情報システムの変遷による品質マネジメントの在り方に関する一考察
～確実性から不確実性へ、演繹から帰納へ、規律から自律へ～

佐藤　孝司

A Study of Quality Management in the Transition of Information Systems: From Certainty to Uncertainty, from Deduction to Induction, from Discipline to Autonomy

Takashi Sato

要約

IT 技術の急速な進展により、情報システムは、システム構成要素や利用する IT 技術が大きく変化した。そして現代のクラウド技術や AI 技術を利用したシステムでは、従来の企業内業務のシステム化に比べてシステムの達成すべき目標が不確実になった。この変化に伴い、システムの開発スタイルも開発工程を順番に進む演繹的スタイルから、システムの目標達成の度合いを検証しながら繰り返し開発する帰納的スタイルに遷移している。一方、情報システムは社会的基盤を支えるまでに利用対象が拡大し浸透してきたため、情報システムに不具合が発生すれば社会生活に大きな影響を及ぼしかねない。そのため、情報システムの信頼性はますます重要になってきている。
しかし、システム開発における品質を確保する考え方は、レガシーシステムの長い開発の歴史の中で確立されたものから脱却できていない面もある。システムの目標が不確実になり、開発スタイルが帰納的に変化したことにより、システムの品質を確保するためには、従来の画一的な品質プロセスの規律を遵守する考え方から、品質をより柔軟にマネジメントする考え方への変革やそれを実行できる自律的な組織への変革が必要になってきている。
本論文では、これらの情報システムの変遷に伴う開発スタイルの変化と品質確保の考え方を整理し、品質マネジメントから組織運営までのあるべき姿を考察し提案する。

Abstract

Information systems have changed dramatically with the rapid development of information technology, both in terms of system components and the IT technologies used. And in systems using modern cloud technology and AI technology, the goals to be achieved by the system have become more uncertain compared to the conventional systemization of business operations within a company. In line with this change, the system development style has shifted from a deductive style, in which the development process proceeds in sequence, to an inductive style, in which the system is developed repeatedly while verifying the degree of achievement of the system's goals. On the other hand, information systems have expanded and penetrated to the point where they support

social infrastructures, and any failure in an information system can have a significant impact on social life. Therefore, the reliability of information systems is becoming increasingly important.

However, in some aspects, the concept of ensuring quality in system development has not broken away from the one established in the long history of legacy system development. As system goals become uncertain and development styles change inductively, ensuring system quality requires a change from the traditional approach of adhering to a uniform quality process discipline to a more flexible approach of managing quality and an autonomous organization capable of implementing this change. This is a new era.

In this paper, we summarize the changes in development styles and the concept of ensuring quality that accompany these changes in information systems and discuss and propose the ideal form for everything from quality management to organizational operation.

1．情報システムの変遷　～レガシーシステムから DX 時代のシステムへ～

情報システムは、1960 年代頃から大型汎用計算機（メインフレーム）をユーザーである各企業に設置して、企業の経営情報処理や財務会計処理などの業務をシステム化して運用していたレガシーシステムの時代に比べて、現代の DX（Digital Transformation）時代では、様々な IT 技術の進展に伴い、システム構成や開発のスタイルが大きく変化してきた。

レガシーシステムは、IT メーカーがハードウェアからソフトウェアまでの全てを自社で開発して､ユーザー企業にシステムを設置する“垂直統合”型のスタイルであった。DX 時代のシステムは、2000 年代頃のオープンソースソフトウェア（OSS）の台頭や 2010 年代頃のクラウド化の大きな波により、OSS やクラウドサービスを積極的に利用することで、IT メーカーが全てを自作しない“分散流用”型のスタイルに変わった。この変化は、システムの開発手法にも大きな影響を与えることになった。

レガシーシステムの開発では、開発途中で不具合やユーザー要件の変更が発生すると、修正や変更のための後戻り作業によりコストが増大し納期も遅れるリスクが増す。不具合などの後戻り作業の発生を防ぐために、ウォーターフォール型の開発スタイルを適用して、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ことが肝要であった。DX 時代のシステムでは、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 システムを例に挙げると､その開発手法はレガシーシステムのそれ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、アジャイル型開発スタイルを適用して、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”システムのゴール（目標）を随時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ことなどから、AI の精度を予測できない“不明確なゴール”のため､結果を検証しながら AI モジュールのチューニングを繰り返し開発する“帰納的な開発”を行う。このように､DX 時代のシステム開発の大きな特徴は､“不明確で変化するゴール”を探りながら開発とリリースを繰り返すことである。

2．情報システムの変遷における課題

～“進み始めた「デジタル」、進まない「トランスフォーメーション」”（[1] のサブタイトルを引用）～

情報システムの変遷の特徴は、レガシーシステムが特定のユーザー企業の業務をシステム化する”明確なゴール”を提供することに対して、DX 時代では、この半世紀の IT 技術の急速な進歩により、クラウドシステムや AI システムなどのように新たな価値を不特定多数のユーザーに対して、システムの到達すべき絶対的な正解値が曖昧な“不明確なゴール”の状態で提供する形態に変化してきたことである。日本の情報システムがこの DX 時代のシステムへの変化にどれほど追随できているのか、その実態について独立行政法人情報処理推進機構（IPA）の調査結果 [1] を参照すると以下のとおりである。

・システムをすべて移行してレガシーシステムが残っていない企業は 12% であり、米国は 23% である。
・半分以上のレガシーシステムが残っている企業は、米国が 23% に対して日本は 41% もある。

上記は調査の一部であるが、このデータから日本の DX 化が米国に比べてかなり遅れていることがわかる。日本の企業にはレガシーシステムが多く残っていることが、DX の推進を阻んでいる可能性が大きい。つまり、DX のキーワードが巷間に溢れており、DX 時代の社会が確立されつつあるのかと臆測するも、実態はそうではない。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く、また、8 割以上の企業がレガシーシステムを抱えている [2] ため、DX 化着手への足かせになっているのが実態のようである。

