クラウド推進をトリガーにしたシステム開発改革

座間哲也

CONTENTS

要　約

1　日本においては、DXと呼ばれる新技術によるビジネスや社会の変革が諸外国と比較して遅れている。また、日本企業のクラウド導入は非常にスローペースであり、導入済み企業であっても既存システムの置き換えにとどまっていることが指摘されている。

2　「既存業務の効率化」を目的としていたこれまでのシステム開発と、「新しい価値の探索や発見」を目的とするDXシステム開発では、開発の進め方や開発チームに求められる行動原理が大きく異なる。

3　クラウドを既存システムの置き換えに使うのではなく、クラウドならではの特性を活かす形で活用することにより「高速な仮説検証」を実現し、DXシステム開発の目的である「新しい価値の探索や発見」を促進できる。

4　高度にクラウドが活用されたシステムの開発には、これまでのような大人数のプログラマーは不要である。代わりに、クラウドやオープンソースの特性をよく理解し、適切に組み合わせて要求を実現することができる高スキルな少数のプログラマーが必要である。

5　DXが進むにつれて、構築されるシステムのユーザーは多様化・複雑化する。このような状況でユーザーに継続的に価値を提供するには、これまで以上にユーザーからのフィードバックの観察と素早い対応が不可欠である。

6　DXに向けたシステム開発改革を成功させるためには、「もう一つのDX」といわれるDeveloper eXperience（開発者体験）を向上させ、国内外の企業や公的機関との優秀なプログラマーの奪い合いを制する必要がある。

Ⅰ DX推進力とクラウド活用力の相関関係

経済産業省の「DXレポート」が「2025年の崖」として問題提起しているように、いわゆる「DX（デジタルトランスフォーメーション）」と呼ばれる新技術によるビジネスや社会の変革の推進が、日本は欧・米・中といった諸外国と比較して大きく遅れている。また、日本企業のクラウド導入は非常にスローペースであり、導入済みの企業においても、多くはその利用用途が既存のシステムの置き換えにとどまっていることがガートナージャパンの2020年５月14日プレスリリースなど、多数の調査結果で指摘されている。

クラウド移行を進める上での考え方の一つに、「リフト＆シフト方式」と呼ばれる手法がある。まずは、オンプレミスのシステムを単純にクラウドに引越（＝リフト）し、その後、クラウドに適した形に作り変える（＝シフト）手法である（表１）。日本においては、多くの企業が第一段階のリフトで停滞しており、クラウドならではの価値を発揮するシフトの段階に至っていない。一方で、諸外国の企業ではリフトの段階は過ぎており、シフトによって情報システムとビジネスとの関係やシステム開発そのものを変革し、新たな価値の創出に成功している。

本稿では、日本におけるDX推進力の低迷とクラウド活用力の低さの相関関係を軸に、クラウド化をトリガーにしたDXシステム開発改革の推進について提言する。

Ⅱ これまでのITとDXの相違点

これまでのITは、「既存業務の効率化」を目的に構築されてきた。システムが創出すべき効果は比較的明確であり、計画可能であった。従来型のシステム開発では、開発工程は数年単位の遠い先まで綿密に計画され、その計画を手戻りなく正確に実行することを行動原理としたマネジメントが行われてきた。確立されたマニュアルによって行われる「組み立て作業」を大人数で実行することが可能であり、１つのプロジェクトに大人数のプログラマーが投入されてきた。また、一度構築したシステムを長い年月にわたって使い続け、一定の期間を経た後に、改廃する点も特徴的である。

一方で、「新しい価値の探索や発見」を目

表1　クラウドのリフト&シフト

	リフト	シフト
クラウド活用の仕方	オンプレミスでのシステム稼働環境をクラウド上に再現する	クラウドならでの価値を出しやすいよう環境を構成する（オンプレミスの再現を目指さない）
クラウド活用の目的	コスト削減 アセットライト	（左記に加えて） 業務やシステム開発プロセスの革新
開発運用スキル・ノウハウ	オンプレミスでのスキル・ノウハウをある程度転用可能	新たなスキル・ノウハウの取得が必要
活用するサービス形態	IaaS中心	SaaS、PaaS、マネージドサービス中心
日本と諸外国の差	多くの日本企業が<u>この段階にとどまっている</u>	諸外国（欧・米・中）は<u>この段階まで進み、新たな価値を創出している</u>

