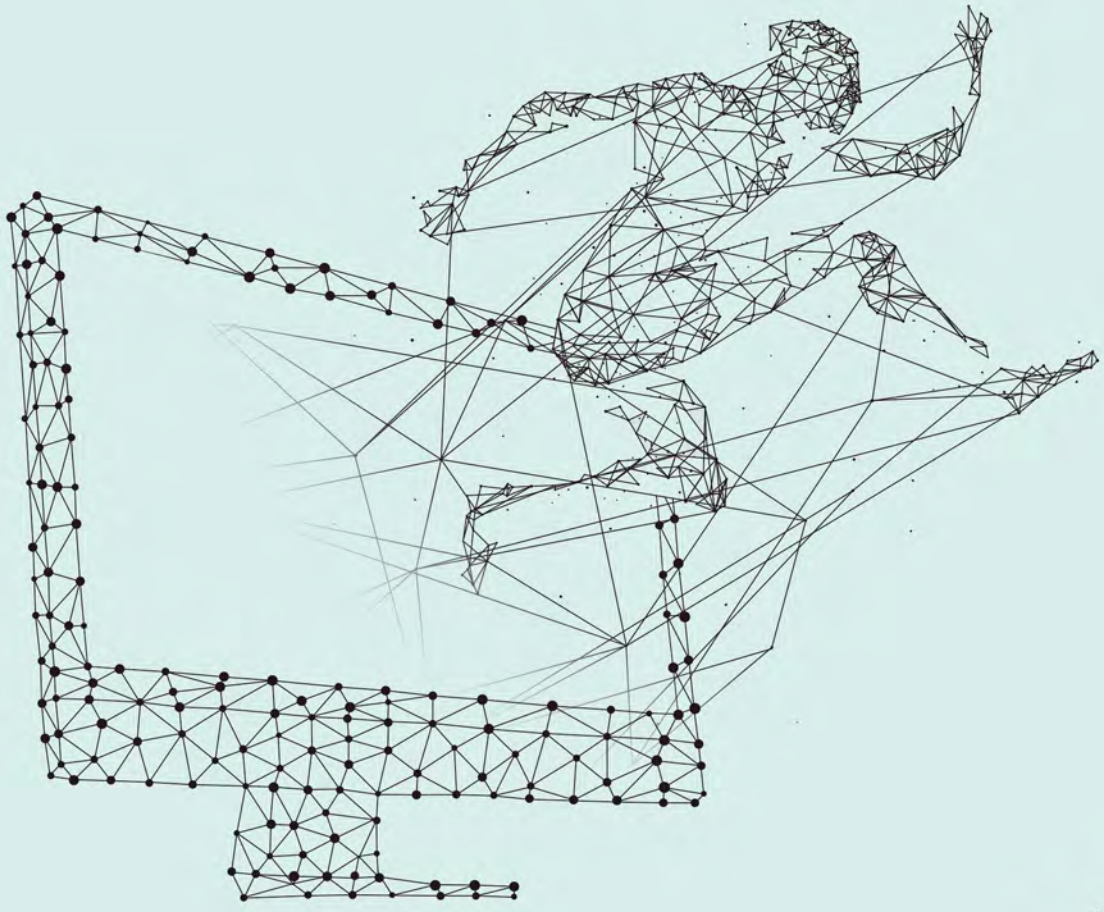
ベリサーブナビゲーション

Veriserve Navigation

DX時代に求められる ソフトウェア品質とは?

ベリサーブ アカデミック イニシアティブ 2020 DAY 2



2021 March Vol.22

DX時代に求められるソフトウェア品質とは?

ベリサーブ アカデミック イニシアティブ 2020 特集

Contents

表紙デザインコンセプト

前号のコンセプトである「最先端の研究をする人」がインプットした知識を元に「新たなチャレンジをしていく人」に変革する様子を表現しました。21号と22号で一つのストーリーになるように仕上げています。

Veriserve Academic Initiative 2020

ベリサーブ アカデミック イニシアティブ

DX時代に求められるソフトウェア品質とは？

ベリサーブ アカデミック イニシアティブ 2020 レポート

品質向上のリーディングカンパニー・ベリサーブがお送りする「ベリサーブ アカデミック イニシアティブ」は、最先端の研究や現場からのリアルなレポートなど、ソフトウェア開発のフロンティアからの研究報告によって明日の「品質の創造」を考えるイベントです。
5回目となる今回は、新たな取り組みとして場をオンラインに移し、2020年12月8日、9日の2日間にわたって開催しました。ベリサーブナビゲーションVol.22では、2日目の講演内容をダイジェストでお届けします。

2020年の講演プログラムについて

DX時代への突入に伴い、あらゆる企業が変革の機会を迎え、時代のニーズに沿った、新しい観点でのソフトウェア品質と、その品質のつくり方が問われています。
ベリサーブアカデミックイニシアティブ2020では、「DX時代に求められるソフトウェア品質とは？」というテーマのもと、この分野の第一人者である研究者や開発者の方々4名にご講演いただいたほか、ベリサーブ社員4名が当社の考え方や取り組みをご報告しました。
2日間のプログラムは以下の通りです。

12月8日（火）

基調講演
及川 卓也氏（Tably株式会社）
正解がない時代のプロダクト開発

技術セッション
佐々木 方規（株式会社ベリサーブ）
テストエンジニアがテストベース視点で考察した「Testing for 人工知能」

技術セッション
伊藤 由貴（株式会社ベリサーブ）
テスト自動化現場の「いま」と、次のステップに進むために

ゲストセッション
中野 直樹 氏（株式会社LIFULL）
ウェブサービス開発における品質改善の取り組み2020

12月9日(水)

基調講演
伊本 貴士氏（メディアスケッチ株式会社）
AI・IoT時代におけるプロジェクトマネジメントと品質管理のあり方

技術セッション
松木 晋祐（株式会社ベリサーブ）
AIとソフトウェア品質技術

技術セッション
桑野 修（株式会社ベリサーブ）
DX時代のITサービスに要求される「安心・安全な品質」とは？
～より安心・安全なITシステムの構築を支援する品質エンジニアとしてのアプローチ～

ゲストセッション
鷲崎 弘宜 氏（早稲田大学）
パターンQA to AQによるAgile Quality(アジャイル品質)への変革と事例

AI・IoT時代における プロジェクトマネジメントと 品質管理のあり方

メディアスケッチ株式会社
代表取締役

伊本 貴士氏
いもと たかし

メディアスケッチ株式会社代表取締役、サイバー大学専任講師、日経ビジネススクール講師、ふくい産業支援センター特別相談員、AI・IoT評論家。NECソフト株式会社、フューチャーアーキテクト株式会社などを経て、2009年にメディアスケッチ株式会社を設立。主に、企業への技術コンサルタントや共同研究開発を行う。教育分野においても数々の講演実績がある。フジテレビ ホンマでっか!? TVはじめメディアへの出演実績多数。

はじめに

現代は第四次産業革命（AI革命）の時代と言われます。AIの劇的な進化、AIとロボットの融合、IoTの普及は私たちの想像を超える速度で進み、今やこれまで当たり前と思っていた一般常識やビジネス習慣が通用しない新しい時代に突入しようとしています。このような革新的な時代において、企業は新たなビジネスを切り開かなくてはなりません。一方で、プロジェクトマネジメントや品質管理の在り方についてはますます複雑になりつつあります。本講演では、ソフトウェア開発において今後どのようなことについて気を付けていくべきかという点についてお話させていただきます。

産業革命とイノベーション

01 社会構造の変化にどう対応するか

はじめに、OpenAIという団体が作ったロボットの映像をご紹介します（図1）。この映像は、5本の指を持ったロボットが、片手でルービックキューブを回しながらパズルを完成させていくというものです。

実はこのロボットは、プログラムではなく、人工知能の制御により動いています。何万回、何百万回という試行錯誤の末、ルービックキューブを器用に解くことができるようになったのです。このことは、ロボットとAIの融合技術の進化により、すでにロボットは人間よりも器用に動くことができるようになっているかもしれない、という可能性を示唆します。

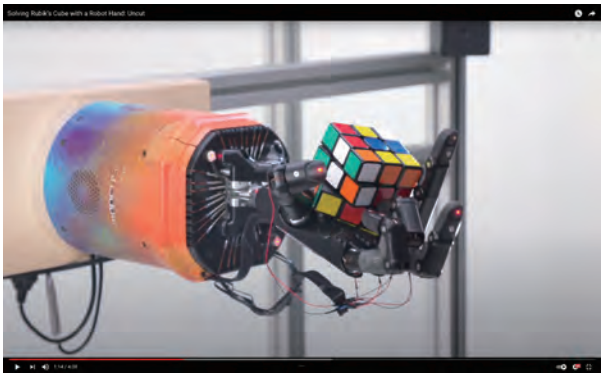


図1 出典：https://www.youtube.com/watch?v=kVmp0uGtShk&t=1s

このような技術が進むと、従来多くの人が就いていた事務作業や軽作業はAIやロボットに取って代わられ、この結果社会構造は二極化していきます。こうした時代に企業を存続させ、成長し、経済を支えていくためにはAIやロボット、IoTといったテクノロジーを核としたイノベーションへのいち早い取り組みが必要不可欠となります。

また、少子高齢化が進む現代では、労働人口が少なくなり一人一人の作業負担が増えていきます。人が少なくなる中で収益を維持しようとすると、企業はスタッフ一人一人の頑張りに頼るようになりがちです。しかし、人間は精神的にも体力的にも限界があり、こうした人への依存を続けていくと事故が起こりやすくなります。それは、結果的に企業に大きな損害をもたらすことになってしまいます。特に、AIやIoTの時代は開発や品質管理などあらゆる面で作業が複雑化し、やるべきことが増えていきます。事故を防ぎ企業を守るためには、ツールやテクノロジーの力を借りながら人への依存度を下げ、マネジメントの効率性を上げていかなければなりません。

ではこのような時代に対応していくには私たちはどうすればいいのでしょうか。

02 イノベーションのジレンマ

企業の成長にイノベーションは不可欠ですが、実際にはイノベーションに成功する企業ばかりではありません。ハーバード・ビジネス・スクールのクレイトン・クリステンセン教授は、その著書『イノベーションのジレンマ』で、従来製品の改良に重点を置く「持続的イノベーション」と、従来製品の価値を破壊して新しい価値を生み出す「破壊的イノベーション」と2つの概念を掲げ、特に大企業は持続的イノベーションに終始しがちになり、これが新興企業の前に力を失う理由になっていると指摘しています（図2）。このような、

巨大企業が新興企業の前に力を失う理由
クレイトン・クリステンセンが、1997年に初めて提唱

大企業では

持続的イノベーション ＞ 破壊的イノベーション

- 株主の意向が優先（目先の利益優先）。
- イノベーションの初期は市場が小さく、価値がないように見える。
- イノベーションの初期は不確実性も高く、価値がないように見える。
- 既存事業を営む能力が高くなることで、異なる事業が行えなくなる。
- 既存技術を高めることと、それに需要があることは関係がない。

図2 イノベーションのジレンマ

イノベーションのジレンマというものを早期に打破して新しい事業を生みだす。ということを私たちは考える必要があるのです。

イノベーションを成功させるポイントとしては、まずモチベーションの高いスタッフだけを選りすぐって少数精鋭でプロジェクトを進めること、トップに立つ企業の責任者がイノベーション推進への強い意思を持っていること、そしてテクノロジーに投資をしていく意識を明確にすることなどが挙げられます。特にテクノロジー＝コストと考えている企業にはイノベーションは困難です。テクノロジーへの投資は同時に教育への投資であり、ここから人材が育つのだと考えるべきです。このようにイノベーションというのは、技術力の問題以前に、組織の考え方、企業文化、プロジェクトメンバーの選定方法が大きな影響を及ぼすことを私たちは知っておくべきでしょう。

LeanとDevOpsのマネジメント

それでは、イノベーションを進めていく上で、具体的にどうすればいいのでしょうか。ここでは「Lean」と「DevOps」という二つのマネジメント手法を紹介します。イノベーション改革とは非常にリスクが高いものという認識が一般的ですが、このリスクをできるだけ抑えてゼロに近づけていく手法として今アメリカで注目されているのがこのLeanとDevOpsです。

01 Leanとは

まずLeanですが、これはマサチューセッツ工科大学（MIT）がトヨタ生産方式をもとに開発した方法論で、かつて世界を席巻した日本の製造業の強みを分析し、その中心にある思考方法をあらゆる業種に適用できるよう一般化したものになります。

Leanという英語には「贅肉を取り除いた」という意味があるのですが、製造工程にあるさまざまな無駄を排除していくことで開発リスクを抑えることを目的としています。ベースとなるのはLean思考と呼ばれる五つの考え方で、さらにここからLean生産方式、Leanソフトウェア開発など七つの生産管理手法が定義されます（図3）。Lean思考に基づいてソフトウェア開発やサービス開発をする際の前提として、注意しなければならない二つの要件があります。一つは顧客の需要は必ずしも明確ではなく、時には顧客自身も気付いていない（潜在的な）需要があるということ。もう一つは、社会情勢がめまぐるしく変化する中で、人々の価値観や需要も絶え間なく変化していくということです。

では、どうしたらいいのでしょうか。Lean開発の哲学としては「継続的な開発とリリースの繰り返し」が提言されます。物を作ってリリースしたら開発は終了し、次の保守フェーズに入るという考え方はもう昔のものです。これからはできるだけ最低限のスモールなものを短期間・低コストで作り、それを顧客に使ってもらいながら、要望を吸い上げて修正したり、付け加えたりしてバージョンアップをしていきます。そうすることで本当に使いやすい、いいものができます。このような繰り返しにより、顧客が本当に求めているものが最低限のリスクで作られます。これがLean開発の哲学になります。

また、修正、開発、テストという一連の過程を繰り返す際、人が一所懸命チェックしたり、テストしたりしているという状況が続くと、そこにミスを犯すというリスクが生じます。従って、これからは品質管理やテストというものはできるだけ「人依存」を排除することが重要になります。「人」を排除するのでなく、「人依存」を排除するということです。もちろん自動化テストツールや品質管理のマネジメントをする人は不可欠なのですが、誰かが頑張ればいいというスタンスからいち早く抜け出すことを考えなければ、Lean開発は実現できません。

