アジャイルからDevOpsへ
― 歴史的観点から見たDevOpsへの進化 ―

開発と運用が連携し、アプリケーションのリリース・サイクルの短期化を目指す「DevOps」。この新たなテーマへの関心が近年になり急速に高まっています。背景には市場の変化がさらに加速する中、業務基盤であるアプリケーションの新規開発や変更をより早いペースで可能にするアジャイルの重要性が、あらゆる企業で増していることがあります。またクラウドや仮想化環境を中心とする新しい運用インフラの登場によって、より早いサービスを提供できることになったことも大きな要因です。本稿では、歴史的観点からアジャイルとそのキーとなるテクノロジーを振り返り、DevOpsへ発展していく流れを解説します。

1. 日本におけるアジャイルに対する意識

IBMが2013年に発表した「C-Suite Study 2013」[1]では、「CEOが考える自社に影響を及ぼす外部要因」のトップは、2012年に引き続き「テクノロジー」になりました（**図1**）。また特徴的なところでは「市場の変化」がトップ2に返り咲いています。日本のCEOに限定すると、逆に「市場の変化」「テクノロジー」の順になり、最新のITを用い、いかに他社に先駆けて競争力のある製品・サービスを迅速に市場に出すかがカギになっていることが分かります。

市場の変化、ビジネス・ニーズの変化に対し、柔軟性とアジリティーを持った製品・サービスを提供する。こう聞いた時、「アジャイル」を思い浮かべる人は多いのではないでしょうか？　1990年代後半から提唱・実践され始め、2001年に宣言された「アジャイルソフトウェア開発宣言（アジャイル・マニフェスト）」をよりどころに、さまざまなアジャイル手法が開発されました。

ここに面白いデータがあります。2009年から開催されているAgile ConferenceのIBMセッションでは、毎年アンケートでアジャイル開発の取り組み状況を聞いていますが、2013年では実施中企業は38%、取り組み予定を含めると約8割の企業がアジャイルを採用しようとしています。2009年の結果では同数値は約35%であり、5年を経過したいま倍以上の企業でアジャイルに対する意識が高まっていることになります（**図2**）。日本では大規模な一括請負契約が多く、欧米とは契約慣習が違うことからアジャイルの適用が難しいと言われてきましたが、IPAから非ウォーターフォール型開発に適したモデル契約書が公開されたり[2]、各種イベントでアジャイル適用の成功、失敗事例が講演されることで、開発ベンダーだけでなくシステムを発注するユーザー企業もアジャイルに注目してい

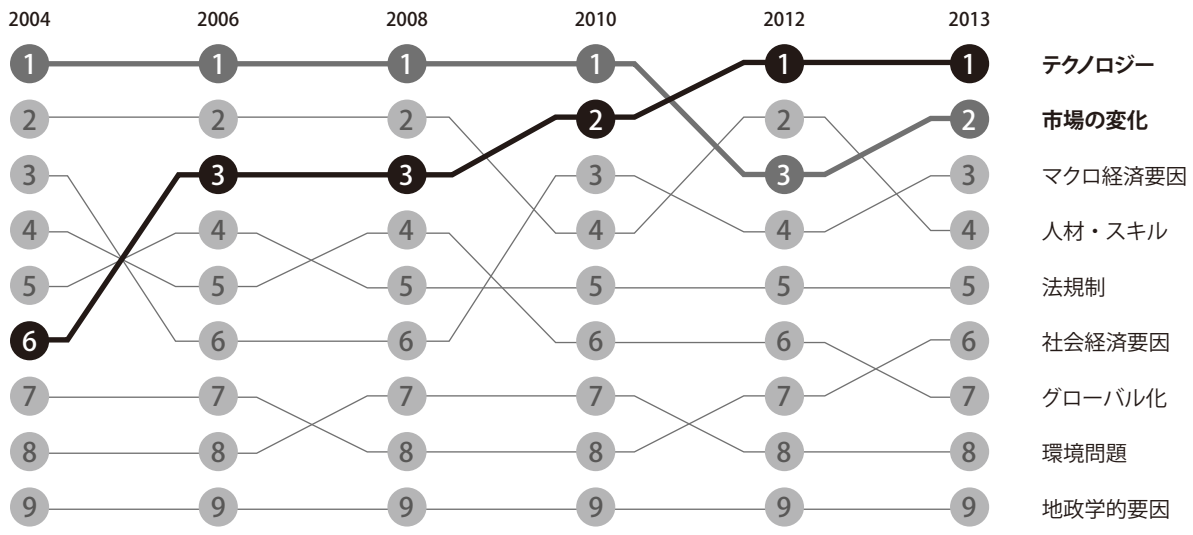

図1. IBM Global CEO Studyに見る自社に影響を及ぼす外部要因 (2004–2013)

