなブランチやリリースバージョンが混在するため，作業は困難を極めるようになっていた．開発よりも品質保証にかかる時間の方が長くなることもあった．そこで，DevOpsのプラクティスを導入することを決め，QA用の環境を廃止し，デプロイを自動化し，本番環境ですぐにテストが走るようにした．併せてQA担当を専門の部署から各プロダクトに移し，バグ混入からバグ検出までのリードタイムを短くできるようにした．

2.3 楽天におけるアジャイルと自動化

楽天においては，2010年頃からチームへのアジャイルコーチングが行われるようになった[15],[16],[17]．英語公用語化した2012年以降は海外の複数のアジャイルトレーナを招いてさまざまなアジャイル研修を行うとともに，サンフランシスコに開設したアジャイル開発センタを始めとした海外の拠点やグループ企業に開発者を派遣し，実地研修を行った．トップマネジメントの後押しもあり，アジャイル開発を行う開発チームは増加し，テストの自動化も進められていった．また，2012年にはそれまで部署ごとに分散していたソースコードのリポジトリやチケット管理システムを全社共通システムに統合した．

開発チームのアジャイル適用が進むに従い，エンタープライズアジャイルで言われるような課題が顕在化してきた．たとえば，開発工程のみ改善しても，品質保証やインフラの改善がなければ，作ったものを素早く安定的にリリースすることができない．

そこで，こうした課題を克服するための取り組みが行われた．

継続的システムテスト[18]は，検索機能において，テスト環境の整備とエンドツーエンドテストの自動化を行い，開発から品質保証までを一貫して確認できるようにする取り組みである．開発チームがコミットしたソースコードをテスト環境にデプロイし，すぐに利用者や利用システムとのインタフェース経由でシステム全体の自動テストが走るようにした．

継続的システムテストの主なねらいは，①複数のチームが絡む開発での最終成果物の確認，②ドメイン固有言語（DSL）によるテスト開発コスト削減，③テストシナリオから環境要因を分離して別に管理，の3つであった．検索機能の特徴として，テストに膨大なデータセットが必要になる点がある．このためリモート接続した本番環境でしかテストできず，開発者が自らテストを実行することが難しい．そこで，テストケースとテストデータのサブセットを作成することで，各開発者の手元の仮想環境上でテストできるようにした．システムテストも毎日，自動で実行できるようになり，リリース直前で発見されるバグの数は減り，バグの平均対応時間も5日から2日へと大幅に縮小した．

インフラ分野でも自動化を進めている．サーバ構築（プロビジョニング）担当は，構成管理ツールを導入してサーバ構築の手順の共通化を行った上で，Chefを導入し手順を自動化した[19]．その結果，サーバの構築から，OSや共通性の高いミドルウェアの設定までが自動化され，自動化タスクのソースコードも社内公開されるようになった．社内向けPaaS（Platform as a Service）基盤も構築し，開発者自身が直接サーバ構築やデプロイを行える環境が用意された．

トラベルサービスの担当は，まずアプリケーションデプロイを自動化したが，さらなる効率化のために，インフラ／環境構築／アプリケーションデプロイの3階層のすべてをそれぞれ自動化した[20]．

このように楽天では，ソフトウェア開発分野での自動化が進むと同時に，リリースや品質保証の分野でもそれぞれ自動化が進展した．

楽天では，開発者チームの自動化が進むとともに，システムテスト，インフラ構築，デプロイと設定の各分野でそれぞれに自動化が進展した

3. Dolphin Project：Dev/Test/Ops の統合

Dolphin Projectは，これまで独自に自動化や効率化を進めてきた開発担当（Dev），運用担当（Ops），テスト担当（Test）を連携させ，さらに効率化するプロジェクトである．サーバインフラからデプロイ基盤，テストのプロセスまでカバーするフレームワークよりなる．

3.1 Dev/Test/Ops の役割を整理する

担当者の連携を考える上で，Dev/Test/Opsがそれぞれ担う必要のある非機能要件や品質特性について，Webサービスの持続可能な成長を支えるという観点で整理し直した．

