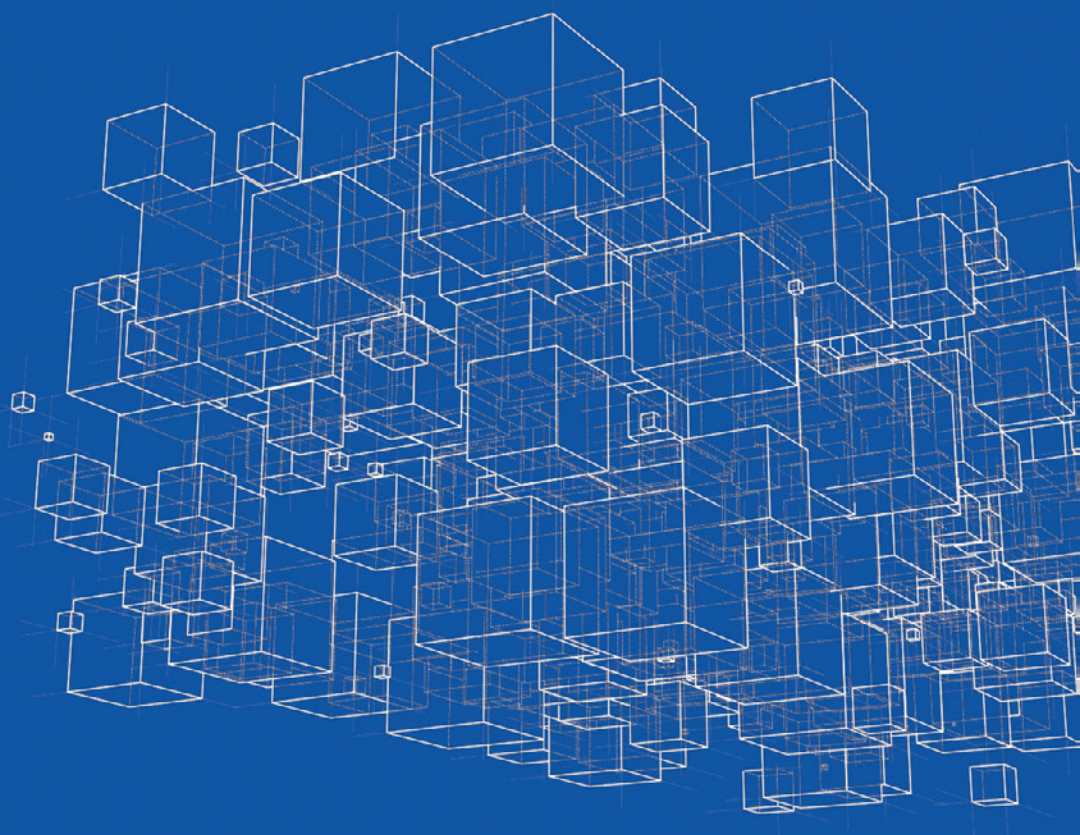
ベリサーブナビゲーション

Veriserve Navigation

テスト自動化の最前線



2019 December Vol.18

テスト自動化の最前線

テスト自動化が当たり前となってきた現在。

自動化を適用する製品・サービス領域、
テストするフェーズ、マネジメント方法は、
日々、進化している。

進化を続けるテスト自動化の最前線を特集する。

Contents

1-1 モバイルアプリテスト自動化の現在

松尾 和昭氏
まつお かずあき

現在、米HeadSpinにてSenior Software Engineerとして働く。
Software Engineer in Quality/Test Engineerとして従事していた（クックパッド、ACCESS）。
Appiumのコアコミッター。

はじめに

2019年4月、ブラウザ自動化ツールとして代表的なOSS(オープンソースソフトウェア)であるSelenium(セレニウム)の公式カンファレンスであるSeleniumConfが日本で開催されたように、日本国内においてもさまざまなテスト自動化に関わる事例を広く聞くことができるようになってきました。その内容も、導入から運用、さらにはより長く保守していくための経験話と広がりを見せるようになってきたように思えます。

筆者はこれまでに組み込み、Web、モバイルアプリとさまざまな領域を経験してきました。いずれもソフトウェアのテストや品質保証に関わる範囲を主軸に、時には開発やマネジメントも行ってきました。現在はHeadSpinという米ベンチャーにソフトウェアエンジニアとして所属し、主にモバイル環境を対象とした自動化プラットフォーム開発を行っています。また、OSSの領域ではAppium(アピウム)と呼ばれるネイティブアプリ向けのテスト自動化ツールの開発にコアメンバーの1人として関わっています。

本稿では、筆者の経験の中から特に、モバイル周りの経験を中心としたテスト自動化の昨今の話を共有いたします。また、国外で開発が進む自動化に関わるサービスやHeadSpinが提供するサービスの例をご紹介します。これらは英語圏が中心の舞台ですが、自動化に関わらず、英語圏におけるさまざまな活動に興味を持つきっかけとなると幸いです。

国内における自動テスト活用の現状

1 数値から見たシステムテスト自動化の現在

日本国内におけるシステムテストレベルのテスト自動化がどの程度普及しているのかを集計した調査結果があります。この結果は筆者も所属するテスト自動化研究会(システムテストの自動化に取り組む技術者が、研究結果を持ち寄って情報交換を図る任意団体)が、2013年よりほぼ毎年開催しているシステムテスト自動化カンファレンスの参加希望者を対象に行っているものです。多様な業態や役割、経験年数を持った平均300人を超える人が回答しています。

この結果はテスト自動化研究会のサイトに公開されているため、誰でも閲覧可能です。結果をいくつかご紹介します。

例えば、システムテスト自動化自体の経験が無い人の割合は年々減ってきているようです(図1)。

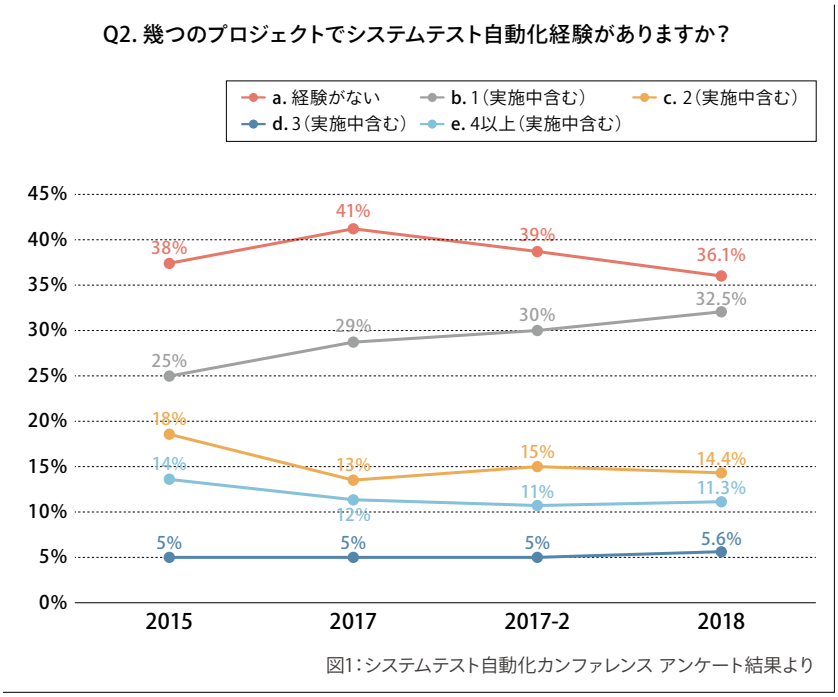


図1:システムテスト自動化カンファレンス アンケート結果より

Q4. システムテストの自動化はうまくいっていますか？

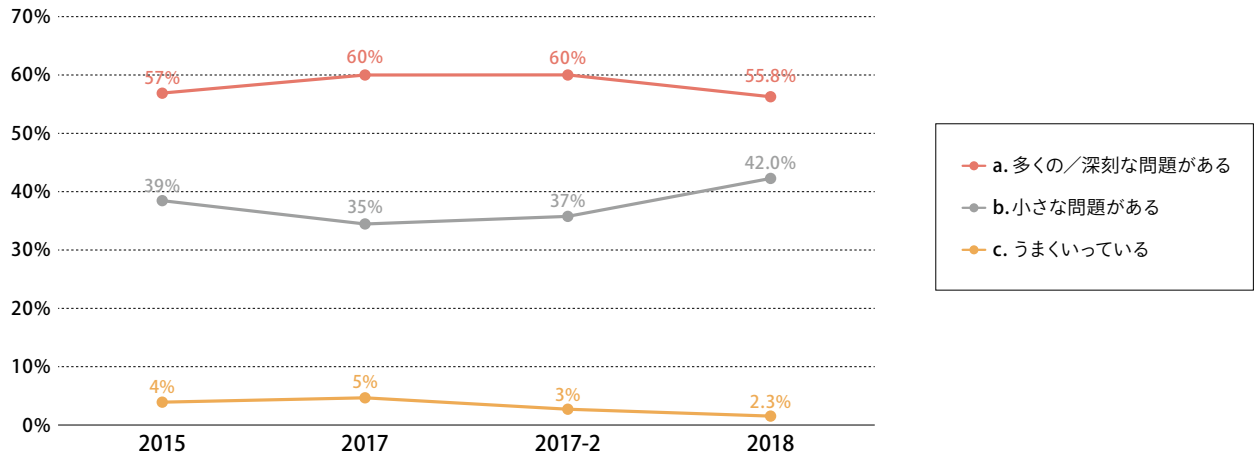


図2：システムテスト自動化カンファレンス アンケート結果より

Q5. システムテストの自動化について、今どんな問題をかかえていますか？（複数選択可）

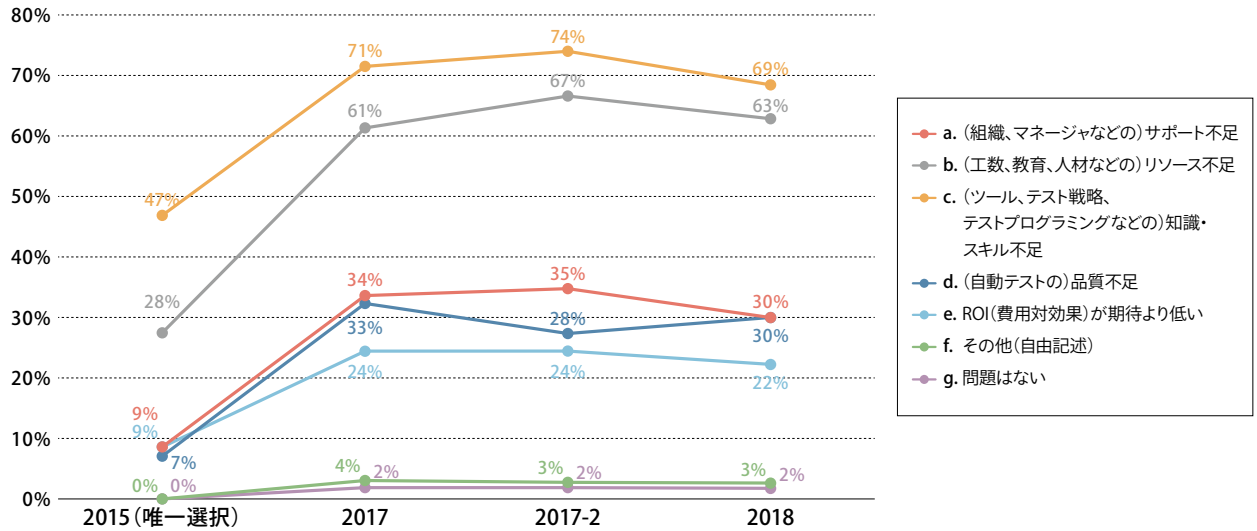


図3：システムテスト自動化カンファレンス アンケート結果より

また、テスト自動化に対して遭遇する問題の大きさも、徐々にですが小さくなってはいるようです。小さな問題には直面しつつも、経験を積んだ人、テスト自動化システムの開発に関するスキルを身に付けた人が増えているのだと見て取れます(図2)。

抱えている課題の内容に関して経験やスキル不足が上がっていますが、テスト自動化に関する経験が増えてきていることで減少傾向にあるようです(図3)。

これらの結果からも、システムテスト自動化は徐々にですが確実に広がってきているようです。テスト自動化に関わるシステムを安定稼働させるには通常の開発同様、プログラミングなどの開発スキルが必須です。それらの知識や経験に対する課題に対しても徐々に改善されているようです。

2 自動化の中心はWebアプリ、機能性のリグレッションが主

筆者が同カンファレンスなどにおいて話を聞く限りでは、主に自動化の対象となっているところは機能性に関するリグレッションテストのようです。それ以上の活用に関して言及する事例もありますが、まだ少数のようです。特に、Webアプリの開発においてはサーバ側のツールも充実しているため、性能計測や負荷試験を自動化し、継続的にCI/CD(Continuous Integration/Continuious Delivery:製品の実行環境構築や、テスト、リリースまでのプロセスを自動化することで、ビルドやテスト、リリースを頻繁に繰り返し可能にする手法)の一部として組み込み運用する話も聞きます。

他方、モバイル領域はまだWebに比べると挑戦途中のようです。まだモバイルアプリ関係には手を付けたいができていない、もしくは取り組み始めたばかりという話を多く聞きます。

モバイルアプリのテストの世界

本章では、特にモバイルアプリの開発における話に焦点を絞ります。

1 スマホアプリの開発・リリース間隔の特徴

現在のモバイルアプリの大半はAndroidもしくはiOS向けに開発されます。AndroidはGoogle社により、iOSはApple社により開発されています。Android、iOS共に、ほぼ毎年何らかの機能が追加・変更されます。それに伴い、Android/iOS向けのアプリ開発環境も大きな変更が発生してきました。開発に関わる人々は、自社のサービス開発のほか、OS側で発生する変更への追従も気にする必要があります。

モバイルアプリのリリース頻度はWebアプリに比べると多くの場合は低くなります。Webアプリでは1日に何度もリリース可能な環境を構築することも可能ですが、モバイルアプリでは現状はそのような仕組みを作ることは困難です。特に大きな要因としてGoogle社やApple社が管理するアプリストアからアプリを配信する必要があり、その審査に時間がかかるためです。

ただし、週に数回(審査に要する時間に依存しますが)程度であればリリースが可能です。企業によっては

高頻度なリリースが可能な環境を実現しています。

高頻度なリリースが可能な環境構築にはビジネスに関わる施策からの要求も大きく関わります。Webの場合、ある程度の簡単な修正であればものの数分程度で本番環境へ反映可能な環境が構築可能になっています。何らかの施策を実装する場合、リリースを自分たちの意思で決め、任意のタイミングで本番環境に配信することが可能です。一方で、モバイルアプリは先に述べたようにリリースまでに審査などの時間を要するため、アプリリリースまでに自分たちの制御下に置けない要素があります。そのため、リリース可能な状態になるまでの時間を短縮すること自体が、何らかの取り組みを世に出す機会を増やす第一歩に繋がります。