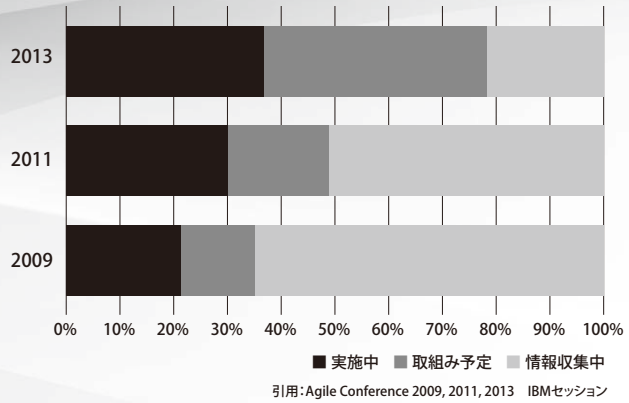

図2. アジャイル開発の取り組み状況の遷移

ると言えます。

ここからアジャイルの発展を振り返り、そこで発生したさまざまな課題を新たな手法やテクノロジーによって乗り越えた、DevOpsへとつながっていく歴史的経緯を見ていきたいと思います。

2. 歴史的観点から見たDevOpsへの進化

（1）2001年～2005年：XPの浸透

アジャイル開発の中で有名なものにXP（エクストリーム・プログラミング）があります。5つの価値と開発チームが行うべきいくつかのプラクティス（習慣、実践）が定義されており、ドキュメントよりもソースコードを、組織的開発の歯車となることよりも個人の責任と勇気を重んじる、人間中心の開発プロセスであるとしています。1999年にKent Beckが執筆した「XP入門」[3]により特に開発者によって熱狂的に受け入れられました。ドキュメントを不要だとはしていないまでも動くソースコードを重視し、またアジャイルの特徴である短期的な反復開発により細かく計画を見直し修正していくことで、顧客から早期にフィードバックを得ようとしました。

XPのプラクティスの数は初期には12でしたが、数度の改定を経て19に増えています。この中でもDevOpsにつながる重要なものとして「継続的インテグレーション（Continuous Integration：CI）」[4]があり、開発の生産性や品質向上を目指しました。

CIの適用により、

- プログラマー当たりの開発LOC（コード行数）が90％上昇
- 欠陥率が36％低下

という興味深いレポート[5]が公開されています。

継続的インテグレーション（Continuous Integration：CI）

システムの結合は非常に労力がかかる作業で、プロジェクトの終盤で行うと品質問題を引き起こし、多くの場合プロジェクトの遅延の原因になっています。このことからCIはチーム・メンバーがそれぞれの作業を早期に次々と結合し、結合にかかわる不具合を早期に発見しようとする考え方です。通常は自動化のためのCIツールを導入し、開発者が構成管理ツール（SCM）に開発したコードを提出するとそれを検知し、あらかじめ設定されていた作業（コンパイル、ビルド、テストなど）を自動的に実行します。その結果はダッシュボードやメールなどの手段で開発者にフィードバックされます（**図3**）。コードを提出後にバックグラウンドで常に指定した作業が実行されることで、デグレードを防止し、修正部分を安全に取り込むことができます。CIを実現するJenkinsなどのオープンソースや多くの商用製品が登場し、IBMでもRational製品群（Rational Team ConcertやRational Build Forge）でサポートしています。

