# IT システムの変遷からみた DX 時代のあるべき開発手法の提言

## Proposals for Ideal Development Methods in the DX Era Based on the Transition of IT Systems

佐藤 孝司（京都情報大学院大学）
Takashi Sato（The Kyoto College of Graduate Studies for Informatics）

**Abstract**

IT システムは，メインフレーム世代のレガシーシステムから，DX（Digital Transformation）時代の新たなデジタル技術を利用したシステムに移り変わっている。レガシーシステムは，ソフトウェアからハードウェアまでの全てを IT 企業が自社で揃える“垂直統合”型のスタイルであった。DX 時代の IT システムは，1990 年代のオープン化や 2010 年代のクラウド化という大きな波により，オープンソフトウェアやクラウドサービスを利用する“分散流用”型のスタイルに変化した。この変化は，IT システムの開発手法にも影響を与えた。レガシーシステムの開発では，開発途中で問題や顧客の要件変更が発生すると，後戻りの作業によるコストが増大する。そのため，ウォーターフォール型開発により各工程の成果物の完成度を高くしながら着実に開発を進める。DX 時代の IT システムでは，DX 時代のテクノロジーの代表としてクラウドシステムと AI システムを例にとると，その開発手法はレガシーシステムとは大きく異なってくる。不特定多数の利用者を対象とする場合のクラウドシステムの開発では，アジャイル型開発と短いサイクルでリリースを繰り返すことで，不明確で変化するゴールを把握することに注力する。AI システムの開発では，AI モジュールの生成が確率的な要素を含むため，繰り返し開発を行う“帰納的な開発”になる。このように，DX 時代の IT システム開発の大きな特徴は，“不確実なゴール”に向かって開発をすることである。

本稿では，DX 時代の IT システムとしてクラウドシステムと AI システムを取り上げ，そのシステムの特徴をレガシーシステムと比較することで，DX 時代の IT システム開発を成功に導くための重要な要素を提案する。

IT systems are shifting from the legacy systems of the mainframe generation to systems that use new digital technologies in the DX (Digital Transformation) era. The legacy system was a "vertically integrated" style in which the IT company provided everything from software to hardware on its own. IT systems in the era of DX have changed to a “distributed diversion” style that uses open software and cloud services due to the big wave of openness in the 1990s and cloud computing in the 2010s. This change has also affected the methodology of IT system development. In the development of legacy systems, when problems or changes in customer requirements occur during development, the cost of backward work increases. For this reason, the waterfall development method is used to ensure a high degree of completion of the deliverables in each process. For IT systems in the DX era, taking cloud systems and AI systems as examples of technologies in the DX era, their development methods will be very different from those of legacy systems. In the development of cloud system for an unspecified number of users, the focus is on understanding the unclear and changing goals through agile development and repeated releases in short cycles. In the development of AI systems, the generation of AI modules involves probabilistic elements, so the development is iterative and inductive. As you can see, the main characteristic of IT system development in the DX era is to develop toward an “uncertain goal”.

This paper discusses cloud systems and AI systems as IT systems for the DX era, and compares the characteristics of these systems with legacy systems to propose key elements for successful IT system development in the DX era.

## 1. 今，IT システムに求められるもの

近年は，あらゆる産業において新たなデジタル技術を利用したビジネスモデルの展開のために，DX（Digital Transformation）の推進が求められている。DX とは，“第 3 のプラットフォーム（クラウド，モビリティ，ビッグデータ／アナリティクス，ソーシャル技術）を利用して，新しい製品やサービス，新しいビジネスモデルを通して，顧客エクスペリエンスの変革を図ることで価値を創出し，競争上の優位性を確立すること”[1] である。しかし，実際には，DX の必要性を認識しているものの，どのようなデジタル技術をどう活用すべきかを模索している企業が多い。一方，8 割の企業が「レガシーシステム」（図 1）を抱えているため，DX 化着手への足かせになっているという調査結果もある [2]。