2 iOSとAndroidのバリエーション

iOSに関わる開発では、サポートするOSバージョンは2メジャーバージョン、端末の多様さはそれらのOSがサポートする端末であることが多いです。これはAppleが開発環境であるXcodeの最新版には最新の2メジャーバージョンのOSシミュレーターしか含まれないという事情があります。そのため、リリースされるアプリの多くも2メジャーバージョン、期間にして約2年前のOSまでサポートすることを1つの目安にできます。ただし、企業によってはユーザの多くがより古いOSを利用しているために、より多くの種類を考慮するところもあります。

Androidに関する開発のOSのサポート期間はより長い傾向があります。最近では、多くのユーザ数をもつアプリであってもAndroid OS 5系未満のサポートを終了するものが増えていますが、Android OS 5系ですら2014年後半に世の中に出たものです。そのため、5年程度昔のOSをサポートする必要があることもまだ多いです。アプリ開発者の視点からの見ると、端末の差異はOSが吸収してくれることが増えてきたものの、それでも例えばカメラなどのようなハードウェアの機能を利用する機能ではiOSよりはるかに多くの機種の確認が必要な場合があります。

このように、モバイルアプリ開発を取り巻く環境は短時間でのリリースを求められながらも、毎年更新される開発環境、さまざまなOSバージョンと端末の組み合わせを考慮しながら開発する必要があります。1つのアプリを開発する人数も、数人という規模から、世界的に使われるアプリでは1つのアプリを何十人単位の開発者が並行して開発を行う環境まで存在します。

3 テスト自動化による支援

前述のより短いリリース環境を構築する上で、開発に関わる諸々の自動化は必需品となります。人手に依存する箇所が多いと、開発環境をスケールすることが難しくなるためです。短時間でより多くの実行を行うには自動化は避けることが難しく、テストに関わる活動も例外ではありません。

例えばAndroidアプリのテスト自動化の場合を考えてみます。ここでいうテスト自動化は想像しやすいようにモバイル端末に描画されるGUIを操作しながら行う類のUIテストを想像してください。あるテストセットが端末の画面解像度に依存しないテストコードとして実装されている場合を考えます。ある端末向けに用意した一連のケースを実行するテストセットは、2台、3台と容易に実行端末を拡大することができます。端末が近くにない場合も、ネットワーク越しに実行端末を確保可能な環境を用意すれば、拡大はさらに容易になります。日本国外へ実行端末を広げることも容易です。これらが手動の場合、人手の確保とその実行に時間をかけたり、実行時に参照するテストケースの表現の精度に結果が依存することもあります。

人気のアプリは機能拡大の際、急激に開発者が増加します。販売戦略上、開発期間を伸ばしリリース頻度を下げる方法はあまり取ることができないことも多いです。そのため、週1リリースを保ちながらも、その開発コードは毎月のように万単位で増減を繰り返すというような状態が発生する可能性があります。このように短時間で修正・確認を繰り返す場合、保守可能な自動テストの価値が非常に上がります。

4 モバイルアプリのテスト

自動化ツール

モバイルアプリの自動化ツールは、必ずAndroid、iOSプラットフォームが提供するテストツール(以下、「公式ツール」という。)を利用します。モバイル向けのサードパーティ製テストツールもありますが、それらは何らかの形で公式ツールを経由しています。その上で、公式ツールでは実現できないことを実現したりしています。特定の開発環境に特化したものも存在します。

筆者も開発に関わっているAppiumは、長く開発され続けている自動化ツールの1つです。

現在、OpenJS Foundation配下でOSSとなっています。Appiumの特徴の1つは、Appium自体が多様な端末を自動化するためのハブになるということです。つまり、Android、iOS、Windows、Tizenなどの多様なOSをもつ端末を対象に、Appiumを介して自動化を支援可能です。AppiumはSeleniumの影響を受けて開発されているため、Selenium WebDriver APIをサポートするクライアントライブラリを利用してテストコードを書きます。公式からはRuby、Python、JavaScript、Javaなどの多様な言語向けのライブラリを提供されています。

これらの他にもモバイルアプリ自動化ツールが多数存在しますが、筆者が英語圏含めた話を聞く限りでは公式ツールとAppiumが大多数の勢力を持っているようです。

自動化ツールを利用して構築する自動テスト環境も1つのシステム開発です。そのため、安定したシステムを組むためにはアプリ開発やサーバ構築の経験は不可欠となります。ある程度の方式に沿ってテストコードを書くことは可能なことも多いですが、それらの環境を運用するためのシステムを構築するには開発経験やその知識が必要です。

国外の事情: 国際カンファレンスを通して

2019年4月にSeleniumConfが東京で、6月にはAppiumConfがインドで開催されました。筆者は運営委員、発表者としてそれぞれのカンファレンスに参加してきました。これらのカンファレンスについてこの場を借りて情報を共有します。

1 SeleniumConf 2019 Tokyo

SeleniumConfは、毎年ヨーロッパもしくはアメリカで行われているSeleniumの国際カンファレンスです。SeleniumConf Tokyoはアジア初の開催となったカンファレンスです。(但し、インドではローカルカンファレンスとして毎年開催されているためアジア初の文脈より除かれています。)全体参加者数は400人を超え、少なくともそのうち半数は国外からの参加となりました。

SeleniumConf Tokyoでは

CfP[*1]により発表者を募りました。英語と日本語で登録可能としたところ、英語圏より200件以上、日本語圏より20件程度の応募がありました。内容は、海外からは自動テスト入門、うまくテストコードを書くには、長く運用した気付きなど、自身の経験を他者に共有するものが多く、日本語からは事例発表に近しい、運用などの話が多かったように思えます。最終的には、約20ヶ国から発表者を集めることになりました。発表内容はSeleniumに関係するコードの書き方から、運用や組織の話など幅広く採択しました。もちろん、基調講演としてSeleniumの将来の話もありました。

興味深い発表として、Windows環境におけるブラウザテストを仮想環境上で実現することでスケールを容易にするというような、Seleniumの運用環境を提供する技術的な話もありました。このような発表は、

Seleniumがすでに15年ほどの長い年月をかけて発展し、多様な環境で共通して利用可能な自動化ツールとして多く利用されていることを背景に生まれたものです。

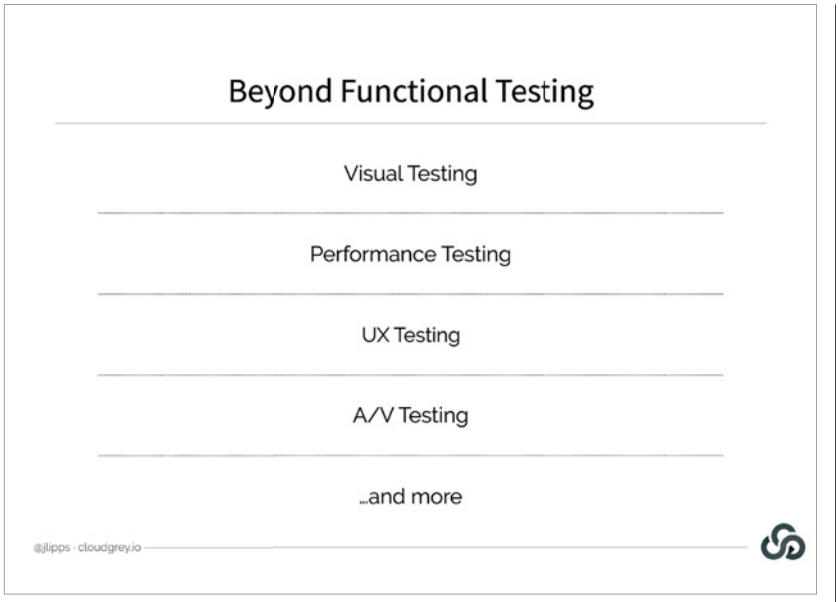
今からでも可能なOSSへの貢献として公式ドキュメントへの修正から

*1: Call for Proposal:講演内容をまとめた資料(プロポーザル)の募集を呼びかけるという意味から、プロポーザル提出による講演の公募を指す。

始める、小さいところから始めて継続していく、問題を口にするだけではなく一緒に発展させる、というような話もありました。長く開発されているOSSならではの話ではありますが、OSSの抱える問題もわかる話題です。

また、参加者が英語で互いに議論している姿が会場の至る所で見られました。その話題の中心は経験や技術情報の共有、Seleniumや関連ツールの開発者に会って直接感謝を伝えるなどです。

筆者は2019年10月に行われたSeleniumConf London 2019にも参加してきました。Tokyoでの開催は話の内容を技術的な話題を中心にすることで、アジア圏に自動化という文脈を広める狙いもありました。一方、Londonの発表では、マネジメント、自動化が効果的なところを思い出そうというような、自動化が進んだ上で再度テスト・品質を考えるという話の流れが多かったように思えます。Londonはヨーロッパ圏からの参加者が多く、各々が議論は英語を中心としつつも、各自の雑談はそれぞれの国の言語で行っている風景も散見されました。



Appium 公式カンファレンス より

2 AppiumConf 2019

Appiumの公式カンファレンスはインドのバンガロールでありました。2018年にロンドンで第1回目が開催され、インド開催でAppiumConfは2回目となります。SeleniumConf同様、Appium周辺に関わること、今後の展望の話が発表の大半を占めました。

基調講演では、Appiumの開発をリードしているJonathan Lipps氏が、「Beyond Functional Testing」というキーワードのもと、現状のモバイルアプリ周辺のAppiumが関わっているテスト自動化周りの発展について話をしました。現在、多くの「自動テスト」についての話題は画面操作を自動化することで機能性を確認することが多いです。講演ではそれだけではない「自動テスト」をより拡大するための挑戦の現状について取り上げました（上図）。

例えば「Visual Testing（視覚テスト）」の例として画像識別に人工知能を利用したApplitools社のツールを取り上げ、より賢く画像判定を行う話がありました。また、「Performance Testing（性能テスト）」や「A/V Testing（A/V機能のテスト）」の話も大きなトピックの1つとして取り上げられました。このPerformanceやA/Vの話では現在筆者が働くHeadSpinが取り上げられました。（後ほどどのようなサービスか説明します。）AppiumConfではモバイルやIoT

を連携した機器のテストについて、既存の自動化されたテストを補助・発展させる挑戦が広がりつつある話がありました。筆者はセッションを持っていたため他の話を全ては聞いていませんが、Appiumに関わる技術的な踏み込んだ話、より使いやすくする話、事例やAppium以外の話もありました。全般的に「Appiumを利用してみた」ではなく、より高度に利用するには、自分たちはこう利用しているという話をする場であったように感じました。

リグレッションテストを越えた取り組み

前章において、AppiumConf 2019では1つの機能性の確認の次の段階として、Performanceなどの評価に挑戦しているという話がありました。現在、モバイルアプリ、特にリリース物に対して行うテストでは自動化スクリプトによりGUIから操作を行い、シナリオベースの動作確認を行う類の機能性のテストが多いです。それらの実行を多機種環境で支援するサービスもいくつか存在します。

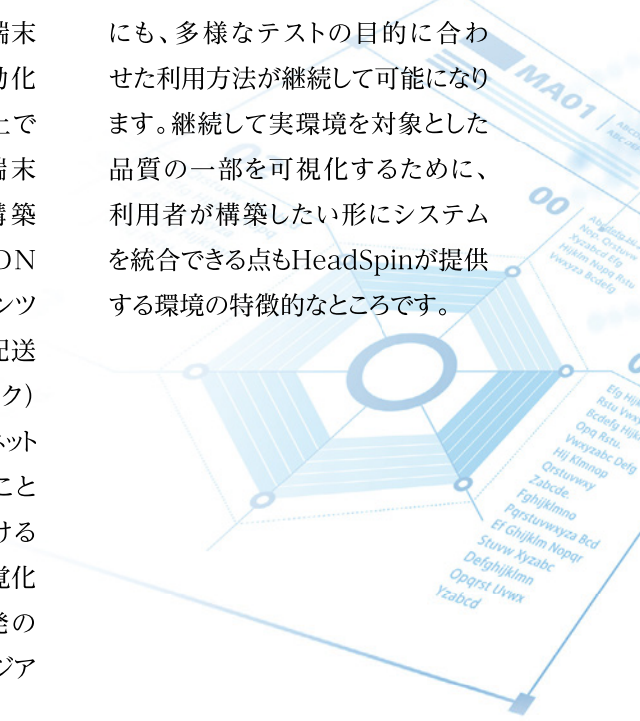
本章では、その実行環境をより発展させるために挑戦している、著者が所属する米ベンチャー企業のHeadSpinの事例を少し紹介します。

1 継続した性能計測・モニタリングとして活用される自動化されたテスト

HeadSpinはモバイル環境における顧客体験と性能の要求に対するソリューションを提供している企業です。HeadSpinは世界150箇所以上に、合わせて22,000台以上のSIMカードを搭載したモバイル端末を保持し、それらの端末に対する自動・手動テストの実行や、ユーザ体験や実世界における性能計測といった機能を提供します。この「性能計測」には端末から発せられるHTTPリクエストのキャプチャー、CPUやメモリーなどの資源の使用率などがあり、それらをAndroid・iOS端末を対象に取得可能です。

実際のさまざまな地域に物理端末が存在するため、定期的に自動化スクリプトを世界各国の端末上で実行させることで、モバイル端末主体のモニタリング環境を構築することも可能です。とあるCDN（Web上で送受信されるコンテンツをインターネット上で効率的に配送するために構築されたネットワーク）を提供する企業の場合、自身のネットワーク経路の評価に利用することで、実モバイル端末環境における自分たちのネットワーク品質を視覚化可能です。モバイルアプリ開発の場合、リリース前の段階であるアジアの国でテストを実行することで、どれほどのネットワーク遅延が実際に存在するのかということを評価し、リリースに望むことが可能です。もちろん、リリース後の定点観測にも利用可能です。

このように、HeadSpinの提供するソリューションの1つとして提供されるテスト実行環境は、開発プロセスの場所にとらわれず利用可能です。同環境では、Appiumをはじめとした多数の自動化ツールを利用できます。そのため、テスト対象をアプリストアから配信されているものから、開発しているリリース版、開発用モジュールともものを選びません。むしろ、自動化されたテストが存在することで、多様なシナリオに対して機能性を確認するという目的以外にも、多様なテストの目的に合わせた利用方法が継続して可能になります。継続して実環境を対象とした品質の一部を可視化するために、利用者が構築したい形にシステムを統合できる点もHeadSpinが提供する環境の特徴的なところです。



おわりに

本稿では、国内のテスト自動化の最近の傾向を数値情報から見てみました。その中で、筆者が特に関わりが深いモバイル領域、Appium周りの話、SeleniumConfやAppiumConfという英語を主としたカンファレンス話を通してさまざまな昨今の自動化話に触れました。また、AppiumConfで取り上げられたテストにおける昨今のテスト自動化の挑戦例として、筆者が所属するHeadSpinのサービスを紹介しました。

日本語圏において、ソフトウェアテストの世界でも自動化をはじめとしたツールの実装に対する力量が注目されて久しいです。一方、英語圏の自動化に向けて取り組んでいる人たちの多くは以前より、さまざまなツールの開発を通じ、お互いにコミュニケーションを取っています。そのため、英語圏ではより多くの情報に触れることができます。活動の幅も広がります。本稿をきっかけに英語圏の情報や、シリコンバレーをはじめとしたさまざまなソフトウェアテストにまつわる話に興味を持ち、議論や挑戦へ繋がれば幸いです。

本記事の内容に関するご質問、お問い合わせは、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。

1 2 実践！アジャイル・DevOpsテスティング

藤原 大氏
ふじはら だい

楽天株式会社、株式会社メルカリを経て現在はフリーランスのアジャイルコーチ、エンジニアリングマネージャとして活動。『リーン開発の現場』の翻訳者でもありアジャイルな創造的、継続的、持続的なソフトウェア開発の実現に向けて奮闘中。週末に娘と息子とお昼寝しながら世界のビーチや離島を旅する夢を見る。
Web：https://daipresents.com/service/

はじめに

「自動化」はソフトウェア開発のみならず、自動車産業やそれ以外の領域のトレンドになっています。本稿では、筆者がモデレータとして参加した2019年3月 JaSST(ソフトウェアテストシンポジウム)'19 Tokyoにおけるパネルディスカッション『テストの未来、品質の未来 ～自動化はテスター撲滅の夢を見るか?』と、2019年7月IT検証フォーラムで講演させていただいた『アジャイル・DevOps時代の品質組織づくり』の内容を基に、今求められているアジャイルなテスト(以下、アジャイルテスティング)の概要と実践を紹介させていただき、次世代の品質組織について検討します。

なぜ今アジャイルテスティングが必要なのか?