MITによってトヨタ生産方式を元に研究開発された方法論。
製造業向けだけではなく一般化された手法として様々な分野で研究が進んでいる。

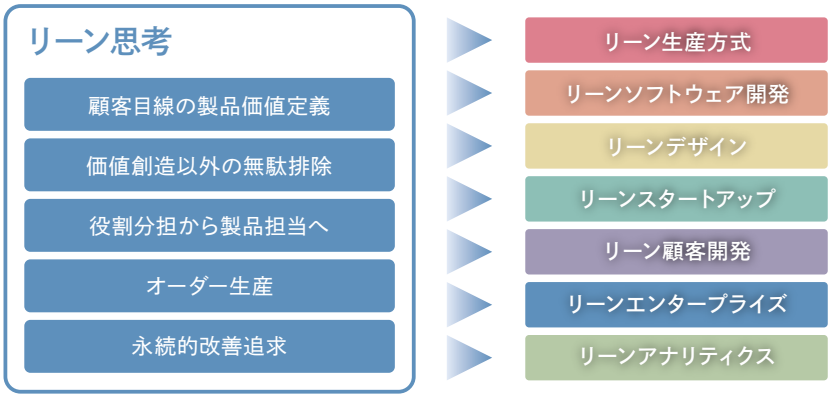


図3 リーン思考（Lean Thinking）

02 DevOpsとは

次にDevOpsは、急激な社会の変化に対して最低限のリスクで対応するという点においてはLeanと一致していますが、効率化のポイントが少し異なります。DevOpsは従来、開発（Development）、運用（Operations）、品質管理（QA）の連携が不十分なために多くの無駄が生じていた点に着目し、この三者を常に連携させながら繰り返していくことで効率的な開発を目指すという考え方になります（図4）。DevOpsでは、市場において常に優位性を築くことを目的としています。例えば、ある商品を作ってリリースするとそれを見たライバル企業が同じような物を作ります。しかし、常に優位性を築くためにはすぐにまたこれを超える商品を開発しなければなりません。従って、一つの開発が終わってリリースするとすぐに次の開発が始まります。ソフトウェアを迅速にデプロイし、これを繰り返し継続的に行うことが求められます。

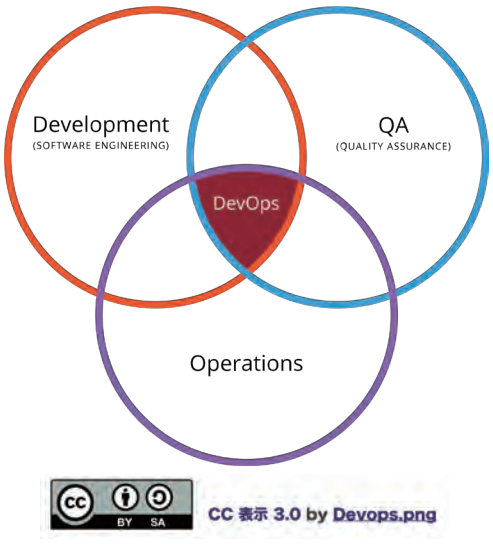


図4 DevOps

またDevOpsでは、基本的な思想としてPDCAサイクルの繰り返しによる段階的なリリースを前提としており、ベータ版（またはアルファ版）をリリースし、実際の利用者に利用してもらうことを通して「潜在的なニーズ」を引きだし、迅速に修正していくことで真に求められるものを開発していきます。これは日本の企業に多く見られる「完璧なものを作り上げなければならない」という考え方とは正反対です。「まずメインの機能だけを実現したら、それ以外の細かい部分は後でいい」――これがベータ版の考え方です。

DevOpsではそのパフォーマンスの計測方法にも特徴があります。従来のパフォーマンス計測がコーディング量や労働時間といった「量」で計測していたのに対し、DevOpsではデプロイ頻度やデプロイの失敗率といった「成果」で計測します（図5）。前述したようにDevOpsの考え方ではベータ版を出してそれに対する顧客の要望を取り入れていきますが、その要望にどれだけ応えられたか、どれだけ喜ばれたか、その「成果」で評価されることになります。逆に労働時間やテスト項目数が多くても、それは顧客価値につながるとは限りません。このような評価を繰り返すことにより、製品やサービスは顧客にとってより価値の高いものになるわけです。

DevOpsでは、成果や進捗度は「量」ではなく、
デプロイ数や、デプロイの失敗率など「成果」で測る

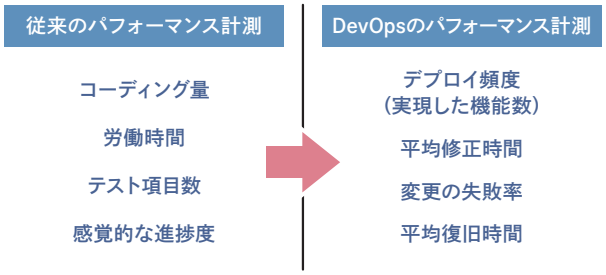


図5 DevOpsにおけるパフォーマンス計測

03 繰り返し開発における品質管理とテスト

次に品質管理とテストの話になりますが、正常系、異常系のテスト以外にもやらなければならないテストはいくつもあります（図6）。例えば、性能についての問題はリリース当初は基準を満たしていても、改修を繰り返した結果、ボトルネックとなり性能基準を満たすことができない、という結果を招きかねません。このような問題を防ぐためにはテストツールなどを用いてパフォーマンスの計測を継続する必要があります。このように、段階的なリリースを前提とした開発では、テストにかかる工数はますます増大していきます。そのため、可能な限りテストは自動化する必要があるのです。また、テストツール以外にもソースコードの修正とデプロイを頻繁に行う上で有効なのがバージョン管理ツールです。これは「いつ、誰が、何かを書いたか」をツールで確認でき、常に共同作業で効率よく修正していくために必要なツールで、有名なものとしては「git」などがあります。人への負担や依存が増えると必ず事故が起きると前述しましたが、これからの開発においてはこういった開発支援ツールを使いながら、組織の文化や開発手法を変えていく必要があるかと思います。

正常テスト	全ての機能をテストする。
異常テスト	考える事が可能な異常をテストする。
性能テスト	性能低下とボトルネック、リソース状況を確認する。
負荷テスト	性能限界とその際の動作について確認する。
障害テスト	ハードウェア障害の動作について確認する。
セキュリティテスト	外部からのサイバー攻撃について確認する。

図6 主なテストの種類

IoTとAIの品質管理

01 IoTと品質管理

最新のIoTやAIを導入した時に、品質管理はどのように変わるのでしょう。

ご存じの通り、IoTの世界では、電化製品や自動車などさまざまな物がインターネットとつながり、頻繁にデータをやりとりしながら連携していきます。この時、製品は「モニタリング」「制御」「最適化」「自律性」という四つの機能を持つと言われています。従来モーターなどの駆動部品とICだけで動いていたハードウェアには、センサーが必要になり、通信モジュールが必要になり、さらにOSが入りというようにコンピュータ化していきます（図7）。当然、品質管理は複雑になり、テスト対象もどんどん増えていきます。またハードウェア開発に

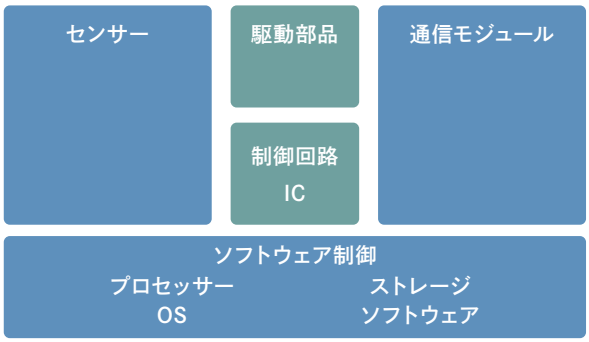


図7 ハードウェアのコンピュータ化

おいては、従来のようなすべての部品を自社で作る「すりあわせ開発（インテグラル型開発）」は通じなくなり、他社製の部品を組み合わせてそこに自社のオリジナルソフトウェアを載せる「組み合わせ開発（モジューラー型開発）」が主流になってきます。この時、製品のテストをどのように行うかが課題になります。まず他社から仕入れた部品に対して、どのように動くのかといった検証を行う必要が生じます。また、セキュリティも今後は非常に重要になってきます。仕入れた電子部品のセキュリティに問題はないか、通信は安全か…。こうしたことから、今後IoTテクノロジーに関わる電子部品はセキュリティの機能を持つようになってきます。例えば、STマイクロという会社が提供している「STM32」というチップに対し、同社ではセキュアブートやセキュアインストール/アップデート、データ保護機能といったIoT機器のサイバーセキュリティをサポートする「STM32 Trust」というシステムを提供しています。つまり、チップ自身がセキュリティ対応機能を持っているわけで、IoT製品の開発を行う企業が組み込む部品を選定するに当たっては、こうした品質管理のしやすさも重要な要素になりつつあるのです。

02 AI開発と品質管理

最後にAIの話をしたいと思います。

人工知能には、二つのフェーズがあります。一つが「学習」です。学習は、過去のすでに答えの出ているデータから問題を出し人工知能に予測させます。答え合わせの結果が正解であれば人工知能は考え方を維持し、不正解であれば新たな判断を付け加え新たな法則で再度予測します。もう一つのフェーズは、学習後に、まだ答えの出ていない新たなデータに対して予測するというというものです。図8は人工知能の学習回数に応じた正解率の推移を示したものですが、学習段階の正解率（青線）は学習するほど上がっていきます（縦軸の1=100%）。一方、未知のデータに対する評価（緑線）は、5〜6回までは順調に上がりますが、その先はなかなか上がっていきません。そこから先、人工知能の精度を上げていくには、学習の仕方を変えたり、アルゴリズムを変えたりといったチューニングが必要になります。人工知能というものは、プログラムを開発した段階ではまだ何もできていません。通常のプログラム開発においては、ソフトウェアは開発と同時に成果を出すことができます。それに対し、

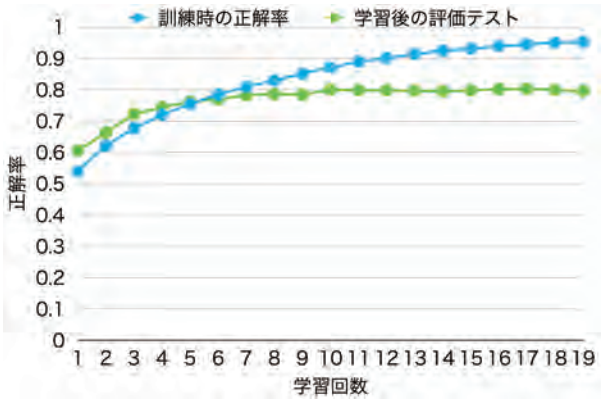


図8 人工知能の学習回数と正解率の計算

人工知能は、そこから学習をさせ、チューニングすることで賢くなっていきます。このように考えてくると、人工知能は非常に開発の時間もかかり、コストもかかります。成果が100%保証されるということもありません。従って人工知能に対しては、予算編成の段階からしっかりとした投資計画作成しておく必要があり、リリース後の運用フェーズにおいてチューニングと監視など継続的なサポートが必要になることを見越した品質管理体制が求められます。

おわりに

IoTや、AIを含む製品においては、テストする対象が増えたり、リリース後の管理工程が増えたりと、品質管理は今後ますます複雑化していきます。これからの開発では、本講演でお話したような生産管理手法やツール、テクノロジーなどを駆使しながら自動化とプロジェクトマネジメントの効率化を推し進めていかなくてはなりません。従来のような人への依存を排除し、人が「顧客目線での価値を追求する」といった、よりクリエイティブな仕事にシフトする必要があるでしょう。このようにして的確なイノベーションを進めることは、次世代に飛躍するための必須条件になると考えます。

本記事の内容に関するご質問、お問い合わせは、**株式会社ベリサーブ 広報・マーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。

パターンQA to AQによる Agile Quality(アジャイル品質)への変革と事例

鷲崎 弘宜氏
わしざき ひろのり

早稲田大学 グローバルソフトウェアエンジニアリング研究所 所長

早稲田大学 研究推進部 副部長・グローバルソフトウェアエンジニアリング研究所所長・教授。国立情報学研究所客員教授。株式会社システム情報 取締役(監査等委員)。株式会社エクスモーション 社外取締役。ビジネスと社会のためのソフトウェアエンジニアリングSE4BSほかの研究、実践、社会実装に従事。2014年からアジャイル品質パターンQA to AQの編纂参画、2020年からCodeZineにて「QA to AQ:アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革」翻訳連載中。

はじめに

2014年、Joseph Yoder氏やRebecca Wirfs-Brock氏が中心となり、アジャイル品質の考え方と推奨される活動を「QA to AQ」というパターン集として発表しました。QAはQuality Assurance、AQはAgile Qualityで、伝統的な品質保証からアジャイル品質の考え方・行動へ変革していこうというメッセージが込められています。私自身もこの活動に参画し、パターンの編纂や拡充のほか、日本語への翻訳展開も進めています※。本講演では、このQA to AQの概要に加え、実際の事例についても解説します。

※「QA to AQ：アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革」https://codezine.jp/article/corner/813

アジャイル品質パターン QA to AQ

I パターンとは何か？

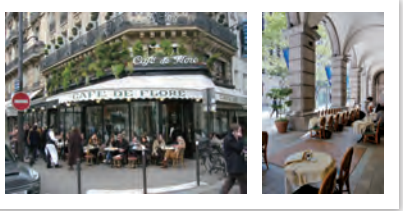
最初に、そもそも「パターン」とは何かをご説明しましょう。これはChristopher Alexanderという建築家が提唱した「パターン・ランゲージ」に端を発しています。

図1の①の写真、左側はパリを代表するカフェ、右は神戸のカフェです。世界中にはたくさんのカフェがあり、心地良いカフェというのは地域によって少しずつ異なります。しかし、よくよく見ると共通の部分があって、地域や場所に限定されない良さというものも存在します。それをドキュメント化し、別の場所で具体化して再利用することはできないでしょうか。この二つのカフェの良い共通点を整理したのが、②の部分です。

ここには「名前」「問題」「解決」として、地域や場所に依存しない普遍的な内容が記述されています。つまり、こういう問題はこうして解決すると良い、という考え方が一つの「パターン」としてまとめられています。

ここで注目すべきは、原典の表題がパターン・ランゲージ、つまり「言語」であるという点です。個々のパターンを語彙＝ボキャブラリーとみなし、その組み合わせによってコミュニケーションや新たな計画に役立てることができます。③がその一例で、ここで強調されているいくつかの単語、実はその一つ一つがパターンであり、これらの組み合せによって一つの都市計画がまとめられています。これが元々の建築理論におけるパターンが目指していたものです。

