ソリューション技術通信　ベリサーブナビゲーションでは、さまざまな分野で検証に携わるベリサーブが「今とこれから」を皆様にお届けします。

VERISERVE NAVIGATION

2018
July
Vol.14

巻頭特集

テストと保証
～品質保証サービス変革への提言～

山本 修一郎 氏

巻頭特集

テストと保証
～品質保証サービス変革への提言～

山本 修一郎氏
やまもと しゅういちろう

国立大学法人 名古屋大学
大学院情報学研究科 教授

1. はじめに

本稿では、テストと保証の関係について考えます。まず、エンタープライズアーキテクチャの概念を用いて、テスト項目を体系的に整理する方法をご紹介します。次に、保証ケースを用いて品質評価業務を高信頼化することにより、テスト業務を変革する方法をご紹介します。

2. テスト項目を体系的に整理する方法

まず、本章に登場する用語について説明をしたのち、整理の方法について説明を行います。

エンタープライズアーキテクチャ（EA）

エンタープライズアーキテクチャ（EA）とは、企業活動にITを含めた全体最適の考え方に基づき体系化することにより、ビジネス変革を効果的に実現する考え方のことを指します。この思想は欧米を中心に取り入れられ、米国政府、石油業界、保険業界などで幅広く適用されています。

具体的なアプローチとしては、経営意思に立脚した業務プロセスとITシステム体系を下記①～③のアーキテクチャに従って体系化していくことでより良いサービスを実現していきます。

①ビジネス・アーキテクチャ（BA）
②情報システムアーキテクチャ（IA）
③テクノロジーアーキテクチャ（TA）

本章では、このような体系化の考え方を利用してテスト項目を整理していきます。まず、EAの各階層を行（横軸）とし、疑問詞（5W1H）を列（縦軸）とするマトリクスに一般的なテスト観点を割りあてたものをテストの参照フレームワークと呼びます。テスト参照フレームワークを表1に示します。

BA行では、業務の中で、システムがどのように利用されるかという観点で実施するテスト対象を整理しています。IA行では、ソフトウェアがどのように実現されているかという観点で実施するテスト対象を整理しています。TA行のテストではシステムが実行される技術環境の観点で実施するテスト対象を整理しています。

このように、テストをEAの各階層と疑問詞（5W1H）で体系的に整理することで、テスト設計を効率化できるだけでなく、テスト業務知識の効果的な分類や、テスト対象の重複を削減できる可能性があります。また、この2次元のテスト参照フレームワークに、業務分野ごとに整理する次元を追加すれば、3次元のテスト参照フレームワークを構成することができます。例として、組み込みシステム分野の共通例題の中の、危険運転警告機能のテストを取り上げてみます。
（以下、共通例題より抜粋）

【概要】
自転車を対象に、走行状況センシング、環境走行センシング、危険運転警告、事故回避などの機能により自転車事故を防止するシステムを考える。
【主な機能】
危険運転警告機能
- 上記のセンシングデータを活用して、マイコンユニット（組込みソフトウェア）で危険運転を判別し、運転者ならびに周囲の歩行者、自動車などに警告する。
  警告は運転者ならびに自転車周囲者に適切なタイミングで適切な間隔で断続的に行う。
- 危険運転としては速度超過、ふらつき運転、急ブレーキ連続運転などを考える。
  ただし、これらは、自転車の走行条件ならびに走行環境条件などによって、多様な判断が必要となる。

このような機能に対して、どのようにテストを構成すればいいのでしょうか？

テスト参照フレームワークを用いれば、1からテストの観点を考えることなく、包括的なテスト項目を構成することができるようになります。（表2）

この章では、EAアーキテクチャの考え方をベースとした2次元のテスト参照フレームワークの紹介と、その適用例について説明を行いました。テストをEAの各階層と疑問詞（5W1H）を用いて体系的に整理することで、テスト観点を効果的に表現することが可能となり、包括的でかつ、効率の良いテスト設計に利用することができます。