1 正しいものを作る時代へ

アジャイルな(俊敏な)開発やテストは、これまでずっと求め続けられてきました。

これまでは「作れば売れる時代」だったかもしれませんが、モノがあふれるようになり、欲しい物が手に入りやすくなると、「何を作るべきか」や「何を作らないべきか」を深く検討しなければなりません。これはつまり、「何を作れば正しいのか」という意思決定が難しくなり、顧客のニーズを探しながら、正解を探し続けていく開発が主流になってきたのだと思います。

さらに、正解を探すためのフィードバックも重要です。そのためには素早く仮説となるソフトウェアを提供し、その反応をすぐに受け取りさらに改善する・・・といった開発サイクルが必要になります。

そこに登場したのがアジャイル開発です。変化の激しい時代の中で、柔軟に対応できる開発プロセスが注目されるようになります。それは、スクラムによって一般的になり、最近ではアジャイル開発、スクラム、DevOpsといったキーワードが、IT系情報サイトにて登場しない日はありません。

キーワードに共通するのは、継続的に、漸進的にサービスやプロダクトを作り続ける開発スタイルです。つまり、開発には終わりはなく、ちょっとずつリリースサイクルに収まる規模でより重要と仮定される機能「だけ」を作りながら、開発や運用を続けることで、より「正しいもの」を作ろうとするプロセスに近づいています。

2 アジャイル開発に最適な技術の進化

そんな時代の中で、インフラ面ではDockerに代表される軽量な仮想化技術や、Kubernetesのように柔軟で拡張性の高いインフラ管理のプラットフォームが発達し、CircleCIやBitriseといったクラウド上で手軽に利用可能な開発支援サービスが出揃ってきたため、ソフトウェア開発においてCI/CD(Continuous Integration:継続的インテグレーション*1,Continuous Delibvery:継続的デリバリー*2) が一気に浸透しつつあります。これは、技術的にアジャイル開発を実装しやすくなったということです。

CI/CDによって、これまでの開発プロセスに自動化が当然のように組み込まれていきます。具体的には、エンジニアのソースコードがSCM(ソフトウェア構成管理ツール)にコミットされるたびに、アプリケーションがビルドされ、そのアプリケーションに対して自動テストが実行され、テストが通ったならそのまま

*1: ソフトウェア開発において、ビルドやテストを頻繁に繰り返し行うことにより問題を早期に発見し、開発の効率化や品質の改善、納期の短縮を図る手法。

*2: CI(継続的インテグレーション)の拡張。CIがアプリケーション実行環境の構築や、テストまでを実行するのに対し、継続的デリバリーではリリースまでのプロセスを自動化する。

本番環境へとデプロイされる･･･。といったビルドパイプライン*3が実現できるようになりました。

3 アジャイルテスティングの登場

CI/CDのような技術の発展や、アジャイル開発やDevOpsのような連続する開発プロセスによって、「どうテストするか?」や「どう品質を担保するか?」という命題にもパラダイムシフトが必要になります。

2018年、アメリカのアナハイムで開催されたSTAR WEST Conferenceにおいて、次のような話題が議論されました。継続的な開発が行われるということは、最後にまとめてテストする従来の手法では、そこがボトルネックになりがちです。よって、テストというフェーズをなくし、継続的にテスト活動する、つまり、継続的テストの実現が求められます(図1)。

変化の激しい連続するプロセスにおいて「テストフェーズ」のボトルネックを避けるために、「アジャイルテスティング」という従来の考えを大きく変える発想が登場します。

「アジャイルテスティング」という言葉自体は、10年以上前の2008年に出版された『実践アジャイルテスト:テスターとアジャイルチームのための実践ガイド*4』で取り上げられたことで注目されるようになりました。

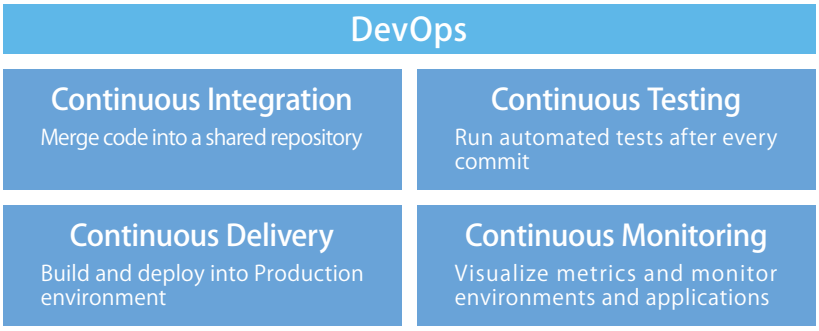


図1:DevOpsを支える4つの継続的なアクティビティ

しかし、書籍の内容は「アジャイルテスティング」の考え方や概念が中心の解説であったため、具体的にプラクティス(習慣・手法)や導入方法、導入後の組織のあり方などについてイメージできなかった方も多かったのではないでしょうか。

メルカリで私が所属していた部署には、QAエンジニアだけでなくSET(Software Engineer in Test:テスト環境の構築やテスト自動化などを推進するQAのためのエンジニア)も所属しています。「アジャイルテスティング」について世界の最先端を理解するために彼らと手分けし、2018年に主要な世界のソフトウェアテスト関連カンファレンスに赴き、情報を集めました。

4 アジャイル開発やDevOpsにおける継続的テスト

アメリカのサンディエゴで開催された Agile 2018 Conferenceでは、Ebay社のテストエンジニアがDevOpsにおけるテストのあり方を紹介していました。ここでは、終わりなきプロセスの中に登場するあらゆる活動にテストが必要であり、それはテストエンジニア、テスター、QAエンジニアだけでカバーできるものではないことがよく分かると思います(図2)。

*3: ソースコードのコンパイル、モジュール配置、自動テスト実行など、ビルドの一連のプロセスを分割して順番に自動で実行する仕組み。プロセスを適切に分割することにより失敗時の影響が限定され、復旧までの時間を短くすることができる。

*4: 原書:Agile Testing: A Practical Guide for Testers and Agile Teams, Janet Gregory, Lisa Crispin, Pearson Education, 2008

アメリカのアナハイムで開催されたSTAR WEST Conferenceでも、先ほどご紹介したように同じような話題が議論されました。ここからも、継続的にテスト活動することが求められるようになることが伺えます。

オランダのハーグで開催されたEuroSTARでは、自動化ありきのセッションが多く、スポンサーのブースは自動テストのサービスでいっぱいでした。さらに、QAエンジニアのキャリアプランとしてSETや自動化エンジニア(Automatorと呼ばれていた)が多く紹介されており、そういったキャリアチェンジをした方のセッションが人気となっていました。

まとめると、アジャイルなテストを実現するためには、誰もがテストに関わる環境づくりが必要で、継続的な開発に合わせて継続的なテストが必要です。そのためには、SETやAutomatorによるテスト基盤の構築というような、新しい役割やスキルがますます求められるようになっていきます。

実践 アジャイルテスティング

では、実践的なアジャイルなテストについて、メルカリでの経験をもとに紹介させていただきます。

1 継続的テストのためのテスト自動化

継続的なテストを行い、テストによる素早いフィードバックを提供するためには、自動化は必然です。もし、人手をかけて継続的テストができるならば、それも一つの手ですが、現在はテストの自動化技術も発達してきたため、属人性を排除し、ミスや漏れが生まれにくいテスト実行が可能です。

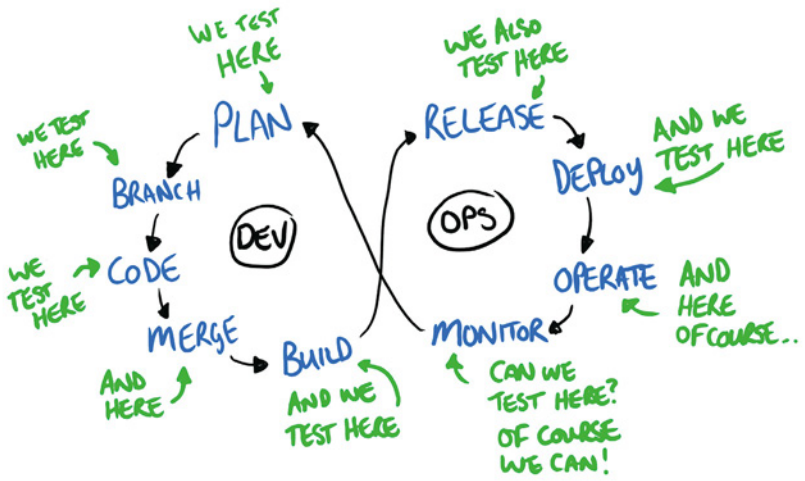


図2:Continuous Testing in DevOps by Dan Ashby.
https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/

メルカリでは、リグレッションテストからスマートフォンアプリのテスト自動化を進め、現在は、自動化されたリグレッションテストがAndoridアプリとiOSアプリ、それぞれ170シナリオほどあります。

まだそれぞれ100シナリオぐらいがマニュアルテストとして残っていますが、その理由として、外部システムとの連携があり自動化しづらい構成になっていたり、スマホアプリのスワイプやスクロールといった、操作性の確認が求められるなど、自動化しづらい部分があるからです。

ただバグを速く見つけるだけではなく、合わせて、早い段階からQA活動することによる、バグを作り込まない活動が必要となるでしょう。

これらはシステム自体の改善(モック化など)や、ツールの活用(Visual Validationなど)で軽減できるようになってきました。いかに「テストしやすい」システムにするかは、QAエンジニアの仕様検討参加や、SETによるアーキテクチャ改善やリファクタリングなど、継続的な活動が大切になってきます。

現在自動化されているケースの自動化は、できるところから全部、

短期間で集中的に行いました。現在は、夜間実行した結果を朝確認する流れになっていますが、大きな問題がなければ、日々15分ほどの確認・修正作業が運用コストで済んでいます。

こういった話をするとよく質問されるのが、テスト自動化におけるROI(投資収益率)についてです。日本では、初期や運用コストの課題や、自動テストをどう普及するかが主題になりがちですが、海外では、すでに「自動化ありき」で物事を考える傾向があります。

自動化の初期コストは高くなりますが、テスト実行を技術的に高速にする方法も安価になってきているので、繰り返し実行すればするほど、そのROIは高くなっていきます。また、一気にやれるなら一気に進めることで、早い段階で繰り返し作業を激減させることが可能です。

2 テスト自動化のための自動テスト基盤構築

テスト自動化のためには、その基盤の構築が必要です。そこには先に説明したビルドパイプラインとも連携し、常に最新のソースコードを用い、ビルドからテストまでをシームレスに実行する環境が必要です(図3)。

基盤を支える要素は、大きく分けて以下のようなものがあります。

- テストデータ作成基盤
- テスト実行基盤
- テスト結果データ基盤

継続的なテストを実現するためには、「繰り返し使えるテストデータ」を作成する仕組みが「テストデータ作成基盤」として必要です。メルカリでは、ユーザ登録など、実際のアプリと同じAPIをテスト用に呼び出せるラッパーAPI郡を作成し、パラメーターを渡すだけでデータを作成する仕組みなどを構築しています。

実行基盤は、実行環境の構築からアプリケーションのソースコード取得、ビルド、シナリオ実行に至るすべての部分をテストする、いわゆるスマホアプリのエンドツーエンドテスト(E2Eテスト)基盤をCI(Jenkins)で構築しています。アプリは毎回クリーンな環境でビルドされ、クリーンな端末(エミュレータやシミュレータ、実機など)で並列実行されます。

テスト結果をどう集約するかはまだ試行錯誤していますが、Allure FrameworkやTestRailといったサービスを使い、テスト結果データ用データベースとして、そこにデータを溜めていくようにしています。

データが溜まっていくと「どのテストが不安定か」といった分析ができるようになります。また、不安定な箇所はバグが起こりやすい箇所かもしれないため、エンジニアにもフィードバックでき、その結果実施したリファクタリングによって、大きな改善が見込めるかもしれません。

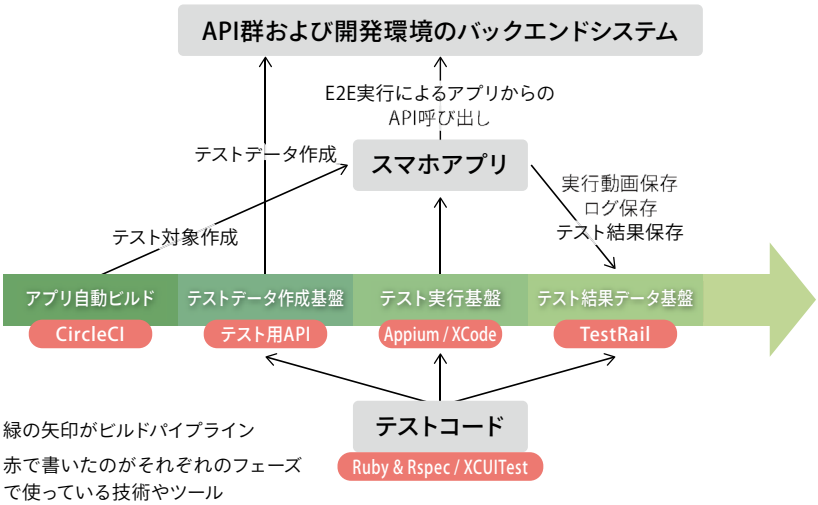


図3:テスト自動化のための自動テスト基盤構築

3 テスト自動化や自動テスト基盤構築のための人材育成

自動テストは人間のように、ポチポチとブラウザやスマホをタップしながら、ユーザを作る必要はありません。APIがあるなら直接呼び出してユーザーデータを作れるため、高速に実行できます。そのため、テストケース作成は、従来型のマニュアル思考ではなく、自動化思考で設計できる人材が必要です。