表2　従来のシステム開発とDXシステム開発の比較

	従来のシステム開発	DXシステム開発
システム開発の目的	既存の事業をより効率良く、より管理しやすく	新しい価値の探索・発見
開発の進め方	綿密に計画し、正確かつ手戻りがないよう開発する（ウォーターフォール型）	早めに失敗し、失敗から学ぶ（アジャイル型）
開発チームの構成	人海戦術型	少数精鋭型
システムのライフサイクル	明確にフェーズを区切る （開発が完了した後に運用し一定期間後に改廃する）	明確にフェーズを区切らない （リリース後も常に運用しながら開発・改修を続ける）

的とするDXシステム開発は、トライ・アンド・エラーを前提とした行動原理で運営される。数年先までの固定した計画より、素早くフィードバックを得ることを優先し、高度なスキルと経験による柔軟かつ迅速な対応が求められるため、マニュアル化が難しく、人海戦術に不向きである。そのため、開発チームは少数精鋭型の構成となる。成果であるシステムの「消費期限」が短く、一度リリースした後も短いスパンで改修を行ったり、場合によっては完全に撤退したりすることを前提に、プロジェクトが運営されることも特徴的である。

第Ⅰ章で述べたクラウドへのシフトによる改革とは、言い換えれば、これまでの「計画通りに正確に組み立てる」ことを行動原理としたシステム開発から、「高速な仮説検証」を行うことを行動原理としたシステム開発へのシフトチェンジである（表2）。

Ⅲ 高速な仮説検証を支えるクラウドの特性

パブリッククラウドは、「高速な仮説検証」のニーズに合わせて設計されている。これまでのインフラ再現を目指す「リフト」ではなく、クラウドならではの特性を活かす「シフト」に切り替えることにより、「高速な仮説検証」を促進することができる。本章では、実際に当社で行われたプロジェクトを事例として、クラウドへの「シフト」がどのように「高速な仮説検証」を支えるかについて述べる。

このプロジェクトは、NRIシステムテクノの顧客であるメーカーの取引先の卸店が発行した伝票を基に、顧客の社内システムへのデータ入力業務を自動化・代理で行うサービスの開発プロジェクトである。具体的には、まず紙ベースで受領した印刷もしくは手書きの帳票をOCR（光学的文字認識）で文字起こしする。次に、その内容に基づいてRPA（ロボティクス・プロセス・オートメーション）を用いて、顧客の社内システムへのデータ入力を代替する仕組みである。

本システムの開発時、開発チーム最大の課題となったのは、「卸店側の表現で書かれた商品名」と「メーカーの商品マスター上の商品名」の紐づけであった。卸店から発行され

る帳票に記載されている商品名は、多くの場合、メーカー側の商品マスター上の商品名とは一致しない。卸店側のマスターに登録されている商品名が印字されている場合や、商品の通称が手書きで記入されている場合など、メーカー側で管理している名称との「表記の揺れ」が発生する。人間はこのような「表記の揺れ」を考慮して正しくデータ入力することができるが、プログラムによって行うことは難しい。この課題をどう乗り越えるかが最大の懸念であった。

当初の設計では、「レーベンシュタイン距離」という計算式を用いて、対応する商品名を推定する手法を用いた。その手法によって正答率は85%程度となったが、メーカー側の反応は、「これではとても使えない」といったものであった。せっかく自動化しても、15%も誤入力が発生するようでは、その発見と修正のために、かえって余計な人的コストがかかってしまうことが理由であった。

ここから、このロジックの正答率を上げるための試行錯誤が始まった。具体的には「正答率を95%以上」とし、「一度発生した誤読を次回以降発生させない」といった条件を満たすための施策を考える必要があった。

この施策として、

- 誤読の内容と対応する正しい商品名を蓄積しておき、その情報も推論に入れる
- メーカーの工場から各卸店への出荷実績データを取り入れて精度の向上に活かす
- 帳票に記載されている卸店側の商品コードを手掛かりに推論の精度を上げる

など、さまざまな「仮説」がプロジェクトチーム内で立てられ、最長でも1カ月以内にロジックを修正し、その精度を検証した。当然、中には逆に精度を下げてしまう施策もあり、その場合は前のバージョンへの切り戻しを行った。前のバージョンを完全に捨て、一から新たに設計し直すこともあった（図1）。