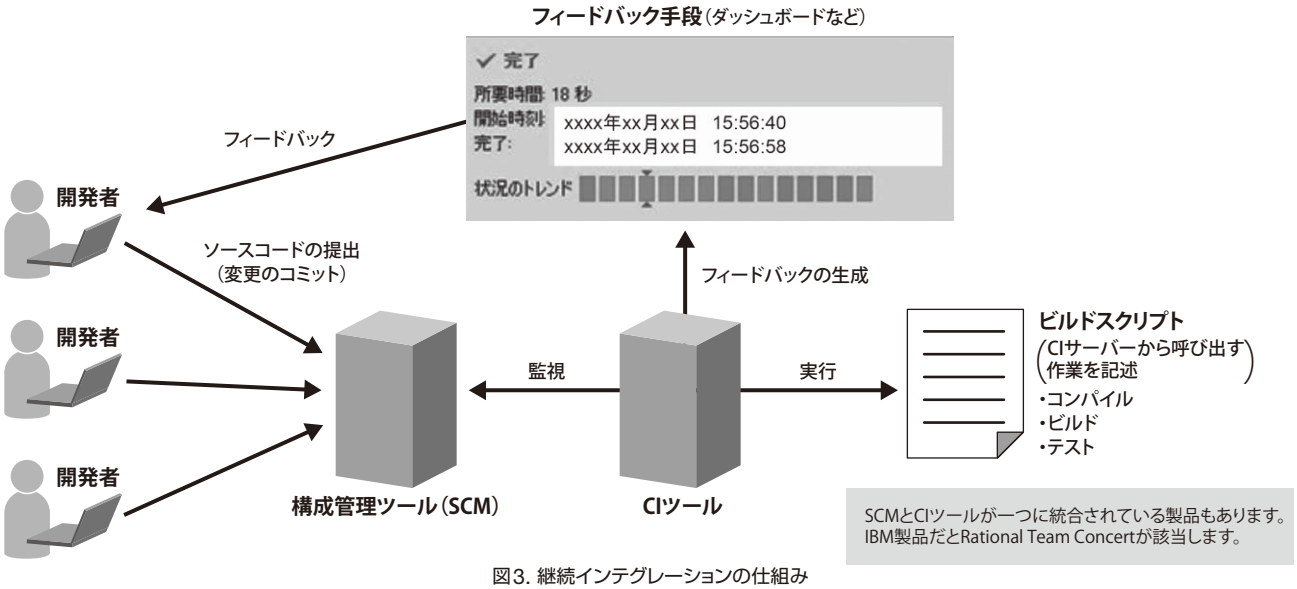

図3. 継続インテグレーションの仕組み

(2)2006年～2011年:スクラムと継続的デリバリー

XPはプラクティスの集合体であり、開発の流れを明確に示していません。一方、1993年Jeff Sutherland、John Scumniotales、Jeff McKennaによって提唱されたスクラムは開発プロジェクト管理の枠組み(フレームワーク)を提供します。XPのように具体的なプラクティスや開発手法には踏み込まないため理解しやすく、さまざまなプロジェクトに適用しやすいことから、現在世界中で最も多く使われているアジャイル手法です。そのためプロジェクトによっては、スクラムとXPを組み合わせるハイブリット型を採用するケースもあります。スクラムの用語や仕組みは多くの書籍やWebサイトで説明されていますのでここでは省略します。

スクラム+XPまたCIにより、ビジネス価値のあるソフトウェアを頻繁にリリースしていく土壌ができました。しかしここで新たな疑問が浮かんできます。

- ●開発で作った成果物をテスト環境～ステージング環境～本番環境に、失敗せずより迅速にリリースするにはどうするか?
- ●開発期間が1年半の場合に構築したリリース・プロセスを1ヵ月サイクルに短縮したとしても機能するか?

2011年にIBMが調査会社に委託して行ったサーベイ結果では、十分なテストができないため障害が本番に入ってしまうことを経験した企業が45%。またアプリケーションのテスト環境構築に時間がかかりすぎていると感じている企業が41%と、開発チームのCIだけでは解決できない課題が出てきました。そこでCIをさらに推し進める考え方として継続的デリバリー(Continuous Delivery : CD)[6]が登場しました。

継続的デリバリー(Continuous Delivery : CD)