レガシー時代の情報システムは、ユーザー企業の既存業務を IT 化する業務効率化が主な目的であったが、1960 年代から 2000 年代までにはこの IT 化がほぼ一巡すると、ユーザー企業の事業の方向性に大きな変革が行われない限り、情報システムを変更する必要はない。もちろん、業務の細かな変更に伴う小さなシステム変更は継続的に行われているにしても、一旦、企業の業務を IT 化すれば、そのシステムの基盤はそのまま継続して利用される。ただし、DX 時代の特徴の一つであるクラウド化を行うことにより、システムの基盤の大きな変更が必要となる。それでも、このレガシーシステムのクラウド化も一巡してしまえば、システムはしばらく使われ続けることになる。これらのシステム化の作業はあくまでも既存の業務の IT 化というシステムのゴール（目標）が明確であることも特徴としてとらえておく必要がある。

したがって、ビジネスインテリジェンスによる”新たな価値“を提供するシステムを構築することが、DX 時代のシステムの目的の一つとすると、現状では十分な DX 化が実現できているとはいえない。上述のとおり各企業はレガシーシステムの足かせという呪縛により、DX 時代のシステム構築が期待通りに進んでいないのが実態である。

3．ソフトウェア品質の実態　～現代の情報システムの信頼性状況～

情報システムのソフトウェアの品質がどのような状況にあるのか？品質は良いといえるのか？日本の情報システムにおけるソフトウェアの信頼性について、2016 年度～ 2021 年度のデータの傾向性をそれ以前のデータと比較した IPA の調査結果 [3] を以下に示す。

1. 近年まで継続していた信頼性の向上傾向は見られなくなった
2. 一方、生産性の低下傾向は弱めながら継続している
3. 工期が少し長くなっているが、その他のデータに大きな変動はない

この調査では定量的なデータも公開されており、それによると、情報システムのリリース後のソフトウェアの不具合（ソフトウェアのバグ）は 10 万行で 1 個（中央値）、1 万行で 1 個（平均

値）である。開発したソースコードの規模が 10 万行を超える情報システムは全システムの 4 割弱あることから、世の中の情報システムの約 1/4 はリリース後に 1 個以上の不具合が発生していることになる。システムの規模は提供する機能により大きく異なるため一概には言えないが、少なくとも 10 万行以上の規模をもつシステムはかなり大規模で複雑なシステムであるといえる。一般的な機能を備えた Web アプリケーションの規模は、数千行から数万行程度であると言われていることを考慮すると、10 万行を超える大規模なシステム開発が現代でも半数近く存在すること、そして、そのような大規模なシステムの約半分は 1 個以上の不具合があるということは、その 1 個の不具合が原因でシステムの停止などによる社会的に大きな影響を与えることになりかねないリスクが小さくはないということを認識しておく必要がある。

4．ソフトウェアの品質確保に対する考え方の変遷　～情報システムの変遷とともに～

前項の IPA 調査によるソフトウェアの品質の状況は、情報システムの品質が芳しいとは言い難いことを示している。それは、現代社会の基盤を支える重要な情報システムが多数存在することから、システム運用中にソフトウェアのバグによる不具合が発生したときに、システムに与える影響、さらにはその影響が社会生活に与える影響は計り知れない。したがって、ソフトウェアの品質を確保することは、時代の変遷を経てもシステムの開発においては常に最も重要な要素の一つであることには違いない。ソフトウェアの品質を確保する手段は、レガシーシステムの時代からさまざまな考え方が浸透しているが、DX 時代になるとシステム化の目的であるゴールが曖昧で変化を伴うことが、システム開発のやり方にも変化を与えることになり、その結果、ソフトウェアの品質確保の考え方にも影響を与えることになる。

次項より情報システムの変遷に伴い品質確保の考え方も変化していることを述べる。特に、レガシーシステムの時代に確立されたといってもよい品質確保の考え方が、その後の DX 時代のシステム開発においては、どのように変化しているかを述べている。

5．レガシーシステムのソフトウェア品質確保の考え方

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。このシステムの開発は、比較的に多くの開発要員で長期間を要する大規模な開発が多く、いわば“重厚長大”な開発である。この時代のシステム化の主な目的は、各企業や官公庁などの組織体において手作業で行われていた事務処理などの業務を IT 化することであった。コンピューターの用途が、政府などの特別なユーザーによる高度な技術計算の利用から、一般企業などの利用に拡大した。したがって、IT 化の対象となるユーザー企業の業務を詳しく分析して、そのビジネスロジックを検討しプログラミングで実装するという業務システムの開発が主流になった。この特徴は、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が明確であることである。したがって、開発の初期段階にその要件を明確に表現することが期待される。そして、明確にした要件を確実にシステムとして実現するために、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進めるウォーターフォール型スタイルで開発が行われた。

この開発スタイルの課題は、開発の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、そこから誤りを修正して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。

この課題を解決するためには、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大幅な後戻りを発生させないことである。これらの課題の対処方法では、多くの失敗や経験を経た結果、レガシーシステムの開発におけるソフトウェアの品質確保の考え方が以下のように確立していった。

(1) 段階的に動作確認を行う（V 字モデルの各テスト）

ウォーターフォール型の開発では、上流の工程において設計を段階的に詳細化して最後の段階でプログラミングを行う。プログラミングが終わると下流工程のテストに入るが、テストといっても単工程ではなく、上流工程の各工程のアウトプットが正しく動作することを確認するために、上流工程の詳細化とは逆に段階的に大概化した複数工程のテストを実施する。これにより、上流工程の設計や実装の各工程のアウトプットに誤りがないことを段階的に確認する。

(2) 当該工程の完成度を高める（レビュー）

ウォーターフォール型開発の最大の欠点は、誤りを検出したときに誤りを埋め込んだ工程まで後戻りして設計や実装をやり直すことである。これを防ぐには、テストを充実するだけでは解決できない。なぜなら、テストで検出した誤りを修正するための後戻り工数が大きいことには変わらないからである。また、システムのすべての動作経路をテストで確認するには、テストケースが指数関数的に増えて天文学的な数になるため実際には不可能である [4]。

そこで、テストを実施する前の段階において、各設計工程のアウトプットを可能な限り完成度の高いものにして、作り込んだ誤りをできるだけ早い段階で除去するほうが効率的である。そのための方法として、各工程のアウトプットをレビューすることが重視されるようになった。各工程のアウトプットに作り込まれた誤りをレビューによってすべてを摘出するという考えである。