約1年にわたる試行錯誤を経て、なんとかこのロジックは前述の2つの条件を満たし、サービスインすることができた。

これまでのIT開発では、「前のバージョンに切り戻す」や「前のバージョンを捨てて、

図1　データ入力自動化サービス開発時の課題

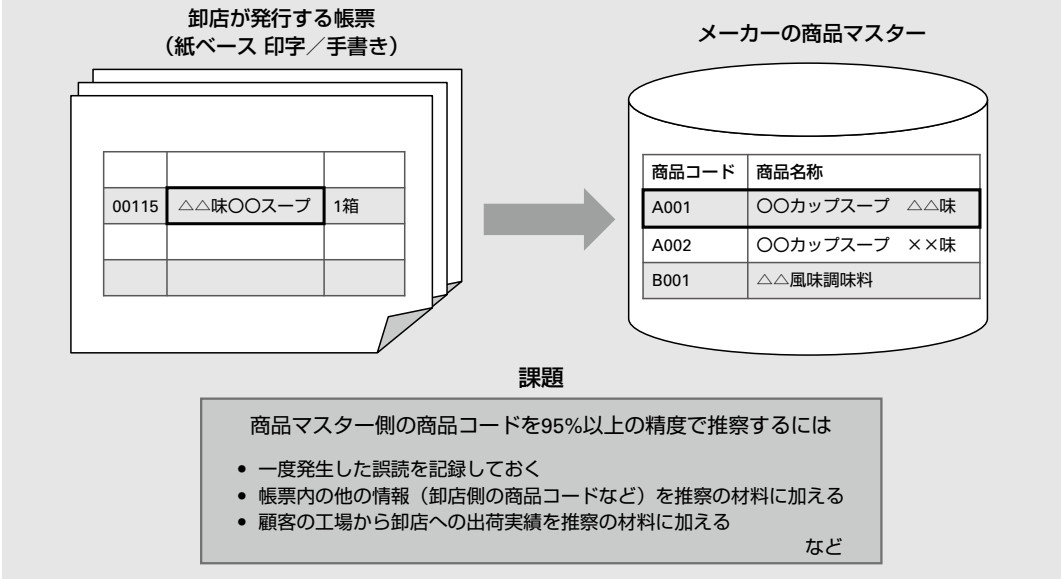
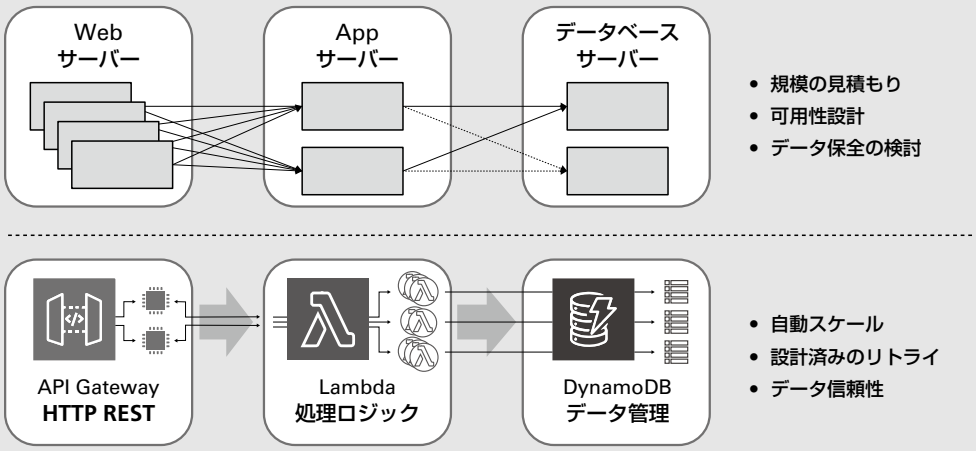


図2　これまでの方式との対比



出所）AWSジャパン「サーバレスのおさらい」（https://pages.awscloud.com/rs/112-TZM-766/images/20200827_serverless_essential.pdf）より抜粋

新たに作り直す」といった手戻りは、大きなコストであった。しかし、試行錯誤を何度も繰り返すためには、手戻りは避けられない。つまり、手戻りに伴うコストを大幅に下げなければ、高速な仮説検証はコスト面で現実的ではないということである。