継続的デリバリー(CD)とは、CIを通じて出来上がったビルド済み、テスト済みの成果物を、テスト環境やステージング環境、本番環境に継続的にリリースすることを示します。目的は、開発から運用に信頼されたソフトウェアを渡し、さらにプロセスを可視化し大部分を自動化することにあります。価値のあるソフトウェアを早いうちから徐々に提供することは、顧客のフィードバックを増やし、ビジネス・ニーズの変化に柔軟に対応できることにつながります。またフィードバック情報は、開発だけでなくテストや運用チームにも伝搬され、組織間のコラボレーションを促進します。当初はアジャイルに適用するために開発されたものですが、フォーカスしているのが適切な可視化と素早いフィードバックによる共同作業の改善なので、ウォーターフォールなどの非アジャイルにも適用されています。

このようにビルド、デプロイ、テスト、リリースを通した全体のプロセスがエンド・トゥ・エンドで自動化されるように実装されることを、デプロイメント・パイプラインと呼びます。ここで重要なのは「ビルドは1回限りとし、できた成果物をあらゆる環境で利用する」ことです。実際のプロジェクトでは環境ごとにコンパイル、ビルドを行うことがありますが、何らかの差分が紛れ込むリスクや設定の不備でアプリケーションの振る舞いが変わるかもしれません。これが原因となったバグが本番まで残ることもよく聞かれる話です。またシステムを構成するモジュールの各バージョンを集めたものをスナップショットと言い、このセットをコピーしていくことで品質を保証します(**図4**)。

IBMでは2013年にCDをサポートするツールを提供するUrbanCode社を買収し、Rational製品群に統合しました。

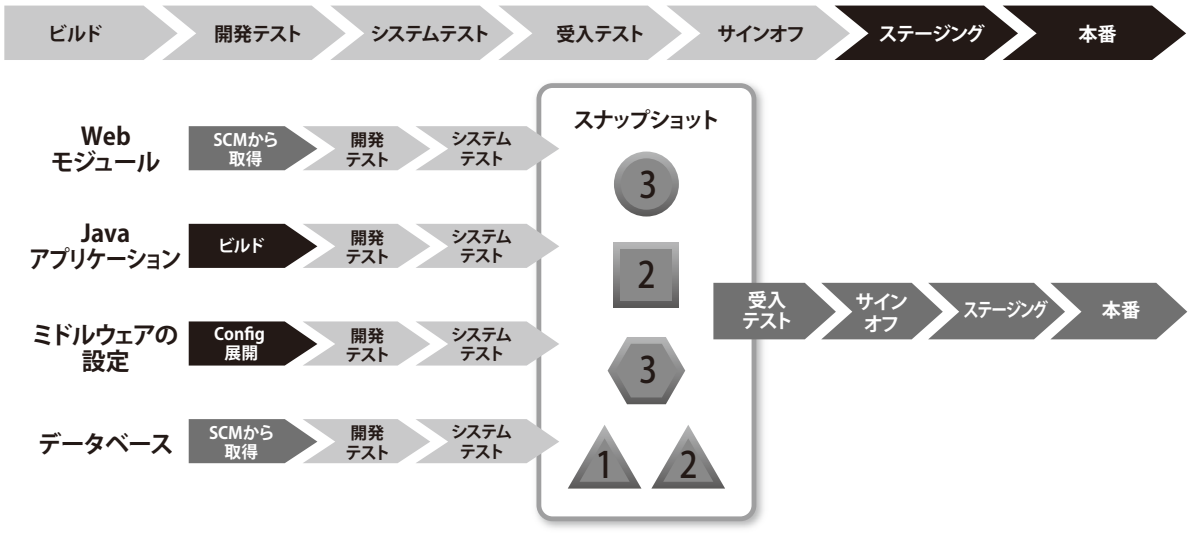

図4. デプロイメント・パイプラインとスナップショットの活用

CDを支える重要なテクノロジーとして、クラウドや仮想化環境があげられます。クラウド・サービスの普及により、インフラの調達・提供期間が大幅に短縮されました。OSやミドルウェアの仮想イメージの配布を手動で行うことはミスも多く、失敗したときの環境の再作成も困難で現実的ではないと思います。そこで「Infrastructure as Code（コードとしてのインフラストラクチャー）」という考えが生まれてきました。つまり、今まで手順書を基に手動で行ってきたインフラの構成管理を、スクリプトや外部ファイルに記述し自動的に行えるようにしたのです。これらはソフトウェアのコードのように管理でき、変数化することで柔軟にプロビジョニング（展開）することも可能です。またイメージをテンプレート化して管理することで、必要に応じていつでも複製できるようになりました。IBMでは、IBM SmarterCloud Orchestrator（SCO）やIBM PureApplication System（IPAS）でこの考え方を取り入れています。またオープンソースではChefやPuppetが有名です。