この考え方は、アジャイル開発やオブジェクト指向の設計パターンなど、ソフトウェア開発にも実際に応用されていて、さまざまな広がりを見せています。



②

名前：**路上カフェ**

問題：忙しい都会において、衆目の中で合法的に腰を下ろし、移りゆく世界をノンビリと眺められる場所がほしい。

解決：各近隣にカフェが開かれるよう促すこと。カフェにはいくつかの部屋を設け、にぎやかな歩行路に向けて解放し、座ってコーヒーなどの飲み物を取りながら、移りゆく世界を眺められるような親しみのある場所に仕立てること。

[Alexander77] C. Alexander他著，
平田翰那訳，“パタン・ランゲージ”，鹿島出版，1977

③

街の中心部には**活動の拠点**として**小さな広場**を設けよう。
その広場には、訪問者や近隣の人々が気軽に集い落ち着けるように、いくつかの**路上のカフェ**が集まってひらかれるように奨励しよう。それぞれのカフェは、オープンエアで楽しめるように**街路への開口**を持たせられるとよい・・・

https://en.wikipedia.org/wiki/Coffeehouse#/media/File:Caf%C3%A9_de_Flore.jpg
https://ja.wikipedia.org/wiki/%E3%82%AB%E3%83%95%E3%82%A7#/media/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:Former_Foreign_Settlement_of_Kobe_Hyogo_Japan_A55_03s3.jpg

図1 建築におけるパターン・ランゲージの例

Ⅱ なぜパターンなのか？

アジャイル品質の考え方や活動をパターンとして整理すべき理由は二つあります。一つ目は、品質とは特定の行動ではなく、日常的に繰り返される事柄の中で成立するのだということです。ちなみに、哲学者のAristotle（アリストテレス）が残した言葉に次のようなものがあります。

「Quality is not an act, it is a habit.」

彼は、質とは一度の行為ではなく習慣、日々の反復であると述べています。品質というものを捉える上で、このマインドは非常に大事です。

二つ目は、アジャイルの源流にはさまざまな説がありますが、私見では先ほどのパターン・ランゲージが一つ、もう一つはEdwards DemingらのPDCAによる品質管理です。実際に、初期のアジャイル開発であるEVO（EVOlutionary Development）は、反復的なPDCAサイクルをソフトウェア開発に応用したものです。これらを原点としていることから、アジャイル品質をパターン集として整理することは理にかなっていると言えます。

Ⅲ QA to AQの概要

これらを踏まえ、アジャイル開発における品質への取り組みとそのヒントを23のパターン集としてまとめたものが「QA to AQ」で、大きく4つのカテゴリに整理されています（図2）。

図2：中核パターンのカテゴリ

Ⅳ 中核パターン

「中核パターン」は、基盤となる考え方や進め方で、二つのパターンで構成されています（図3）。

中核パターン

他のパターンを用いるうえでの基礎

アジャイル品質プロセス	品質記述および理解する軽量な方法などを通じて品質保証をプロセスに組み入れる
障壁の解体	開発チームにおいて品質保証メンバと残りのメンバ間の障壁をなくし、皆が品質プロセスに関わる

図3 中核パターン

Ⅴ アジャイル品質プロセス

図4はアジャイル開発で一般的なScrumをベースに、アジャイル品質プロセスを説明したものです。Scrumのプロセスでは、初めにプロダクトバックログを作成しますが、その準備段階となる製品ビジョン/ロードマップの作成時点から、重要な品質シナリオを特定します。プラットフォームや主要機能だけでなく、この段階から品質を考慮しておくわけです。次のバックログ管理にも品質の項目を含めます。また、スプリントの中はもちろん、フィードバックの前にも品質テストを実行します。こうしたサイクルの繰り返しが、推奨される品質プロセスとなります。

Ⅵ 障壁の解体

アジャイル開発では、常に変化に適応しながら短いサイクルで開発と評価、改善が繰り返されていきます。そのため、品質の確保は専門家が担うという従来の考え方から脱却し、チーム全体で取り組むことが必要となります。ここで重要になるのが、障壁の解体です。早い段階からQA担当者をチームに加え、全てのスタッフと利害関係者がコミュニケーションを妨げる障壁を取り壊しながら品質に携わっていく。そして、結果として得られた品質には、チーム全体が評価されるような体制作りも必要です（図5）。

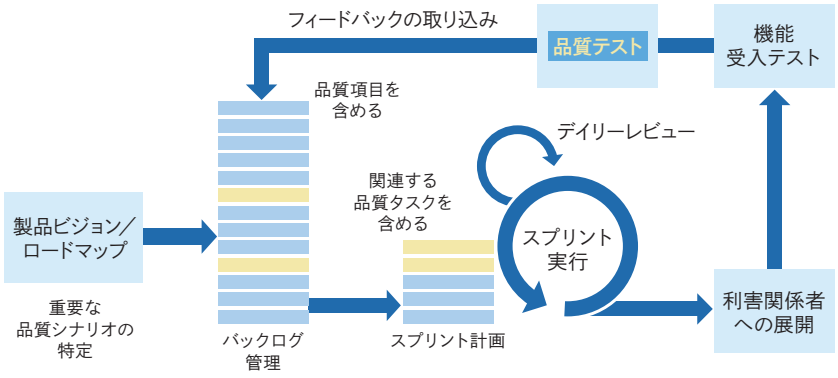
Ⅶ 品質のアジャイルなあり方

中核パターンで提示した考え方を、具体的な体制や手法としてまとめたのが「品質のアジャイルなあり方」です。ここでは、推奨する進め方やロールを9つのパターンで示しています（図6）。

アジャイル品質プロセス

重要なシステム品質の検査をアジャイルプロセスにどのように組み込み、QA担当者はプロセスのどこへ入ればよいでしょうか？

アジャイルプロセスの一環として、システムの品質を理解し、記述し、開発およびテストする方法を構築しましょう。



鷲崎, 長谷川, 濱井, 小林, 長田, 田村, 陳, QA to AQ：アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革, CodeZine, https://codezine.jp/article/corner/813

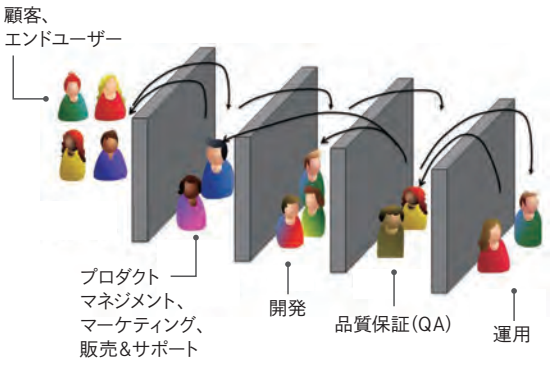
パターンの見方：図表最上部はパターンのタイトル、タイトルの下、左側には取り扱いたい問題、右側の太字部分には解決を表記しています。

図4 アジャイル品質プロセス

障壁の解体

アジャイルチームはどのようにしたら障壁を取り除き、品質に関してよりアジャイルになることができるでしょうか？

早い段階でQA担当者をチームに含めるなど、さまざまなアクションを通じてコミュニケーションを妨げる障壁や壁を取り壊しましょう。QA担当者をスプリントの一部にして、チームに取り込み、品質に対してチーム全体に報酬を与えましょう。



鷲崎, 長谷川, 濱井, 小林, 長田, 田村, 陳, QA to AQ：アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革, CodeZine, https://codezine.jp/article/corner/813

図5 障壁の解体

品質のアジャイルなあり方

アジャイルプロセスにおける品質保証のあり方や役割

QAを含むOneチーム	最初からQA担当者をチームに含める
品質スプリント	品質の測定や改善に焦点をあてたスプリントを設ける
プロダクト品質チャンピオン	チームが重要なシステム品質特性に集中できるよう支援する人をチームに含める
アジャイル品質スペシャリスト	特定の品質スキルがチームに不足しているなら、品質スペシャリストをさまざまなタイミングで投入
品質チェックリスト	システムで共通の、一貫して満たすべき品質特性の期待値を記したチェックリスト
品質作業の分散	QA担当者以外も品質タスクに巻き込み、負担を調整
品質エキスパートをシャドーイング	品質エキスパートがタスクを実行している間に、さまざまな人にフォローまたはシャドーイング
QAリーダーとペアリング	開発者と他のアジャイルチームメンバーを品質保証担当とペアを組ませ、品質関連のタスクを完了
できるだけ自動化	環境を整え、ツールを使用して、バリューを高める部分をできるだけ早く自動化

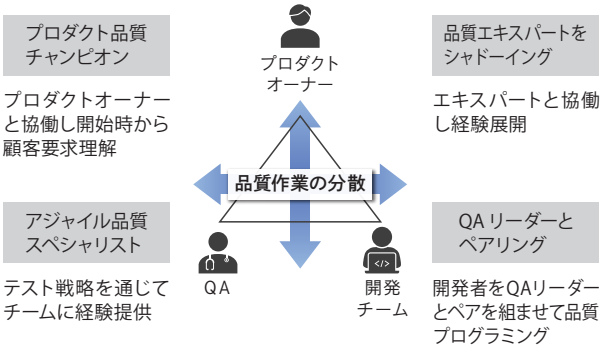
図6 品質のアジャイルなあり方

これらのパターンのうち、特に重要となるパターンについて具体的に説明しましょう。パターン「QAを含むOneチーム」は、中核パターン「障壁の解体」についてチーム構成の面で具体化したものといえます。図7のように、プロダクトオーナー、開発メンバーのみならず最初からQA担当を含んだOneチームでは、中央にあるようなトライアングルが構成されます。QA担当は「プロダクト品質チャンピオン」や「アジャイル品質スペシャリスト」として活動します。開発チームのメンバーは「品質エキスパートをシャドーイング」もしくは「QAリーダーとペアリング」などの活動を通じて、必要に応じて品質作業の分散を行います。これは、負担を調整するとともに、品質に関する知識や作業手順をチーム内で共有し、知見として浸透させていくことにもつながります。

QAを含むOneチーム

QA担当者が開発チームの一部ではない場合には、多くの問題が発生する可能性があります（「我々」対「彼ら」シンドロームの発生）。

アジャイル品質チームでは、最初からQA担当者をアジャイルチームの一部として含めましょう。



Joseph Yoder, “品質面でアジャイルであるために”,第2回enPiT-Proスマートエスイーセミナー: アジャイル品質保証と組織変革, 2018
鷲崎, 長谷川, 濱井, 小林, 長田, 田村, 陳, QA to AQ:アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革, CodeZine, https://codezine.jp/article/corner/813

図7 QAを含むOneチーム

品質への取り組みを高速、かつ変化に適応可能な状態で進めるために、プロダクトの価値に直結する部分は、できるだけ自動化するべきです（図8）。それも、可能な限り早く自動化することが大切です。ただし、何もかも自動化するのではなく、きちんと計画を立てて進める必要があります。いち早く自動化すべきなのは、ビルド・結合・テストなどの繰り返し行う作業です。また、ミスが起こりやすい作業や品質のメトリクス、コードの臭い[*1]の検出なども自動化することを

できるだけ自動化

重要な品質特性について、迅速なフィードバックを支援し、現在の状態を共有し可視化するために、ツールや環境をどのように確立できるのでしょうか？

環境を整え、ツールを使用して、バリューを高める部分をできるだけ早く自動化しましょう。開発の終盤まで自動化のタスクを先送りしないようにしましょう。

スクリプト化または自動化する対象

- 繰り返し行う作業（ビルド、結合、テスト等）
- 前の作業が終わったら動かすもの
- ミスを起こしやすい作業 / 退屈な作業
- 環境のセットアップ等
- 品質のメトリクス（パフォーマンス、信頼性、セキュリティ等）
- コードの臭い検出
- アーキテクチャ適合性
- 長時間かかるもの

※メモ / チェックリスト

- 発生する頻度は小さいが重要なもの
- 自動化が難しいもの
  → 一貫性が無いもの
  → 毎回、考えないといけないもの

Joseph Yoder, “品質面でアジャイルであるために”,第2回enPiT-Proスマートエスイーセミナー: アジャイル品質保証と組織変革, 2018
鷲崎, 長谷川, 濱井, 小林, 長田, 田村, 陳, QA to AQ:アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革, CodeZine, https://codezine.jp/article/corner/813

図8 できるだけ自動化

お勧めします。一方で、発生頻度が少ないもの、その都度考える必要のあるものは自動化には不向きと言えるでしょう。

また、ストリームとして流れていくアジャイル開発にも立ち止まる時間は必ず設けられていて、そこで少し振り返って次の改善につなげるといったことを繰り返していきます。そのタイミングで、システム上で一貫して満たすべき品質特性については、品質チェックリストを用いて確認を行います。ただし、リストがあまり過大なものになると、チェックそのものが目的になり形骸化する恐れがあるため、本当に重要なところに絞り、極力具体的な期待値を含めた形にすることをお勧めします。

図9はその実例で、あるWeb系の開発に使用したチェックリストです。パフォーマンスの項目には、「500ms未満」というように具体的な期待値が含まれています。さらに、誰もが同じようにチェックを行える状態になっています。

＊1：プログラミングにおいて、システム上で深刻な問題を引き起こす可能性のあるソースコードの兆候を比喩で表現したもの。

品質チェックリスト

システムが進化し続けても、システムの品質要件が保証され、見落とされないようにするにはどうすればよいでしょうか？

システムで共通の、一貫して満たすべき品質特性の期待値を記したチェックリストを作成し、先に進む前に振り返る時間で確認しましょう。

開発リリースチェックリストの例

コード品質

- ☑ 全てのコードがコード基準を順守していること
- ☑ 全てのコードにエラーや警告がないこと
- ☑ 適切なロギングがコード全体で実装されていること
- ☑ 考え得る全ての例外が適切に処理されていること
- ☑ メモリリークを起こすコードになっていないこと
- ☑ 全てのテスト用とデバッグ用のコードが削除されていること
- ☑ コードが適切に記述されていること
- ☑ 全ての不要コードが削除されていること
- ☑ 全ての単体テストがエラーにならず実行できること
- ☑ 全ての新規コードとコードの変更箇所について単体テストがされていること

アーキテクチャ

- ☑ 完全なドキュメントとアーキテクトの署名なしに、WebサービスのAPIが作成・変更されないこと
- ☑ 完全なドキュメントとアーキテクトの署名なしに、Webサービスのデータ構造が作成・変更されないこと
- ☑ 完全なドキュメントとアーキテクトの署名なしに、データベース構造が作成・変更されないこと

パフォーマンス

- ☑ 全てのウェブページは、実稼働時の負荷でも、500ms未満でレンダリングされること
- ☑ 全てのレポートは、実稼働時の負荷でも、500ms未満で生成されること
- ☑ 全ての検索SQLは、実稼働時のデータ量でも、500ms未満で結果を返すこと