これは，TestやOpsの非機能要件や品質特性をシステムに反映するという新しい役割がTestやOpsに求められるようになったためである（図1）．

- Dev：新しい機能を動作するコードの実装
- Test：機能を持続的にテストする仕組みの実装
- Ops：機能を持続的にデプロイするための仕組みの実装

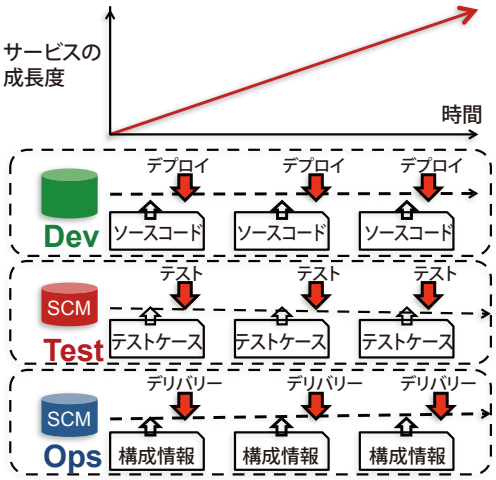

図1　Dev/Test/Ops の役割分担と作業タイミング

従来の組織体制下では，それぞれ以下の役割分担になっている．

- Dev：新しい機能を実装する
- Test：機能をテストする
- Ops：機能をリリースする

この役割分担は，「実装が終わった機能をDevからTestに」「テストが終わった機能をTestからOpsに」と，「機能」をチームからチームへと受け渡していくため待ち時間が発生するという欠点がある．

また，Dev側でアジャイルの導入や自動化が進んでも，TestやOpsが旧来のQAやオペレーションを続けていると，前述のYahoo!の事例のようにQAがボトルネックになるといった問題が発生する．この問題の本質的な原因はサービスの持続可能な成長のための非機能要件や品質特性といった観点が抜けているためである．

テスト自動化やInfrastructure as Codeといったコンセプトを導入するにあたり，TestとOpsの役割は大きく変わりつつある．以前はDevが要求分析と設計に責任を持ち，TestやOpsはそれらをDevの定義の範囲内でテストと運用を行っていた．しかし，自動化によってTestやOpsのタスクのソフトウェア化，コード化が進むと，TestabilityやOperabilityといったTestとOpsの観点もシステムアーキテクチャに反映することが可能になった．

3.2 目指すべき非機能要件の定義

Dolphin Projectでは，既存の非機能要求と品質特性のフレームワーク[23],[24]を参考にして，目指すべき非機能要件を定義した．Deliverability（リリース可能），Operability（運用可能），Testability（テスト可能）の3つのカテゴリに分類した．このカテゴリの定義例を以下に示す．

- Deliverability：1週間に1度リリース可能なこと

- Operability：ログ監視システムより5分以内に障害の切り分けが可能なこと
- Testability：1週間に1度，品質評価データを更新可能なこと

Deliverabilityの「1週間に1度リリース可能なこと」という要件は，DevOpsを定義したFlickrの「10 deploys per day」（1日10回リリースできること）を参考に，社内規程の品質保証を考慮して当面の目標として定義したものである．従来でも3カ月に1度はリリースしていたが，これを1週間へと短縮する．ビジネスには試行錯誤が必要である．各種の変更をより短期で行うことができれば，より多くの試行錯誤が可能になり，ビジネスの成功の確率を高めることができる．

また，この要件に合わせ，Testability要件としても1週間に1度以上テストを実行し，評価データを更新できなければならない．1週間に1度という目標はまだまだFlickrの水準に遠いが，まずは現実的な目標設定をし，着実にクリアすることが重要である．