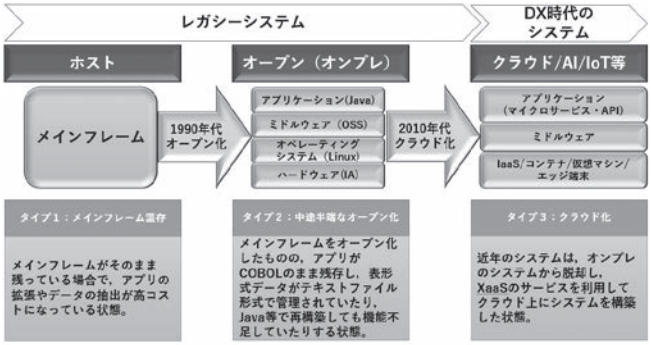


**図 1　IT システムの変遷 [2]**

IT システム利用者がシステムに期待している事柄について調査した結果 [3] では，IT システムに求める重視事項が変化をしている。「業務支援・情報系」および「Web・フロント系」システムでは，2016 年度に比べ 2019 年度には，これまで常に上位にあった「品質」を重視するポイントが減少し，反対に「開発スピード」を重視する傾向が全分野において強くなっている（表 1）。

**表 1 システム構築時の重視事項の変化 [3]**

| | 「品質」を重視する割合 | | | 「開発スピード」を重視する割合 | | |
|---|---|---|---|---|---|---|
| | DI 値（重視の割合が高い－低い） | | DI 値増減 | DI 値（重視の割合が高い－低い） | | DI 値増減 |
| | 16年度 | 19年度 | | 16年度 | 19年度 | |
| 基幹系 | 73.6 | 75.2 | 1.6 | ▲54.2 | ▲48.1 | 6.1 |
| 業務支援･情報系 | 43.2 | 37.3 | ▲5.9 | ▲5.2 | 3.2 | 8.4 |
| Web･フロント系 | 42.8 | 39.8 | ▲3.0 | 10.5 | 23.9 | 13.4 |
| 管理業務系 | 68.6 | 70.5 | 1.9 | ▲61.7 | ▲47.5 | 14.2 |

DX 化における企業の悩みと，この IT システム利用者のシステムへの期待調査の結果によれば，次のような状況が浮かび上がる。すなわち，DX 化により“新たなデジタル技術を利用した新たなビジネスモデルの展開”をいそぐ企業にとって，新たなデジタル技術を導入した DX 時代の IT システムは，“開発スピードを重視”して，“どう活用すべきかを模索”しながら，開発され利用されなければならない，と解釈できる。

本稿では，この IT システム利用者の状況をふまえて，DX 時代の IT システムのテクノロジーの代表といえるクラウドシステムと AI システムの特徴に着目した開発について，これまでのレガシーシステムと比較することで，DX 時代の IT システム開発を成功に導くための重要な要素を提案する。

## 2. レガシーシステム開発の特徴と課題／対処法

レガシーシステムは，メインフレーム時代からの長い伝統をもつシステムである。このシステム開発の特徴は，比較的に大人数で長期間をかけて，大規模なシステムを開発する場合が多く，いわば“重厚長大”な開発であるといえる。そして，システム開発のゴールとなるシステムで実現すべき要件は，開発初期段階に明確であることが期待される。開発手法は，各開発の工程を“滝（ウォーターフォール）”のように順番に着実に実施して完成させるウォーターフォール型開発が主に用いられる（図 2）。

この開発手法の課題は，開発の途中に誤りを発見した場合に誤りを埋め込んだ工程まで作業を後戻りして，そこから開発作業をやり直さなければならないことである。したがって，開発の途中で要件の変更やシステムの修正が想定以上に頻発すると，後戻

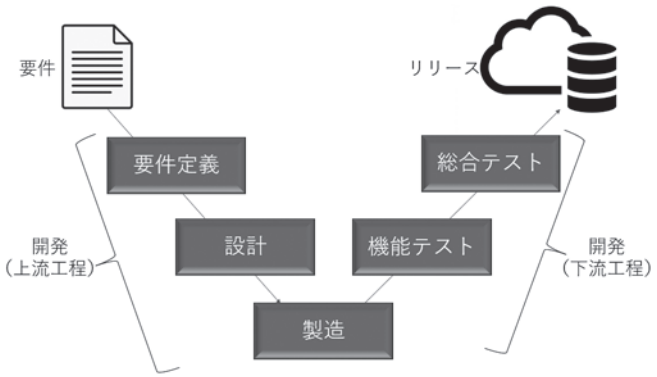


**図 2　レガシーシステム開発のプロセス例**

表2　ソフトウェア品質管理メトリクスの例 [5]