Joseph Yoder, “品質面でアジャイルであるために”,第2回enPiT-Proスマートエスイーセミナー: アジャイル品質保証と組織変革, 2018
鷲崎, 長谷川, 濱井, 小林, 長田, 田村, 陳, QA to AQ:アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革, CodeZine, https://codezine.jp/article/corner/813

図9 品質チェックリスト

Ⅷ 品質の特定

どの品質が重要なのかを見極め、それをきちんと書き下して共有するのが「品質の特定」で、8つのパターンで構成されています（図10）。

重要な品質の発見

顧客の代表なども含む利害関係者とともに、何が重要なのかをきちんと定義します。

品質シナリオ

重要な品質について、特に性能、負荷、セキュリティなどの非機能要件を誰もが理解できる平易な表現で記述しておきます。図11は、品質シナリオの例です。開発プロセスの早い段階で大まかな基準の品質シナリオを作成しておくことが必要となります。

品質の特定

重要な品質の発見	重要な利害関係者とチーム会議して最も重要な品質特性についてブレインストーミング
品質シナリオ	早い段階で、手軽な方法で、重要な非機能要件を扱う大まかな品質シナリオを作成
品質ストーリー	個別の品質特性のストーリーを作成し、それらをバックログに追加
測定可能なシステム品質	品質を測定し、必要な精度と正確さで記述する適切な方法を定義
品質の折り込み	品質受け入れ基準をユーザーストーリーに添付
着陸ゾーン	品質特性の受入れ可能な着陸ゾーン（最低、目標、優良）を定義して使用
着陸ゾーンの再調整	測定結果やベンチマークに基づき再調整
着陸ゾーンの合意	目標値に関する情報に基づいた合意

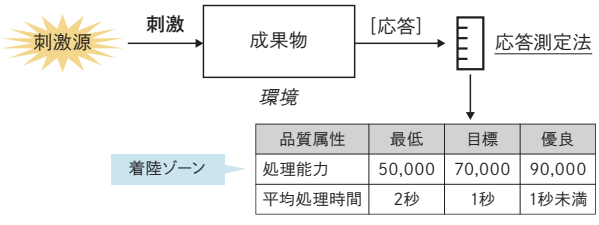
図10 品質の特定

品質シナリオ

パフォーマンスに関する記述

・端末の利用者は、**ピーク時**に**報告をWeb経由で要求**し、1秒以内に［報告を受理］する。
・待ち時間減少のため**新サーバ**を2人週以内で［追加する］。

保守・拡張性に関する記述



着陸ゾーン

品質属性	最低	目標	優良
処理能力	50,000	70,000	90,000
平均処理時間	2秒	1秒	1秒未満

Joseph Yoder, “品質面でアジャイルであるために”,第2回enPiT-Proスマートエスイーセミナー: アジャイル品質保証と組織変革, 2018
鷲崎, 長谷川, 濱井, 小林, 長田, 田村, 陳, QA to AQ:アジャイル品質パターンによる、伝統的な品質保証からアジャイル品質への変革, CodeZine, https://codezine.jp/article/corner/813

図11 性能、負荷、セキュリティなど重要な非機能要件については、早い段階で大まかな基準の品質シナリオを作成しておくことが必要

品質ストーリー

やや抽象度の高い記述である品質シナリオをより具体的にしたもので、品質にフォーカスしたユーザーストーリーと言っていいでしょう。これをプロダクトバックログなどに追加して、スプリントで回していきます。

測定可能なシステム品質

ISOやJISにもうたわれていますが、品質は「測定可能」でなければなりません。セキュリティが高い、パフォーマンスが高い、といった曖昧な表現ではなく、それがどの程度なのかを明確にする必要があります。

品質の折り込み

品質の受け入れ基準を具体的に設定します。先出の品質シナリオや品質ストーリーに記述する、あるいは機能面に着目したユーザーストーリーに、この機能の実現に当たり、品質の基準はこうなります、という形で記載してもよいでしょう。

着陸ゾーン

品質の受け入れ基準には、「着陸ゾーン」という考え方がお勧めです。変化に適応できることが特徴のアジャイル開発では、受け入れ基準をピンポイントで設定するのは困難な場合もあります。そこで、最低値・目標値・優良値といったように、着地点に一定程度、幅を持たせるという考え方です。これは処理能力など、さまざまな品質属性について記述することが可能です（図11）。

着陸ゾーンの再調整

着陸ゾーンは、ニーズやベンチマークの変化に応じて適宜再調整を行います。

着陸ゾーンの合意

当然ですが、着陸ゾーンは利害関係者と事前に合意しておくことが必要です。

Ⅸ 品質の可視化

重要な品質特性は、開発プロセスのあらゆる段階で「見える化」し、メンバー全員が常時気を配って取り組むことが大切です。これを4つに整理したのが「品質の可視化」です（図12）。

品質の可視化

重要な品質特性を可視化しチームメンバーに気付かせる

品質 ロードマップ	製品の機能を開発し、ロードマップを進化させながら、品質特性とそれをサポートするアーキテクチャにいつ対処するか計画
品質バックログ	品質項目（品質シナリオ、品質ストーリー、ユーザストーリーへの品質の折り込み）を作成してバックログに追加
システム品質 ダッシュボード	品質特性を継続的に測定し可視化・監視する仕組みを用意
システム品質 アンドン	誰もが質問する必要なく、仕事中や休憩中に見ることができる品質特性のステータスのディスプレイを用意

図12 品質の可視化

Ⅹ 品質ロードマップ

製品ロードマップに品質特性を組み込み、いつどのように対処するかを明記します。図13はその例で、負荷分散・セキュリティ・安定性・応答性などを、それぞれの要件や重要性に応じて計画に組み入れています。これらは状況の変化に合わせて随時見直しを行う必要があります。

Ⅺ 品質バックログ

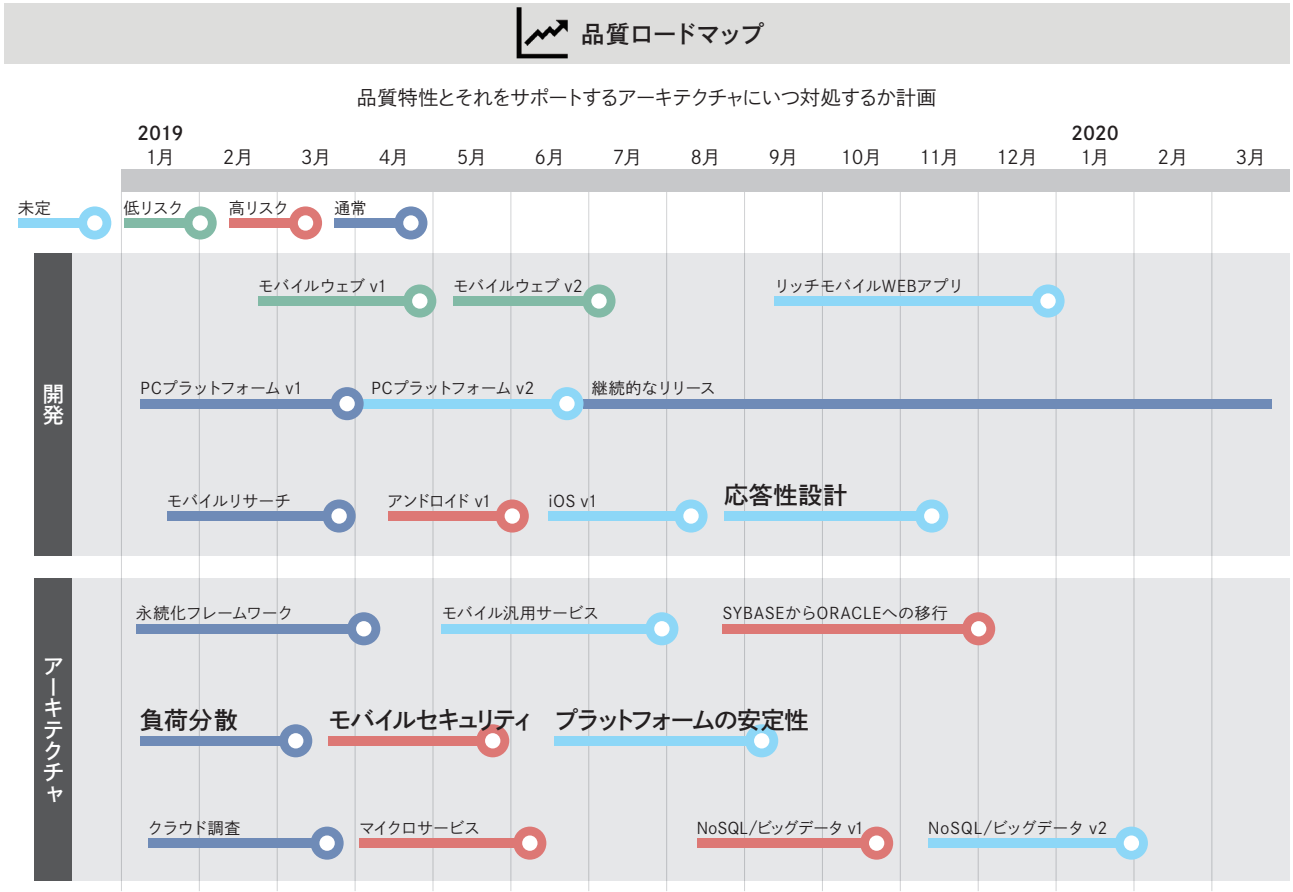
品質項目をプロダクトバックログに追加します。機能だけではなく、品質に関する事柄もタスク管理に組み入れて、優先順位を付けながらスプリントで回していきます。

Ⅻ システム品質ダッシュボード

品質の様子をグラフや表などで可視化して、推移やアラート情報などをチームで常時確認・共有できるようにします。

XIII システム品質アンドン

ダッシュボードよりも簡易的に、例えばランプの点滅のようなレベルで誰もが直感的にステータス確認ができるディスプレイがあれば、なお良いでしょう。



Joseph Yoder, “品質面でアジャイルであるために”, 第2回enPiT-Proスマートエスイーセミナー: アジャイル品質保証と組織変革, 2018

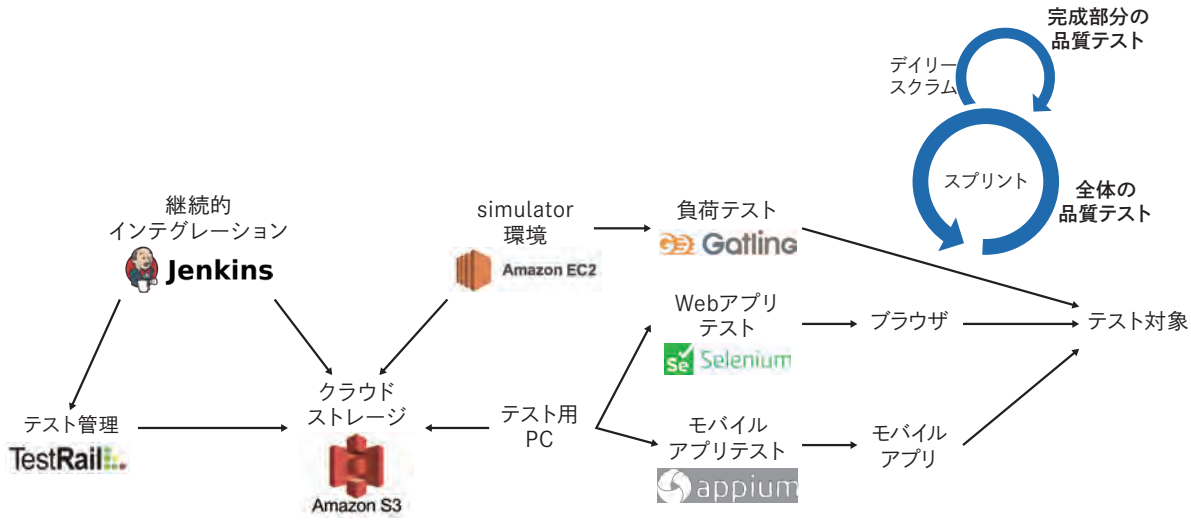
図13 品質ロードマップ（一部抜粋）

QA to AQの事例

QA to AQの取り組みには、実際に日本国内でも活用事例がいくつか出てきています。

Ⅰ「できるだけ自動化」

繰り返し実行するテストの自動化と継続的な自動化を実現するために、継続的インテグレーションからテストの管理、テストの実行までを自動化します。また、全体テストをスプリントに、部分的なテストを日々の活動にそれぞれ組み込んでいます(図14)。

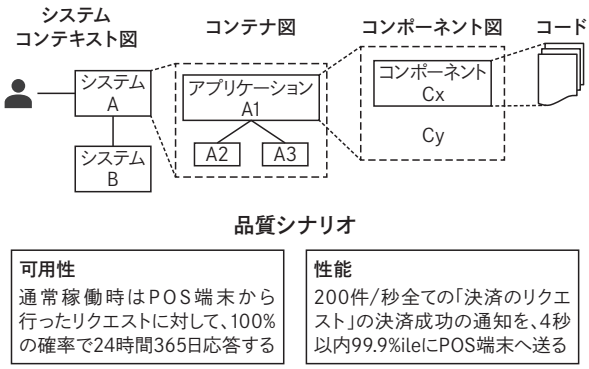


長田武徳,アジャイル品質パターンの利用 事例, スマートエスイーセミナー: アジャイル開発と品質, 2020年8月20日
https://smartse.connpass.com/event/178629/ https://en.wikipedia.org/wiki/Jenkins_(software) https://logodix.com/logos/
https://www.gurock.com/testrail/ https://aws.amazon.com/jp/s3/ https://gatling.io/ https://www.selenium.dev/ http://appium.io/

図14「できるだけ自動化」の利用事例

Ⅱ「品質シナリオ」

アーキテクチャの開発にC4モデル[*2]を用い、そこで求められる品質要求に具体的な数値を含めた品質シナリオを併記することで定義しています(図15)。



品質シナリオ

可用性	性能
通常稼働時はPOS端末から行ったリクエストに対して、100%の確率で24時間365日応答する	200件/秒全ての「決済のリクエスト」の決済成功の通知を、4秒以内99.9%ileにPOS端末へ送る