階層	When	Who	What	How	Where	Why
BA	業務契機	業務担当者	業務対象	業務手順	業務環境	業務理由
IA	システムイベント	システム	論理情報	処理手順	システム環境	システム要件
TA	技術イベント	技術要素	物理情報	技術手順	技術基盤環境	技術制約

表 1 テスト参照フレームワーク

階層	When	Who	What	How	Where	Why
BA	危険運転警告	運転者	危険運転	警告認識処理	乗車環境	安全運転基準
IA	危険運転判別	コンポーネント※	センサ情報	運転判別処理	ソフト環境	安全運転要件
TA	危険運転検知	センサ	センサデータ	センサデータ受信手順	車両環境	センサ条件
	危険運転警告	アクチュエータ	アクチュエータ制御信号	アクチュエータ制御手順		アクチュエータ条件

※情報システムの構成要素のことを指す

表 2 危険運転警告機能のテスト参照フレームワーク

3. テスト業務を変革する方法

ソフトウェアの信頼性を向上するために、第三者による検証サービスが日本でも提供されるようになってきています。たとえば、複数の企業から提供される製品間の相互接続テストなどは、製品開発企業ごとにテストするよりも、独立した検証サービス企業が実施する方が経済的と言えます。

本章では、EAの考え方に従い新サービスを開発する仮想的な構築例を紹介します。ここで登場する新サービスは、ITシステム開発部門が開発したITアーキテクチャについて、品質評価部門が「期待される特性を持つことを保証する」というサービスについて取り上げます。

TOGAFとO-DA

第2章で説明を行ったEAを構築するフレームワークの一つにTOGAF(The Open Group Architecture Framework)があります。このフレームワークはグローバルなIT標準団体であるオープン・グループのアーキテクチャ・フォーラム(アーキテクチャ技術部会)が永年にわたり開発してきた、エンタープライズの経営意思に立脚したITシステム体系(ITアーキテクチャ)を策定するための手法およびツールです。このフレームワークでは、戦略レベルでのビジネス計画を基にして、フェーズAより順に策定をすすめていくことで、「ビジネスの思想に基づくITシステムまでを含めた業務プロセス」を具体化していきます。

フェーズA　アーキテクチャ・ビジョン
フェーズB　ビジネス・アーキテクチャ
フェーズC　情報システム・アーキテクチャ
フェーズD　テクノロジー・アーキテクチャ

尚、TOGAFには、初期フェーズや、移行、評価、要求管理などを合わせると全部で10のフェーズがありますが、本章では、サービスの開発に必要となる4つのフェーズを取り上げるため、他のフェーズについては説明を割愛します。

TOGAFを用いてEAを構築する一つの方法としてO-DA(Open Dependability through Assuredness)というものがあります。これは、「保証ケース」という文章化手法を用いて、ステークホルダーと合意形成した状態を保つことにより、プロセスの効率化を図るものです。後ほど説明する新サービスの各プロセスは、O-DAを基にしており、評価依頼部門であるITシステム開発部門との効果的な連携や、知見の可視化を実現します。まず、本文中に登場する、「保証ケース」について説明を行います。

保証ケース

保証ケースとは①~③のようにして、最下位の主張から段階的に最上位の主張を立証する文書化の手法を指します。この手法は主張を保証するための議論の過程が成果物として残るため、ステークホルダーとの合意形成や、議論の可視化に有効であると言えます。