レビューの欠点は、レビューで対象物の誤りや漏れを指摘する量や質がレビューする技術者のスキルや経験に依存することである。原理的にはソフトウェアのバグがないことを証明できない（チューリング , On Computable Numbers, with an Application to the Entscheidungsproblem（計算可能な数について、および決定問題への応用について）,1936 年）といわれるように、“完全な設計（仕様）”を表現することは不可能である。したがって、設計者がスキルや時間の制約の中で可能な限りを尽くして作成した成果物であっても、どこかに誤りや分析 / 詳細化不足や曖昧さなどが残ることは否めない。レビューにおいても最適な人材配置の制約や時間的 / 物理的な制約などにより、レビューにより“完全な設計（仕様）”に仕立て上げることにも限界がある。特に、レビューでレビュー対象物に記述されていることの誤りは気付き易いが、記述に漏れがあった場合は、記載されていないことを指摘することは経験の豊富な技術者がレビューをしなければ難しい。また、レビューは非常に神経を使う人的な作業であるので、レビューの進行を効率よくスムーズに運用できないと、レビューの参加者は疲れて本来指摘すべきことも指摘できなくなる。

しかし、レビューでは、本来指摘すべきことを見逃すようなことがあってはいけない。そのために、国際標準化機関でもレビューの実施手順を定めており [5]、参考にすることが多い。また、実施手順に従って着実にレビューが実施できるように、次に示すようにプロセスを管理することも効果的である。

(3) 開発作業の計画と実行を確実にする（プロセス管理）

当該工程の完成度を高めるための施策としてレビューを行い、当該工程における問題点をすべて摘出してしまう方法は理にかなった考え方ではあるが、実際には理想的なレビューを継続的に安定して実施することは非常に難しい。レビューの参加者のスキルや経験により指摘内容のレベルに差が生じるし、レビュー時の参加者の心理状態に左右されてしまったり、当該開発者は早く次の工程に進みたい心理が働くので、レビューの重要性を理解しつつもレビューの時間が充分に取れないまま終了したり､レビュー作業を簡略や省略してしまう場合もある。レビューに限らず、他の作業においても同様なことがいえる。ソフトウェア開発は人的作業であるので、やるべき作業を怠ってしまうことが誤りを埋め込むことにつながることは明白である。そこで、やるべき開発作業の計画を明確にして、計画したことを確実に実施できていることを確認するために、プロセスを管理することが品質確保につながる。

プロセスを管理する方法は、プロセスの計画や各工程の管理などを誰がいつどのように実施するかを組織で取り決めて実践する。また、IOS9000 や CMMI（Capability Maturity Model Integration 能力成熟度モデル統合）を活用して第三者によるプロセス監査を行うことで、自組織の弱点を改善してプロセスを成熟させていくことも重要である。熟練者を“ベテラン”や“匠”と称するように、組織自身も少しずつ改善を積み重ねて成熟していくことで、レベルの高い組織プロセス能力が培われるようになる。プロセス管理を主体的に実施する組織は、納期優先の心理が働きがちな開発者組織ではなく、品質保証部門のような開発組織とは別の第三者組織が役割を担うと、より効果を発揮する [6]。

(4) プロセスや品質の見える化（品質管理）

ソフトウェアの開発は結局、論理設計をすべて人的作業で行うため、プロセスの実施内容や品質の状況などの開発の実態を把握しにくい。そこで、ソフトウェアの開発作業においても、ハードウェア的な品質管理手法を取り入れて“見える化”することで、開発状況を客観的に把握して共有できるようにする。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 [7] した手法である。この品質管理を組織内で徹底して運用することにより品質が向上した例 [8] も多くみられる。例えば、前述したレビューの運用が充分に実施されたかを客観的に把握するために、レビューの時間数や指摘数などを記録することがよく行われている [9]。さらに、定量的管理だけでなく、開発時のさまざまな定性的な情報も分析し管理することで、品質確保をさらに効果的に実施した事例 [10] もある。
このように、ソフトウェア開発の作業状況の見える化の品質向上への寄与は大きい。

6．オープンシステムのソフトウェア品質確保の考え方

レガシーシステムでは、IT メーカーがハードウェアからソフトウェアまでのシステム構成の全てを自作する垂直統合スタイルであったが、1990 年代の頃からメインフレームを使用しないオープンシステムが台頭してきた。オープンシステムでは、ハードウェア、基本 OS、ミドルウェアなどを自由に組み合わせて構成する水平分散スタイルに変わった。代表的なシステム構成例では、標準化されたハードウェアとして CPU は Intel、基本 OS は OSS（オープンソースソフトウェア）の UNIX（後に LINUX)、ミドルウェアの言語やデータベースも OSS で構成されたシステムである。

レガシーシステムは、システムに組み込むすべてのソフトウェアを開発組織が自前で作成していたので、システムに問題が発生すると、システムを開発した組織が責任をもって問題に対処する。問題個所の設計書やソースコードを見直して、原因を調べ、誤りを修正し、修正版をリリースする。

しかし、オープンシステムでは、同じような対処ができない。ソフトウェアの側面では、OSS の開発者（実態はコミュニティ活動が多い）には基本的に個々のユーザー先で発生した問題を調査する責任はない。OSS を取り込んでシステムを開発した組織が責任を持つことになる。ここにユーザーとの摩擦が生じる。ユーザーには OSS を利用しているかどうかなどは知る由もなければ、OSS だから調査できないなどという理屈は通らない。そこで、システム開発側は、OSS を利用するときには、その品質を把握して悪い品質のものは利用しないという判断をしなければならない。OSS を利用することにより、自らソフトウェアを開発しない代償としてのリスクは極めて大きい。このリスクに対して、OSS の品質確保をどのように考えるのかを以下に列挙する。

(1) 徹底的にテストをする

自作開発のように設計書やソースコードをレビューするようなソフトウェア開発の上流工程における品質確保の手段を取れない（ただし、公開された OSS のソースコードを理解することは可能）ので、テスト工程において品質確保の手段をとる以外にない。したがって、利用する OSS をシステムに組み込んだ状態で、徹底的にテストをする。テストの工数比率がレガシーシステムよりも大きくなるが、その分、ソフトウェアの上流工程の開発コストが削減されているので、そのコストバランスを注視する必要がある。