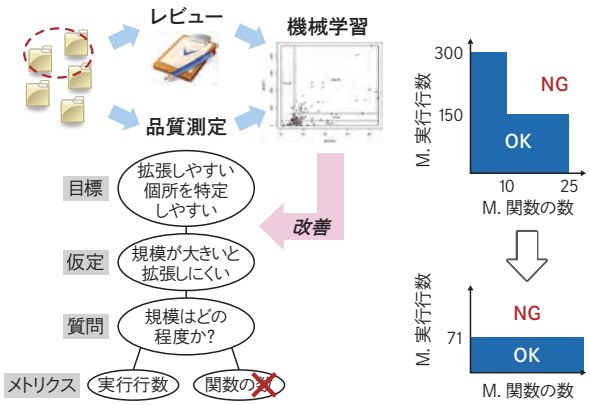
長田武徳,アジャイル品質パターンの利用 事例,
スマートエスイーセミナー: アジャイル開発と品質, 2020年8月20日
https://smartse.connpass.com/event/178629/

図15「品質シナリオ」の利用事例

＊2:Ionut Balosin氏が提唱しているソフトウェアアーキテクチャの表記法。コンテキスト(context)、コンテナ(containers)、コンポーネント(components)、コード(code)というレベルの異なる4つの図を用い、必要箇所に絞って表現することにより、アーキテクチャを簡潔に表現することができる。

Ⅲ「着陸ゾーンの再調整」

品質基準の見直しには何らかの根拠が必要になりますが、必要なデータがある程度取得できる場合は、機械学習などを交えて調整するのがお勧めで、図16はその一例です。保守性に関する品質要求を捉える際に、従来は実行行数や関数の数を指標としていました。しかし、別のレビューデータと突合した機械学習の結果、関数の数は無視して、実行行数にもっと厳しい基準を適用すべきという見直しが図られました。

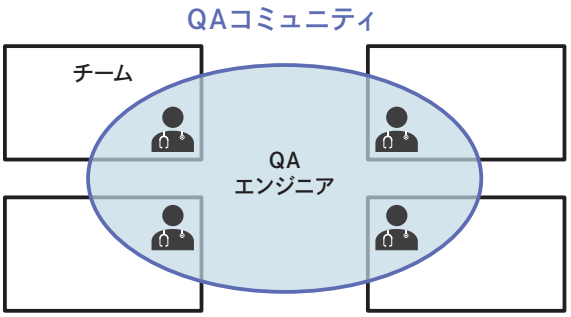


N. Tsuda, H. Washizaki, Y. Fukazawa, Y. Yasuda, S. Sugimura, "Machine Learning to Evaluate Evolvability Defects: Code Metrics Thresholds for a Given Context," 18th IEEE International Conference on Software Quality, Reliability & Security (QRS 2018)

図16「着陸ゾーンの再調整」の利用事例

Ⅳ 組織とプロセス変革への取り組み

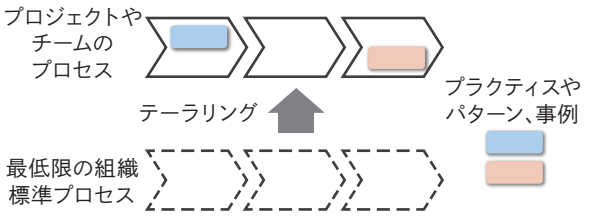
アジャイル品質の取り組みを推進する上で最も重要なのは、やはりOneチームの構築と、品質作業の分散だと思います。これについても、参考になる事例をご紹介しましょう。一つ目はQAコミュニティで、これはQA担当者をチームに送り込むのではなく、逆にチームメンバーの中からQAエンジニアを選出し、その人たちとQA部門がチームを超えて交流する環境を作るというものです。これによって、QAの知見を広く共有し、Oneチームへの布石にしようという試みです（図17）。



長田武徳,アジャイル品質パターンの利用 事例, スマートエスイーセミナー: アジャイル開発と品質, 2020年8月20日

図17 QAコミュニティを活用しQA部門がチームを越えて交流するための環境作り

もう一つはテイラーアップと呼ばれるもので、組織標準のプロセスを強制せず、最低限の標準を守った上でチームに適したプロセスを独自に構築するという手法です。そこにアジャイル品質のパターンやプラクティスを適宜組み入れて、各プロジェクトやチームで活用する仕組みを作っています（図18）。



小林浩, アジャイル品質パターンによる伝統的な品質保証(Quality Assurance)からアジャイル品質(Agile Quality)への変革（適用事例）
https://smartse.connpass.com/event/178629/

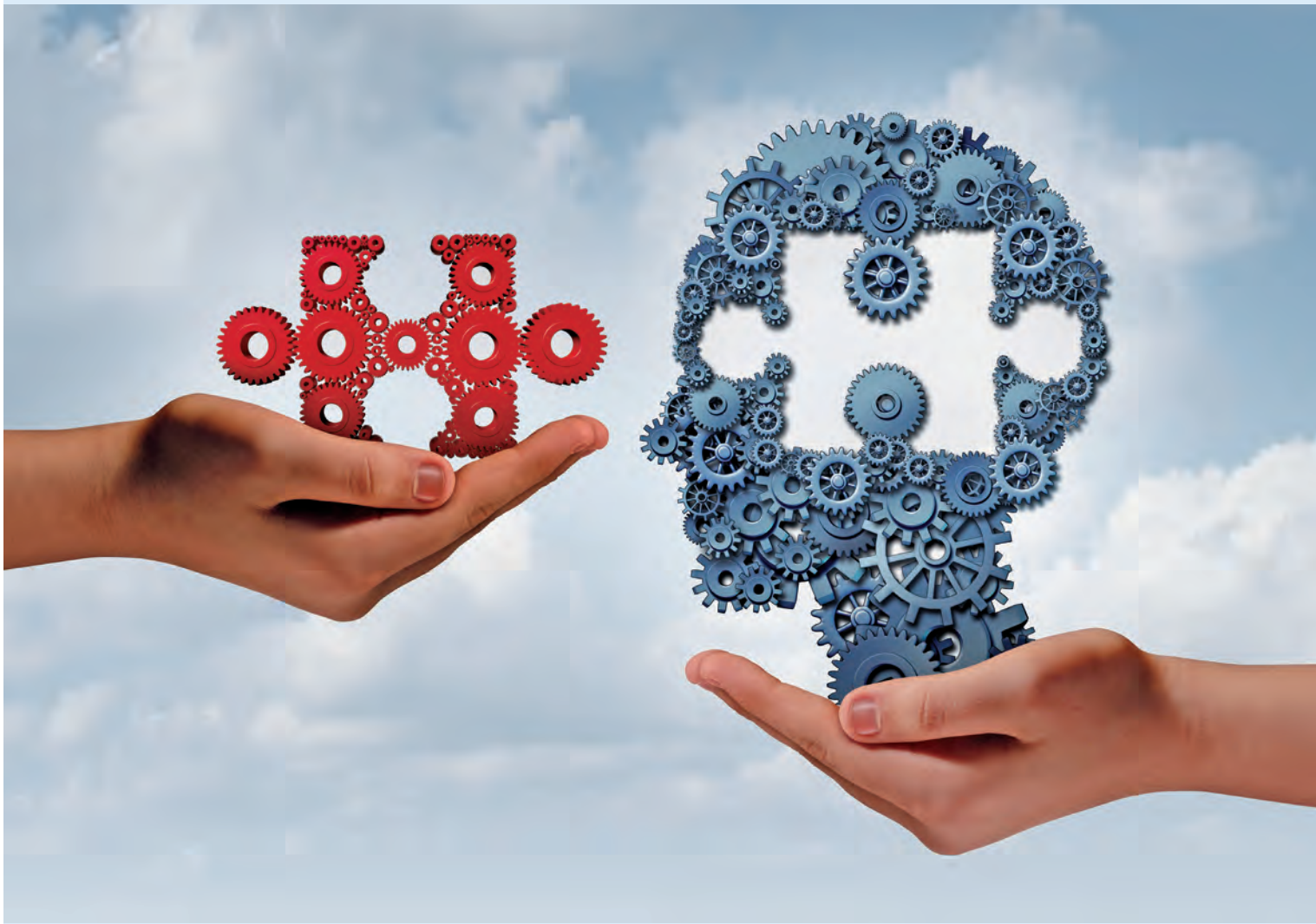
図18 テイラーアップを用いた仕組み作り

おわりに

このQA to AQの取り組みは、現在も編纂・拡充を続けています。ぜひ皆さんからも良い事例や知見をご提供いただき、私たちと一緒に皆さんにもこの活動の推進にご協力を願えればと考えています。本講演が、そのきっかけとなれば幸いです。

本記事の内容に関するご質問、お問い合わせは、**株式会社ベリサーブ 広報・マーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。

AIとソフトウェア品質技術



松木 晋祐
まつき しんすけ

株式会社ベリサーブ 研究企画開発部 部長

独立系ソフトウェアベンダにて、テストオペレータから品質部門統括まで、ソフトウェアテストと品質保証にまつわるさまざまなロールを経験後、ベリサーブへ参画。現在はソフトウェア品質にまつわる製品の研究企画開発部門を担当。
NPOソフトウェアテスト技術者振興協会 会員 /JaSST東京 実行委員 / テスト自動化研究会 ファウンダー
東京電機大学 CySec 講師 / W3C CSSWG コントリビューター / 認定スクラムマスター

はじめに

本講演では、QA4AIとAI4QAという二つの考え方・アプローチを軸に、ここ数年AI技術がソフトウェア品質技術に及ぼした影響を具体的な活動・製品とともに振り返り、さらにAIが共存する今後の品質技術の方向性についてお話しさせていただきたいと思います。

I 概念の整理

AIと品質保証技術の関わりについては、これまで「AI by QA」「AI with QA」などさまざまな呼称がありましたが、近年では「QA4AI」と「AI4QA」という言葉・概念で統一されてきたように思います。二つの概念の違いを整理すると、「QA4AI」はAIが搭載された製品に対して品質技術をどのように適用していくかという「考え方」なのに対し、「AI4QA」はAI技術を品質技術にどのように活用していくかという「アプローチ」です。昨今、AI4QAを具体化したものとして、AI技術の搭載によってより高い精度で、より効率の良い処理を実現した「より良いツール」がテスト自動化ツールを中心に数多く登場しています。

品質保証技術を俯瞰してみると、ソフトウェア品質のマネジメントからテストの実行まで、技術の領域は多岐に及びますが、AIが得意とする分野は現在まだ一部にとどまっています。AIを活用した新しいツールがどの技術をカバーするものであるかを意識して初めて、その効果を最大限に活かせるようになります。「AI技術を品質保証技術にどう活かすか」について考えることが、テスト技術者の役割であると言えます。

QA4AI

QA4AIに関して、AIプロダクトに対する品質保証の考え方がまとめられている代表的な二つのガイドラインをご紹介します。

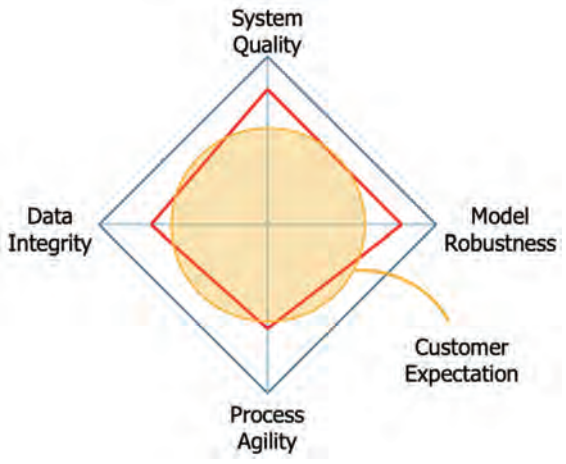
I AIプロダクト品質保証ガイドライン

一つは日本の有識者団体であるAIプロダクト品質保証コンソーシアムによる「AIプロダクト品質保証ガイドライン[*1]」です（2019年初版発行、2020年8月改定）。この中では、AIプロダクトに対して普遍的に適用しうる「5つの軸」を策定し、製品ドメインごとに注力すべき軸のバランスを変えるというアプローチ（図1）を中心に、説明可能AI（XAIなどと呼ばれます）技術への言及、実践的な品質技術のカタログ、各ドメインにおける具体的な成果物などが掲載されています。

中でも品質技術のカタログには、いくつかの軸に対応した性能指標やデータの評価、頑健性、公平性について、具体的にどのように品質保証を行うべきかという指針が示されています。特に、教師あり学習モデルに対する性能指標は非常に具体的に記述されていて参考になります。

AIプロダクト品質保証ガイドラインの5つの軸

- Data Integrity ―― そもそもデータの質は適切か？
- Model Robustness ―― モデルは頑強であるか？
- System Quality ―― AIが組み込まれる製品全体の品質
- Process Agility ―― 試行錯誤は迅速であるか？
- Customer Expectation ―― 顧客の期待は適切か？



出典：http://www.qa4ai.jp/QA4AI.JaSST-Tokyo.20190327.pdf

図1 AIプロダクト品質保証ガイドラインの5つの軸と顧客の期待とのバランスを見る評価の一例

＊1：http://www.qa4ai.jp/QA4AI.Guideline.202008.pdf（2021年2月15日参照）

Ⅱ 機械学習品質マネジメントガイドライン

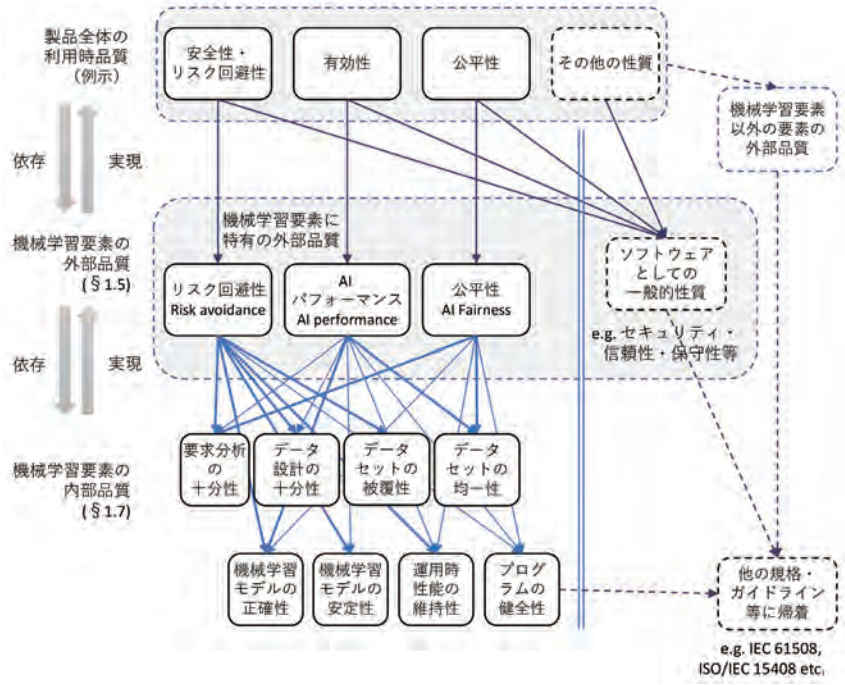
もう一つは、国立研究開発法人産業技術総合研究所（AIST）と国立情報学研究所（NII）が民間企業や大学と共同でまとめた「機械学習品質マネジメントガイドライン[*2]」（2020年6月初版発行）です。こちらにはAIを採用するシステムのライフサイクル全体における品質マネジメントについて、必要な取り組みや検査項目が掲載されています。ISO25010-10利用時品質モデルや、機械学習要素に対する外部品質、内部品質の測定法を定義しており、これまでISO25000シリーズを運用してきた企業などにとっては違和感のない構成で、受け入れられやすいのではないかと思います（図2）。