デプロイメント・パイプラインに仮想化技術をどのように取り入れられるでしょうか。システムのバグが発見されたとします。その原因にありがちなものとして、

●アプリケーション・コードのバグ
●テストの実行ミス、無効な期待値の指定
●デプロイ、リリース手段の問題
●インフラ（OSやミドルウェア）の問題

があり、インフラの設定も要素の一つになります。再現できるインフラが残っていて自由に利用できればよいですが、一から構築しなくてはならないケースも出てきます。そんな時、仮想化技術を利用することが効果的です。

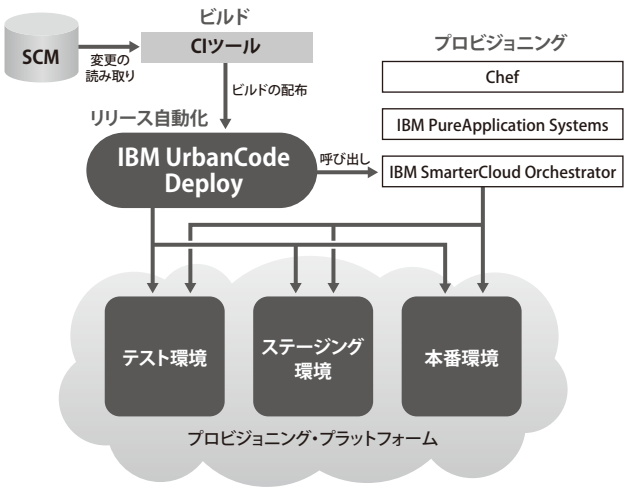

図5. IBM UrbanCode Deployを中心とした
継続的デリバリーの実現

図5はCDをサポートするIBM UrbanCode Deployと、仮想化のプロビジョニングをサポートするSCOやIPASを連携したイメージです。SCOやIPASで作成したテンプレートをUrbanCode Deployにインポートします。UrbanCode DeployからSCOやIPASを呼び出して適切な仮想イメージをプロビジョンして構築した環境に対し、デプロイメント・パイプラインを走らせるということも既に実現しています。SCOやIPASの代わりにChefを呼び出して、プロビジョンを行うことも可能です。

（3）現在：そしてDevOpsへ

ここまでCI、CDというDevOpsの重要なテクノロジーを見てきましたが、疑問がわいているかもしれません。

CDで開発から運用に信頼されたソフトウェアを渡し、本番環境にリリースできるようになりました。また、デプロイメント・パイプラインの状況は可視化され、共有できるようになりました。これこそDevOpsではないでしょうか？

DevOpsは、Development（開発）とOperation（運用）を組み合わせた言葉です。そのため開発と運用を一体化するというコンセプトと考えると上記もDevOpsに他なりません。

一方、DevOpsの目的について改めて考えてみます。CEOの関心が、最新のITを用い、いかに他社に先駆けて競争力のある製品・サービスを迅速に市場に出すかにあるとすると、

「高品質で価値のあるソフトウェアを効率的で素早く、信頼できるやり方で継続的にリリースし、開発して終わりではなく、ユーザーの利用状況やシステムの稼働状況をフィードバックし、次の商品提供や機能改善につなげること」

と言えるのではないでしょうか。実際、2013年7月に米調査会社のForester Researchが発行したレポート[7]によると、

●DevOpsは開発や運用だけでなく、顧客や事業部門（Line of Business）を含めたコラボレーションに広がり、市場の変化に対応した迅速なリリースが求められている
●顧客への直接のフィードバックがイノベーション・サイクルをドライブする