(2) OSS のコミュニティの活動状況から品質を推定する

OSS の開発はボランティアベースで運営しているコミュニティが多い。このコミュニティの活動状況、例えば、バグ FIX 版（バグを修正したバージョン）がどれくらいの頻度で提供されているか？コミュニティの情報発信は活発か？活動停止状態にないか？などの情報をもとに、品質を推定する。この場合、バグ FIX が多いことを品質が悪いと判断するのではなく、頻繁に修正されていることは、多くの利用者が存在していることであり、OSS の品質が向上していることを示しているととらえる。

OSS が台頭してきた当初は、OSS を利用することは、ハイリスク・ハイリターン（低品質のリスクが大きいが、品質に問題がなければ大幅に開発工数が削減できるというリターンも大きいという意味で使用）であった。しかし、ソフトウェアの特徴は“劣化しない”ことであり、OSS も利用され続けることによりコードが洗練されてくるので、OSS が台頭し始めてから 30 年以上も経過した今の時代には、OSS の品質が大きな問題として取り上げられることは当時よりも格段に減少したようにみえる。今では、当たり前のように OSS を利用している。もちろん、OSS の品質が万全でバグがゼロであるわけではないし、セキュリティ脆弱性に関しては今でも重要な問題であることには変わりがない。しかし、今の時代に OSS を利用しないでシステムを作成することは不可能といってもよい。ソフトウェアの核となるプログラム言語とその豊富なライブラリィはほぼすべて OSS であり、レガシー時代のようにシステム開発組織が自ら開発する場面は、システムの核となるビジネスロジックを含むアプリケーション以外にはないといってよい。ある意味、OSS が市民権を得たといってもよいだろう。

このように自らソフトウェアを作る場合とOSSを利用することで自ら作らない場合の比率が反転して、OSSを可能な限り利用する開発のしかたは、ソフトウェアの品質確保の考え方を大きく変化させた。OSSを組み合わせることにより、自ら作成しているコードの比率が小さいため、コードの論理的な誤りを正すための品質確保の活動も小さくなる。その結果、自ら作った部分の論理的な誤りを探すこと（後述する当たり前品質）よりも、ユーザーインタフェースなどのように、よりよく見せる工夫やより分かりやすく見せる工夫など（後述する魅力的品質）をより重視する開発にシフトしていった。

品質の面では、ソフトウェアは利用者が増えて様々な使われ方を経ることで、ロジックのバグが修正され洗練されていく特徴があることを踏まえれば、OSSが世界中で利用されて洗練されていく過程における品質と、特定企業むけのビジネスロジックを特定の企業にだけある程度決められたパターンで利用されている場合の品質は、どちらの品質がより洗練されていくかを相対的に考えることも、OSSを利用する上では検討すべき重要な要素である。そのときの条件は利用者が多いということであり、著名なOSSであればある程度の年月を経て多くの利用者により洗練されていることが想定できるが、例え先進的なロジックを持つOSSであったとしても年月を経ていない場合や利用者が多くない場合（特殊な形態でしか利用されない）は、品質面から採用するかどうかを慎重に検討する必要がある。

多くの利用者により長期間にわたり利用されたソフトウェアは、新しいアルゴリズムやロジックを追加しなければバグが減少して当然である。そしてDX時代のシステムは、OSSを存分に利用した上に、クラウドシステムのＥＣサイトアプリケーションであれば、商品を並べ購入クリックにより決済のロジックに遷移するというような、同じパターンの処理や画面遷移を持つ場合が多い。このようなシステムでは、特に新規性の高いビジネスロジックを除けば、アルゴリズムの信頼性よりも、商品の見せ方などの画面イメージや操作性などのユーザーインタフェースの設計が中心となり、多くのユーザーにクリックしてもらうための外部設計に注力することのほうが重要になってきている。

7．クラウドシステムのソフトウェア品質確保の考え方

2010年頃より，AmazonのAWSを筆頭にGoogleやMicrosoftなどのクラウドサービスが浸透してきた。ネットワークの高速化やセキュリティ機能の高度化に伴い、クラウドサービスを利用することでサーバーなどのインフラ運用を気にすることなく、提供するサービスに集中したビジネス展開が可能となった。したがって、情報システムの開発は、Webアプリケーションやスマホアプリケーションのように、ほぼ常にオンラインで動作するクラウドシステムの開発にシフトしていった。クラウドシステムの特徴は、レガシーシステムが特定の利用者に限定されるのに対して、スマートフォンなどのインターネット端末を使ってインターネットショッピングをするように不特定多数の一般利用者が容易に利用できることである。さらに、SNSの普及により利用者のさまざまな声（対象への評価）が容易に、時には安直に共有される時代になったことである。

クラウドシステムによる不特定多数のユーザーに対するサービスを提供するための開発は、特定ユーザー（特定企業）むけの明確な要件を実装するレガシーシステムの開発とは大きく異なる。その一番の違いは、オンライン上のソフトウェアを容易に何度でもリリースして置き換えることができる点にある。クラウドシステムではCI/CD（Continuous Integration /Continuous Delivery）により、

継続的にシステムを統合・テスト・リリースできる環境にあるので、機能のバージョンアップやバグの修正版などを容易にリリースができる。したがって、システムの開発スタイルもアジャイル型開発が適合する。

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て開発できたり、要件の変化に柔軟に対応して開発できたりする長所がある。しかし、一方では、繰り返し開発の終了判断が難しくなり、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。

したがって、これらのクラウドシステムの特徴を踏まえると、品質確保の考え方もレガシーシステムの考え方とは必然的に異なる。以下にクラウドシステムの品質確保の考え方を挙げる。

(1) 自動テストの網羅性の検証

繰り返し開発することにより何度もリリースできることから、“バグがあればすぐにリリースし直しをすればよい”という甘い考えに陥り易い開発スタイルであることも否めない。しかし、サービスを利用した人がバグによる悪い影響を受ければ、そのサイト（ソフトウェア）の信頼性が低下することは言うまでもない。したがって､容易に何度もリリースできる繰り返し開発では、CI/CD の中で実施すべき自動テストの内容や網羅性が十分であるかの検証が信頼性を確保する上で非常に重要になる。