AI4QA

ここからは、AI技術を品質保証技術に応用して、その精度や効率を向上する「AI4QA」のさまざまなアプローチについて見ていきます。

Ⅰ AI自動テストツール

まず、AI4QA分野において著しい進化が見られるものの一つに自動テストツールがあります。進化の段階を三つの世代に分けると（図3）、第一・第二世代はその過渡期でもあり、テスト対象の仕様変更に伴う、テスト実行スクリプトの保守などの面において課題も少なくありませんでした。第三世代になりAI自動テストツールが登場すると、テスト対象のコード変更に対する影響箇所の特定や、画像解析を利用したテストコードの自動生成などが可能になり、これまでの課題はスマートに解決されるようになりました。



機械学習品質マネジメントガイドライン 第1版 図2:製品品質実現の全体構造出典:https://www.cpsec.aist.go.jp/achievements/aiqm/

図2 機械学習品質マネジメントガイドラインでの製品品質実現の全体構造

＊2：https://www.cpsec.aist.go.jp/achievements/aiqm/

一方で、AI自動テストツールによって次々にテスト作成が行われるようになった結果、テストケースが数万件という単位で膨張する事態が発生します。テストの実行時間が長大化し、テストレポートの仕分けなど人間がチェックすべき要件も数千件単位で増加します。ここから考えると、次世代のAI自動テストツールは「チェックをいかに効率化するか」あるいは「実行するテストの数をいかに減らすか」のいずれかのアプローチを歩むことになると思われます。

Ⅱ 自動テストの実行時間を短縮：Launchable

「実行するテストケースを絞る」というアプローチにトライしているのが「Launchable」（Jenkinsの開発者・川口耕介氏が開発したクラウドサービス）です。このサービスはテスト対象製品のソースコードの変更に基づいて起こるテストケースの失敗（Failed）を予測し、失敗する可能性の高いテストケースを提案します。ソースコードの変更と関連の深いテストだけを実行することにより自動テストの時間を短縮することができます。

プログラマがソースコードをコミットする際、リグレッションテスト（変更したコードが他の機能に害を及ぼしていないことを確認するテスト）を長時間かけて全ケース実行するよりも、ある程度高い精度のテストケースのみに絞り込んで実行し、結果が出るまでの時間を短縮することを重視する。万が一、不具合が起こった場合も、対応の確認まで短時間で済むという安心感を開発者に与え、スピードを落とさず開発できるようになる。このようなコンセプトがLaunchableの面白いところだと思います。

Ⅲ ニューラルファジング

次にニューラルファジングというアプローチを紹介します。こちらは「Microsoft Security Risk Detection（MSRD）」に含まれる製品です。ファジングは、ソフトウェアに対して予測不可能な入力データ（fuzz）を与えて不具合の発生を確認する手法です。かねてより機械学習と非常に相性のいい技術だと思っていたところ、やはりツールとして具体化する企業が現れました。

ニューラルファジングは、グレーボックスファザー（テスト対象のプログラム構造は意識しないが、過去の挙動をもとに以降の入力データをガイドする機能を有するファジングツール）に機械学習のアプローチの一種であるディープニューラルネットワーク*3を活用することでソフトウェアの脆弱性を発見しようとする試みです。「American Fuzzy Lop」（AFL）というオープンソースのグレーボックスファザーで実験を行ったところ、ニューラルAFLは通常のAFLに比べてより多くのクラッシュが確認でき、高い検証成果を得ました。ニューラルファジングは現在非推奨となっていますが、2020年末にはオープンソース化して再登場する予定になっています。

世代	世代を構成する主な技術	代表的な製品	課題
第一世代	キャプチャリプレイ 画像認識	AutoTester WinRunner	コードの自動生成による保守の煩雑さ、実行時の不安定性
第二世代	UIオブジェクト（locator） データ・キーワード駆動	Ranorex eggplant	やや軽減されたものの保守コストは依然として高い。自動化に開発技術力が必要になったことによる人材難
第三世代	AutoDetect（locatorの半自動取得） Self-healing	mabl Autify Magic Pod	旧2世代の課題を解決。大量のテストレポートの仕分け、テストの実行時間の長大化

図3 AI自動テストツールの三世代

＊3：GitHub社が運営するクラウドの開発プラットフォーム。ソースコード管理やコードレビュー、プロジェクトの管理など、チームでのソフトウェア開発に便利な機能が備わっている。 https://github.co.jp/

Ⅳ コードレビューの自動化

コードレビューを自動化する試みとしては、Amazon.com社の「Amazon CodeGuru」という製品があります。これはAmazon.com社内の数十万のプロジェクトやGitHubに載っている1万件以上のオープンソースのプロジェクトのコードレビューをもとに、教師あり学習モデルを利用して、コードの問題があると思われる部分、パフォーマンス低下につながりそうな部分を指摘してくれるサービスです。コードの品質に関してCodeGuru-Reviewerが指摘してくれるポイントを図4にまとめています。

GitHubはオープンソースのプロジェクトであれば、プログラマがコミットしたコードに対して、プロジェクトに参加している人がどういったコメントを付けているか誰でも見ることができます。似たようなコードには似たような指摘が入るため、これを学習すれば成果が上がると考えられるわけですが、実際にアイデアを具現化したところが素晴らしいと思います。

Ⅴ ベリサーブの取り組み

ベリサーブの取り組みについてもいくつか紹介します。弊社では技術者のテスト観点についての知見をまとめた「テスト観点のデータベース」を有しています。現在、このデータベースを利用し、テスト設計の際、テスト対象の仕様文章を入力すると、仕様に適用できそうなテスト観点を推薦してくれるサービスを開発し、社内提供しています。先述した「Amazon CodeGuru」に近いもので、熟練のテスト技術者の描くテスト観点をフォローできるメリットがあります。その他、システムやネットワークから吐き出される膨大なログから不具合につながる箇所を検出する技術なども研究を進めています。

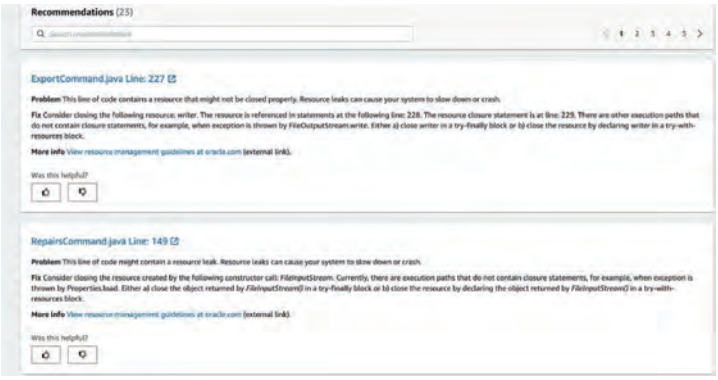
今後のAIと品質技術

ここからは今後の「with AI」と呼ばれるAIが当たり前にある時代の中で、私たち品質技術者がどのように活動していくべきかについて考えていきたいと思います。

Ⅰ DevOpsへの組織的な投資で生じる「差」

今後、開発→品質保証→開発というプロセスはさらに加速していきます。開発と品質保証のターンアラウンドタイムもますます短縮化され、同時に、開発と品質保証の境界はより曖昧になっていきます。先ほどからご紹介しているような品質保証技術を活用できる組織（エリートパフォーマー）と活用できない組織（ローパフォーマー）では、製品の新バージョンを届けるまでのリードタイムに大きく差が生じ、その差は実に100倍と言われます。

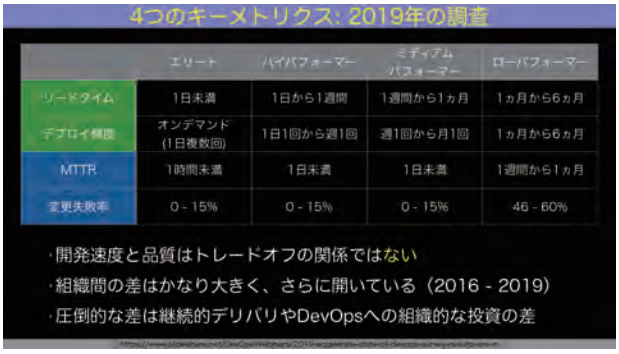
AWSのベストプラクティス	AWS API（ポーリング、ページ区切り、など）の使用方法を修正します。
Javaベストプラクティス	一般的なJava言語とライブラリ機能の使用方法を修正します。
同時実行	機能上の障害を起こしている同期不良、もしくは、パフォーマンスを低下させている過剰な同期などを検出します。
デッドロック	同時実行されるスレッド間の割り当てをチェックします。
リソースリーク	リソースの処理方法（データベース接続の解放など）を修正します。
機密情報のリーク	個人識別情報（ログインしているクレジットカードの詳細など）の漏洩を検出します。
一般的なコード上のバグ	Lambda関数読み出し時にクライアントを作成していないなどの、発見しづらい問題を検出します。
クローンコード	統合することでコードの保守性を高められる可能性のある、重複コードを特定します。
入力検証	信頼されないソースから送られる、不適当な形態の、もしくは悪意のあるデータをチェックします。

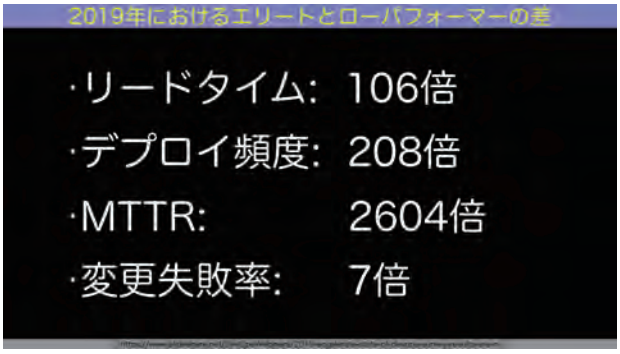


出典：https://aws.amazon.com/jp/codeguru/features/

図4 Reviewerがコードの品質に関する問題を指摘する様子

今後のAIと品質技術





出典:JaSST'20 Niigata 基調講演 和田卓人氏 https://twitter.com/t_wada/status/1310496998341005312
引用元:https://www.slideshare.net/DevOpsWebinars/2019 accelerate state of devops survey results are in

図5 エリートパフォーマーとローパフォーマーのリードタイム比較

図5はJaSST '20 Niigataにおける和田卓人氏（タワーズ・クエスト株式会社）の基調講演資料の引用ですが、リードタイムを含め4つのキーメトリクスが紹介されています。しかし、これらの差は個人の資質だけの問題ではなく、DevOpsへの組織的な投資の差による部分が大きいとされています。エリートパフォーマーがパフォーマンスを向上させるために用いる技術の多くは、従来の品質技術をベースにAIを活用した新しい技術でフォローできる領域です。“安全な実装”または“その実装が安全であること”をツールでフォローできる体制をいかに整えられるかという点に着目すべきだと思います。

Ⅱ DevOpsをつなぐQAの役割

それでは、私たちは今後何をするべきでしょう。折しもDevOpsの中心に入るエンジンとして、AI4QAという心強い武器が登場しました。DevOpsはDevelopmentとOperationからなりますが、本来この間には「QA」があるべきです（DevQAOps）。DevOpsによっていくら早く走れるようになっても事故を起こしては意味がありません。DevからOpsに渡す時に、価値を確定するQAが入るシーンはこれから増えていきます。その時、「今回のリリースに対して何が十分なのか」をAIの力を借りて判断し、ジャスト・イン・タイムで必要なテストを届ける。今後はこういったインテリジェントなQAが必要になってくると思われます。

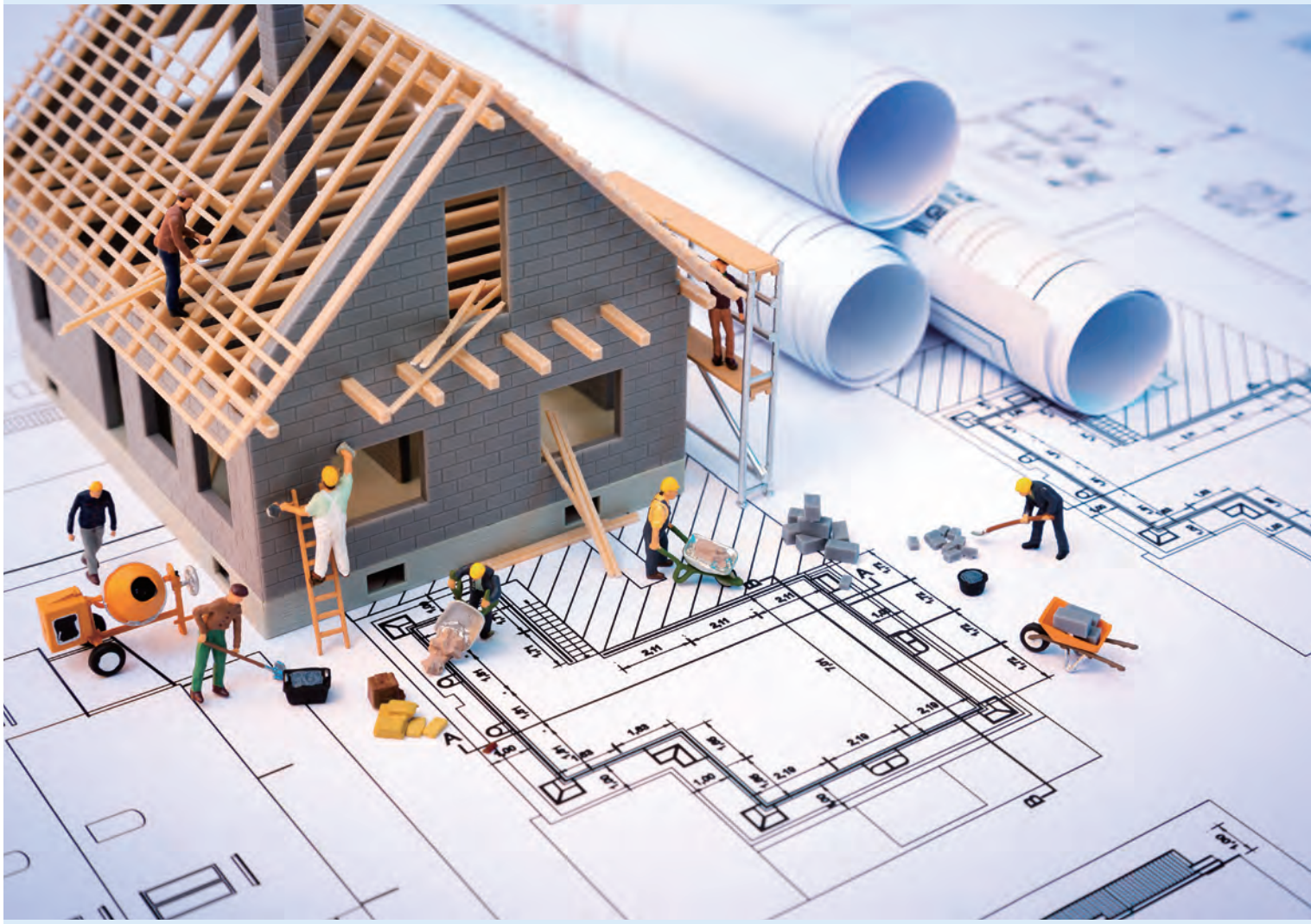
おわりに

開発のスピードが求められる現代は、ソフトウェアテストをはじめ品質技術をAI4QAのような形でソフトウェア化すること、またそれを支えるための指針を示す技術者の育成が急務となっています。同時にそれはベリサーブの使命でもあります。私たちはこれまで培ってきた技術やノウハウを最大限に活かしながら、これからもお客様とともにソフトウェア品質技術の未来を創っていきたいと思っています。

本記事の内容に関するご質問、お問い合わせは、**株式会社ベリサーブ 広報・マーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。

DX時代のITサービスに要求される「安心・安全な品質」とは?