3.3 Conwayの法則の回避

自動化がアーキテクチャレイヤごとに行われてしまう背景には，機能別組織がある．「組織はアーキテクチャにしたがう」（Conwayの法則）[21]という現象は，歴史

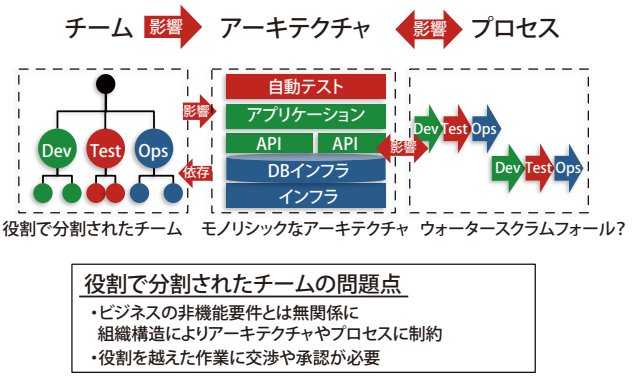

図2　機能ごとの組織構成の問題点

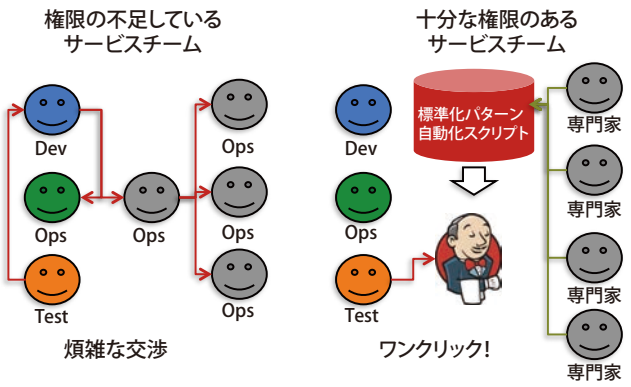

図3　標準パターンと自動化スクリプトによる権限付与

上繰り返されてきた．楽天でも，Dev/Test/Opsはそれぞれ別の組織に別れており，チーム構造，アーキテクチャ，プロセスは互いに影響していた（図2）．

機能ごとの組織構成は，アーキテクチャの階層を下から順に積み上げる開発プロセスと相性が良い．Devが要求分析と設計を行い，TestとOpsは設計に基づいてテストとリリース作業を行うという分担も明白である．このままアジャイルを適用すると，各チームスプリント単位でタスクを順に受け渡す形になり，動的なスケジュール調整が起こらず，後工程に手待ちや調整のムダを発生させる．また，担当をまたぐような課題での調整に時間がかかる．

DevOpsでは，TestやOpsの担っていた非機能要件や品質特性の設計と確認をコードで表現し，誰でもいつでも何度でも，それを実行すれば要件を満たすこと確認できるようになる．

そこで，Dolphin Projectでは，標準パターンと自動化スクリプトを整備することで，設計と実行を分離し，実行の権限を作業者に渡せるようにした（図3）．

3.4 デプロイメントパイプライン

Dolphin Projectは部門内の開発チームに，共通のデプロイメントパイプラインを提供する．パイプラインとは，非機能要件を満たすように組み合わせられたツール群のことである（図4）．

個々のサービスのビジネス要件には標準パターンから選んで使ってもらうが，既存に適切なパターンがない場合はDolphin Project担当が新たに作成する．

Opsに関連するフローを拡大したのが図5である．Devがリポジトリにある環境構築スクリプトの標準パターン（Cookbook）から，必要な修正を行って，変更提案（Gitのプルリクエスト）を行い，Opsがそれをレビ

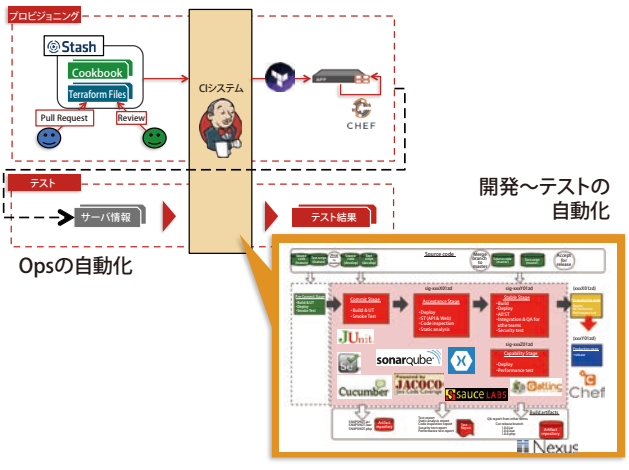

図4　デプロイメントパイプライン

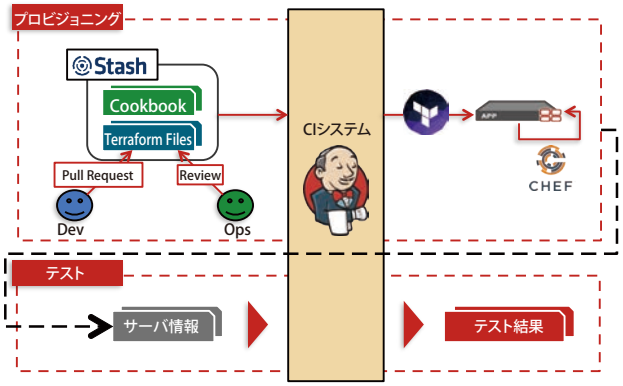

図 5　デプロイメントパイプラインの Ops 部分

ューする．レビュー済みの環境構築スクリプトは継続的インテグレーションシステム（CI システム）により自動的に各環境で実行され，サーバが構築される．Test は構成ファイルやソースコードに対するテストを書くが，これも CI システムによって自動実行され，レポートに反映される．