| No. | データ項目 | 単位 | 定義 |
|---|---|---|---|
| 1 | 開発規模 | Line | 新規規模+修正規模．削除規模は含まない．開発規模に加えて，新規再利用規模と継続再利用規模を測定しておくとよい．<br>参考：ソフトウェア全体規模=新規規模＋修正規模＋新規再利用規模＋継続再利用規模 |
| 2 | 工数 | 人時/KL | 開発に費やした工数(人時)/開発規模(KL)．場面に応じて，以下のように使い分ける．<br>全工数＝開発に費やした工数(人時)/開発規模(KL)<br>上流工程工数＝上流工程に費やした工数(人時)/開発規模(KL)<br>テスト工程工数＝テスト工程に費やした工数(人時)/開発規模(KL)<br>工程毎の工数＝各工程に費やした工数(人時)/開発規模(KL)<br>※工数はレビュー工数を含む |
| 3 | レビュー工数 | 人時/KL | 設計およびコードに対するレビューに費やした工数(人時)/開発規模(KL)<br>場面に応じて，上流工程レビュー工数，工程毎のレビュー工数を使い分ける． |
| 4 | バグ数 | 件/KL | 出荷前に摘出したバグ数/開発規模(KL)<br>場面に応じて，全バグ数，上流工程バグ数，テスト工程バグ数，工程別のバグ数を使い分ける．<br>バグ1件ごとに，作り込み工程と摘出工程を分析し，作り込み工程毎のバグ数，摘出工程毎のバグ数を計測しておくとよい． |
| 5 | 上流工程バグ摘出率 | % | (上流工程バグ数/全バグ数)×100 |
| 6 | テスト項目数 | 項目数/KL | テスト工程で実施するテスト項目数/開発規模(KL)<br>場面に応じて，工程毎のテスト項目数を使い分ける．<br>今回開発する機能に対するテスト項目数を新規テスト項目，既存機能に対するテスト項目数を既存テスト項目数と呼ぶ． |
| 7 | 生産性 | Line/人時 | 開発規模(Line)/全工数(人時)． |
| ※KLは，Kilo Lines of Codeの略<br>※KLあたりで使用する開発規模は，実績規模が確定するまでは計画規模を使用する | | | |

りの作業が多く発生するため，コストが大幅に増大し，開発期間も予定を超過してしまう。

この課題に対処するためには，各開発の成果物を可能な限り完成度の高いものにして，誤りを混入させないことや，後工程で要件の変更による矛盾等を発生させないことである。開発途中における要件の変更をできるかぎり発生させないためには，開発の上流段階において，利用者も交えて，できるかぎり要件の詳細化を図る。このようにして，レガシーシステムの開発では，開発初期段階において，システムのゴールを明確にし，各工程の成果物の完成度を高め，後戻りの少ない開発を行うことが重要である。

また，このシステム開発を成功に導くために，構造化設計やオブジェクト指向設計等による大規模で複雑な仕様を合理的にモデリングする設計手法が用いられる。さらに，後戻りを防ぐための各工程の品質と進捗を管理する手法に，ハードウェア開発の品質管理手法を適用している。このソフトウェア品質管理手法は“ソフトウェア工場”と称され，日本のソフトウェアが高品質であることに世界が注目した [4] 手法である。このように，開発手順やルールを明確にし，厳格な品質管理を行うためのソフトウェア品質管理メトリクス(表2)を研究し運用している [5]。

## 3. クラウドシステム開発の特徴と課題／対処法

2010 年代より，自社内にシステムを構築するオンプレミス型のレガシーシステムの開発は次第に減少し，システム開発は，Web 上に構築するクラウドシステム開発にシフトしていった（図 1)。