～より安心・安全なITシステムの構築を支援する品質エンジニアとしてのアプローチ～



桑野 修
くわの おさむ

株式会社ベリサーブ ソリューション事業部 事業部長

1995年にベリサーブの前身となる株式会社CSKの検証事業部門に入社。2001年に株式会社ベリサーブ設立と共にベリサーブへ転籍。その後は、主に業務システムやWebサービス分野の顧客に対する検証事業部門を担当し、その中で、性能・セキュリティを中心とした非機能要件に関する検証サービスの立上げを行う。現在は、前述の非機能要求系（特にIoT分野のセキュリティ対策）の他、ソースコード解析、OSS検査ツールなど、品質向上に寄与する各種テストツールを活用した検証サービスを提供。

はじめに

私は1995年の入社以来、性能やセキュリティの分野における品質向上に数多く携わってきました。本講演では、こうした分野における弊社の活動をご紹介し、皆様の課題解決に役立つヒントをご提供できればと考えています。

DX時代のITサービスに求められる品質とは?

この数年、「DX」という言葉をたびたび耳にするようになりました。AI・5G・IoT・クラウド・ビッグデータなどの要素技術を組み合わせることで、さまざまなサービスがデジタル化され、新たな価値がもたらされています。

利便性の高いサービスが次々と登場する一方、これらに深刻なインシデントや障害が起きるケースも散見されています。特にセキュリティ関連のトラブルは増加傾向にあり、個人情報の漏えいは年間数千件のレベルで発生(図1)、大手決済サービスにおいても、ユーザーに金銭的な被害をもたらす事案が起きています。

また、アクセスの集中によるシステム障害も頻発しています。記憶に新しいところでは、コロナ禍に対する公的助成金制度で受付システムの停止や障害によるデータ損失などが起こっています。このように、誰もが知るような巨大ベンダや国・地方自治体が提供するシステムでも、こうしたトラブルは後を絶ちません。

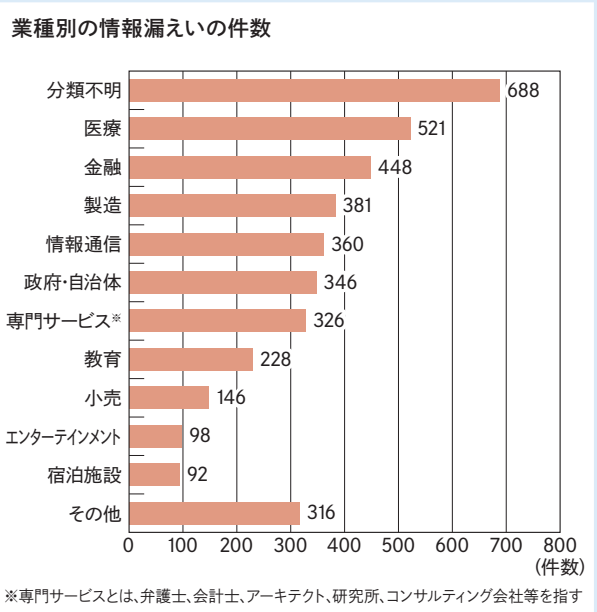


図1 業種別情報漏えいの件数
(独立行政法人情報処理推進機構「情報セキュリティ白書2020」
https://www.ipa.go.jp/security/publications/hakusyo/2020.html
(参照2021-02-25))

I 非機能要件とは?

ソフトウェアやサービスには、「機能要件」と「非機能要件」が存在します(図2)。機能要件は、何を実現するのかを文字通り機能として記述したものです。一方、非機能要件は機能に依存しない特性で、時に暗黙的にしか定義されない要件を指します。その代表が性能やセキュリティで、先に挙げたようなトラブルは、まさにこの非機能要件に関わるものです。

機能要件	非機能要件
要件・システム要件といった機能面	機能面**以外**全般
システム設計	システムアーキテクチャ設計
「要件に対するシステムのふるまい」として記述され、**ブラックボックス・機能モデル**として説明される	**機能に依存しない**全体的な**特性**について「システムが要件を満たさなければならない」の形で記述される

■ **非機能要件の具体例**

- ◆『非機能要件要求仕様定義ガイドライン』 :JUAS発行
- ◆『非機能要求グレード』 :IPA/SEC発行

図2 機能要件と非機能要件

	非機能要件要求仕様定義ガイドライン	非機能要求グレード
発行元	JUAS 日本情報システムユーザー協会	非機能要求グレード検討会(NTTデータ、富士通、NEC、日立、三菱電機インフォメーションシステムズ、沖電気) →2010年IPAへ譲渡(普及目的)
発行時期	2008年発行	2010年IPAで発行 2018年改訂(主にセキュリティ要件:背景は以下) ・新たなセキュリティ脅威の増大、仮想化技術の拡大
概要	10種類の非機能要求を定義　測定可能な指標:230項目	6つの大項目　35の中項目
要求・要件項目	1.機能性　4.効率性　7.障害抑制性　10.技術要件 2.信頼性　5.保守性　8.効果性 3.使用性　6.移植性　9.運用性	1.可用性　4.移行性 2.性能・拡張性　5.セキュリティ 3.運用・保守性　6.環境・エコロジー

図3 非機能要件について明示的に合意するためのフレーム

【2001年 ISO/IEC 9126】

品質特性	品質副特性
機能性	合目的性、正確性、相互運用性、機密性、標準適合性
信頼性	成熟性、障害許容性、回復性、標準適合性
使用性	理解性、習得性、運用性、魅力性、標準適合性
効率性	時間効率性、資源効率性、標準適合性
保守性	解析性、変更性、安定性、試験性、標準適合性
移植性	環境適応性、設置性、共存性、置換性、標準適合性

【2011年 ISO/IEC25010 SQuaRE】

品質特性	品質副特性
機能適合性	機能完全性、機能正確性、機能適切性
性能効率性	**時間効率性、資源効率性、容量満足性**
互換性	共存性、相互運用性
使用性	適切度認識性、習得性、運用操作性、ユーザーエラー防止性、UI快美性、アクセシビリティ
信頼性	**成熟性、可用性、障害許容性、回復性**
セキュリティ	**機密性、インテグリティ、否認防止性、責任追跡性、真正性**
保守性	モジュール性、再利用性、解析性、修正性、試験性
移植性	適応性、設置性、置換性

図4 品質特性と品質副特性の変化

II 非機能要件の標準化の取り組み

暗黙的である非機能要件についても、可能な限り顧客とベンダ間での明示的な合意が必要です。これを促進するため、標準化の試みが国内外で実施されています(図3・4)。

しかし、こうした努力にも関わらず、非機能要件に起因するトラブルは今も続いています。非機能要件の品質確保は、それほど難易度が高いものだということがご理解いただけるかと思います。

この非機能要件に対し、テストや検証をサービスとして提供する弊社がどのように取り組んできたかをご紹介します。

非機能要件とテストプロセス

図5は、90年代後半に弊社が当時の親会社の検証部門として事業を開始した頃の活動を表したものです。一般的なV字型モデルの中で行われるテストとは別に、専門のチームによる独立したテストプロセスを定義し、開発のスタート段階からテストを考える体制を作りました。

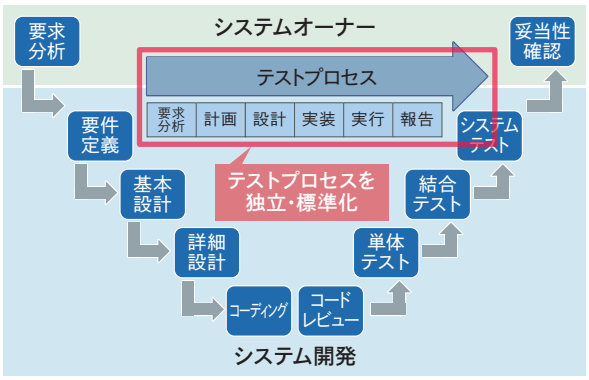


図5 システム開発におけるテストプロセスの独立

1.ボリュームテスト	テスト対象に対して、大容量、多数量、多種類のデータ処理を要求した場合でも、正しく動作することを確認するテスト
2.ストレージテスト	テスト対象の利用するリソースが不足しそうな状態、または不足している状況においても、正しく動作することを確認するテスト
3.高頻度テスト	テスト対象に対して、単位時間内に処理能力の限界を超えるような多数の処理を要求した場合でも、正しく動作することを確認するテスト
4.累積稼動テスト	テスト対象に繰り返し長時間処理を要求した場合でも、正しく動作することを確認するテスト
5.構成テスト	テスト対象と同一環境内のシステムを構成する要素の削除・置き換え・属性変更を行った場合、または別の要素を同一環境内に追加した場合に、テスト対象が正しく動作することを確認するテスト
6.両立性テスト	テスト対象をある環境に対して追加、削除、置き換え、属性の変更を行った場合に、その環境内のシステムを構成する要素が正しく動作することを確認するテスト
7.データ互換性テスト	テスト対象に対して利用できるフォーマットのデータを処理させた場合に正しく動作することを確認するテスト
8.障害対応テスト	障害発生時の影響が局所的なものに抑えられているか、致命的なものにならないか、または障害発生時の影響をどこまで回復できるかなど、発生する可能性のある障害に対応できるかを確認するテスト
9.ユーザビリティテスト	テスト対象のユーザーインターフェイスについて、視認性と操作性の良否を評価するためのテスト
10.セキュリティテスト	**テスト対象への不当な行為を阻止するための対策が適切になされているかを確認するテスト**
11.ドキュメントテスト	ユーザに提供されるドキュメント(マニュアル)の技術的な正確性を評価するテスト

図6 11個のシステムテストカテゴリ

I テストアプローチの進化と変遷

初期の活動で課題となったのは、不具合の発見にチームのメンバー間でバラツキがあることでした。これを解消するにはプロセスの上位にある「設計」フェーズの強化が必要と考え、テスト手順の標準化に取り組みました。具体的には、過去に見つけた不具合をグルーピングして分析し、有効であったアプローチに名前を付けていき、最終的に11個の「システムテストカテゴリ」に集約しました(図6)。信頼性(耐障害性)やユーザビリティ、セキュリティなど、すでにこの時点で非機能要件的な問題を取り扱っていたことがお分かりいただけると思います。

この標準化の効果もあり、徐々に品質改善の実績を積み重ねていく中で、2000年代前半には開発の初期段階である要求分析の時点から積極的に関与する案件が増えていきました。お客様は、この業務やサービスをIT化したい、といった機能要件の定義はしていても、実施するテストや品質の在り方には明確な答えを持っていない場合も多かったためです。

図7は、こうした場合に私たちが採った代表的なアプローチの一つです。品質特性にはISO/IECの国際標準を、それに対するテスト項目には先にご紹介した11個のシステムカテゴリを使ってマトリクス化しています。

これにより、お客様や開発チームとの間で、品質とテストに関する共通認識を早期に醸成することが可能になりました。また、設計の品質向上に寄与したほか、私たちのテスト活動自体にも良い影響をもたらしたと記憶しています。

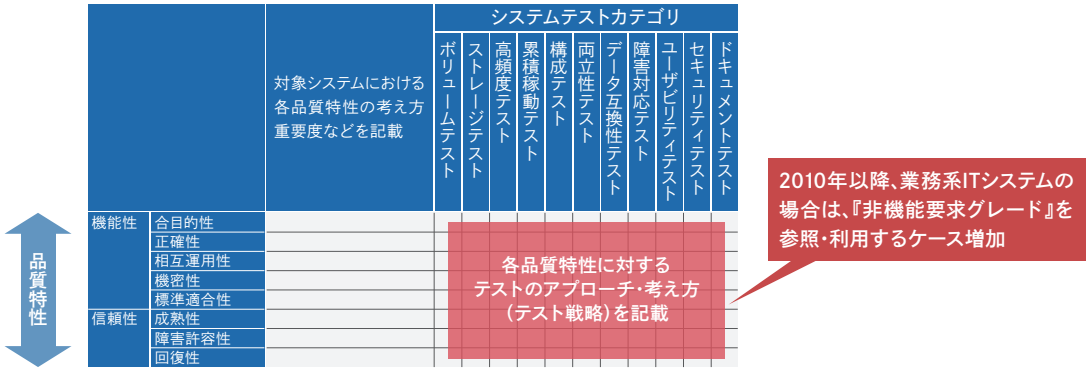


図7 テストカテゴリと品質特性を利用したテストのアプローチ

Ⅱ 非機能要件テストのサービス化・水平分業

ここからは、現在私たちが実施している非機能要件テストの概要についてご説明します。端的に言って、専門家である私たちにとっても非機能要件のテストは簡単ではありません。これには、大きく二つの要因があります。