それぞれの自動化スクリプトは，既存の各部門で行われてきた自動化の取り組みで得られた知見を反映して作成された．

CI システムの中では，さまざまなテストのためのミドルウェアを組み合わせ，Commit（コミット），Acceptance（受入），Stable（安定稼働），Capacity（容量）の 4 つのステージを形成している（図 6）[22].

3.5　Dolphin Project の効果

これらの仕組みを構築し，1 つ目のプロジェクトに適用したところ，ビルド・サーバプロビジョニング（デプロイ含む）・機能テスト・非機能テスト・レポート作成まで非常に素早く表示できるようになった．これまでは 150 時間以上のリードタイムが必要であったが，1 時間を下回る結果となった（図 7）.

Dev と Ops と Test のすべての工程を一括して自動実行することによって，リードタイムを大きく短縮することができた．この時間を使い，品質の向上やビジネス成功に向けて試行をより多くこなせるようになると期待している．

4.　成功要因の分析

まず 1.2 節で述べたエンタープライズアジャイルの課題に即して対応を整理する（表 1）.

4.1　チーム運営上の課題への対応

まず Dolphin Project がほぼすべて内製開発で行われている点は成功要因として挙げられる．社内のエンジニアが数多くのオープンソースや商用ソフトウェアを試し，コンパクトなチームで仕様決定・技術選択・実装・テスト・デモを行った．東京とバンガロールの 2 拠点合わせてもチームの人数は 12 人に満たない．少人数でアウト