このプロジェクトでは、クラウド上にサーバやネットワークを構築して自身で維持・管理する方式ではなく、プログラムが実行される時間のみ共有のリソースを間借りする「サーバレスアーキテクチャ」という方式を用いることにより、次に挙げるような3つの手戻りに伴うコストを削減することができた。

なお、本稿においては「インフラ」という用語はシステムの開発や稼働に必要なサーバ、ネットワークなどのあらゆるリソースを総称するものとする（図2）。

1　インフラ設計コスト

素早く仮説検証を実施するには、システム開発を開始するまでに発生するイニシャルコストを極力抑える必要がある。

従来の方式では、将来の利用者数や利用頻度の増大をある程度予測してサーバの数やリソースの量を計画するなど、「将来に備えるためのイニシャルコスト」をかける必要があった。一方、「サーバレスアーキテクチャ」においては、利用の増大に伴って半自動的にリソースの拡張が行われるよう設計されているため、「まずは動かして様子を見る」という選択をすることが可能である。

2　インフラ運用コスト

これまでの設計を捨て、新たに作り直すという判断をするには、それまでの開発に費やしたサンクコストを可能な限り抑制する必要がある。

クラウドサービスのリソースは、原則的に使った分だけ費用が発生する「従量課金制」である。「サーバレスアーキテクチャ」であれば、実際に開発や検証に利用した時間のみコストが発生し、プロジェクトが動いていな

い夜間や休日の「環境維持費」はほとんど発生しない。

実際、事例で紹介したサービスのクラウド利用料が月額100ドルを超えたのは、サービスインしてから半年ほど経過した後であった。プロジェクト開始から半年後に大きな方向転換を迫られた場合、1万ドル×6カ月投資したインフラを「捨てる」判断をすることは難しいが、100ドル×6カ月の投資を「捨てる」選択をすることは比較的容易である。

3 人為ミス対策コスト

ロジックの変更をシステムに適用し検証するという作業を何度も繰り返すためには、ソースコードやデータの変更、テスト、本番化および失敗時の切り戻しといった管理に伴うコストを削減する必要がある。

これらは、いわゆる「人為ミス」が起きやすいポイントである。従来型のシステム開発では「手順書」を整備し、綿密な「レビュー」を行い、スクリーンショットなどの形で「証跡管理」をすることで人為ミス防止を試みてきた。

一方、多くのクラウドサービスはAPI（アプリケーション・インタフェース）と呼ばれるプログラム連携機能を有しており、ソースコードによるリソースの管理が可能となっている。言い換えれば、「プログラムを書くことによる管理の自動化」をする前提でサービスが設計されているということである。そのような特性を活かしてインフラ管理、テスト、リリースなどの作業を徹底的に自動化することにより、「人為ミス」のリスクとその予防のためのコストを大幅に抑制することが可能である。

Ⅳ DXプロジェクトの開発者に求められる能力と生産性

高速な仮説検証を行う上で、クラウドへの「シフト」による最大のメリットは、開発チームの少数精鋭化である。事例に挙げたプロジェクトチームのプログラマーは、当初、筆者1人だけであったが、1人であったからこそ仮説検証に伴う頻繁な方向転換に柔軟に対応できたともいえる。その後、技術教育を兼ねて若手プログラマーによるチームを構成したが、プログラマーの数は最大でも6人であった。NRIシステムテクノ以外の事例を挙げれば、Instagramという世界中で人気のある写真共有サービスは、Facebookに買収された当時、アクティブユーザー3000万人のアプリケーションを13人の会社で運営していた。同サービスは、ユーザー数1000万人の頃までは、たった5人で運営されていたことでも知られている。

オープンソース文化が発展し、さまざまなパブリッククラウドサービスが安価で提供されている昨今、われわれプログラマーは多くのコードを書く必要がない。問題解決の本質となるコアな部分や、ビジネスの中でオリジナリティの高い部分は自分で実装するが、それ以外の部分はオープンソースやパブリッククラウドサービスなどをうまく連携させることによって、コードを書かずに実現することができる。