クラウドシステムの特徴は，オンプレミス型のレガシーシステムが IT 資源を自社内で所有する形態であることに対して，クラウドシステムでは，インターネット上で IT 資源やサービスを利用する形態に変わったことである。インターネットショッピングのように一般の人が容易に利用できるサービスが急激に拡大したことも特徴の一つといえる。この一般向けにサービスを提供するクラウドシステムでは，不特定多数の利用者のニーズの把握が困難になるとともに，ニーズの変化するスピードが速まった [7]。この点が，従来の特定顧客の要件をシステム化してきたレガシーシステムと大きく異なる。レガシーシステムでは，利用者の要件を満足するように開発を行い，一度リリースしたら，その後は多少の機能改善を含むものの，長期にわたり主機能を利用し続けることを前提としており，システムの開発はゴールを明確に把握できていた。

しかし，一般利用者向けサービスを提供するためのクラウドシステムでは，ニーズの把握が困難になるので，開発においてシステムのゴールを明確にできない。したがって，クラウドシステム開発は，以下の課題がある。

- 不特定多数の利用者を対象にする場合，ゴール（要件）が不確実になる。
- そのため，サービスの利用状況を常に分析して，利用者のニーズの変化を素早く把握する必要がある。
- 利用状況の分析により把握したニーズに素早く対応するためには，1 回のリリースをスピーディに行わなければならない。

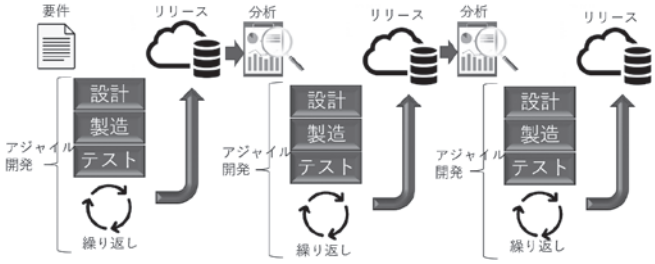


**図 3　クラウドシステム開発のプロセス例**

この課題に対処するためには，レガシーシステムのウォーターフォール型開発ではスピーディな開発に適さないため，俊敏さを重視したアジャイル型開発が浸透した。

アジャイル型開発のスクラム手法では，少人数の開発チームで，短期間に小さな機能の開発を繰り返すことや，繰り返しのたびに，どの機能を優先して実現するかを判断しながら開発するので，クラウドシステム開発に適している。

さらに，サービスの利用状況を分析し，変化するニーズを把握して次の開発につなげるように，開発とリリース運用を繰り返す DevOps（Development Operations）手法により，ニーズの変化に追随したスピーディな開発を実現する（図 3）。

アジャイル型開発では，小さな機能の開発を繰り返すことで，常に搭載すべき機能の優先度を吟味し，要件の変化に柔軟に対応できる長所がある一方，繰り返し開発の終了判断が難しくなり，却って開発期間が延びてしまうなどの問題が発生する場合もある。この対策として，システム全体の仕様を明確にして，各開発工程を厳格に管理することを得意とする従来のウォーターフォール型開発の長所を，アジャイル型開発に取り混ぜたハイブリッド型の開発を実施している事例も多い [7][8]。

## 4. AI システム開発の特徴と課題／対処法

第 3 次 AI ブームのきっかけは，機械学習のディープラーニング（深層学習）による特徴量を AI が自ら学習する方式の成功による。近年あらゆる産業において AI を搭載したシステムや製品が拡大している。AI がこれほど急速に浸透した要因には，深層学習のさまざまなモデルを具備したオープンなライブラリを世界中の研究者が開発・公開し，誰でも容易に利用できる環境によることも大きい。

深層学習の AI モジュールを生成する場合には，推定や判別の対象を学習するデータのパターンが AI モジュールの精度（正答率，適合率等）に影響するため，準備するデータの質や量が重要である。AI モジュール生成後の評価により初めてモジュールの精度を把握できるため，事前に精度を予測することは難しい。さらに，リリース前の評価データでは良い性能であると判断しても，現実の世界のデータが必ずしも評価データと同じではないので，リリース後の性能が評価時よりも低下する可能性もある。