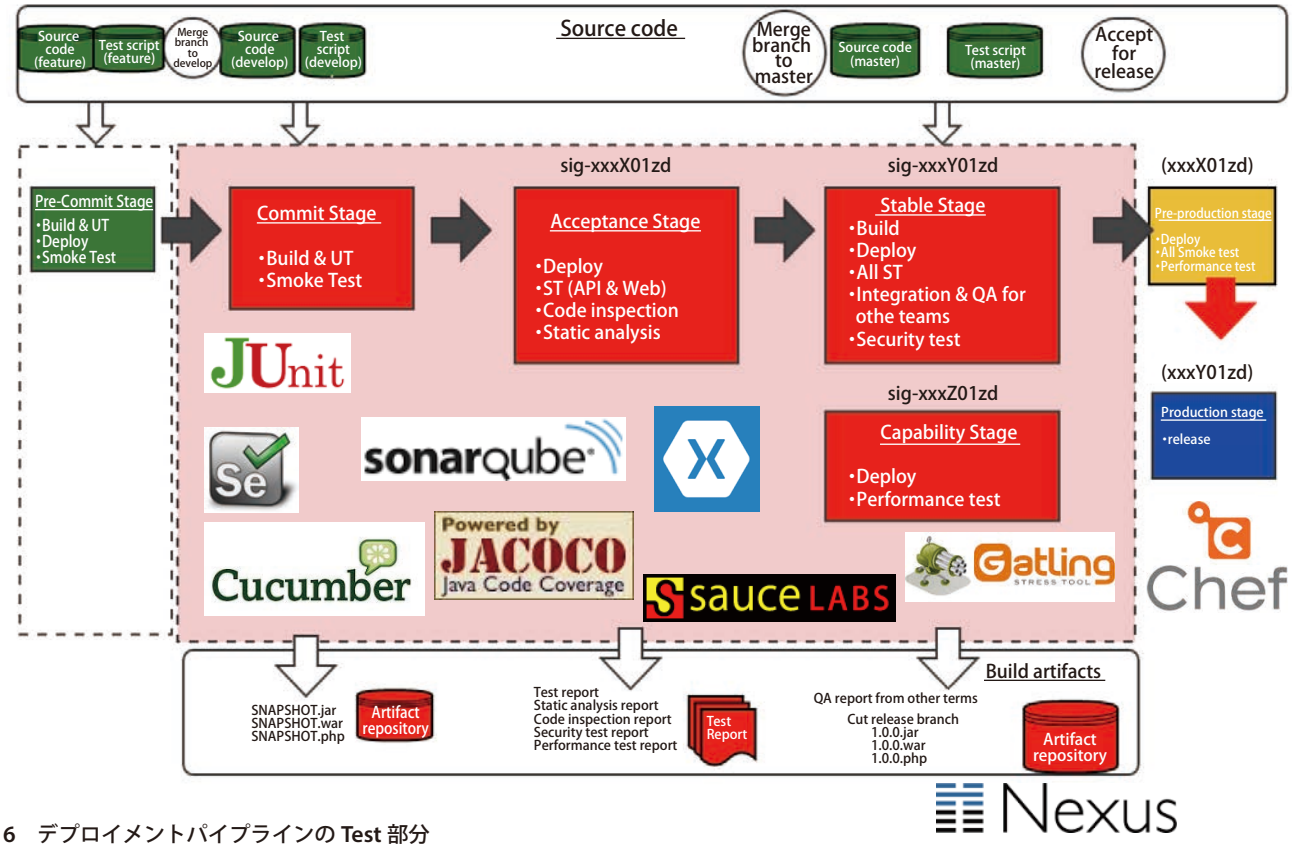

図 6　デプロイメントパイプラインの Test 部分

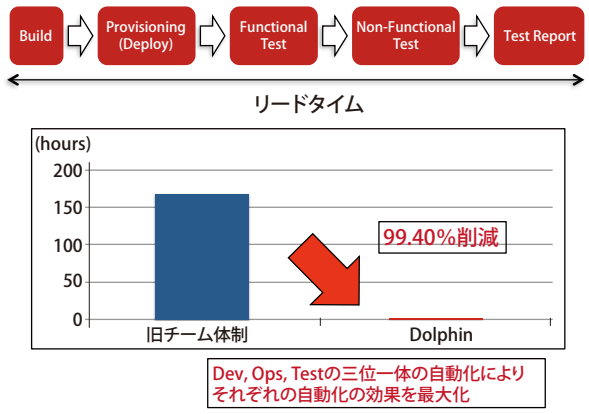

図 7　全体工程のリードタイムを削減

表 1　エンタープライズアジャイルの課題への対応

A：チーム運営上の課題			
A-1 プロジェクチーム解散時の記憶喪失	スモールチームの維持	事例共有会，技術勉強会，コミュニティ	自動化とリポジトリの共有
A-2 チームの熟成不足		スクラム研修，メンバのスキル分析	定期的な計画と振り返り
A-3 スキルとプロセス合意の不足	事例共有会，技術勉強会，コミュニティ		(今後)テストのコーディング研修
B：チーム外との調整の課題			
B-1 チーム外との調整負担	明確なプロダクトオーナー	事例共有会，技術勉強会，コミュニティ	関係者への成果物のデモとフィードバック
B-2 承認プロセスが足を引っ張る		マネジメントからの支援	
B-3 予算管理や受発注契約と開発プロセスの不一致			
C：組織全体の課題			
C-1 評価制度のミスマッチ		マネジメントからの支援	(今後) 人材部門との協業
C-2 指導者の不足	海外カンファレンス参加	新人研修を通じた情報共有とリーダ育成	
C-3 情報共有不足と不信感	事例共有会，技術勉強会，コミュニティ		

プットを続けるために，積極的に省力化・自動化を行う意識が高まった．

インフラの自動化についても，数年に渡って Chef の導入から地道に行ってきた成果を活用している．必要なスキルは関係者の中に備わっていたため，事例共有会や技術勉強会をきっかけに，部署の垣根を越えて協力し，フレームワークとしてまとめ上げた．

また，DevOps やアジャイルの文化やスキルを学べるよう，カンファレンスなどでの発表，技術勉強会の開催や参加，また技術ニュースレターの配信などが進められた．

チームの協業の方法としてはスクラムを採用し，研修の開催をして知識を共通化するとともに，定期的な計画と振り返りを行ってチーム自身が自ら考え，改善する文化を作った．スキルマップを用いてチームメンバのスキル分析を行って相互に補完する方法を話し合った．