まず、かつてのシステム開発はフルスクラッチが主流で、開発者が内部構造をすべて把握している場合がほとんどでした。しかし、最近は短納期化などの影響で、さまざまなモジュールやサブシステム、マイクロサービスを利用することが増えています。その結果、個々のブロックの構造は開発者にも理解が難しく、仮に性能劣化が起きた場合でも原因がどこにあるか不明なケースが出てきています。

もう一つは繰り返しになりますが、性能やセキュリティに対する定義付けが、お客様自身にも明確でないことです。性能を例に取ると、「このシステムは5万人が使える」とお客様が言ったとしても、それ以上の定義が何もない場合があります。アクセス集中への対処としては、ではそのシステムを10秒間で500人が同時に利用した時にはどうなのか、といったことを要件として定義しておくべきなのですが、これが欠けているプロジェクトが多いため、まずはテストをする私たちが「あるべき要求」を整理するところから入る必要があります。

セキュリティの場合も状況は同じです。攻撃者のアプローチは千差万別で、システムをどう守るかという問いに対して体系的な回答を持つ方は多くありません。そのため、こうした要件についても、テストをする側の私たちが定義しなければならないケースがあります。

こうした状況から、非機能要件のテストについては水平分業による専門チームでの対処を基本としています。具体的な手法としては、特殊なツールを使用して、同時に大量のアクセスを生成したり、システムへの疑似攻撃を試みたりします。これにはプログラムが実施する通信の内容を深く理解しておく必要があり、一般的なテストエンジニアが持っていないような、内部のロジックに踏み込んだ知見が求められます。

さらに、テストの結果として応答時間の劣化やシステムの停止が起きた場合に、その原因や対策について設計側が簡単には見極められないケースも多いため、私たちが問題の解析や改善に対するアプローチを示唆することも必要です。

このような専門的なサービスを必要なタイミングで提供することで、プロジェクト全体のQCD最適化に貢献することを目指しています（図8）。

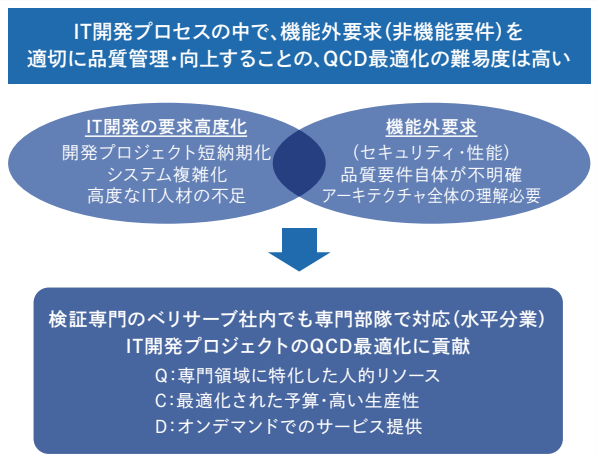


図8 非機能要件のテストにおけるアプローチ

Ⅲ 非機能要件に対する支援領域の拡大

非機能要件に対する私たちへの要望は、徐々に開発の上流へとシフトしていく傾向にあります。品質の問題が下流工程で発覚すると手戻りが大きくなるため、上流からそのリスクの低減を図ることが求められています。具体的には、非機能要件の定義と設計への反映をレビューしたり、プログラムがセキュリティを担保した構造になっているかをソースコードレベルで解析したりと、テストの前段階で品質を上げるアプローチを行っています。

さらに、ここ2～3年は、非機能要件を開発のライフサイクル全体でコントロールする支援も行っています。特にセキュリティ分野では、システム開発がスタートする前の要求分析の段階から「どんなリスクがあるのか」という脅威分析を行うことがさまざまなガイドラインで推奨されるようになったり、製品のリリース後、システムが使用しているコンポーネントに脆弱性が発覚するというニュースが増えたりしたことから、システム開発プロセスの前後の工程である、要求分析や脆弱性管理を含む運用支援への依頼も増えてきています。

また、大規模なプロジェクトで複数のチームが、それぞれが担当するサブシステムの開発・テストを行っている場合、サブシステムごとに性能試験を実施する必要が生じることがあります。そのようなケースでは、プロジェクト全体のマネジメントを支援するPMOチームなどに参画して、テスト全体のコーディネートを支援するケースも出てきています(図9)。

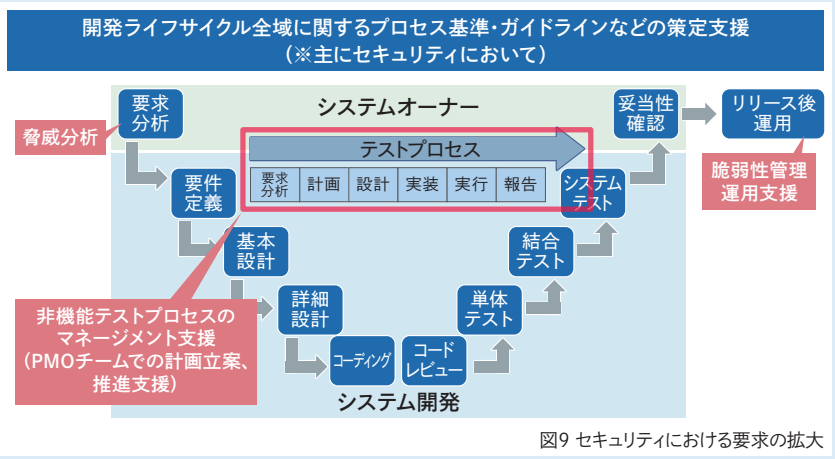


図9 セキュリティにおける要求の拡大

図10 セキュリティに関する業界標準

Ⅳ セキュリティ要求の増大と必須化

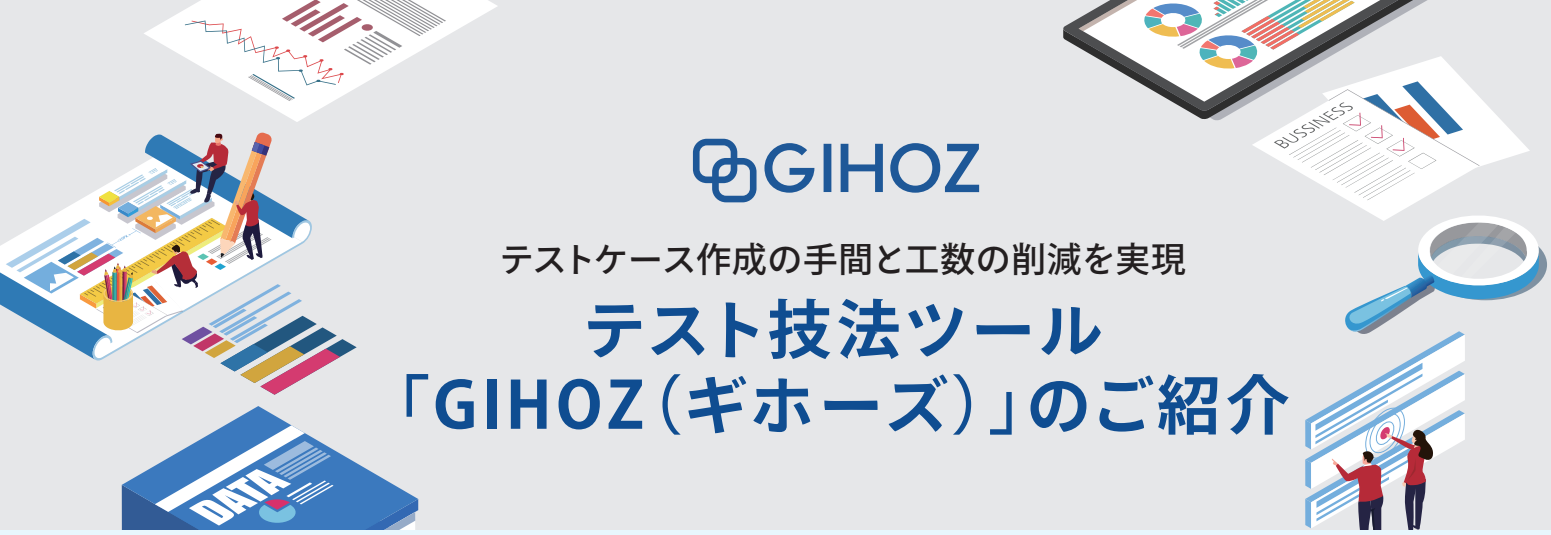
非機能要件の中でも、セキュリティに関する要求はここ数年で急速に高まり、業界ごとに設けられた国際標準や規制への準拠が必須となっています(図10)。自動車関連業界などでは、ISOのセキュリティ規格に準拠していない製品は、2023年以降には販売できなくなる方向で法制化が進んでおり、弊社でもその対応への支援を行う案件が増えています。

セキュリティが他の非機能要件と大きく異なるのは、意図的に何らかの障害を起こそうとする攻撃者への対応が必要である点です。このため、もう一段高い要求分析や品質のマネジメントが必要になってきています。

おわりに

非機能要件への対処は、単にテストをするだけでなく、システム開発全域にわたるコントロールが必要です。それには開発に関わるすべてのステークホルダー、特に上位層がリスクマネジメントの意識をしっかり持つことが非常に重要だと感じています。本講演でご紹介した課題と、私たちのアプローチにご興味をお持ちいただき、意見交換の場をいただければ幸いです。

本記事の内容に関するご質問、お問い合わせは、**株式会社ベリサーブ 広報・マーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。



テストケース作成の手間と工数の削減を実現

テスト技法ツール「GIHOZ（ギホーズ）」のご紹介

はじめに

ベリサーブは2020年11月、ブラウザ上で各種のテスト技法が利用できるクラウド型ツール「GIHOZ（ギホーズ）」のオープンβ版をリリースしました。サイトにアクセスするだけで誰でも手軽に、テストの目的にかなったテスト技法が利用できる、他に類を見ないユニークなツールです。

テスト設計の手間と工数の削減を目的としたこのツールは、どのようなきっかけで生まれたのでしょうか。今後の展開を含め、プロダクトオーナーの谷崎浩一（研究企画開発部 サービス開発課）に聞きました。

テスト技法を手間暇かけずに利用できないか

きっかけは、クラウドで各種テスト技法を一括で提供するツールがないねという会話からでした。テスト技法ごとの個別ツールは存在しますが、それらは各提供元からさまざまな形で公開されており、データのフォーマットはまちまちです。利用時にそれぞれインストールして設定しなければならないため、常駐するお客様先の変更などでPCの環境が新しくなるたびに同様の手間がかかります。「一カ所にアクセスするだけで各種のテスト技法が使えるようになるとテスト設計がはるかに効率的になる」との思いから、企画・開発がスタートしました。

必要な道具がそろう「大工さんの道具箱」に

イメージしたのは「大工さんの道具箱」です。“金づち”や“かんな”といった大工仕事に欠かせない道具は、あちこちに置いておくより一カ所にまとめた方が必要なタイミングですぐに取り出せ、作業がはかどります。テスト設計版の「道具箱」も、テスト担当者や開発担当者にとっての道具（＝テスト技法）がいつでも使える状態でそろっている、そんなツールを目指しました。そこから名前も、技法の複数形でギホーズ＝GIHOZと決まりました。

現在サポートしているテスト技法は、現場でよく使われる次の4種類です。

- **ペアワイズテスト**
- **状態遷移テスト**
- **デシジョンテーブルテスト**
- **境界値分析**

作成したデータはクラウド上に保存されるため、いつでもどこからでもアクセスできます。テストケースはCSV形式でダウンロードできるため、汎用の表計算ソフトでも利用可能です。

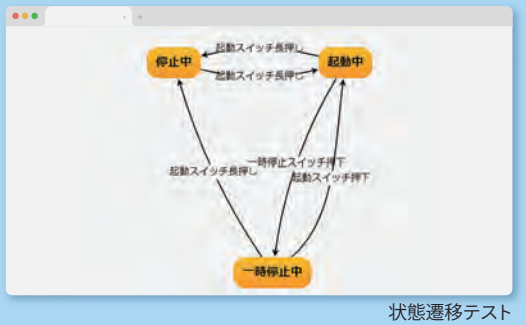
テスト技法のデータ活用に向けて

GIHOZのこれからの開発計画ですが、まず他のユーザーをチームに招待する機能を追加する予定です。これによってクラウド上でのテストケース共有が可能となり、レビューなどチームでのテスト活動が進めやすくなります。

ペアワイズテスト

デシジョンテーブルテスト

境界値分析



状態遷移テスト

また、新たな「道具」としてクラシフィケーションツリー法や直交表などの追加も検討しています。

さらに、当初から視野に入れていたテスト技法のデータ活用に向けても開発を進めています。汎用の表計算ソフトでテストケースを作成すると、見た目は整っていてもフォーマットはまちまちでデータとして扱いづらく、活用するにはひと手間もふた手間も要します。

これに対してGIHOZでは、JSON*1形式でデータを入出力するためのAPI*2の公開を検討しています。GIHOZで作成したデータを他のテストツールから呼び出してテストの自動実行に使ったり、テストケースの入力条件を外部から与えてGIHOZで組み合わせ生成をしたりといったことが容易になります。「テスト技法のデータが標準化され、さまざまなツールがデータ連携できるようになって効率的な開発やテストができる」。GIHOZによってそんな世界が実現することを目指しています。

おわりに

本稿では、GIHOZ開発のきっかけや狙いを通してその特長をご紹介しました。β版はアカウント登録のみでどなたでも無償でご利用いただけますので、実務担当者はもちろん、テスト技法を学習中の方にも役立てていただけます。多くの方にご利用いただき、お気付きの点やご意見・ご要望をお寄せいただければ幸いです。

本ツールのご利用はこちらから
（アカウント登録のみでご利用になれます）

GIHOZ公式ページ

＊1：JavaScript Object Notationの略。データ交換を行うための記述形式の一種で、プログラム言語を問わず利用できる。テキスト形式で可読性が高く、単純で軽量なことなどが特徴。

＊2：Application Programming Interfaceの略。あるプログラムの機能やその管理するデータを、外部のプログラムでも利用できるようにするための接続仕様（手順や形式を定めたもの）。

お問い合わせ先　株式会社ベリサーブ　GIHOZサポートチーム e-mail：support@gihoz.com

品質を創造する企業

Veriserve Navigation 2021年3月号（ベリサーブナビゲーション）
編集・発行 | 株式会社ベリサーブ　東京都千代田区神田三崎町3-1-16 神保町北東急ビル9F
お問い合わせ | 広報・マーケティング部　TEL：050-3640-8194　MAIL：verinavi@veriserve.co.jp