このような深層学習 AI モジュールを含む AI システム開発の特徴は，“帰納的な開発”である。AI モジュールの生成（図 4）は確率的手法（初期化，確率的勾配降下法，ドロップアウト等）が使われ，開発の成果となる AI モジュールの精度が，開発のゴールとして正解であるかの答え合わせができない。評価結果の精度が芳しくなければ，モデルの選択やチューニングを変更して，モジュールの生成と評価を繰り返しながら，精度向上を目指す。開発前に，このモデルで，このチューニングをすれば，この精度になるというゴールとしての正解を予測でき

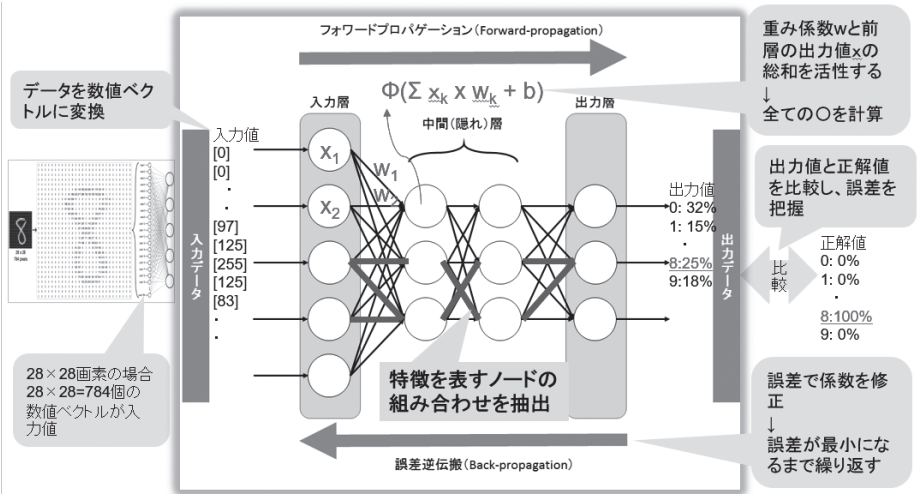


**図 4　深層学習 AI モジュール生成の模式図 [9]**

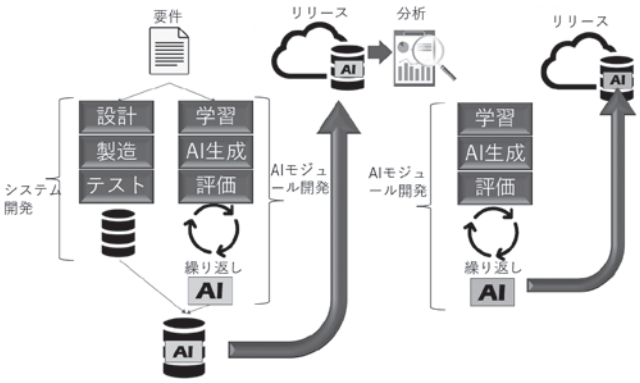


**図5　AIシステム開発のプロセス例**

ない。

特徴の2つ目は，リリース後にも精度を観測して必要に応じて再開発をすることである。たとえ開発時に高い精度を得たAIモジュールであっても，実際の現場に適用して同じ高い精度を示すとは限らない。したがって，リリース後の適用状況を監視し，精度を常に観測して，現場の時間経過や環境変化により対象データの性質が変わったと判断すれば，開発時に学習したデータを現場に適合した最新のデータに置き換えて，再度学習から再開発をしなければならない（図5）。

これらの特徴は，レガシーシステム開発のようにゴール（要件）を開発初期段階に明確にできないため，“不確実なゴール”に向かって開発をすることになる。このようなAIシステムの特徴をふまえた開発の課題を整理すると以下のようになる。

・AIモジュールを生成するための学習データと，現実の世界のデータとの差異を推測していかに学習前に十分に準備できるか。
・AIモジュール生成が確率的手法によるため，同じ結果が得られない不確実さがあるなかで，精度をどれだけ高くすればリリースできるのかという判断基準をどうするか。
・AIシステムをリリース後に，学習データが現実の世界のデータと差が生じ，モジュールの劣化を判断して再開発を行うタイミングをどうするか。