今後の課題としては，より良いテストを書くためのテストのコーディング手法の普及が挙げられる．

4.2　チーム外との調整の課題への対応

チーム外との調整については明確なプロダクトオーナーを立て，調整・説明・承認などのプロセスを進めた．

ミドルマネージャがプロジェクトリーダとして直接指揮を執り，承認プロセスと予算システムについて連携部署との交渉を進めた．マネジメントからの支援の効果の定量測定は難しいが，一般的にマネジメント層の支持の不足でアジャイルが失敗するケースが多いことを考えると，意味があったと考えている．

成果については関係者への成果共有会を開催し，実機デモを行った．担当執行役員も参加しフィードバックを行うなど，継続的な支援を行った．

4.3　組織全体の課題への対応

人事評価については従来通り，マネジメントが個別の人事評価への成果の反映を行っている．しかし，チームとしての成果の評価や，チーム活動に必要なスキルセットの分析については，人材部門と協業し，さらに検討を進めていく必要がある．

楽天はエンジニアの海外カンファレンス参加を支援しており，米国で行われた Agile Conference での Microsoft と Yahoo! の事例に刺激を受け，これが社内の DevOps の活動の連携に繋がった．彼らを自社カンファレンスに講演者として招いた．併せて社内講演や，担当との個別セッションを実施している．社員は英語が堪能で，また日本人社員も多いため，国内外問わず，実践者と直接の相談が可能であった．

技術勉強会としては，Dev 視点でインフラの事例を共有する DevOps 会議を実施した．部門を越えて 20 名ほどのエンジニアが集まり，社内で利用可能なインフラの事例を共有し合った．実際に使ってみてどうなのか，今後の体制はどうなりそうかといった，技術にとどまらない業務情報が交換された．

社内外の事例や新技術を共有する TechTalk や GogoTalk，開発部内の共有会（〆会），毎週行われる全社会議 Asakai の場で DevOps や関連技術の事例や技術紹介を行い，部署や立場を越えてノウハウや課題認識の共有，相談が行われるようにしている．Ops の自動化担当がサーバ構築の自動化について発表した際には，それま

でDevからは保守的に見えていたOpsが，実際は最先端の自動化を行っていることが分かり，それからエンジニア同士の協働が始まった．

エンジニアの新人研修のプログラムを変え，各部門の技術担当がリーダとなって直接指導するプロジェクトベース研修を行った．参加者への技術普及と，リーダの育成を両立する試みである．リーダ意識の向上につながったとの意見が出ている．

「評価制度のミスマッチ」については，今後の課題であり，人材部門や標準化部門と協調して取り組んでいく予定である．

5. おわりに

Dolphin Projectは，Dev/Test/Opsの自動化施策をまとめ上げ，アプリケーションのデプロイパイプラインにかかる時間だけでなく，テストとサーバ構築の時間をトータルに短縮する仕組みを構築した．

本稿では，エンタープライズアジャイルの潮流と，楽天でのアジャイル導入の歴史を紐解きながら，その概要と成果を分析した．

謝辞　本稿の作成にあたり，発表の許可ならびに，技術的なご助言をいただきました，プロジェクト責任者のFernando Paulo氏，プロジェクトリーダである千葉俊彦氏に深く感謝いたします．