といったように、DevOpsは開発や運用だけにとどまらず、ビジネスの視点まで含めるべきと提唱しています。

IBMではDevOpsを、ビジネスの視点に基づくソリューションの企画・設計から始まり、開発とテスト、デプロイ

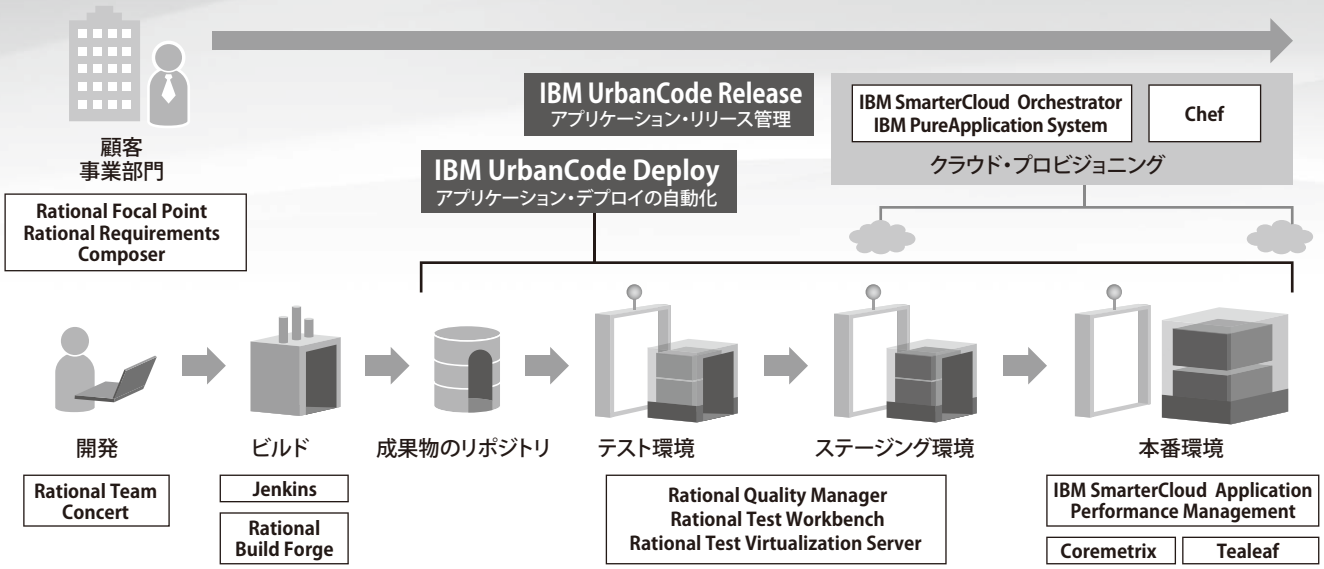

図6. DevOpsライフサイクルを支えるツール

とリリース、モニタリングとその結果を関係者にフィードバックするというデリバリー・ライフサイクル全体を最適化することとして位置付けています（詳細は今号の「海外寄稿：Introducing DevOps」[8]をご参照ください）。開発ソリューションを提供するRational製品を核にして、前述のUrbanCodeやSCO、IPASなどの仮想化技術、インフラのモニタリングをするIBM SmarterCloud Application Performance Management、Webマーケティング最適化ツールであるCoremetrix、顧客体験分析ツールのTealeafなど、DevOps実現に向けトータルなソリューションを提供することができます。もちろん既存で使っているツール（例 JenkinsやChef）を組み合わせて利用できるオープン性も備えています（**図6**）。

もちろんDevOpsはツールを導入するだけでできるものではありません。組織間のプロセスや意識が異なるため、そのギャップを埋めるための「文化の醸成」という面が強調されることも多く、多大な労力や時間がかかりそうなため導入に躊躇している企業が多いことも確かです。

IBMでは、これまでの経験からDevOpsのプラクティスと成熟度モデルを定義し、企業が現在成熟度のどこに位置しているか（AS-IS）、今後どこを目指すのか（To-Be）をお客様と一緒に議論し、フォーカスする領域を定め、今後のロードマップ策定の支援をすることができます（**図7**）。