メルカリでは、当時存在したマニュアルテスト用リグレッションテストケースをそのまま自動化していきましたが、今やり直せるのであれば、自動化用にテストケース設計から始め、自動化で補えない場所をマニュアルテストで埋め合わせていく形を取ります。

そもそも、人間が実行しやすいテストケースが、自動で実行しやすいとは限りません。例えば、Aをやってからに BとCをあわせて確認したほうが効率がいい・・・といったちょっとしたTIPSは、暗黙知となりがちで、新しい人から見れば「なぜこんなルートで確認しているのだろう?」となってしまいます。

自動テストにおいては、マニュアルテストの職人的なスキルよりも、ソフトウェアエンジニアリングの基礎力が求められます。プログラミング的な考え方(構造的なプログラムの書き方やコンポーネントの共通化など)が重要で、小さな単位で高速に実行できるため、効率良いルートは必要ありません。よって、シンプルで独立した記述になり、誰でもテストの観点を理解できるシナリオへと育っていきます。

メルカリでは「全員自動化」を合言葉に、QAエンジニアもテスト自動化に関わっていく体制を取っていました。しかしながら、QAエンジニアによる自動テスト設計はなかなかうまく行かず、テストコードのコーディングはさらに敷居が高いものでした。

しかし、国内でもQAエンジニアからSETやAutomatorへのキャリアを考える人は増えてきており、メルカリでも、「マニュアルテストだけでなく自動化というキャリアも積みたい」と従来型のテストに危機感を持ったQAエンジニアを1名採用し、SETチームに配属して育成しながら自動化を推進してもらっています。

おわりに

冒頭でご紹介した『テストの未来、品質の未来~自動化はテスター撲滅の夢を見るか?』というパネルディスカッションにおいて、興味深いアンケート結果がありました。事前に有識者数十名にアンケートを取らせていただいたのですが、「テスターの仕事は変わらない」と答えた人は0人。つまり、全員が「テスターの仕事は変わる」と考えているようです。よって、「テスト」だけをサービスとして提供する部署や企業の価値は今後下がっていく傾向にあり、スキルとしての価値も同様に下がっていくだろうと考えています。

テスト自動化は、これからの開発プロセスに必要な「継続的なテスト」に必要となる新しい力のひとつです。

SETやAutomatorといった役割が目立ちますが、テスターやQAエンジニア、テストエンジニアもテスト自動化に関わる時代がやってくる予感がします。

テストや品質の未来は、まだ明確に見えてきてはいませんが、今回紹介させていただいたテスト自動化などの事例がどんどん登場し、「アジャイルテスティング」が実現できる未来は近いと期待しています。

本記事の内容に関するご質問、お問い合わせは、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。

1-3 ベリサーブが考えるテスト自動化プロジェクトのマネジメントとは

伊藤 由貴
いとう よしき

株式会社ベリサーブ
IT企画開発部自動テスト推進課 所属
新卒で入社以来、テスト自動化ツール開発や、組み込み機器・エンタープライズシステム・Webアプリケーションのテスト自動化を経験。現在は社内におけるテスト自動化技術の普及と強化を担う。

はじめに

ベリサーブは、テスト・検証の専門会社としてさまざまなお客様の元でテスト自動化プロジェクトを行ってきました。それらの豊富な実績と経験を踏まえ、テスト自動化で管理すべき工程を標準プロセスという形でまとめました。合わせて、その標準プロセスがいかにテスト自動化のマネジメントに役立つかについて紹介いたします。

背景・経緯

過去、ベリサーブではさまざまなお客様の元でテスト自動化支援を行ってきました。自動化を支援し始めた頃、当社の支援内容の質が常駐するエンジニア個人のスキルに依存していることがありました。

このとき問題となっていたのは、

「テスト自動化を進めるために必要な作業内容、管理すべきことがわからない」
「テストの全体像が把握できていない」

という点です。

結果として、手戻りが大きくなるといったことが発生していました。

あるお客様ではうまくいき、別のお客様ではうまくいかないといったことをなくすため、テスト自動化の標準プロセスを定義することにしました。標準プロセスは、テスト全体で、今、何を検討すべき工程かを把握し、テスト自動化で、誰が・いつ・何をすべきかを明確にします。

標準プロセスを定義するまでの過程

標準プロセスを定義する以前から、社内には「テスト自動化エンジニアのロールおよびスキル定義」が存在しました（表1）。

しかし、これだけでは実際のプロジェクトを進める上で役割の違いがわかりづらいという声があり、テスト自動化に必要な作業の検討には不十分でした。そこで、各ロールと結びつく形でテスト自動化のプロセスを定義し、どのプロセスをどのロールが担当するのかを明確にすることで、これらの問題が解消できるだろうと考えました。まずはベリサーブ社内でテスト自動化の技術推進活動をしている有志の一人がプロセスのたたき台を作成しました（図1）。

このたたき台に対し、他の技術推進活動のメンバーからフィードバックを受け、それらを反映させる形で進めていきました。

例えば、このたたき台では、以下のようなフィードバックがありました。

ロール	主な業務内容
テスト自動化プランナー	自動テストシステムの要件定義や、ツール選定など
自動テストプロジェクトマネージャ	テスト自動化プロジェクト全体のマネジメント
自動テストシステムアーキテクト	自動テストシステムの設計・構築
自動テストシステム運用エンジニア	自動テストの運用・保守
テスト自動化エンジニア	自動テストのコーディング等

表1：テスト自動化エンジニアのロールおよびスキル定義

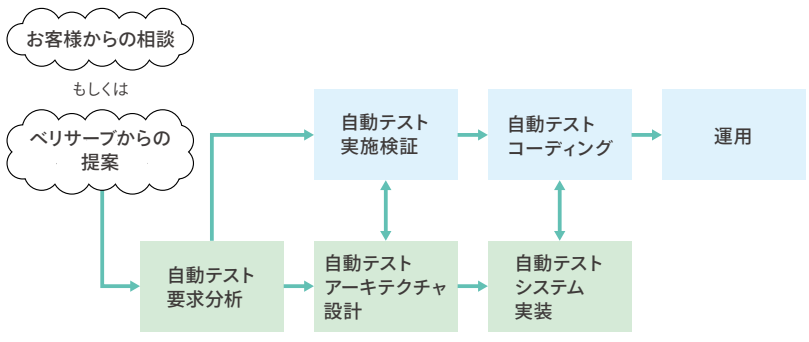


図1：標準プロセスのたたき台

- 自動テストスクリプトの話、自動テストシステムの話、自動テスト全体の話が混在している
- 自動テストのコーディングは、実際には設計と実装を含んでいるはず

さらに、テスト自動化のプロセスはそれ単体で存在するわけではなく、テスト全体のプロセスと対応付けが必要だと考えたため、手動を含むテスト全体のプロセスを含めることにしました。プロセスの大枠を作成し、さらに言葉の定義や意味合いを社内外の定義と整合性がとれるようにするなどの調整を行った結果、定義したテスト自動化のプロセスが図2です。

テスト自動化のプロセス定義

ここからは、実際に定義したテスト自動化のプロセスについて具体的に説明します。

現在ではさらに詳細化および一部変更を予定していますが、大まかなプロセスの骨組み自体は前述の図の通りです。

1 全体像

テスト自動化はそれ自体を独立して考えるものではなく、サービスやプロダクトのテスト全体の中で考えるものです。そのため、手動テストのプロセスと対応付けてテスト自動化のプロセスを定義しました。

大きなプロセスとして以下の3つを設定しています。

- 自動テストの実装・運用に関するプロセス
- 自動テストシステムの構築・運用に関するプロセス
- マネジメントプロセス

自動テストの実装・運用に関するプロセス(3～6)(図中≪自動テストプロセス≫)では、自動テストツールやライブラリを使って自動テストを作り、運用する流れを定義しています。

自動テストシステムの構築・運用に関するプロセス(7～10)(図中≪自動テストシステムプロセス≫)では、実装した自動テストを実行して結果を確認するCI(Continuous Integration:継続的インテグレーション*1)などの仕組みを構築して運用する流れを定義しています。

マネジメントプロセス(11)(図中≪マネジメントプロセス≫)は1本の流れになっていますが、テスト自動化を含んだプロジェクト全体のマネジメント・コントロールを行います。本質的にはシステム開発のプロジェクトの運営と同じですが、作業工数の見積や考えられるリスクの洗い出しなどに関しては、テスト自動化に関する知見が必要になります。

*1:ソフトウェア開発において、ビルドやテストを頻繁に繰り返し行うことにより問題を早期に発見し、開発の効率化や品質の改善、納期の短縮を図る手法。

2 全体要求分析とPoC(12)

テスト自動化プロジェクトの最初の工程では、プロジェクト全体の要求分析とPoC(Proof of Concept:概念実証)を行います。

要求分析(1)は、主に【テスト自動化プランナー】が担います。本工程では以下のような点についてヒアリングや検討を行い、全体の計画を立てます。

- 現在の開発プロジェクトにおける課題は何か
- そのうちテストを自動化することによって解決できるものはどれか
- どういった効果を目指すか

要求分析の内容を基に、テスト自動化に使用するツールの候補選定も行います。

PoC(2)は【テスト自動化プランナー】と【自動テストシステムアーキテクト】が担当します。本工程では、要求分析で検討した全体の進め方や選定したツールで小規模なトライアルを行い、テスト自動化の導入可否の調査と、テスト自動化の費用対効果の検討を行います。

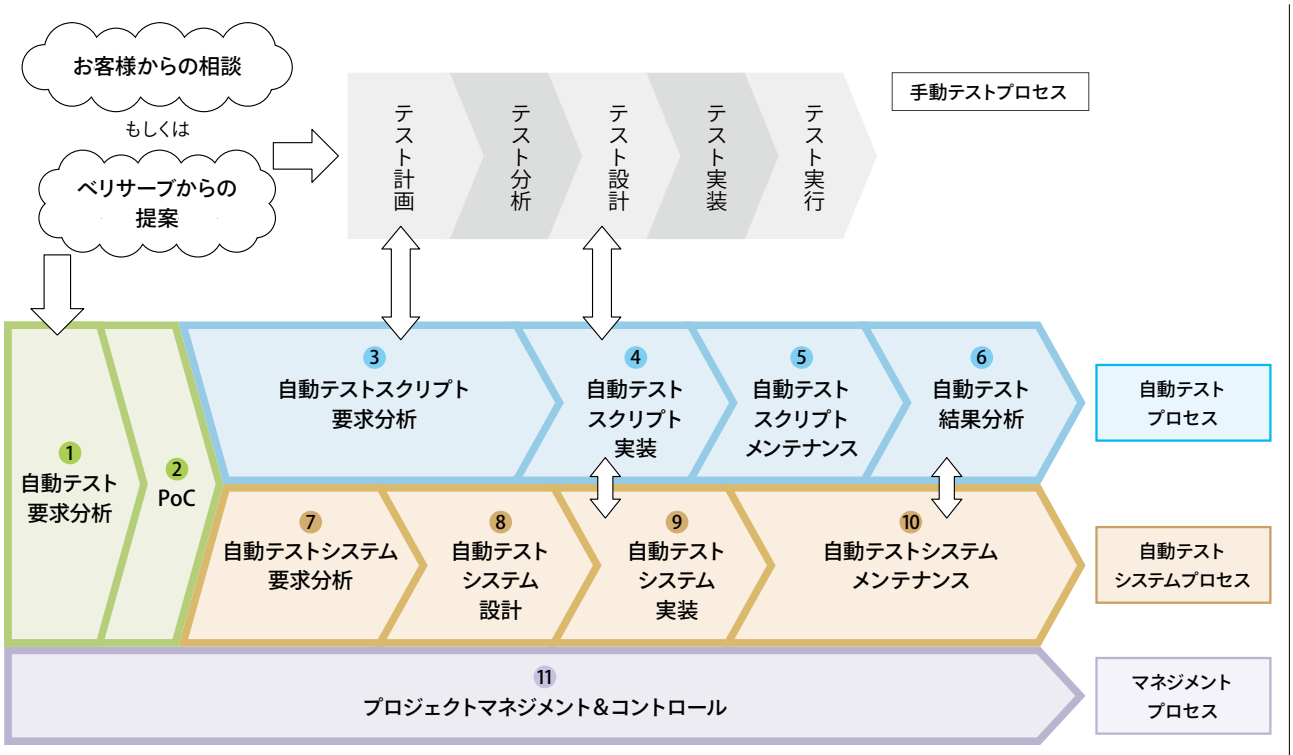


図2：テスト自動化の標準プロセス

3 自動テスト実装～運用（③～⑥）

PoCの後、自動テストの実装～運用のプロセスが続きます。

テストの自動化を行う流れは、システム開発によく似ています。自動テストが満たすべき要件をまとめ、設計を行い、その後実装を行います。

最初の自動テストスクリプト要求分析（③）は、【テスト自動化プランナー】が担当します。本工程では、以下のような点について調査・検討を行います。

- 自動テストの改修や保守を誰が行うのか、また担当予定者のスキルはどのくらいか
- 使用するフレームワークやライブラリの制限はあるか
- テスト自動化チームでの開発ルールは何か
- その他、PoC時に発見した課題の対応方法

事前に行った全体要求分析（①）と重複する部分も出てきますが、PoC（②）を行ったことによって方針が変わった点や、新たな課題が必ず出てきます。PoCの結果を踏まえた上での自動テストスクリプト要求分析（③）を行います。

次の自動テストスクリプト実装（④）は、主に【テスト自動化エンジニア】が担います。本工程では、テスト自動化ツールまたは何らかのプログラミング言語とライブラリなどを使用して、自動テストスクリプトを書いていきます。「テスト自動化作業」と言うとこの段階をイメージしがちですが、本稿で説明しているように実際にはその前後の作業が多く、かつ重要でもあります。

自動テストスクリプトメンテナンス（5）は、【自動テストシステム運用エンジニア】が担当します。本工程では、実装した自動テストが安定して正しく実行されているか日々の結果を確認したり、自動テストが動かなくなっている場合に問題点の調査と対応を行ったりします。一度作成した自動テストのメンテナンスが必要になるタイミングには、以下のようなものがあります。

- 既存のテストスクリプトの実行効率改善
- テスト対象ソフトウェアへの機能追加や改修
- 自動テストツールやライブラリのバージョンアップ
- テスト対象が動作しているOSやブラウザのバージョンアップ

このように、一度テストを自動化しても、使い続けるには継続的なメンテナンス作業が必要になります。

自動テスト結果分析（6）は、【自動テストシステム運用エンジニア】が担当します。本工程では、実行した自動テストの結果を基に分析を行います。分析対象となるのは主に結果がNG（Fail）になってしまったケースで、テスト対象に本当に不具合があるのか、それとも自動テストスクリプト/システム側の問題なのかなどの切り分けが必要になります。分析のために必要な情報の追加や変更、自動テストレポートの表示方法などは、自動テストシステムと連携しながら継続的に改善を行うポイントです。改善は、【自動テストプロジェクトマネージャ】や【自動テストシステムアーキテクト】が中心となって行います。