しかし、われわれプログラマーの仕事は不要になったわけではない。むしろ次に挙げるように、現代のプログラマーにはこれまで以上に高度な能力が要求されている。

表3　人海戦術型と少数精鋭型の開発者に求められる能力の比較

	人海戦術型開発者	少数精鋭型開発者
ナレッジ	入門レベルのプログラミング知識 社内標準・ルールなど	高度なソフトウエア工学 オープンソースやクラウドの世界で起きている技術的革新
スキル	設計書やコーディング規約に従ってソースを記述する力 他人のソースをコピーして似たようなソースを再生産する力	解決したい課題をソースコードでモデリングする力 オープンソースやクラウドの設計思想や特色を理解し、組み合わせる力
マインド	与えられた業務を正確かつミスなく行う 自分の周囲との協業	自ら課題を設定し解決策に向けて試行錯誤する オープンソースやクラウドを通じた世界中の開発者との協業
生産性のキーファクター	作業効率	判断のスピードと正確性

- 数多くあるオープンソースやクラウドを、どのように組み合わせれば問題を解決できるか仮説を立てる構想力
- 世界中の優秀なプログラマーが作成したコードやサービスの特性を読み解くための技術的教養
- スクラップ・アンド・ビルドを繰り返しながら問題解決に向かっていくマインド

クラウドへシフトした後の開発現場において、優秀なプログラマーとそうでないプログラマーの生産性の差は開く一方である。

ショパンのピアノ曲を演奏するコンサートを主催しなければならないときに、ピアノ歴１カ月程度の初心者ピアニストを100人雇うであろうか。高度な能力が必要な仕事に人海戦術は通用しない。同じように、新たな価値を生み出すための高度な仮説検証プロジェクトに必要なのは、１～５人程度の優秀な「プログラマー」であり、単純作業的にコードを組み立てる100人の「作業者」ではない（表３）。

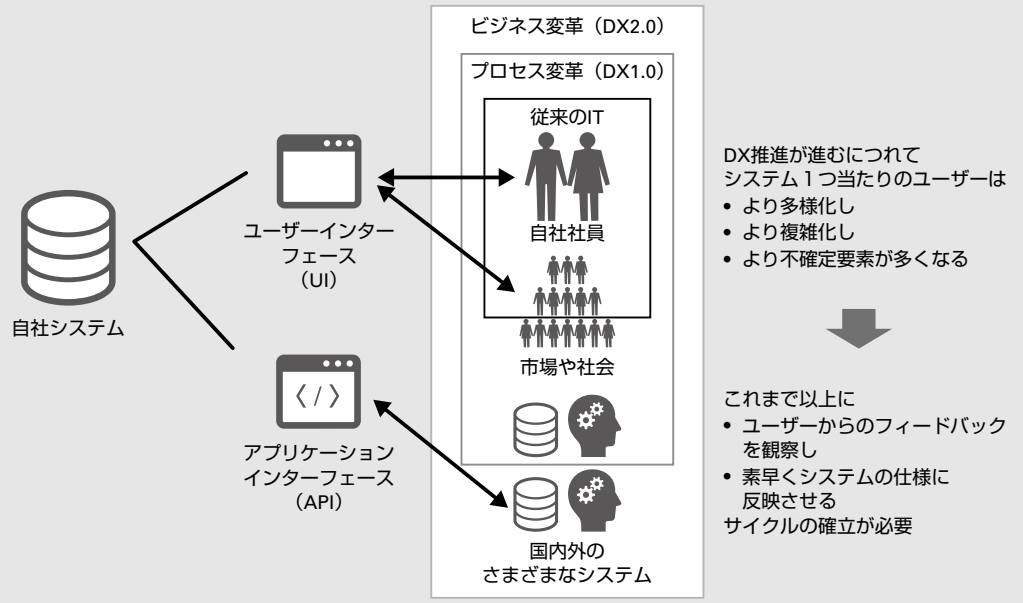
Ⅴ DX2.0を推進するクラウド活用の要点

従来のIT開発から、プロセス変革（DX1.0）、さらにビジネス変革（DX2.0）と進むにつれて、構築されるシステムのユーザーは多様化・複雑化する。自社社員が中心であったエンドユーザーは、徐々に市場や社会そのものへと拡大される。また、業界を横断してエコシステムを形成していくためには、人間のみではなく外部システムや人工知能とも相互に情報のやりとりをする必要がある。

このように多様化・複雑化した不確定要素の多いユーザーに対して継続的に価値を出し続けるには、これまで以上にユーザーからのフィードバックを注意深く観察し、その結果をシステムの仕様へと反映させ続けなければならない（図３）。