また，プロジェクトを進めるにあたり，技術的なご助言と議論への参加をいただきました，菅野裕氏，Chen Xi氏，高橋兵太氏に深く感謝いたします．

参考文献

1) アジャイルソフトウェア開発宣言，http://agilemanifesto.org/iso/ja/（2016年5月17日現在）
2) Cockburn, A.: Agile Software Development: The Cooperative Game (2nd Edition), Addison-Wesley Professional（2006）.
3) Agile 2010 Program “Enterprise Improvement”，http://agile2010.agilealliance.org/enterprise.html（2016年3月10日現在）
4) Scaled Agile Framework (SAFe)，http://www.scaledagileframework.com/（2016年5月17日現在）
5) Disciplined Agile Dlivery，http://www.disciplinedagiledelivery.com/（2016年5月17日現在）
6) Large Scale Scrum (LeSS)，http://less.works/（2016年5月17日現在）
7) Nexus Framework，https://www.scrum.org/Resources/The-Nexus-Guide（2016年5月17日現在）
8) Scaling Agile at Spotify with Tribes, Squads, Chapters & Guilds，http://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify（2016年5月17日現在）
9) アジャイル宣言の背後にある原則，http://agilemanifesto.org/iso/ja/principles.html（2016年5月17日現在）
10) Guckenheimer, S. and Loje, N.: アジャイルソフトウェアエンジニアリング，日経BP社（2012）.
11) Allspaw, J. and Hammond, P.: 10 Deploys Per Day，http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr（2016年5月17日現在）
12) Guckenheimer, S.: Our Journey to Cloud Cadence, Lessons Learned at Microsoft Developer Division，https://www.microsoft.com/en-us/download/confirmation.aspx?id=46920（2016年5月17日現在）
13) Guckenheimer, S. and Ushio, T.: Microsoftが実践したScrum導入7年間の旅．そしてDevOps，https://docs.com/ushio-tsuyoshi/7001/microsoftscrum7devops（2016年5月17日現在）
14) Kraay, E. and Zvinyatskovsky, S.: Yahoo: Pulling an Elephant out of a Tarpit, Rakuten Technology Conference 2015，http://rakutentechnologyconference2015.sched.org/event/4Pbx/yahoo-pulling-an-elephant-out-of-a-tarpit（2016年5月17日現在）
15) 藤原　大：アジャイルリーダーシップと組織改革～楽天のアジャイル開発というリアル～，Developers Summit 2012，https://speakerdeck.com/daipresents/16-b-5（2016年5月17日現在）
16) Ito, H.: Technology-Driven Development: Using Automation and Development Techniques to Grow an Agile Culture, Agile 2014 Experience Report，https://www.agilealliance.org/wp-content/uploads/2015/12/ExperienceReport.2014.Ito_.pdf（2016年5月17日現在）
17) 及部敬雄：アジャイルヒーローは誰だ!?，Developers Summit 2015，http://www.slideshare.net/TakaoOyobe/20150220-developers-summit-2015oyobe（2016年5月17日現在）
18) 荻野恒太郎：15-B-11 安心なサービスの品質改善を実現する為の継続的システムテスト，先進的な設計・検証技術の適用事例報告書2015年度版，https://www.ipa.go.jp/files/000049409.pdf（2016年5月17日現在）
19) 荻野恒太郎，古川貴朗：三位一体の自動化で壊せDevとOpsの壁～アラサーエンジニアの挑戦～，Developers Summit 2016，http://www.slideshare.net/rakutentech/devops-58450617（2016年5月17日現在）
20) Naoi, K. and Shinoda, T.: Rakuten Travel Engineer Talks DevOps, Rakuten Technology Conference 2015，https://youtu.be/UqWFncjrfUs（2016年5月17日現在）
21) Coplien, J. O.：生成的開発プロセス・パターンランゲージ，http://patterns-wg.fuka.info.waseda.ac.jp/japanplop/Translations/GDP/patternIndex.htm（2016年5月17日現在）
22) Farley, D. and Humble, J.（著），和智右桂，高木正弘（翻訳）：継続的デリバリー，翔泳社（2012）.
23) （独）情報処理推進機構技術本部ソフトウェア高信頼化センター：非機能要求グレード，https://www.ipa.go.jp/sec/softwareengineering/reports/20100416.html（2016年5月17日現在）
24) Systems and Software Engineering –Systems and Software Quality Requirements and Evaluation (SQuaRE) –System and Software Quality Models.

川口 恭伸（正会員）yasunobu.kawaguchi@rakuten.com
楽天（株）アジャイルコーチ．1997 年北陸先端科学技術大学院大学情報処理研究科修了．（株）QUICK にてデータメンテナンス業務，　金融機関向け B2B アプリケーション開発，仮想化開発インフラ構築，スクラムでの業務アプリ開発などに従事した後，2012 年より現職．

荻野 恒太郎（非会員）kotaro.ogino@rakuten.com
楽天（株）．2010 年名古屋大学修了．楽天（株）入社．検索基盤開発プロジェクトにて，開発とシステムテスト自動化を 5 年間担当．2014 年より DevOps やテスト自動化の推進を行っている．

古川 貴朗（非会員）takaaki.furukawa@rakuten.com
楽天（株）．2010 年会津大学修了．楽天（株）入社．インフラエンジニアとして，サーバのプロビジョニング作業を 5 年間担当．2013 年よりサーバの構成管理ツールの導入やインフラのコード管理についての推進を行っている．

採録決定：2016 年 5 月 17 日
編集担当：斎藤正史（金沢工業大学）