4 自動テストシステム構築～運用（7～10）

自動テストシステムの構築から運用までのプロセスについて説明します。自動テストの実装～運用と並行する形で進行し、これらの構成は主に【自動テストシステムアーキテクト】が担当します。

自動テストシステムに関しても、最初に以下のような点について要求分析（7）を行います。

- 自動テストの実行タイミング
- テスト実行時間の制限
- 自動テスト実行のインフラ面での制限

次に、PoC（2）や要求分析（7）の結果を基に、自動テストシステムの設計（8）を行います。設計時には、ISTQB（International Software Testing Qualifications Board）のTest Automation Engineerシラバスで定義されているGeneric Test Automation Architectureや、過去の類似プロジェクトなどを参考にします。

アーキテクチャの設計ができたら、次の自動テストシステム実装（9）でシステムを構築します。自動テストスクリプト実装（4）で作成した自動テストを実際に自動テストシステム上で動作させながら進めていきます。

自動テストシステムの構築後は、自動テストシステムのメンテナンス（10）を継続します。本工程で発生するメンテナンスとしては、例えば以下のようなものがあります。

- 自動テストシステムのバージョンアップ
- OS、データベース、ライブラリ、CIツールなど
- 自動テスト失敗時の原因調査と、システム起因の問題点への対応
- 実行効率の改善や、レポート内容の充実

5 プロジェクトマネジメント（11）

テスト自動化のプロセス全体において、要求分析（1）やPoC（2）、自動テストスクリプトの実装～運用（3～6）や、自動テストシステムの構築～運用（7～10）などの各工程のマネジメントとコントロールを行います。この工程は【自動テストプロジェクトマネージャ】が担当します。

テスト自動化を含むプロジェクトのマネジメントを行う場合は、テスト自動化に関する知見が必要になってきます。プロジェクトマネージャは、必ずしも実務担当者と同じレベルでコーディングやシステム構築ができる必要はありません。しかし、JSTQB(Japan Software Testing Qualifications Board)のテストマネージャシラバスにある「テストツールおよび自動化」の章に書かれているような内容について理解しておいたほうが、プロジェクト全体の進行がスムーズになると考えます。

標準プロセス定義の効果

ここまで、テスト自動化プロジェクトにおけるベリサーブ社内の標準プロセスをご紹介してきました。この標準プロセスの設定によって、以下のような効果が得られました。

1 会社としての提供サービス品質が向上した

プロセス定義以前ではテスト自動化全体を担当する【自動テストプロジェクトマネージャ】のスキルに依存していた部分が、担当者のスキルに寄らず一定レベル以上のサービス品質の提供することが可能になりました。

2 テスト自動化に必要な準備や発生する作業についての説明・合意形成がしやすくなった

過去のテスト自動化プロジェクトの際、お客様の中には「本当にこれだけの人数・期間・作業が必要なのか?」「もっと効率よくできるのではないか?」と考える方もいらっしゃいました。しかし、標準プロセスを用いて、いつ・何をすべきなのか、どういったスキルを持ったメンバーが必要なのかを説明することで納得していただくことができ、プロジェクトに関わる全員で認識を合わせながら進んでいくことができるようになりました。

おわりに

今回、紹介した標準プロセス定義は最初期に策定したもので、以降、さまざまなテスト自動化プロジェクトの知見を取り込みながらアップデートしています。しかし、根幹の部分は変わっていません。標準となるプロセス自体は詳細化しすぎると汎用性が無くなるため、ある程度抽象度の高いものにしつつ、テスト対象のドメインや開発スタイルなどに合わせた個別適用例の蓄積などを継続的に行っています。

日本においてもテスト自動化の事例や公開情報が増えてきており、ポイントを絞った工夫に関する情報や「テスト自動化の落とし穴」などのアンチパターンに関する情報は入手しやすくなっています。しかし、テスト自動化全体を通してプロジェクトをどう進めていくかに関してはそれほど多くない印象です。

「テスト自動化に着手したいけれども、どう始めていいかわからない」「どう進めていいかわからない」といったことに迷われている方は、ベリサーブまでご相談ください。

本記事の内容に関するご質問、お問い合わせは、**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。

順序組み合わせテストとIDAUカバレッジ

湯本 剛氏
ゆもと つよし

株式会社ytte Lab
テストリーダーとして数多くの開発に携わった後、テストコンサルタントとしてテストプロセス改善や教育などを行う。現在はfreee株式会社のQAエンジニア。
株式会社ytte Lab代表&コンサルタント、JSTQB技術委員、ISO/IEC JTC1/SC7 WG26幹事としても活躍中。博士(工学)。

はじめに

筆者は、昨年まで社会人大学院生として、ソフトウェアテストの研究を行っていました。研究の成果は、「入出力データの順序情報に基づくブラックボックステスト手法の研究」という論文にまとめました。[*1]この論文の中では、研究成果の一つとして、「順序組み合わせテスト」というテスト技法と「IDAUカバレッジ」というカバレッジ基準について提案をしています。本稿では、このテスト技法とカバレッジ基準を取り上げて説明をしていきます。

筆者は、ソフトウェアテストのコンサルタントをしていた頃、「ゆもつよメソッド」という手法をいろいろなテストの現場に適用してきました。この手法は、筆者の名前から命名したもので、テスト分析とテスト設計をセットにしています。テスト対象機能と処理に期待する結果の組み合わせから、テストの目的に合わせたテストケースを漏れなく抽出する方法です。

今回紹介する「順序組み合わせテスト」技法は、「ゆもつよメソッド」を使ったコンサルティングを行う際に、筆者の経験からテスト設計に

*1: この論文は以下のURLからダウンロード可能です。
https://tsukuba.repo.nii.ac.jp/?action=repository_action_common_download&item_id=47025&item_no=1&attribute_id=17&file_no=1

ついてアドバイスをしていたことの一つで、若い頃から現場でずっとやってきた方法です。これから説明する技法に似た方法でテストケースを抽出している人は多いと思いますが、学術論文になっているものが見当たらなかったため、研究の中で、テスト技法とカバレッジ基準としてまとめ上げました。この技法がどんなものなのかがわかるように、論文よりも簡潔に説明してみたいと思います。

1. 処理をつなげたテストをしていますか？

単一の処理に対するテスト、例えば給与計算システムで給与計算が正しいことの確認や、スマホで音楽が再生できることを確認のため、必ずテストを行うと思います。複数の処理をつないだテスト、例えば正しく給与計算された後に、その計算結果を使った年末調整処理での計算の確認や、スマホで音楽を再生している時に電話が着信したときのようなテストはどうでしょうか。これらの例は作業を想像しやすいのでテストできているかもしれません。しかし、処理をつないだテストを必要なだけ全部しているかと聞かれた時に、そもそも何をもって全部と言えばよいかうまく説明できないといったことはないでしょうか。この問題を解決する一つの方法として「順序組み合わせテスト」があります。まずは、「順序組み合わせテスト」がどのようなテスト技法かを説明していきます。

1-1 処理をつなげたテストの例

例えば、会議室予約のアプリケーションで、予約したい会議室が空いていれば予約ができ、空いてなければ予約ができない、といった仕様項目をテストするとします。この場合のテストは表1に示した2つが考えられます。

また、「予約は予約日時前であればキャンセルすることができる」という別の仕様項目があれば、表2のような予約した会議室のキャンセルができるテストを行うでしょう。

この2つの表で示したテストケースだけでテストケースは十分でしょうか?

2つの表に示したこと以外に、予約をキャンセルした後に、別の人がその会議室を予約できなければならないはずです。そのような場合は、予約のキャンセル処理を行い、その後に予約実行処理をテストします。

1-2 バージョンアップや派生開発で特に必要になる考慮

2つの処理をそれぞれ違う人が開発している場合、その処理をつなげたテストを行うとバグが出やすくなります。それぞれの処理がお互いに影響を与える考慮が漏れるためです。

また、バージョンアップや派生開発などで、テスト対象に変更が入ったときに、機能自体は変わらないため、変更箇所以外への影響を考えないことがよくあります。

例を挙げてみます。予約した会議室のキャンセル処理に仕様変更が入ったことを想定してください。

	事前条件	アクション	期待結果
テストケース1	会議室が空室の状態	予約実行	予約成立
テストケース2	会議室が予約済の状態	予約実行	予約不可

表1:「予約実行」テストケースの例

	事前条件	アクション	期待結果
テストケース1	会議室の予約日時前	キャンセル実行	キャンセル成立
テストケース2	会議室の予約日時以降	キャンセル実行	キャンセル不可

表2:「予約キャンセル」テストケースの例

例えば、「キャンセルは誤って押してしまう場合があるため、キャンセルした後の5分以内であれば、キャンセルを取り消すことができる」という仕様変更が入ったとします。この場合、5分以内にキャンセルが取り消せることをテストしなければなりません。しかし、それだけでは不十分です。例えば以下のようなことが考えられます。

- キャンセル後の5分間は他の人が予約できないこと
- 5分過ぎれば他の人が予約できること
- キャンセルを取り消せば5分後でも他の人は会議室の予約ができないこと

1-3 処理をつなげたテストの技法

同値分割、境界値分析、デシジョンテーブルといったテスト設計に使う技法があります。処理をつなげて順番に行うためのテストを設計する技法としては、状態遷移テストがあります。状態遷移テストは状態の変化に着目し、状態が変わるイベント(イベントは処理であることが多い)をつなげてテストしていく方法です(図1)。

状態遷移テストでは、予約実行と予約キャンセルの2つの処理を順番に実行するテストケースの抽出が、何を状態にするかで変わります。

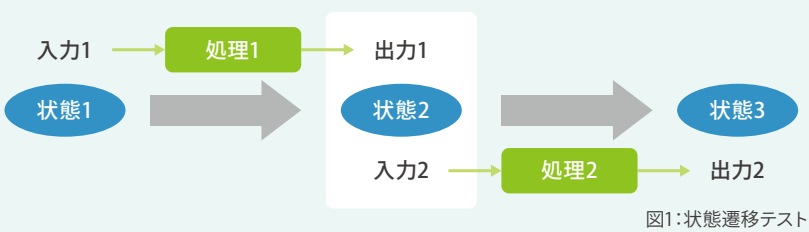


図1:状態遷移テスト

会議室予約アプリケーションの例で言えば、会議室を状態とする場合にはすぐに見つかりますが、予約画面、キャンセル画面といった画面を状態とする場合はその画面遷移がつながっていないとキャンセルした後に予約をするというテストケースが見つからない可能性があります。また、状態の変化を全てテストするとなると、予約とキャンセルだけに着目したテスト以外もカバレッジ対象になってしまい、テストケースが無駄に増える可能性もあります。

状態遷移テスト以外の処理順序を組み合わせるテスト技法として、データを介してつながる処理を組み合わせるという考え方を提案しているのが「順序組み合わせテスト」です。同じデータを介して複数の処理がつながるところに着目し、そこだけを網羅するという考え方です。

2. 順序組み合わせテスト

それでは、「順序組み合わせテスト」の説明をしていきます。「順序組み合わせテスト」では、処理と処理がデータを介しているかどうかに着目します(図2)。

会議室予約の例で言えば、このアプリケーションが持つデータベースには、会議室というエンティティがあり、処理をする際にそのエンティティに対して新しいデータ(複数の情報である可能性もある)を登録したり、そのデータの内容を書き換えたり、そのデータの内容を参照したりします。

バージョンアップや派生開発の場合は、変更対象になる処理があります。その場合は変更対象の処理がアプリケーション内部に持っているデータに対して影響を与えている

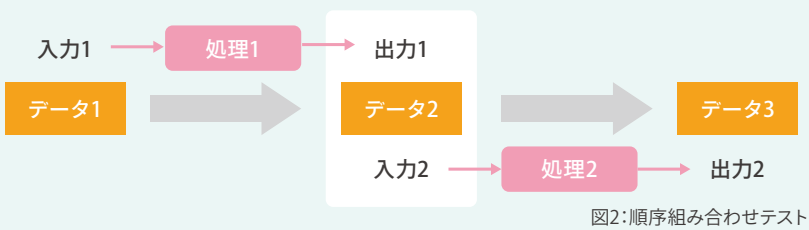


図2:順序組み合わせテスト

ときに、そのデータを介して行われる処理を抽出して、順序を組み合わせてテストを行います。

2-1 「IDAUカバレッジの定義」

「順序組み合わせテスト」は、処理順序を組み合わせる際に、データに着目し、そのデータフローをテストしていることになります。一般的なデータフローテストのカバレッジ基準である全使用法の定義は「データを定義したすべての場所から始まり、データを使用するすべての場所に至るまでのパスセグメントの最低限一つを含むテストケース」となります。この定義を変更対象の処理とデータを介して影響を受ける処理との関係に置き換えて「変更タスクにおいてデータの生成および更新があるデータを使用する波及タスクを2つのタスクを実行するまでに経由するルートにかかわらず最低限一つ含むテストケース」というカバレッジ基準にして、波及全使用法（Impact Data All Used：IDAU）と命名しました。（変更タスクと波及タスクについては後述します。）

2-2 順序組み合わせテストのテストベース

ルールの説明の前に、この技法の入力情報として必要なテストベース（テストケースの素材となる情報の種類）を図3に示します。

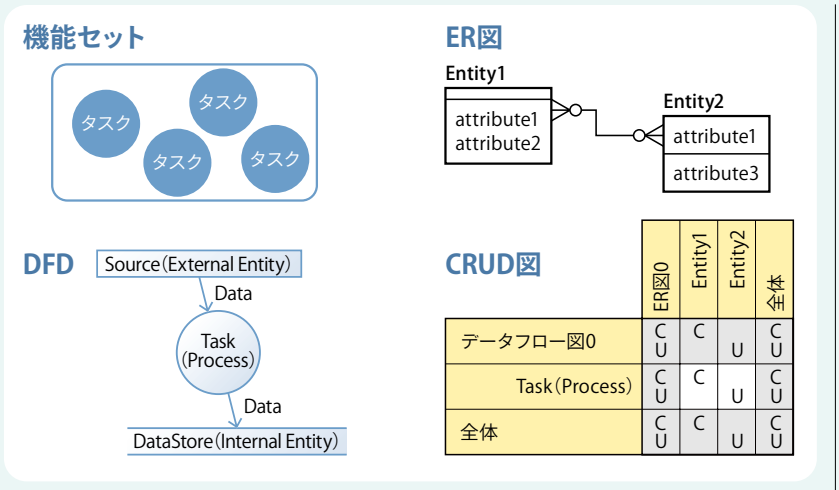


図3：入力情報となるテストベース

機能セット

テストアイテムとなる機能セット（feature set）をテスト対象全体から識別します。機能は複数の処理で実現しています。この処理のことは、以降、タスクと呼びます。テスト設計には、識別した機能セットに該当するDFD、ER図、CRUD図を使用します。

DFD（データフローダイアグラム）

DFDはシステムにおけるデータの流れを表現した図です。DFDの構成要素は、図3に示す通り、源泉（Source）、タスク（Task）、データストア（DataStore）です。テストベースとして用いる場合、テストの範囲はDFDで決まります。