これらの課題への対処には以下が必要であると考えられる。

・学習データやAIモジュールの不確実性を，AIシステム全体の機能と運用プロセスでカバーする（図6）[10]。
◆開発初期段階の“不確実性分析”により，現場の世界のユースケースを十分に研究した上で，学習データを準備する。
◆AIモジュールの不確実性をカバーするためのフェールセーフ等のシステム機能を搭載する。
◆AIモジュール精度の評価に，正解値のある絶対値評価ができない代わりに，最適な方向に向かっているかの差分評価の手順を明確にする。
◆リリース後に精度をモニタリングする仕組みとAIモジュール再作成の判断プロセスの仕組みを構築する。開発時の繰り返しモジュールを再作成したデータ（精度と変更したパラメータ等のログ）をもとに判断基準を作成しておく。

## 5. DX時代のシステム開発への提言

DX時代のシステム開発の代表として取り上げたクラウドシステム開発とAIシステム開発の特徴をまとめると以下のようになる（表3）。

・ゴール（要件）は不確実で曖昧であり，変化する。
・最初からゴールが明確ではないため，開発結果

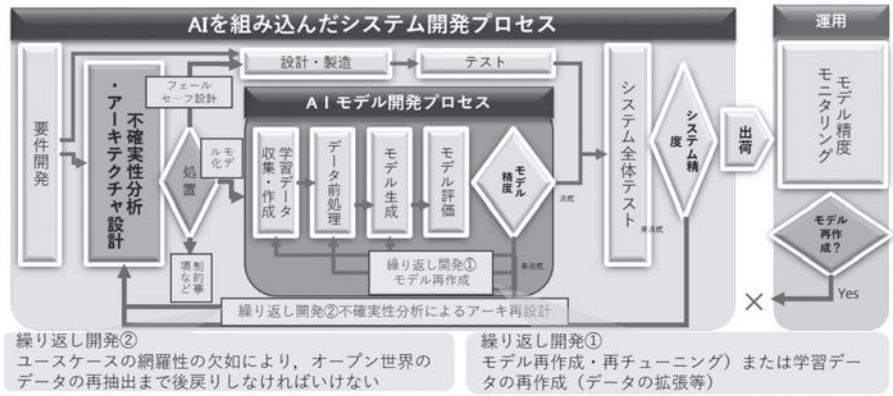


**図6　AIシステムの開発の改善プロセス例 [10]**

表 3 各開発手法の特徴

| | レガシーシステム | クラウドシステム | AI システム |
|---|---|---|---|
| ゴール（要件） | 開発初期段階に特定顧客の要件を明確に把握する | 一般利用者向けサービスのニーズが不明確かつ，変化する | AI モジュールの精度は確率的であり，結果を予測できない |
| 開発プロセス | ウォーターフォール型開発により，後戻り作業が発生しないように各工程の完成度を高くする | アジャイル型開発により小さい機能の開発を繰り返しスピーディな開発を行う | 結果を予測できないため，帰納的開発により開発やチューニングや再学習を繰り返す |
| リリース後 | 機能改修などリリース後の開発は最小限コストに抑える | DevOps により，不明確で変化するニーズに対応するための開発とリリースを繰り返す | 環境変化を監視し，AI モジュールの性能劣化に応じて再学習・開発を行う |

を評価して再開発する帰納的開発である。

・最初からゴールが明確にできないため，開発スピードを上げ，リリースを繰り返すことで，ゴールの変化に対応する。

・ゴールが不確実で曖昧なため，リリース後の評価・分析をタイミングよく行うことで，次のリリースのゴールを設定して，開発とリリースを繰り返す。

これらの特徴へ対処する DX 時代の開発手法として，“アジリティ型組織”（図 7）による開発を提案する。そして，“アジリティ型組織”とは，以下の要素を兼ね備える必要があると考える。

・高速 PDCA による改善スピード向上

PDCA は，計画（Plan）－実行（Do）－評価（Check）－改善（Action）のサイクルを繰り返す伝統的な業務改善方法である。帰納的な開発では，不確実なゴールに対して，この改善サイクルを，“高速”に回すことが重要なポイントであるといえる。ある時点の