回帰テストも含めて、リリース直前のテスト内容に漏れがないことを保証するための検証などの活動を開発プロセスに組み込むことが必要である。このテスト内容は、レガシーシステムに比べると 1 回のリリースに新たに提供される機能は相対的に小さくなるので、レガシーシステム時のテスト工程よりもリリースまでのテストの工数は比較的に少なく済むと考えられ、その点からも漏れや抜けのないテスト内容であることを確実にすることが品質確保につながる。

(2) 上流工程における品質の確保

前項のようにテストにおいて品質を確保するだけでは不十分である。レガシーシステムにおけるウォーターフォール開発でも上流工程における品質確保が重要であったと同様に、繰り返し開発であっても上流工程でのレビュー活動などの品質確保の活動は基本的に必須である。アジャイル開発では､“スプリントレビュー”や“ペアプログラミング”が上流工程の品質確保に相当する。結局、アジャイル開発といえども、1 サイクルの設計、コーディング、テストにおける各作業の品質確保は、ウォーターフォール開発で講じられる品質確保の考え方と変わらない [11]。

8．AI システムのソフトウェア品質確保の考え方

2010 年頃から始まった第三次 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。

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 モジュールの精度を予測できず、開発のゴールとして正解かどう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、AI モデルの選択やチューニングを変更して、AI 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。

二つ目の特徴は、リリース後も AI モジュールの精度を観測して必要に応じて再開発を講じなければならないことである。例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精度を得る保証はない。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、現場の環境変化や時間経過により対象データの性質が開発時に学習したデータから変化してきたと判断すれば、現場に適合した最新のデータを使用して学習するステップから再開発をしなければならない。

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することが難しく、“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 システムの特徴を踏まえた品質確保の考え方は以下となる。

(1) 学習データの網羅性の検証

AI モジュールを生成するための学習データについて、単にデータがバイアスやノイズを含んでいないことの検証だけでなく、学習データと現実の世界のデータとの差異を分析して現実の世界で想定できるデータを網羅的に準備できているかを検証する。さらに、その検証には、AI モジュールの対象物のあらゆるパターンや制約事項を洗い出す不確実性に関する分析プロセスのアウトプットを検証することも必要である [12]。

(2) AI モジュールの精度のリリース判断

AI モジュールの生成が確率的手法によるため、同じ実行結果を得られない不確実さをもつ。したがって、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が難しい。AI モジュールの精度は予測困難なため、実行してみないとわかないという帰納的な開発を繰り返さなければならない。この繰り返しをいつまで続けるべきか、作り直すプロセスが単に AI モデルのチューニングの変更なのか、それとも学習データの再作成が必要なのかなどの判断を適切に行う必要がある。この判断により開発コストやリリース時期を制御しなければならない。ときには、AI モジュールの精度が未知のため、その不確実性をカバーするためのフェールセーフ等の機能を搭載するかどうかの設計レビューも必要である [12][13]。

このような帰納的開発において、その不確実性に対応するためには、AI モジュールの評価が、正解値との照合ができない（テストオラクルがない）代わりに、最適な方向に向かっているかの“差分分析”（後述）による評価を取り入れることである。

(3) リリース後のモニタリングと再開発の判断

学習したデータとは想定外の現実世界のデータの影響により AI モジュールの精度が下がる可能性がある。したがって、リリース後に精度をモニタリングする仕組みと AI モジュール劣化による再作成の判断プロセスの仕組みを構築しておく必要がある。そして、AI モジュールが異常な結果を出した場合にすぐに再作成が必要かどうかを判断できるようにする [12]。

以上により、レガシーシステムから AI システムまでの情報システムの変遷と、それによるソフトウェア品質確保の考え方を述べてきた。これらを整理すると表 1 になる。大きな特徴は、明らかにレガシーシステムの頃に比べて、DX 時代のクラウドシステムや AI システムの品質確保の考え

方は複雑である。レガシーシステムの品質確保の考え方は、プロセス品質確保や定量的な品質管理のように決められたことを確実に遵守することが求められる。これに対して、DX 時代の品質確保の考え方は、プロジェクトの特徴によりケースバイケースで考える必要があり、それぞれの目指すべきゴールを開発者全員が共有して、その時点における最適な品質確保とは何かを考えながら進めていくべき性格のものである。これが、最後に述べる組織作りの考え方にも影響を与えることになる。

表 1. 各システムの開発プロセス等の特徴

システムの種類	主なゴール / 要件の特徴	主な開発プロセス	主なリリース後の特徴
レガシーシステム（独自 HW+ 独自 SW）	特定顧客の業務システム / 組み込みシステム →要件は顧客と明確にする	ウォーターフォール型開発プロセスにより、後戻り作業を抑えるために各工程の完成度を高くする	リリース後は修正や再開発はできる限り最小限に抑え、手離れをよくする
オープンシステム（標準 HW + OSS+ 独自アプリ）	特定顧客の業務システム / 組み込みシステム →要件は顧客と明確にする	OSS で実現した機能が自作部分との連携部の動作や要件を充分に実現できているかを確認するためのテストプロセスを重点的に行う	OSS の部位の問題の切り分けや、問題を特定しても修正されるとは限らないための代替手段を講じる等、必要以上に手間がかかる
クラウドシステム（クラウド基盤 + OSS+ 独自アプリ）	不特定多数の一般利用者向けサービスが浸透し、この場合のゴールは不明確かつ常に変化する	アジャイル型の繰り返し開発プロセスにより、小規模機能の開発とリリースをスピーディに繰り返す	不明確で変化するゴールを把握するために、常に利用者の声や利用状況を分析して、開発にフィードバックする
AI システム（様々な基盤 + OSS+ 独自アプリ）	AI モジュールの精度が確率的で不確定であり、適用時の想定外の環境変化などにより、ゴールが不明確かつ達成度を予測できない	AI モジュールの結果をもとに、帰納的開発プロセスによりチューニングや再学習などの開発を繰り返す	AI モジュールの結果の不適合状況を監視し、性能劣化に応じて再学習や再開発を行う