ER図（ERダイアグラム）

※本稿のテストケース抽出の説明には出てきません。

ER図はシステムにおけるエンティティ間の関係を示す図です。DFDでは表現できないエンティティの詳細化やエンティティ間の関係について示しています。

CRUD図（CRUDダイアグラム）

CRUD図とは、タスクからデータストアへの「生成:C」「参照:R」「更新:U」「削除:D」の操作を示した図です。CRUD図から、DFDとER図では表現されていない、タスクによるエンティティへの操作を知ることができます。ここでは、タスクがデータストアに対して行う操作を特定するためにCRUD図を用います。

2-3 順序組み合わせテストのテスト設計方法

「順序組み合わせテスト」は、DFDでわかる外部からのデータに対するタスクの処理順序と、その時のデータの操作をCRUD図から特定して、テストが必要な順序組み合わせを見つけていきます。

処理の順序は、変更タスク(つまりテスト対象となるメインのタスク)でのデータに対する操作を最初に行い、波及タスク(同じデータを使って処理をするタスク)を次に操作するという順番で組み合わせます。このときの組み合わせのカバレッジ基準を決めており、表3に示す内容で組み合わせのテストを行うかどうかを決めることにしています。

「○」を付けた組み合わせはIDAUカバレッジを網羅するためにテストが必要な組み合わせです。「 - 」を付けた組み合わせは、データストアを介した影響が生じないため、IDAUカバレッジの対象としません。「×」を付けた組み合わせは仕様上ありえない組み合わせであり、ありえないことの確認は「順序組み合わせテスト」技法でなくともテストできるため、IDAUカバレッジの対象としません。

「順序組み合わせテスト」によるテスト設計は、以下の順序で行っていきます。

- 変更タスクの特定
- 波及タスクの特定
- 順序組み合わせの特定

これらを組み合わせたものをテストケースとして仕立てていきます。変更タスクと波及タスクの特定に関してもう少し詳しく書いていきたいと思います。

2-3-1 変更タスクの特定

機能セットから変更タスクとそのデータストア(変更に影響を受けるデータの格納先)を特定します。会議室予約アプリケーションの例で示したキャンセル取消処理(「キャンセルした後の5分以内であれば、キャンセルを取り消すことができる」)の場合は、キャンセル取消が変更タスクに該当します。またキャンセルタスクもキャンセル後に取り消しができるような変更が入る上に、キャンセル後5分が経過したかという判断をするタスクも変更タスクとして追加になるでしょう。そして、それらは会議室というデータを更新します。図4が変更タスクを特定したDFDになります。

2-3-2 波及タスクの特定

変更タスクで操作するデータに対して操作をするタスクがどれになるかは、CRUD図を参照することで見つけることができます。その中から前述した表3のタスク間のデータ共有組合せパターンに該当する組み合わせを波及タスクとして特定します。会議室予約アプリケーションの例では、変更タスクであるキャンセル、5分チェック、キャンセル取消の全てがデータの更新をしており、かつ予約実行も会議室のデータを更新します。変更タスクがU:更新の操作を行い、後続するタスクもU:更新を行う場合は、波及タスクとして組み合わせ対象になるため、予約実行が波及タスクに該当します。図5は、波及タスクも含めたDFDになります。(予約実行同様、U:更新を行うキャンセルとキャンセル取消も、波及タスクになります。)

		変更タスク			
		C:生成	R:参照	U:更新	D:削除
波及タスク	C:生成	×	×	×	○
	R:参照	○	−	○	−
	U:更新	○	−	○	×
	D:削除	○	−	○	×

表3:タスク間のデータ共有組合せパターン

2-3-3 順序組み合わせの特定

変更タスクと波及タスクが明らかになったので、組み合わせを行います(表4)。

会議室予約アプリケーションの例では、5つの処理の順序組み合わせを特定できました。IDAUカバレッジとしては、この5つをテストすれば、全て網羅したことになります。以降、テスト実行ができるようにこれらをテストケースにしていきます。

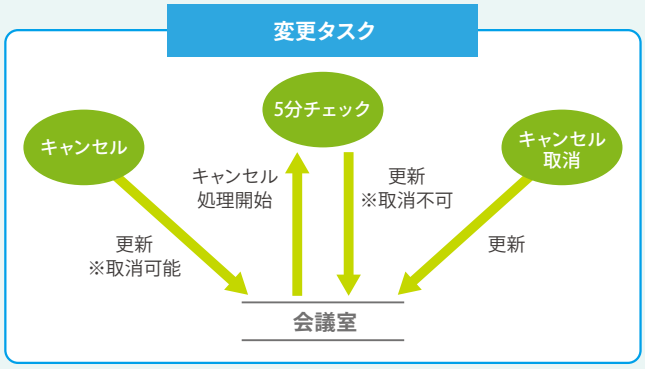


図4：変更タスクの特定

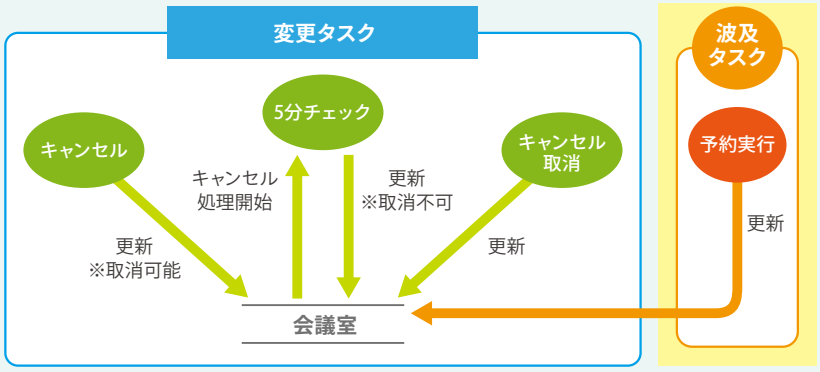


図5：波及タスクも含めたDFD

No	変更タスク	波及タスク	テスト対象の仕様項目
1	キャンセル	予約実行	キャンセルの後の5分間は他の人が予約できないこと
2	キャンセル	キャンセル取消	キャンセルを取り消しできること
3	5分チェック	予約実行	5分過ぎれば他の人が予約できること
4	キャンセル取消	予約実行	キャンセルを取り消せば5分後でも他の人は会議室の予約ができないこと
5	キャンセル取消	キャンセル	キャンセルを取り消した後は、またキャンセルができること

表4：変更タスクと波及タスクの組み合わせ

おわりに

「順序組み合わせテスト」は、データを介す処理同士をつなげてテストケースにしていく技法であるため、例で示したようなデータベースを内部に持つことが多い業務系の情報システムがより適しています。そのため、組み込みソフトウェアのような制御専門のソフトウェアでは、あまり適用の効果が出ません。ただ、昨今の組み込みソフトウェアは、IoTのように、デバイス同士がつながって利用者に価値を提供するようなシステムに組み込まれることが多くなり、そうなると必ずデータを介した処理が出てくるため、この技法が活用できる状況は多くなっていくのではないかと思っています。

「順序組み合わせテスト」は、大学院での研究を通して、テストベースやテストケース作成手順やカバレッジ基準を定義していき、理論的に明確なものになりました。今後、工夫をして、さらに有用にしたいと考えています。そのための現場適用や、適用結果の研究も今後進めていきます。本稿を読んで現場に適用した方がいたら是非、情報共有をさせてください。それによって、この技法をより有用なものにしていければと思います。

本記事の内容に関するご質問、お問い合わせは、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。

工程実施の質を可視化するデファクトスタンダード

～不具合を残存させる「やり方」を正すODC分析～

杉崎 眞弘氏
すぎさき まさひろ

SUGIシステムズエンジニアリング
代表

日本IBM株式会社研究開発部門にて、主に中小型システム、PC、組込みシステムを担当する品質保証部門長を歴任し、品質検証技術、開発プロセス改革の社内展開を主導。後年そうした品質技術を基にコンサルティング部門を立ち上げ、国内企業への品質改善技術の普及に努めてきた。
現在、コンサルティング事業の傍ら、日本科学技術連盟ODC研究会運営委員、一般社団法人UX設計技術推進協会理事を務めている。

はじめに

皆さんは、日々のソフトウェア開発において、開発プロセスの実施の質すなわち「やり方」の妥当性、適格性、十分性について考えたことがありますか?

ソフトウェア開発において、不具合を検出・修正していくことが品質向上に貢献していると信じられてきました。確かに不具合を修正して仕様通りの動作を確認することは、品質検証作業として必要なことではあります。

では、毎度、プロジェクトの終盤で残不具合と格闘せざるを得ないのは、致し方のないことでしょうか?

もっと早い段階で手を打つことができなかったのでしょうか?

突き詰めると、不具合を残存させてしまう「**やり方**」をしているのではないかという考えに行き着きます。

各工程で摘出された不具合の含有要素を分析することで、工程実施の質が「見える化」でき、必要な改善アクションが示唆できるという気付きから考えられたのが、ODC分析(Orthogonal Defect Classification)です。

本稿では、ODC分析のコンセプトとその有効性、および展開活動について紹介します。

01 よくある判断基準

プロジェクトにおいて、テスト工程も終盤になると、不具合の出方が気になります。摘出した不具合数を時系列にプロットしたものが使われていると思います。そして、次のような

状況の時、「そろそろ完了でしょう」という声が聞こえてきます（図1）。

- テストケースをすべて消化した。
- バグを全部修正した。
- そろそろリリース日だ。

ビジネスでやっていることなので、正当な判断だと思います。

しかし、次のようなことがわかったら、その判断は揺らぎませんか?

図2が示していることは、「機能に関わる不具合の比率が、工程を追う毎に相対的に増加している」ということです。

このことは、**「機能的にまだ不安定な状況で、今後も機能に関わる不具合が出続けるリスクがある」**ということを示唆しています。

それを示す理由として、設計レビューなど第1期(設計~コーディング)では、最も多いはずの機能に関する指摘がされず、不具合を残存させたまま第2・3期（単体・機能テスト）のテスト工程で不具合の摘出作業をしていることがうかがえます。

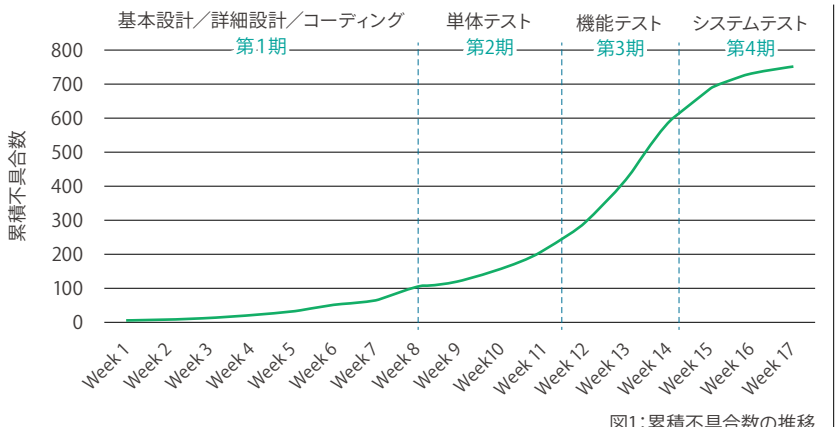


図1：累積不具合数の推移

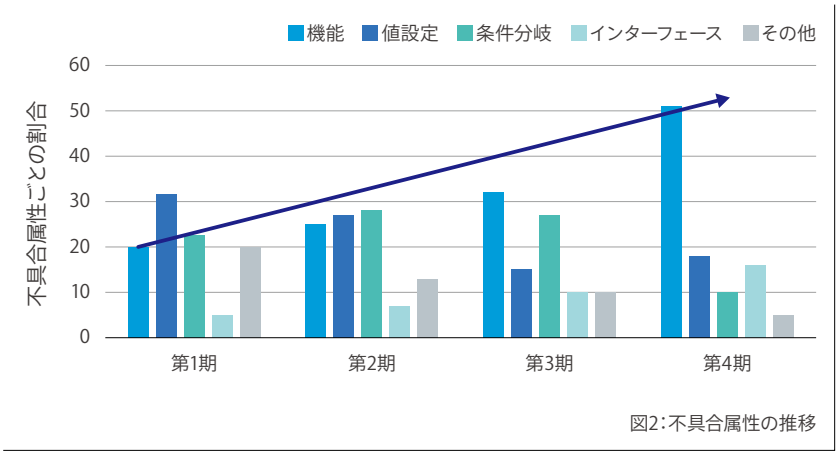


図2：不具合属性の推移

こうした示唆を与えてくれるのがODC分析です。

つまり、ODC分析は、不具合の出方を分析することで、工程内で起きている品質の変化を見える化する手法と言えます。

02 ODC分析の生い立ちについて

ODC分析は、米国IBM社の基礎研究部門であるワトソン研究所（Thomas Watson Research Center, NY）で、ソフトウェア品質について研究をしていたDr.Ram Chillaregeと彼の研究チームから生まれました。同研究チームが、社内の膨大な不具合情報を分析して気付いた不具合と開発プロセスの関係を、1992年にIEEEで論文[*1]として発表し、IBM社内で分析手法化されたものです。

この手法は、長年IBM社内でプロジェクト品質の改善手法として各事業部に普及し成果を上げてきました。近年、社外にもサービスとして展開されるようになりました。公開されている事例として、Bellcore（AT&T）でのプロセス改善への適用事例、またNASAでの品質改善手法としての採用例などがあります。

国内でも私（筆者）自身、IBM時代からコンサルティングのソリューションの一つとして適用しています。

03 不具合について気付いたこと

前述の研究チームが不具合について気付いたことは、大きく次の3つです。

①プロセス実施についての観点

開発プロセスに従ってよく練られた設計や検証を行った場合とそうでない場合では、各工程での不具合の出方に違いがあります。そして各工程の作業の質が後工程への不具合の出方に影響していることに気付きました。

そのことから、前後の工程に求められる改善アクションが示唆されます(図3)。

概して、工程内に不具合が多いのは、「何かおかしい」ことが起こっているというシグナルと考えるべきです。

②不具合属性についての観点

同研究では、不具合を図4のように4つの属性とその相互作用によって生じていると捉えます。

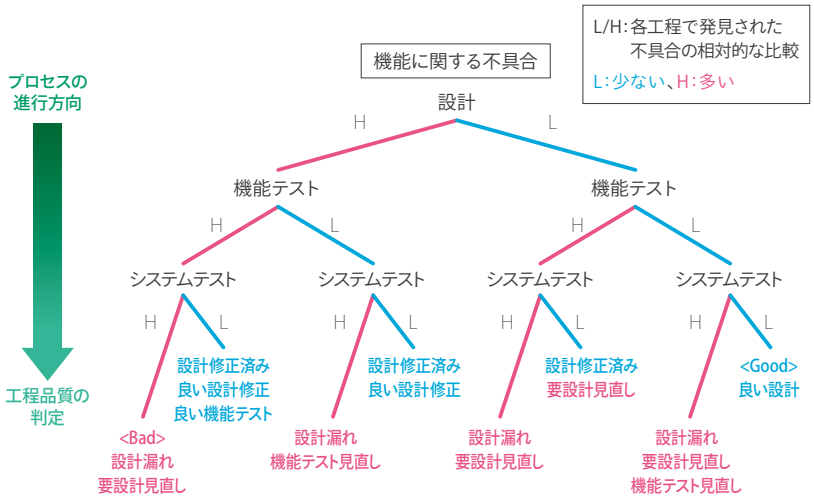


図3:プロセス実施の質による不具合の出方の反応

図4:不具合属性の4つの属性と相互作用

タイプ:不具合の種類の属性でプロセスに関わる不具合(何がどこで修正されたか)を示唆する属性
トリガー:不具合を表面化させた動機の属性
ソース:不具合を含んでいたコードの箇所、工程、ドキュメントの属性
インパクト:ユーザ・顧客への影響の属性(利用時の品質特性)