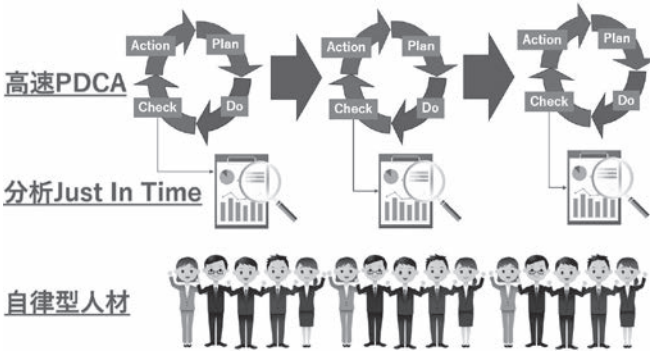


図 7　DX 時代の“アジリティ組織”による開発手法

開発ゴールと現実の期待するゴールとの差分を頻繁にチェックして，変化するゴール，曖昧なゴールに対して短いタイミングで軌道修正を行い，ゴールの差異を最小にとどめる。不確実なゴールを明確にし，また変化に追随する。

このように，変化への迅速な対応や，方向性の誤りに気づいたら軌道修正する形態は，組織の画一的・統一的な戦略はかえって現場との乖離が生じるため効果が薄くなる。全体最適ではなく，個別最適の考え方が重要になる。

・分析 Just In Time による差分分析開発

Just In Time は，トヨタ自動車が開発した「必要なものを必要なときに必要なだけ」供給する生産管理システムである。無駄，ムラ，無理を防ぐ，世界中の生産工場の手本となっている製造方式である。

これに倣い，分析 Just In Time とは，上述の高速 PDCA の評価 Check において，不確実なゴールの現実と期待との差分の分析を遅れの無いタイミングで，必要な情報を確実に分析し，その結果をもとに軌道修正の Action を行うという考えである。

差分の分析とは，例えば，クラウドシステム開発では，一旦クラウド上にリリースしたシステムのアクセス分析により利用状況を分析し，次回のリリースには，どのような差分の機能が必要かを検討する。また，AI システム開発では，AI モジュールの性能改善に，前回のモジュールと今回のモジュールの差分を分析することで全体的なゴールに近づいているかを判断する。

このように適切なタイミングで，その時点に必要な分析と次の対策を施す特徴から，“分析 Just In Time”と名付けた。この分析を Just なタイミングで行うためには，不確実なゴールや帰納的開発に対して，“少ない情報で大胆な仮説を立て，必ずしも事実やデータに立脚しなくても，仮説を検証する仮説思考アプローチ”[6] が，有効と考える。

・自律型人材の育成

人材に焦点をおいて考えてみる。従来のレガシーシステム開発では，ゴールが明確なので，詳細で明確な開発プロセスを開発者全員が遵守することにより，個人のスキル差の解消や底上げを狙う，画一化，統制的な製造業手法であった。開発者はプロセスに則ることに重きを置くため，ともすれば，自らが考

え，工夫する機会を奪いかねない。しかし，DX 時代の IT システム開発の施策と同様に，開発者の人材育成の視点においては，“従来”とは真逆の発想が必要になる。つまり，自律型人材とは，自ら考え行動して，業務を遂行できる人材である。決められたプロセスを忠実に遵守する手法では，不確実で変化するゴールに対して，業務のスピードも質も不足となり，変動対応力がキーとなる高速 PDCA も分析 Just In Time も遂行することは難しい。

そして，自律型人材の育成により，“組織の目的達成のために，組織のメンバー全員が個別に自己決定を行う自律型組織”[11] へと進化を果たす。

表題の“IT システムの変遷”の本質は，“IT システムの実現すべき対象の変遷”である。開発手法は，ゴールの性質を見極めて，その性質に適切に順応することである。そのために，開発手法が変化することは，自然な流れである。一方，人が組織として集団活動を行うと，慣習や伝統，過去の継承，変化に対する拒絶などが生まれやすいのも事実であろう。しかし，ともすれば，開発する組織が開発のゴールを見極める本質的な作業を怠り，従来のやり方を維持して安定を望み，意識的にしろ，無意識にしろ，周囲を見ない，変化を感じようとしない思考に陥ることは，非常に危険であると考える。