3. 今後のアジャイルと DevOps

本稿では歴史的観点からアジャイルを振り返り、DevOpsへ発展していく流れを見てきました。最後に今後のアジャイルとDevOpsについて考えてみます。

現在のアジャイル開発の主流はスクラムとXPですが、一般的には同じ場所で作業し、シンプルな開発をする小規模のチーム（15名未満）に効果があると言われています。密なコラボレーションを行う自己組織化（Self-Organized）されたチームにするには、この規模が限度と言われてきました。しかし最近では、製品の品質やチームの効率、納期の改善などにより、より大規模のチームがそれぞれの環境でアジャイル原則の導入を検討するようになってきています。IBM Software Groupには全世界に5万人を超えるメンバーがいますが、2005年よりグローバル規模でアジャイル・プラクティスの導入を決めプロセスの変革に

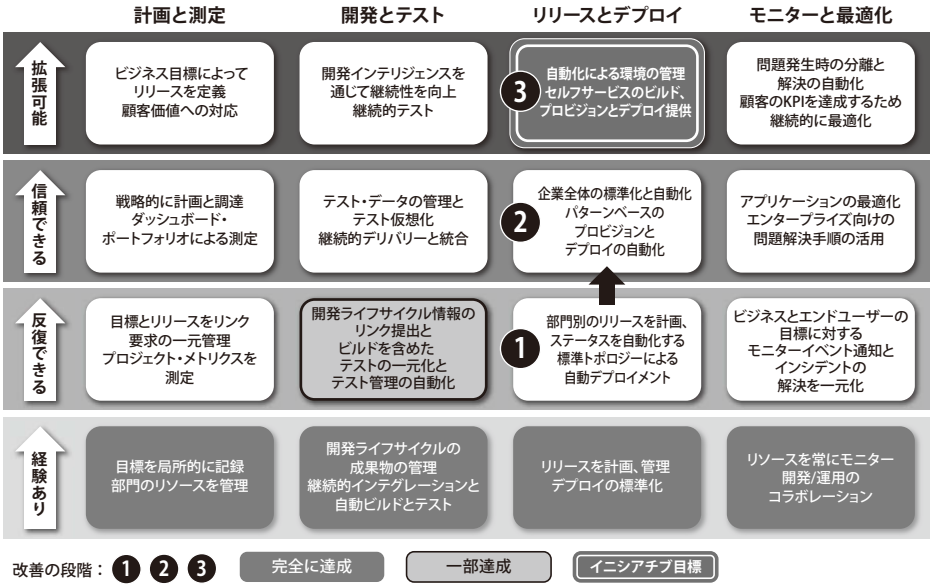

図7. DevOpsプラクティスをベースにした成熟度モデルと改善目標の例

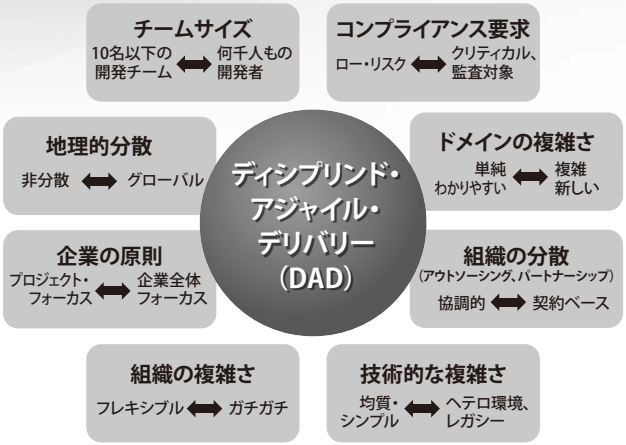

図8. ディシプリンド・アジャイル・デリバリーによるアジャイルの適用範囲の拡大

挑みました。結果として成果物の再利用性の向上によるコスト削減、品質の向上、コミュニティー・サイトの活用によるコラボレーション促進を達成しました[9]。これらの経験を基に2012年には「ディシプリンド・アジャイル・デリバリー（Disciplined Agile Delivery：DAD）」[10]を刊行し、チームの規模、地理上の分散、組織の分散、ガバナンスなどエンタープライズに対応するためのアプローチを提唱しています（**図8**）。またトヨタ生産方式を源流に持つ「リーン」や「カンバン」と、「Scrum」をハイブリッドした「Scrumban」という手法が出現しています。アジャイルは試行錯誤しながらこれまでの課題の解決に挑戦しながら進化し続けており、日本企業でアジャイルに対する意識が高まり適用が増えることは間違いないと思います。