さらに、ODC分析の各属性には、副属性が定義されています。(図5)

③開発プロセス実施の質とODC属性の関係

開発プロセスに従った実施作業の質とODC属性の関係が導き出されています。

例えば、図6にあるように**タイプ属性**の副属性について、不具合が摘出されるべき工程は、開発プロセスの規定からすでに決まっている(期待されている)という見方です。

逆に言うと、然るべき工程以外での頻繁な不具合の摘出は、「何かおかしい」と判断できます。

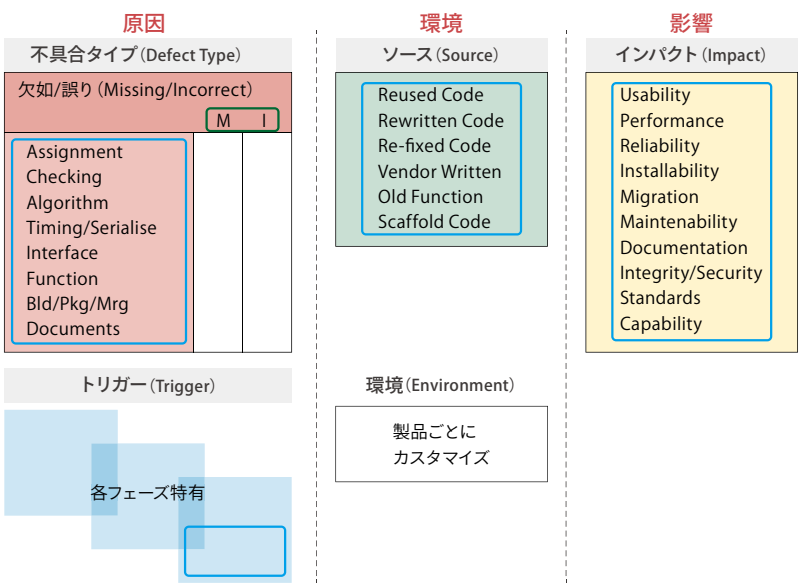
その「何か」とは、開発プロセス規定(期待)と実施作業での「やり方」にギャップがあることを意味します。

- レビューの観点
- テストケース・カバレッジ

などに問題があると考えられ、見直しが必要なことを示唆しています。

また、テスターの知識、経験の差によりテスト範囲に偏りが出る場合が多く、**トリガーの分布**（図7）から、テスターの要員構成には配慮すべきことが示唆されています。

そうした研究チームの気付きから、ODC分析のコンセプトは次のようにまとめ上げられています。



参照：Orthogonal Defect Classification – A concept for In-process Measurement

図5：ODC分析 各属性の副属性

04 ODC分析のコンセプト

ODC分析のコンセプトは、次の言葉に集約されます。

Defect Control is：

A measurement and analysis methodology based on Orthogonal Defect Classification to gain insight and direction for improving software quality.

参照：Orthogonal Defect Classification
– A concept for In-process Measurement

ソフトウェア開発において、その進捗を妨げる主たる要因は不具合です。

その妨げを低減するには、**「不具合を抑制する（Defect Control）」**ことが重要であると考えます。

開発フェーズ／不具合タイプの副属性	基本設計	詳細設計	コード	単体テスト	機能テスト	システムテスト
Assignment（値の設定）			×	×		
Checking（条件分岐）			×	×		
Algorithm（アルゴリズム）			×	×	×	
Timing/Serialize（タイミング/処理順番）		×				×
Interface（インターフェース）		×	×	×	×	×
Function（機能性）	×				×	

図6：タイプ属性の副属性

トリガー ＼ テスターの経験	新人/見習い	製品開発経験	プロジェクト参加経験	複数製品連携経験	熟練/権威
ホワイトボックステスト					
単純パスカバレージ	×	×			
複雑パスカバレージ		×	×		
ブラックボックステスト					
単一機能のカバレージ	×	×			
単一機能のバリエーション	×	×	×		
複雑機能の順序			×	×	×
複数機能の相互作用			×	×	×

図7：トリガーの分布

ここで、「**不具合を抑制する**」とは、この先でも発生し得る同種の不具合を事前に知り、手を打つことで、未然に防ぐ（control：抑制する）ことです。

そのためには、

- まず、不具合を生み出し、残存させている「**やり方**」を発見する。
- その「**やり方**」を改善するためのアクションを策定し、実施する。

ことが求められ、そのための方法論が必要となります。

そこで、ODC分析のコンセプトでは次のように結論付けられています。

「不具合を抑制する（Defect Control）には、ソフトウェア品質改善への洞察と示唆を得るための計測方法と分析方法論が必要である。その基となるのがODC（Orthogonal Defect Classification）での不具合の分類法と意味論である。」

05 ODCの意味

ODC（Orthogonal Defect Classification）という言葉の意味について、参考文献[※1]には下記のように定義されています。

“Orthogonal”という言葉は、「直交」という意味だが、ここでは「お互いに相容れない」という意味で、ODCで分類する4つの属性がお互いに異なる次元のものであるという特徴を表している。

“Defect（**不具合**）”という言葉は、以前から使われていて、「障害」あるいは「欠陥」のことである。

“Classification（**分類**）”という言葉こそ、この議論の核である。

何かを分類（classify）するとは、種類ごとに意味論を持って振り分けることである。

研究チームの成果として、意味論的に不具合を分類（classify）し、分析する事によって、不具合を抑制（Defect Control）することができ、**適用する開発プロセスの改善への示唆が得られる**可能性があることを、探り当てることができました。

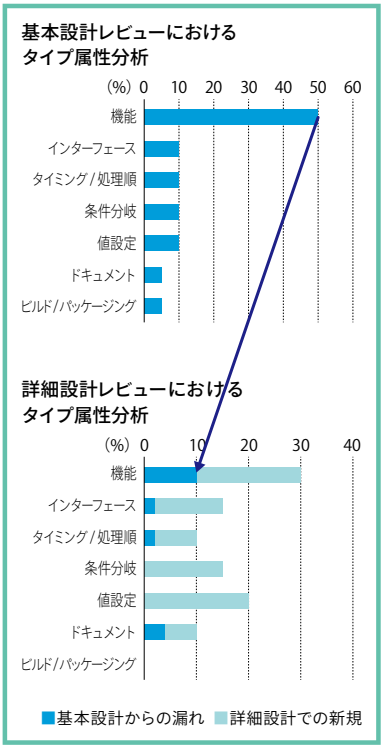
以上のことから、ODC分析の目的は、**開発プロセスの改善**にある、と言うことができます。

06 ODC分析で、何が分かるのか

ODC分析で、不具合を属性ごとに分類・分析すると何が分かるのか、簡単な事例で紹介します。

（1）タイプ属性による分析例

タイプの推移が示唆する工程品質



分析結果

基本設計レビューから、詳細設計レビューにかけて、検出された不具合の内容を比較します。

- 開発工程の初期である基本設計レビューにおいて、集中的に機能に関わる不具合を摘出しているのは、理解できる。

- しかし、次工程の詳細設計レビューでも機能に分類される不具合が大半を占めている。
- そのうちの1/3が前工程の基本設計レビューで摘出されるべき不具合であることから、前工程からの漏れが多発していると言える。

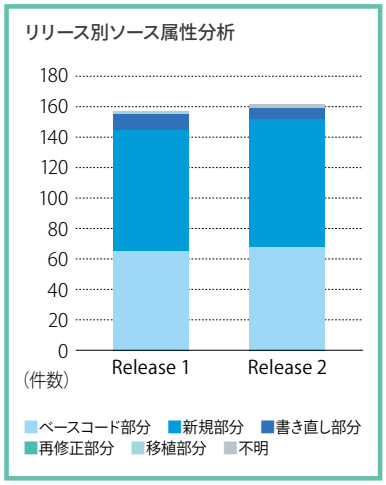
そこから、

- 基本設計でのレビューの観点（特に機能）の見直しが示唆される。
- それに伴う、ドキュメント（機能仕様書）の再レビューの必要性を示唆している。

ということが分かり、改善策の指針となります。

(2) ソース属性による分析例

不具合が摘出された箇所のソース・コードの開発履歴の観点（派生開発の場合）



分析結果

リリース1とリリース2の分析結果を比較します。

- 不具合の摘出量はほぼ同等である。
- しかしながら、ベースコードからの不具合比率が変わらないのは、リリース1の開発時からベースコードの検証が不足していたことが示唆される。

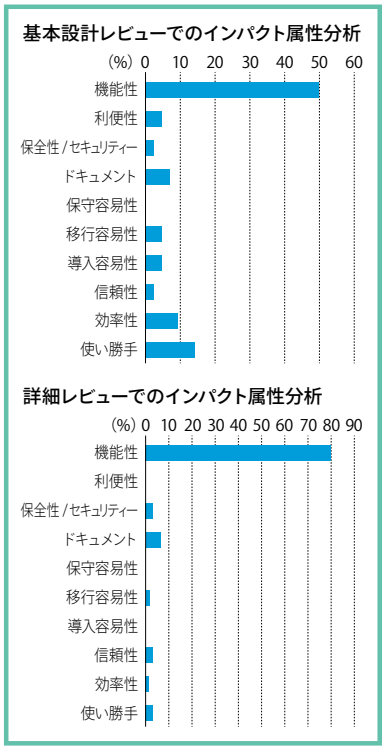
よって、

- 新規機能は注目されるが、ベースコードについても、同じ仕様で妥当かどうか検証すべきである。

と、いうことが分かります。

(3) インパクト属性による分析例

お客様への影響度合いの観点



注）この事例で適用しているインパクトの副属性は、ODC固有の定義であり、ISO25010などの品質特性定義とは異なります。

分析結果

基本設計と詳細設計での設計レビュー時の指摘事項をインパクト副属性に基づいて分析します。

- 機能性に関する検討はよくなされている。
- しかし、設計レビューが機能性に集中し過ぎてシステムの信頼性、性能、あるいは導入容易性、使い勝手など利用時の品質への検討に欠けていることを示唆している。

結果として、

- 設計メンバーのみのレビューでとかく起こりやすい傾向で、ユーザー寄りの視点を持ったレビューアを参加させるべきである。

ということが示唆されます。

以上、3つの属性についての簡単な事例を示しましたが、実際のプロジェクトではさらに具体的な示唆になる場合が多く、また属性を組み合わせて、工程内検証の目的の適合性、妥当性を見ることができます。

07 分析結果が示唆するもの：なぜそう言えるのか

ここで改めて、分析結果の評価を行う際の基本的な見方を紹介します。

ポイントは、適用する開発プロセスの特性の理解にあります。ここではこの特性をプロセス・シグネチャーと呼びます。

開発プロセスは、この通りすれば正しく物が作れると考えられているはずで、そこには工程の定義と目的があるはずです。

それに従って、その工程で主眼として摘出すべき不具合の分布は導き出されます（プロセスの期待）。

例えば、プロセス・シグネチャーとODC分析の**タイプ属性**の分布の関係を事例で示します（図8）。

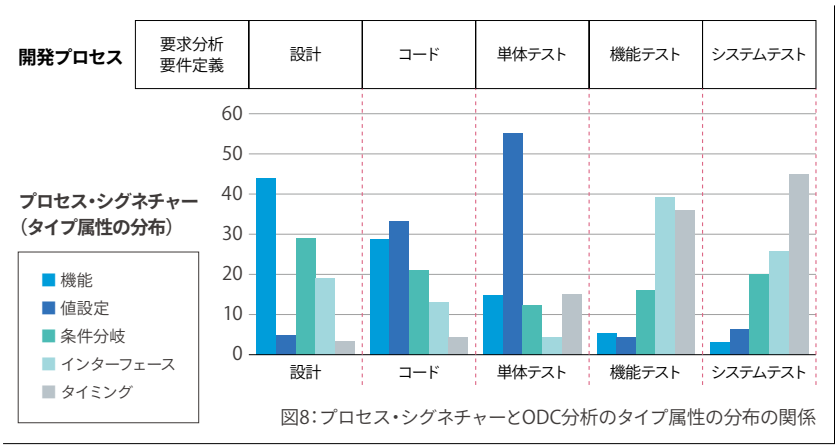


図8：プロセス・シグネチャーとODC分析のタイプ属性の分布の関係

注）この事例は、ある特定の開発プロセスに基づいて策定したプロセス・シグネチャーの事例で、全ての開発プロセスに適合するものではありません。適用する開発プロセスの特性を考慮して、個別にプロセス・シグネチャーを策定する必要があります。

プロセス・シグネチャーは、

- 開発プロセスに依存した固有の期待（こうあるはず）である。
- シグネチャーを定規として、現状に差異がある場合、「何かおかしい？」と判断できる。
- その「何かおかしい？」を分析することで、そこから示唆される改善策「何をすべきか」を見つけることができる。

といった考え方です。

これが、**ODC分析のポイントです。**

08 ODC分析と筆者の関わり

1992年に前述の研究チームの論文[*1]を基に、分析手法化してIBM社内で展開をしようというタスクが始まり、Dr.Ram Chillaregeから各国IBM研究所への召集がかかり、日本からは私（筆者）が参加しました。このタスクでは、コンセプトの共通理解と分析の標準化、社内教育のための講師教育が行われ、修了後、各国自部門への展開に努めて来きました。

09 ODC分析の適用例（私のやり方）

これまで私自身、ODC分析を、電機、デジタルコピー機、医療機器、重工業、携帯機器などのメーカー様にて実施してきました。そのやり方は、次の手順です。

1）お客様の開発現状の分析・理解のため、ODC分析を適用し、そこから示唆される改善施策を協議して策定します。

多くの場合、テスト期間での不具合情報を基に始め、工程移行の判定妥当性を見ていきます。

2）改善施策に沿って開発プロセス、要求分析、設計レビュー、テスト計画・カバレッジ可視化など、「やり方」への改善実施を、実プロジェクトを通して行います。意味合いをチーム内での共通理解を図りながら進めます。

3）「やり方」への改善実施の過程でもODC分析を実施し、「やり方」の妥当性・的確性・十分性の推移を検証し、効果を確認していきます。

その結果、多くのプロジェクトでは、当初の分析結果と比べて明らかな変化が見られ、QCD改善に効果が現れています。

長年の実績ある手法ながら、IBMとしてODC分析についての書籍を残していませんでした。今日に至るまで興味のある方々が詳しく知りたくても、断片的な情報しかなく、手探りで試行錯誤されている方々が多いのが現状ではないかと認識しています。