9．DX 時代のシステムの品質確保の考え方の提案 ～より柔軟なゴール達成型組織を目指して～

DX 時代の情報システムのゴールは不透明かつ不確実であり、さらに時間とともに変化する可能性もあるという特徴をもつ。この特徴をふまえて適切な開発プロセスを講じるとともに、適切な品質確保の考え方を取り入れることが肝要である。適切な開発プロセスとは、クラウドシステムの開発プロセスに適した繰り返し開発であり、AI システムの開発プロセスに適した帰納的開発である。これらの開発プロセスでは、常にゴールの仮設を立ててその仮説を実現する機能を実装してリリースする。そしてリリース後のシステムの稼働状況を監視して、可能な限り短いタイミングで仮設検証を行い、必要に応じて軌道修正の開発をさらに続ける。つまり、繰り返し開発や帰納的開発とは、本番適用の場での仮説検証により最適解を追及し続けるために、開発へのフィードバックを繰り返しながらシステムを成長させていくことである。

しかし、検証の結果は予測不能であり、どれほど開発を繰り返す必要があるのかを見積もることができない。これが、繰り返し開発や帰納的開発などの仮説検証型の開発プロセスの大きな課題である。これは、プロジェクトのリソース計画立案時に適切な予算の見積もりが難しいことにつながる。これも従来の情報システムの開発の考え方とは大きく異なる問題である。

それでは、仮説検証がいつ終わるのかを予測できないというビジネス観点においても悩ましいこの問題をどのように対処すべきか？ここに DX 時代にむけた新しいソフトウェア品質確保の考え方を取り入れるべきと考える。そこで、これまでに述べてきたソフトウェア品質の考え方の変遷を踏まえて、DX 時代においてはケースバイケースで適切にソフトウェア品質をマネジメントするという新たな考え方について、具体的に以下に提案する。

・ **提案１：ゴール達成の方向性の差分分析を行う（新たな品質マネジメント１）**

差分分析とは、動作中のシステムの“前回の結果”と”今回の結果“の差を分析して、向かうべきゴールに近づいているのか、遠ざかっているのかを判断し、ゴールに近づいていると判断した場合は現状の対策を継続し、ゴールから遠ざかっていると判断した場合には、何らかの軌道修正を行い次回の差分分析において、この軌道修正が正しかったかどうかを検証する。これを繰り返すことにより、ゴールが不確実で曖昧な状態であったとしても、ゴールの方向（ベクトル）を見誤らなくすることで、気が付いたらゴールが遠く離れていた、または、見えないゴールを意識せずに無理矢理に進んでしまうという大きなやり直しを回避できる。

では、この差分をどのように把握すればよいか？クラウドシステムであれば、単純に SNS などのユーザーの声（評価）や、すでに多くのデータアナリティクス指標が定義されているようなユニークユーザ数やコンバージョン率などから最適と思われる指標値を観測し、新たな機能をリリースした前後を比較することで、当該機能が有効であったかを判断できる。また、AI モジュールの適合性であれば、これもすでに多くの実証実験を進めるプロセスの中で確立されているように、AI モジュールの適合率などを実際のリリース現場において観測して、その結果によっては AI モジュールのパラメーターチューニングを見直すことや、場合によっては機械学習のデータパターンの見直しから再開発することも発生する。

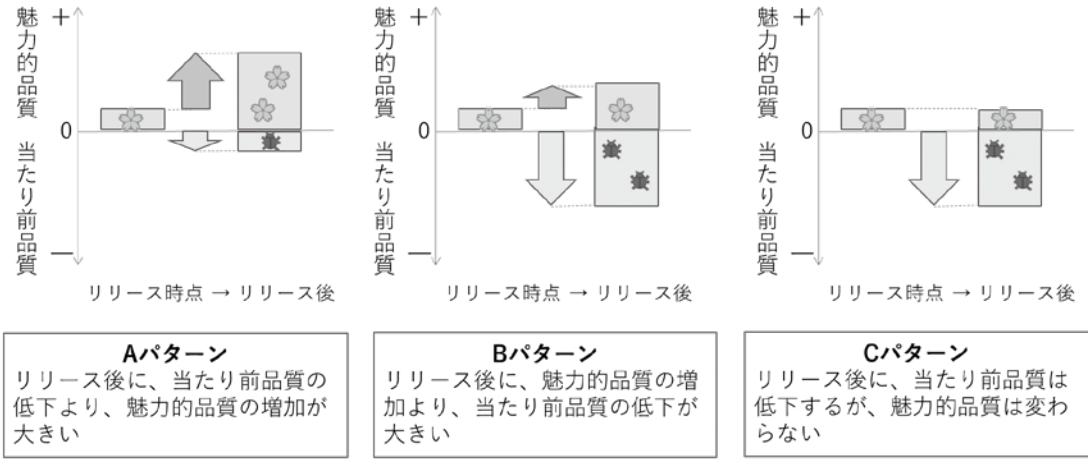
このように、すでに差分分析は多かれ少なかれ開発チームでは行われているはずであるが、これらをシステムの”品質“ととらえて品質をマネジメントする考え方を開発プロセスに定義し、開発チームが役割を分担する。さらに開発チームとは独立した品質保証チームとして組織横断的な役割を任せるということを積極的に行うことも肝要である。この意味は、レガシーシステムの時代にソフトウェアのバグを減らすための活動を開発チームとは別の独立した組織として品質保証チームが担当し、品質の確保を下支えする役割を担っていたことと同じ考え方である。

・ **提案２：魅力的品質と当たり前品質のバランス運営を行う（新たな品質マネジメント２）**

狩野モデルでは､品質は”当たり前品質”と”魅力的品質”に分類される。当たり前品質とは、期待どおりに動作するという当たり前のことができているか、すなわちバグがないかどうか、であり、魅力的品質とは、ユーザーインタフェースがわかりやすく誤操作を防止するデザインや操作性とか、あったらより便利に感じる機能などが相当し、特に魅力的品質が欠けていることをバグとは呼ばない。

この２つの品質の違いは、当たり前品質は最高の状態が（バグが）ゼロであり、バグが存在する分量だけ品質が悪いことを示すマイナスの状態となる。一方、魅力的品質は、魅力的品質が無い最低の状態がゼロであり、魅力的な要素が増える分だけプラスが増える状態になる。つまり、当たり前品質にはプラスの状態はなく、魅力的品質には逆にマイナスの状態がない。