一方DevOpsに話を戻すと、開発と運用の協業をいかにして行うかという議論が続いています。アジャイル開発やCI、CDをベースとしてDevOpsは進化してきた一方、運用ではITサービス・マネージメントのプロセスであるITIL（IT Infrastructure Library）を重視し、相容れることが難しいと言われてきました。しかし最近では、ITILにDevOpsの要素を組み込むことでITIL自体も改善し、頻繁な変化に対応することでより一層の価値を提供する必要性が語られています[11]。日本でも2013年11月に開催された「第10回itSMF Japanコンファレンス/EXPO」で、ITILとDevOpsを議論する講演や寄稿が多数あり、総じてDevOpsとITILの融合が提唱されています。事実、アプリケーション開発・保守の業務ではアジャイル開発が採用されており、IBM社内IT部門では2008年からRational Team Concertをインフラとし、リモート環境によるオフショア開発でアジャイルを適用しています。組織を挙げて推進し、コスト削減や品質向上だけでなく、いかに早くビジネスの価値に変えるかという「Time to Value」につなげています[12]。

DevOpsの取り組みにかかわるステークホルダー（利害関係者）が目的・目標を共有し、それに向けて密にコラボレーションすることで、ニーズに合った製品・サービスを迅速に市場に提供し、日本の競争力向上につながればと切に思います。IBMは、今後もDevOps実現に向けてより一層、製品やサービス、コンサルティングを強化して支援してまいります。

［参考文献］
［1］ IBM : C-Study 2013, 2013年
http://www-935.ibm.com/services/jp/ja/c-suite/csuitestudy2013/
［2］ IPA：非ウォーターフォール型開発に適したモデル契約書の改訂版を公開、2013年
http://www.ipa.go.jp/sec/softwareengineering/reports/20120326.html
［3］ Kent Beck（著）、長瀬嘉秀（監訳）、「XPエクストリーム・プログラミング入門」, ピアソン・エデュケーション、2000年
［4］ Paul M. Duvall, Steve M. Matyas, Andrew Glover（著）、大塚庸史ほか（訳）、「継続的インテグレーション入門」、日経BP社、2009年
［5］ MacCormack, A.ほか : Trade-offs between productivity and quality in selecting software development practices、IEEE Software、2003年
［6］ David Farley, Jez Humble（著）, 和智 右桂ほか（訳）、「継続的デリバリー」、アスキー・メディアワークス、2012年
［7］ Kurt Bittner : Continuous Delivery Is Reshaping The Future Of ALM, Forrester Research, 2013年
http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=RAL14092USEN
［8］ Sanjeev Sharma : Introducing DevOps - Adopting DevOps to achieve Continuous Innovation -, PROVISION No.80, 2014年
［9］ Julie King : アジャイル変革の実施, 2012年
http://www.ibm.com/developerworks/jp/rational/library/common/julie-king-interview/
［10］Scott W. Ambler, Mark Lines（著）、藤井智弘（監修）、「ディシプリンド・アジャイル・デリバリー」、翔泳社、2013年
［11］Grischa Ekart : DevOpsとITILを知的な方法で統合, InfoQ, 2013年
http://www.infoq.com/jp/news/2013/05/integrate-devops-itil
［12］吉竹 正貴 : Time to Value革命- IBM 社内 IT 部門のアジャイル開発、クラウドソーシングの取り組み -, PROVISION No. 66、2010年

日本アイ・ビー・エム株式会社
ソフトウェア事業ラショナル事業部
ITテクニカル・セールス
アドバイザリー ITスペシャリスト

黒川 敦
Atsushi Kurokawa

［プロフィール］
2002年日本IBM入社。ソフトウェア事業にてWebSphere Application Severのテクニカル・セールスとして活動。その後お客様担当SEとしてSOAプロジェクトに参加し、2009年からRationalテクニカル・セールスとして、ソフトウェア・ライフサイクル全般のアプリケーション開発支援を担当。2013年から日本におけるDevOpsとUrbanCodeの立ち上げをリード。developerWorks Rational管理者。中小企業診断士。