## 6. おわりに

～ Do the right thing, Do things right. ～

ソフトウェアを開発することは，何を作るのかを明確にして，そのとおりに実装する，ということである。これは，正しいものを正しく作る（Do the right thing, Do things right）と言い換えることもできる。

本稿で述べたように，DX 時代には何を作るべきか，何が正しいものか，を開発当初から明確にすることが難しくなってきている。その一因には，ソフトウェアがハードウェアと一体化した製造物として提供する形式から，不特定多数の利用者を対象としたサービス化（SaaS: Software as a Service）へシフトしていることも大きく影響していると考えられる。

そこで，ソフトウェア開発者は，PoC（Proof of Concept: 概念実証）を何度も実施して，何を作るべきかを検証したり，DevOps（Development and Operations）やアジャイル開発のようにリリース後も軌道修正を繰り返したりすることで，利用者の嗜好とその変化により近づくように対処してきた。つまり，“正しいもの”を“正しく作る”ためには，何が“正しいもの”かを常に観察し，その変化をとらえること。そして，その変化に適切に対応できるようにすることが“正しく作る”ことになる。

しかし，AI の開発，特に深層学習の開発になると，さらにやっかいなことになる。これは，テストオラクルが無い世界である。つまり，作成した AI エンジンのテストによる判断基準となるべき期待値が定まらない。こうなると，前述の PoC や DevOps という手段だけでは不十分であり，リリースの判断さえも難しくなる。従来，よく利用されていた定量的指標を用いた過去の実績値との比較ができない。したがって，明らかに前提条件が異なる開発においては，定量的（統計的）指標が果たす役割は小さくなる。

そこで，本稿で述べた“分析 Just in Time”を“高速 PDCA”で廻していくという考え方が必要になる。絶対値ではなく瞬間値の差分から向かうべき方向であるかどうかの判断を行い，また，開発作業の最終結果を見るのではなく，開発時やリリース後の随所に観測点を設けて，常に何が“正しいもの”かの実像を追い続けるという，従来とは大きく異なる開発方法の考え方にチェンジしていくことになる。

**参考文献**


[1] IDC Japan，Japan IT Market 2018 Top 10 Predictions: デジタルネイティブ企業への変革，2017

[2] 経済産業省，「DX デジタルトランスフォーメーションレポート～ IT システム『2025 年の崖』の克服と DX の本格的な展開～」，2018

[3] 一般社団法人日本情報システム・ユーザー協会（JUAS），「企業 IT 動向調査報告書 2020」，2020

[4] Michael A.Cusumano, Japan's Software Factories: A Challenge to U.S. Management，1991

[5] 誉田直美，佐藤孝司，森岳志，倉下亮，ソフトウェア品質判定メソッド　～計画・各工程・出荷時の審査と分析評価技法～ , 日科技連出版社，2019

[6] 広木大地，エンジニアリング組織論への招待 ～不確実性に向き合う思考と組織のリファクタリング，技術評論社，2018

[7] 佐藤孝司，纐纈伸子，橘克一，下村哲司，アジャイル開発のスクラム手法における定量的管理の導入事例，日本信頼性学会誌「信頼性」第 41 巻，2019

[8] 英繁雄，奈加健次，平岡嗣晃，前川祐介，関西電力株式会社，ハイブリッドアジャイルの実践，リックテレコム，2013

[9] ソフトウェア品質シンポジウム 2019，AI システムの

品質保証，日本科学技術連盟，2019
[10] AI プロダクト品質保証コンソーシアム 編，AI プロダクト品質保証ガイドライン，QA4AI, 2021
[11] フレデリック・ラルー，ティール組織――マネジメントの常識を覆す次世代型組織の出現，英治出版，2018

◆著者紹介

**佐藤　孝司**　Takashi Sato

京都情報大学院大学　教授
名古屋工業大学工学士（情報工学専攻）
鳥取大学大学院工学研究科会基盤工学専攻終了　工学博士
元日本電気株式会社　主席主幹，上席ソフトウェアプロセス＆品質プロフェッショナル
日本科学技術連盟 SQiP ソフトウェア品質保証部長の会企画委員，2009–
AI プロダクト品質保証コンソーシアム運営委員兼元自動運転技術チームリーダー，2018–