図 1. 魅力的品質と当たり前品質のリリース後の増減パターン



※リリース時点では当たり前品質はバグが無いと判断してリリースされる。A パターンは、クラウドシステムなどの DevOps による開発とリリースを繰り返しながら魅力的品質を向上させ、リリース後に影響の小さい範囲のバグが露呈したとしても、それ以上に魅力的品質を上げることで総合的な顧客満足を下げない。B パターンは、リリース後の魅力的品質より当たり前品質が大きく低下することで総合的な顧客満足が低下してしまう。クラウドシステムでリリース後の魅力的品質の向上のための繰り返しリリースが利用者の暗黙の期待に応えられない状態や AI の精度が本番環境に適合できなかった場合が想定される。C パターンは、レガシーシステムではリリース後に魅力的品質を上げることは困難なため、当たり前品質が低下して顧客満足を下げる可能性のみが残る。

DX 時代は、この魅力的品質と当たり前品質を、システムの特性によりバランスよく制御することが重要になってきている（図 1）。

魅力的品質は様々なプラス要素が考えられるため無限に広がっていく可能性を持っており、ここでシステムの価値が評価されるといってもよい。不確実なゴールに対して新たな価値を追加して魅力的品質を増加させるために、リリース後の利用状況を監視しながら繰り返し開発や帰納的開発を行う開発プロセスが重要になる。例えば、Web アプリケーションの EC サイトなどを例に挙げると、少しでも多くのお客様を引き付け、サイトの滞在時間を増やすなどの魅力的品質をどれだけ伸ばすことができるかが勝負となる。

ただし、魅力的品質に注力するばかりで、当たり前品質を疎かにしてはいけない。クラウドシステムでは DevOps による Web サイトやスマホアプリのアップデートは、魅力的品質を向上するために頻繁に行われる傾向が強くなるが、当たり前品質の側面では、これらのアップデートにより、返って使い勝手が悪くなることやデグレード（修正などの影響により既存の機能が使えなくなること）が発生することは絶対に避けなければならない。たとえ、プラスが増えても、それ以上にマイナス面を大きく引き下げることは、品質マネジメントのバランスの悪い運営であるといえるので細心の注意が必要である。これは、従来からあるデグレードテストを着実に行うプロセスを怠らないという伝統的な当たり前品質確保の考え方を踏襲していることが大前提であることは言うまでもない。注意すべきは、当たり前品質のマイナス面は、単にバグの分量だけではなく、例え 1 件のバグであっても利用者に大きな損害を与えるような問題を引き起こしてはならないことも当たり前品質の最低条件である。

このように魅力的品質と当たり前品質のバランスの良い運営を行うためには、提供するシステムの特性をよく踏まえることが必要になり、表 2 のように整理して適切なバランス運営を行うことが新しい品質マネジメントの考え方として重要である。

表 2. アプリ種類と品質の特徴

アプリ種類	主な用途	主な利用者	主な特徴	魅力的品質	当たり前品質
Web アプリ（フロントエンド側の場合）	ゲーム ,EC（電子商取引）, ニュース , ビジネスなど	一般ユーザーなどの不特定多数の場合が多い	操作パターンがシンプル・操作数が少ない	ユーザー嗜好変化に追随し、見やすい、直観的に操作し易い、追加のサービス強化など	ユーザーの信頼を損なわない程度、または、それ以上に魅力的品質が上回ることが重要
モバイルアプリ（フロントエンド側の場合）	ゲーム , SNS, EC（電子商取引）, ニュース , ビジネスなど	一般ユーザーなどの不特定多数の場合が多い	操作パターンがシンプル・操作数が少ない	ユーザー嗜好変化に追随し、見やすい、直観的に操作し易い、追加のサービス強化など	ユーザーの信頼を損なわない程度、または、それ以上に魅力的品質が上回ることが重要
サーバーアプリ（ビジネスロジックを持つ場合）	ビジネス ,DB アクセス , 課金 / ユーザー管理 , セキュリティ保護など	ユーザー企業などの特定範囲内の場合が多い	ビジネスロジックなどシステムの基幹部を持つ場合が多い	開発時にユーザーと合意した要件以上のことは期待されていない	ビジネスロジック等の基幹部は限りなくバグゼロであることが最低条件

- **提案 3 ：高速 PDCA による目標にむけた改善スピードの向上（新たな品質マネジメント 3 ）**

PDCA とは、計画（Plan）- 実行（Do）- 評価（Check）- 改善（Act）のサイクルを繰り返すことにより業務の改善を進める伝統的な方法である。DX 時代のシステムは、ゴールが曖昧で不確実かつ変化する中で、ゴールを探りながら少しでも今期待されるゴールに近づいていく必要がある。そのためには、繰り返し開発や帰納的開発を行い、可能な限り短いサイクルでシステムの軌道修正を続けることが重要になる。この軌道修正のための改善サイクルを素早く行うこと、すなわち PDCA プロセスを高速に回すことが、ビジネスの成功につながることになる。提案 1 で述べた“ゴール達成の方向性の差分分析”を､高速に PDCA を回すことで実現させる。これにより、短いサイクルで軌道修正を行い、ゴールの差異を最小にとどめながら、不透明なゴールを少しでも明確にし、また変化に追随していくことが可能となる [14]。

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組織活動に対して、組織の画一的･統制的な戦略は却って開発現場の意識との乖離が生じるため効果的ではない。全体最適が非効率で個別最適の考え方が重要になる。これについて、提案 4 で述べる。

- **提案 4 ：ゴール達成型組織の構築**
  ～品質マネジメントの考え方を突き詰めると組織経営と人材育成につながる～

従来のレガシーシステム開発では、ゴールが明確なので固定的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、画一的・統制的な製造業的手法であった。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、自らが考えて工夫をする機会を奪いかねない。つまり、個々に割り当てられた“アウトプット”に注力すればよく、“アウトプット”の先にある“アウトカム（成果）”を意識しなくとも任務を全うしたことになる。しかし、DX 時代のシステム開発では、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、開発関係者一人ひとりがゴール達成（すなわち、“アウトプット”ではなく“アウトカム”）を常に強く意識して、この目的を達成するためには従来の方法にとらわれることなく自由な発想で自ら考え行動する自律型の人材を育成していくことが重要である。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 も差分分析も遂行することは難しい。そして、自律型人材の育成は、“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”[15][16] へと進化を遂げることが期待できる。