フィードバックに素早く柔軟に対応することを目標とするには、従来型の開発プロジェクトはユーザーからシステムオーナー、そして実際にシステムを作成するエンジニアに至

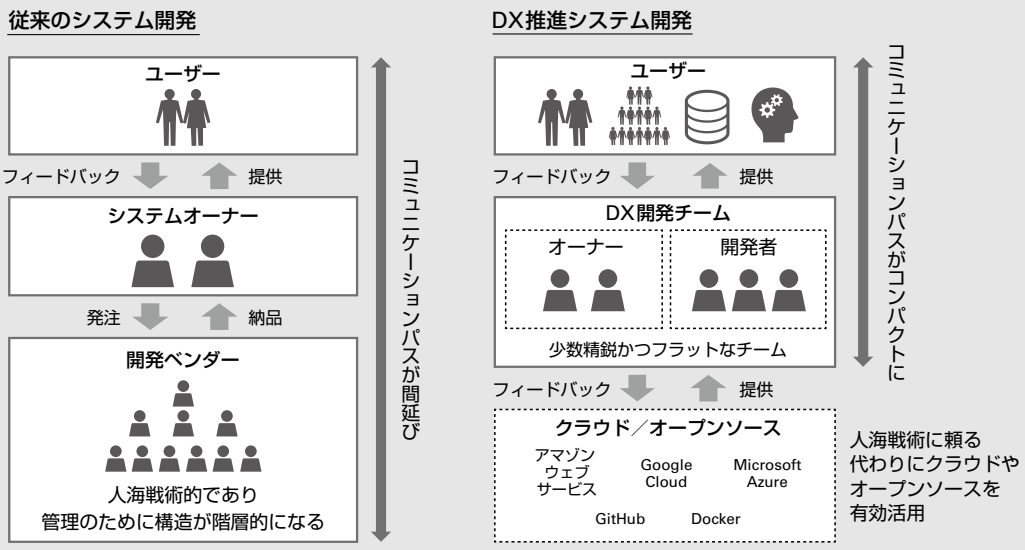
図3　DX推進に伴うユーザーの多様化・複雑化



るまでのコミュニケーションパスが間延びし過ぎている。開発ベンダー側が人海戦術に頼っているためエンジニアの人数が多く、その管理のためにどうしても階層的なチーム構造となる。情報伝達は階層構造の中で伝言ゲーム的に行われるため、コミュニケーションコストが非常に高い。このことが、柔軟かつ高速な仮説検証を大きく阻害してしまう。

一方で前述の通り、クラウドへのシフトが完了したチームは少数精鋭型である。コミュニケーションパスは短縮され、ユーザーからのフィードバックに素早く反応することがで

図4　従来のシステム開発体制とDX推進システム開発体制の比較



きる。

また、システム開発を内製へのシフトや開発ベンダーとの契約の工夫によって、システムオーナーと開発者の関係をよりフラット化することも肝要である。これにより、契約手続きの簡素化などによるスピードアップや、発注者と受注者間の「カベ」の破壊（心理的なものも含む）が実現され、より素早く自律的に活動する開発チームを構成することができる（図４）。

クラウドはただのシステムの引越先ではない。開発チームを少数精鋭化し、迅速かつ柔軟な仮説検証プロセスを回し、DXを推進していくための立派な武器である。DX投資の成否は、クラウド活用を通じてシステム開発のあり方そのものを変革し、ITを自社のビジネスモデルの一部に組み込むことができるかどうかにかかっている。

Ⅵ 優秀なプログラマーを引き寄せる「もう一つのDX」

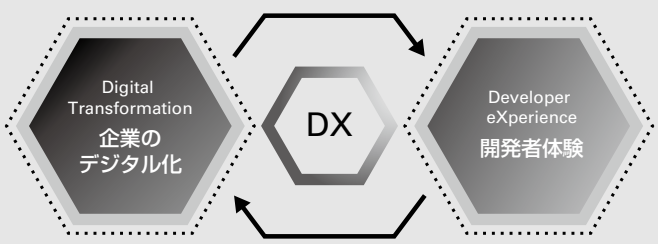
プログラマー全体の人数に対して、高速な仮説検証を推進できるような優秀なプログラマーの割合はそれほど多くはない。DX投資を成功させるには、国内外の企業や公的機関との優秀なプログラマーの奪い合いを制する必要がある。しかしながら、多くの日本のシステムベンダーにおいて、手塩にかけて育てたプログラマーが外資系企業や海外企業に流出してしまうような例が日常茶飯事的に発生している。残念ながら、優秀なプログラマーにとって、現状、多くの日本企業や公的機関は働きがいのある環境ではないということを認めざるを得ない。