そこで、ODC分析のオリジナルを学び実践してきた者として、その知識を広く普及を図ろうと、次のような趣旨で発足した「ODC分析研究会」を通して普及活動を行っています。

10 ODC分析研究会の発足と活動

高い品質のものづくり文化を根底から支える品質管理や品質保証の分野において、検証結果の不具合分析は重要な活動であると考えています。不具合分析は多くの企業や組織で取り組まれていますが、統一された分析手法が定着していません。「不具合」そのものでなく不具合の除去技術や製品の品質評価方法などが組織の壁を越えて議論されにくくなっています。

そこでODC分析に関係する用語や不具合を分類する属性を統一し、共通の言語で品質を語ることができる場として「ODC分析研究会」を一般財団法人 日本科学技術連盟の協力を得て設立しました。
(http://www.juse.or.jp/sqip/odc_workshop/index.html)

ODC分析および研究会に興味がありましたら、問い合わせください。

ここでは、企業や組織の枠組みを超えた不具合分析や品質評価方法の議論と研究を行っています。ODC分析研究会の成果として、教育のための資料や文献、教科書などを提供していきます。

参考文献

*1: Orthogonal Defect Classification - A concept for In-process Measurement
Ram Chillarege, Inderpal S. Bhandari, Michael J. Halliday, etc.,
IBM Thomas J. Watson Research Center IEEE Transactions on Software Engineering
Vol .18, No. 11, Nov. 1992©IEEE

*2: Experiences with Defect Prevention R. Mays, C. Jones, G. Holloway, and D. Studinski
BM Systems Journal, Vol. 29, No. 1, 1990

本記事の内容に関するご質問、お問い合わせは、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。

はじめに

SAP ERPは2025年にメインストリームの保守サポート期限が到来するため、引き続きSAPシステムを利用するためにはSAP S/4HANAへ移行する必要があります。そのため、これから2025年までの間に国内で2000社と言われるSAPユーザー企業の多くがSAP S/4HANAへの移行プロジェクトを立ち上げると見込まれています。

当社では、このようなSAPプロジェクトを抱えているお客様に向けて、

①PMO品質管理支援

②テスト業務支援

③自動化支援

を柱とするSAPシステム検証ソリューションを2019年5月より開始しました。本稿ではこのSAPシステム検証ソリューションについてご紹介します。

SAPシステム検証ソリューションのご紹介

SAP S/4HANA移行プロジェクトについて

SAP S/4HANA移行の前提条件は以下の3点です。

① SAP ERP6 EHP7以上

② UNICODE化

③ DBMSのSAP HANA化

SAP S/4HANA移行については、移行手順や保守サービス利用などの対応方法により、図1の通り4方式があります。

移行に向けたSAPユーザー企業の動きとして、早期にSAP S/4HANA移行を実行する企業は、業務改革を伴うプロジェクトとして実施することが多く、その場合は業務を大きく変革することになるため、図2のように方式1（New Implementation）を採用する企業が多くの比率を占めています。

現状では、大半のSAPユーザー企業がまだSAP S/4HANA移行の検討段階にあります。これから2025年に向けて移行プロジェクトを立ち上げていきますが、それらの多くは、現状の

業務に大きな変更は加えずにSAP S/4HANAへ移行すると思われます。その場合は、方式2（Conversion（1Step））を採用することになります。なお、DBMSのSAP HANA化がすでに完了しているSAPユーザー企業は、方式3（Conversion（2Step））でSAP S/4 HANA移行を進めることになります。

また、検討の結果、SAP S/4HANA移行を行わず、第三者保守サービスを活用して現行のSAP ERPを継続利用する方式4を採用する企業もSAP S/4HANA移行期限の2025年に近づくほど増加してくると思われます。

SAPシステム検証ソリューションの内容

当社のSAPシステム検証ソリューションは、SAPプロジェクトにおけるQCD向上に貢献するサービスで、以下の支援業務から成ります。

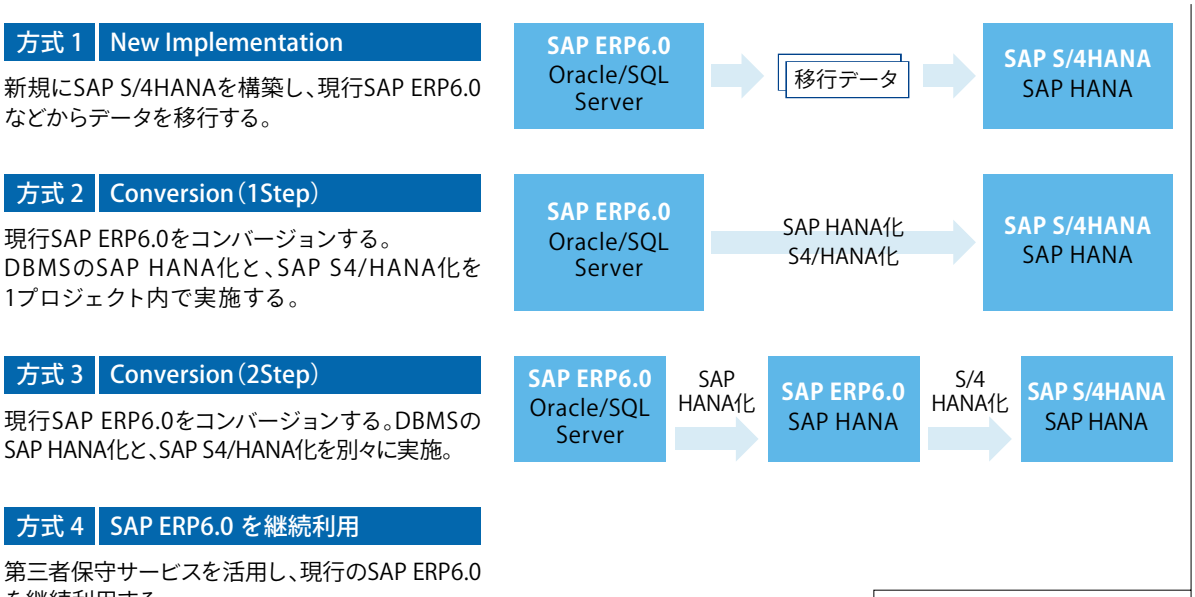
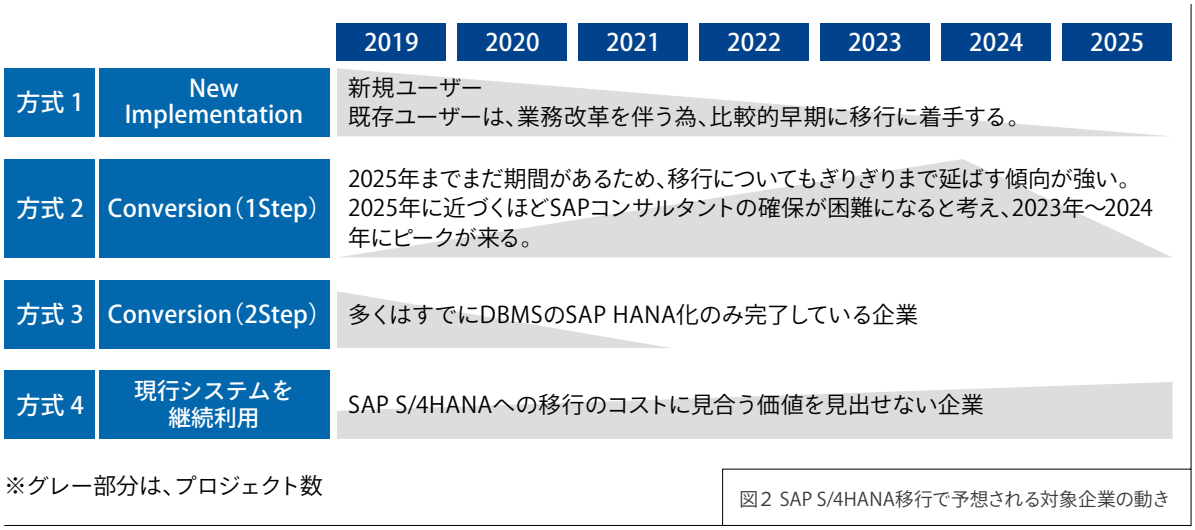


図1 SAP S/4HANA移行方式の種類



※グレー部分は、プロジェクト数

図2 SAP S/4HANA移行で予想される対象企業の動き

PMO品質管理支援

品質管理に高度なスキルを持つ当社の”品質管理コンサルタント”が、お客様のSAPプロジェクトのPMO品質管理チームに参画し、品質管理・品質評価・不具合解析などを支援します。

品質課題をより上流工程で早期に解決することにより、プロジェクトを円滑に推進します。

テスト業務支援

SAPプロジェクトやSAPシステム運用保守における実績を持つ当社の“SAPコンサルタント”が、SAPプロジェクトの上流工程より参画し、テスト方針・テスト計画の策定などを担当します。また“SAPコンサルタント”として、要件定義・ビジネス設計・プロトタイプ検証なども担当します。テストフェーズでは、当社の“SAPコンサルタント”を増員し、テスト設計やテスト実行を担当します。

SAPシステム自動化支援

SAPシステムの自動化を検討するお客様に対して、自動化スコープの策定、自動化環境構築、自動化開発および実行を支援します。機能テスト・シナリオテスト・権限テストなどのテスト実行に限らず、SAPシステムのカスタマイジングやマスタデータ登録も対象としてプロジェクトにおける最適な自動化スコープを選定することにより、工数削減や品質向上、負荷の平準化によるプロジェクトの安定的な遂行といった自動化の効果を最大化します。

自動化ツールはMicrofocus社の以下の製品を使用します。

- Quality Center(以下、QC)
- Business Process Testing(以下、BPT)
- Unified Functional Testing(以下、UFT)

また自動化には、スミテム社のQC自動化開発支援ツール「QC-ACCEL」や「SAPシステム権限テスト自動化フレームワーク」を活用し、効率的な自動化開発を実現します。

SAPシステム検証ソリューションの特長

プロジェクトのQCDを向上

当社の品質やテストに関する高度なスキルにより、より上流工程から品質を作り込み、プロジェクトの円滑な運営を促進し、QCDを向上します。

ユーザーやSIベンダのプロジェクトマネージャー・SAPコンサルタントの負荷軽減(図3)

品質課題の解決を促進し、お客様のプロジェクトマネージャーやSAPコンサルタントの負荷を軽減します。削減工数を別プロジェクトに充てれば、お客様のプロジェクトマネージャーやSAPコンサルタントが、より早期に別プロジェクトへ移行することができます。

自動化効果の最大化

人手による作業のばらつきや人的ミスを抑制し、属人性を排した高いテスト品質を確保します。

自動化は、テストの実行工数を削減します。特にスケジュールが圧迫されがちなテスト工程の負荷を平準化し、安定したプロジェクト運営を実現します。

自動化開発には、QC・BPT・UFTによる自動化に特化した自動化開発支援ツールを活用し、自動化開発工数を削減するとともに、保守性の高い自動テストを構築します。

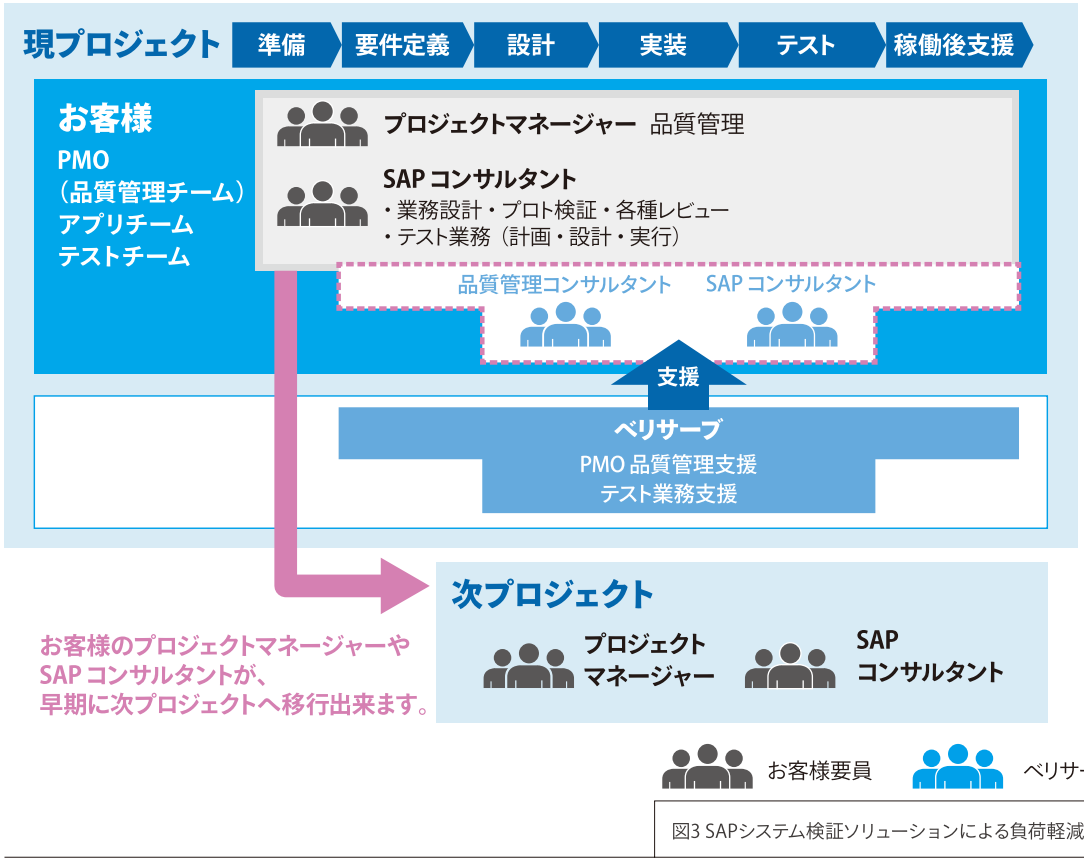


図3 SAPシステム検証ソリューションによる負荷軽減のイメージ

おわりに

「品質の作り込みは上流工程から」というのが鉄則です。品質管理を疎かにし、要件定義で大きな漏れや齟齬に気が付かないままプロジェクトを進行し、下流工程でそれに起因する重大な問題が生じた場合、手戻り工数によって、スケジュールが遅延し、コストも膨らんでしまうことがあります。上流工程から品質を作り込み、次工程への不具合流出を抑えることによって、下流工程での手戻りを軽減し、品質向上にかかるコストを最小化することができます。これを機会に是非、当社のSAPシステム検証ソリューションをご活用ください。

詳しい内容は、以下のお問い合わせ先までご連絡ください。

お問い合わせ先 株式会社ベリサーブ
TEL：050-3640-7523（ITアライアンス推進部） 052-325-5010（中部支社） 06-6223-6110（西日本支社） URL：https://www.veriserve.co.jp/

品質を創造する企業

Veriserve Navigation 2019年12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