10. まとめ

ゴールが不確実な開発では、階層化された組織における強いリーダーがゴールを示して組織を統制する従来のやり方は適していない。経験したことが無い新しい価値を求めるが故にゴールが不確実である訳だから、リーダーの豊富な経験は不確実なゴールの対処法にはそれほど効果を期待できないだけでなく、新しい価値の創出の妨げにさえなりかねない。それよりも、多くの人が意見を出し合いながらゴールを検討するほうが、不確実なゴールの達成に早く近づく。そして、組織の全員がゴールを達成するための意見を自由に戦わせることができる“心理的安全性”を伴う環境を整えることにより、組織に所属するひとりひとりがゴール達成の共通目標をより強く意識して、ゴール達成にむけてアイデアを創出し、仮説検証型の繰り返し開発や帰納的開発を行うことができる。そのためには、開発する組織の規模が大きすぎると非効率になるので、全員が議論しやすい程度の小さ目の規模の組織運営のほうがよい。まさに、アジャイル開発の組織運営と共通している考えである [17][18]。

DX 時代は、IT 技術の進化に伴い、クラウドや AI を使った新しいサービスを提供する情報システムが増加している。そして不特定多数のユーザーを対象とするシステムや AI のように結果を予測できないようなシステムは、明確なゴールが見えない中で開発とリリースを繰り返しながら手探りでゴールを明らかにしていくため、従来のレガシーシステムのゴールとは真逆である。このような情報システムの変遷によりシステム化の目的が 180 度の変化をしていることを十分に認識してソフトウェアの品質の確保の考え方も変化に追随していくことがとても重要である。このことは、経営情報の観点からも次のように同じ方向性を示唆している。“情報システムは、その技術的側面からのみ経営の成果を考えるには限界があり、人的資源や組織構成も情報システムの技術的側面と相互作用しながら、反復的に成長することにより経営を支えていくことが重要な視点であり、さらにシステム開発と運用・保守についても、オーバーラップして再帰的、相互構成的に影響しあうことで、動的なビジネス環境の経営の変革を支えていくものである”[19]。

スマートフォンという端末が充足し、クラウドというインフラが整備された現代では、スマホアプリでどのような価値のある情報サービスを提供できるかが情報システムのこれからの課題であり大きなビジネスチャンスである。多くのビジネスが飽和した現代においては、新しい価値の創出によるビジネスインテリジェンスを狙うどの組織においても、そのための情報システムの環境はほぼ平等に整っている状況といえる。SNS などの普及により利用者の声や評価がビジネスに大きく影響を与えるほどに情報が散乱している。また、このクラウド環境では、DevOps による繰り返しリリースによりアプリを改善・改修が容易にできる環境でもある。この整った環境で、新たな価値を提供する情報サービスは、サービスの価値を高めること、すなわち、魅力的品質をいかに高め続けるか

を、開発時だけでなく、リリース後も常に監視し軌道修正をすることが重要である。

かつて、1960 年代前後の高度経済成長期に日本の自動車メーカーは、生産プロセスの改革や品質管理の徹底により自動車産業を急速に発展させて、“日本品質”と呼ばれる世界的な競争力を獲得した。このときの“日本品質”とは主に故障が少なく信頼性の高さが評価されたこと､すなわち、Made in Japan の“当たり前品質”が世界を席巻した。翻って、近年の“日本品質”が、高級料亭などにみられるようなお客様の想像をはるかに超えた“おもてなし”による満足感を指すとすれば、これはまさに“魅力的品質”の向上である。

これからの情報システムの目指すところは、クラウドシステム等の不特定多数のユーザーに最適なサービスの向上を継続的に提供していくことにより“魅力的品質”を向上することである。サービスの新たな価値を継続的に提供するための作業工程は、サービスのマーケティングや企画などのシステム開発の超上流工程の作業だけでなく、リリース後の運用・保守におけるサービス状況の監視による繰り返し開発や帰納的開発を判断するまでの全工程に及ぶ作業である。したがって、DX 時代の品質マネジメントは、プロダクトマネジメント [20] を行う活動であり､”日本品質“の”おもてなし“による新たな価値の提供を行うには、品質マネジメントとマーケティングとプロジェクトマネジメントが一体化したプロダクトマネジメントを実践していくことにほかならない。

参考文献

［1］ 独立行政法人情報処理推進機構（IPA）社会基盤センター（IKC）,DX 白書 2023,2023
［2］ 経済産業省 ,D X デジタルトランスフォーメーションレポート ,2018
［3］ 独立行政法人情報処理推進機構（IPA）社会基盤センター（IKC）, ソフトウェア開発分析データ集 2022,2022
［4］ Myers, Thomas, Badgett, Sandler, ソフトウェア・テストの技法 第 2 版 , 近代科学社 ,2006
［5］ ANSI/IEEE, ANSI/IEEE1028-1998 IEEE Standard for Software Reviews,1998
［6］ 佐 藤 孝 司 , Comprehensive Evaluation for Quality, Productivity, and Delivery of Software Development Products, The 1st World Congress for Software Quality pp.1-10, 1995
［7］ Cusumano, Japan’ s Software Factories: A Challenge to U.S. Management, 1991
［8］ 佐藤孝司 , 山田茂 , 品質マップによるソフトウェア分析手法の提案プロジェクトマネジメント学会誌 , Vol.18, No.5 pp.35-40, 2016
［9］ 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
［10］佐藤孝司 , 山田茂 , Analysis of Process Factors Affecting Software Quality based on Design Review Record and Product Metrics, International Journal of Reliability Quality and Safety Engineering, Vol.23, No.4 pp.1650011.1-1650011.11, 2016
［11］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
［12］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
［13］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
［14］佐藤孝司 , 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 , 日本応用情報学会誌 Vol.16 pp.26-33, 2022

［15］フレデリック・ラルー , ティール組織 , 英治出版 , 2018
［16］広木大地 , エンジニアリング組織論への招待 , 技術評論社 , 2018
［17］ジョナサン・ラスマセン , アジャイルサムライ―達人開発者への道 , オーム社 , 2011
［18］英繁雄 , 奈加健次 , 平岡嗣晃 , 前川祐介 , ハイブリッドアジャイルの実践 , リックテレコム ,2014
［19］遠山暁 , 村田潔 , 古賀広志 , 現代経営情報論 , 有斐閣アルマ , 2021
［20］メリッサ・ペリ , プロダクトマネジメント , オライリージャパン , 2020