一般社団法人日本CTO協会は、優秀なプログラマーにとって働きがいのある組織・文化・システムを実現できているかどうかを表す度合いを「Developer eXperience（開発者体験）」と名付け、デジタルトランスフォーメーションと対をなす「もう一つのDX」と位置付けた。デジタルトランスフォーメーションの実現には「開発者体験」の向上が必須であり、「開発者体験」の向上のためにはデジタルトランスフォーメーションに向けてのチャレンジが必須である。これら「２つのDX」を車輪の両軸として推進することができなければ、DX投資の成功はあり得ない。同団体は「DX Criteria」というガイドラインを設け、この「２つのDX」への取り組み度合いを測ることができる指標を提供している（図５）。

日本のシステム開発現場では、その歴史上、プログラマーは単純作業者と見なされてきた。その影響は現在でも色濃く残っており、優秀なプログラマーの生産性や創造性を大きく制限するような制度・文化・仕組みがいまだに根強く存在している。「優秀な社員は早々にプログラマーを『卒業』し、プロジェクトマネジメントなどの仕事をやるべきだ」といった価値観はその典型である。

これに対して、諸外国の企業では20年以上

図5　デジタル時代の超高速な仮説検証能力を得るには「２つのDX」が必要不可欠



出所）日本CTO協会 Webサイトより抜粋
https://dxcriteria.cto-a.org/

にわたってソフトウエア工学などの専門性を磨き続けてきた「シニアプログラマー」が活躍し、DX推進をリードしている。これに加えて、高度なシステム開発・運用に必要なツールなどの環境整備や、優秀なプログラマーがほかの関係者と協業して成果を出すためのチームづくりのノウハウ形成など、「開発者体験」の向上に対するあらゆる取り組みが進んでいる。

逆説的にいえば、開発者体験の価値に気づき、その向上に力を入れている企業には、日本でも多くの優秀なプログラマーを引き寄せることができるようになる。これは決して、プログラマーという職種を特別扱いし、過剰にもてはやすべきであるということをいっているわけではない。プログラマーを専門家として認め、ほかの分野の専門家と同じように処遇すべきであるということである。専門家としてふさわしい成果を出せないプログラマーには当然厳しい評価を下す必要がある。

具体的にどのような施策を推進すべきかについては、前述のDX Criteriaや日本CTO協会に名を連ねている企業の事例などを参考にするとよいだろう。欧米と異なり、雇用流動性の低い日本において、優秀なプログラマーとともに働くとはどういうことか、大いにヒントが得られるはずである。

AWSのような主要パブリッククラウドは、優秀なプログラマーが生産性を上げやすいように設計されている。彼らの主要顧客である欧米企業では「開発者体験」を向上し、優秀なプログラマーを囲い込んでシステム開発を行うことが当たり前で、そのニーズに応えるようサービスが設計されている。

日本がこのまま「開発者体験」の価値に気づかずに優秀なプログラマーを海外に流出させ続け、プログラマーを単なる作業員として位置付け、人海戦術に頼っているような状況が続けば、日本はクラウドベンダーたちの想定顧客ですらなくなってしまう。そのような状況を回避するためにも、われわれ日本企業はクラウド化をきっかけにシステム開発そのものを大きく改革しなくてはならない。

参考文献

1　経済産業省「DXレポート」(2019年)
2　ガートナージャパン2020年5月14日プレスリリース
https://www.gartner.co.jp/ja/newsroom/press-releases/pr-20200514
3　日本CTO協会「DX Criteria」
https://dxcriteria.cto-a.org/
4　AWSジャパン「サーバレスのおさらい」
https://pages.awscloud.com/rs/112-TZM-766/images/20200827_serverless_essential.pdf
5　Nicole Forsgren Ph.D.他「LeanとDevOpsの科学」インプレス、2018年

著者

座間哲也（ざまてつや）
NRIシステムテクノ 事業本部 デジタル事業企画部 クラウド推進企画グループ
専門は、ソフトウエア工学・Web認証認可技術・サーバレスアーキテクチャなど
プログラマー、ソフトウエアアーキテクト、クラウドソリューションアーキテクト