①最上位の「主張」を「戦略」を用いて確認可能な粒度まで論理的に分解する。
②最下位の「主張」を「証拠」により確認する。
③すべての最下位の「主張」を確認する。

例として、「システムが要求を満たすこと」を確認するテストについて記述した「保証ケース」を図1に示します。

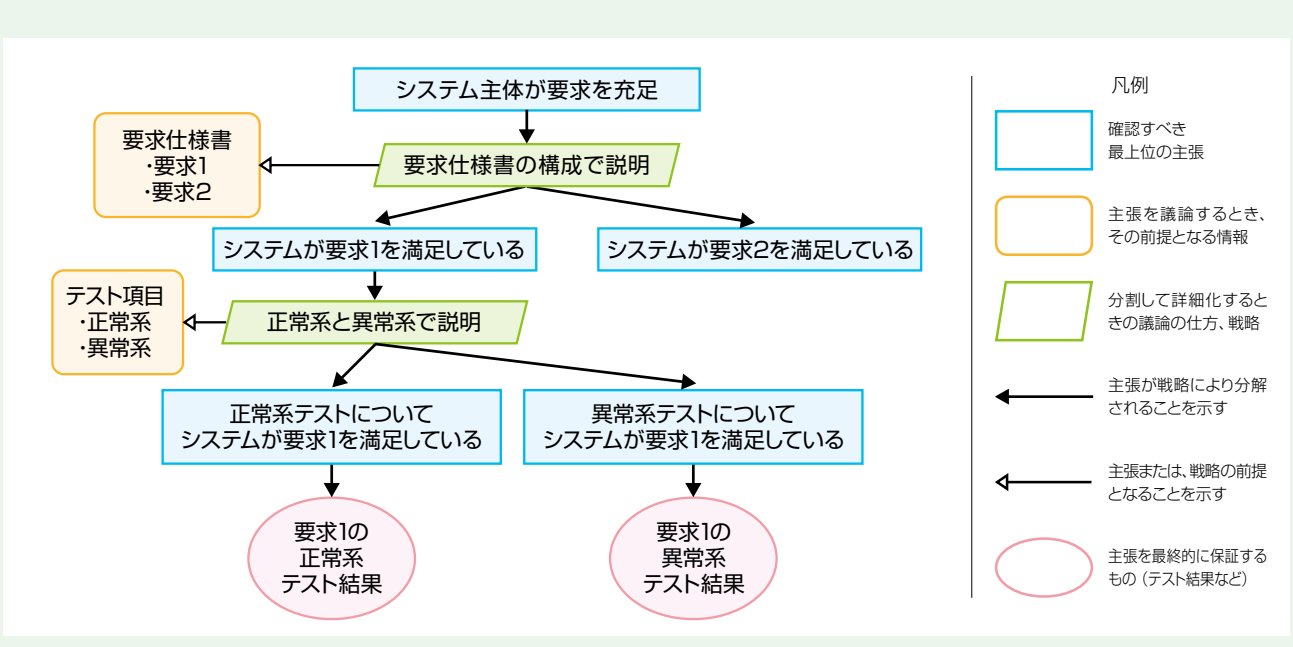


図 1 テストの保証ケースの例

それでは、ここからサービスの構築例についてご紹介します。

3-1. 新サービス検討に至る問題状況

A社の品質保証部の主な業務は、ITシステムの品質保証である。同部門が担当する業務は、ITシステム品質の点検業務である。IT業界では、オープン化の流れにより競争にさらされ、コスト面の競争力が求められている。とはいえ、ITはA社の顧客企業にとって必須の事業基盤であり、高品質のITシステムの提供が最も優先される。そのため、ミドルウェアや既存のアプリケーションをできるだけ再利用するとともに、リスクを確実に把握して対策ができていることを保証できる効率的な品質保証サービスが必要とされていた。

また、ITシステムの品質評価手法は担当者に任せており、有識者のリスク分析知識を可視化できておらず、共有できていなかった。この状況では全体の効率化は難しい。さらに、有識者が人事異動で抜ければ、リスク分析知識は失われてしまう上、最初から未経験者に教育する必要があるが、教育するためにはリスク分析知識が可視化できていなくてはならない。そこで、ITシステムの品質保証を支援するために、保証ケースを利用した「品質保証サービス」を再構築するプロジェクトが発足した。

3-2.「品質保証サービス」の再構築

再構築プロジェクトで作成する新サービスには、EAのフレームワークであるTOGAFで標準化されているO-DAの考え方が適用されました。「既存アプリケーションの品質保証知識を活かした、効率的な品質保証サービスの構築」を方針とし、それぞれのフェーズで検討したことを説明していきます。

フェーズA　アーキテクチャ・ビジョンの策定

アーキテクチャ・ビジョンにおける「アーキテクチャ」とは、新しく開発する「サービス」を指しています。アーキテクチャ・ビジョンの策定フェーズでは、アーキテクチャを作成する目的やステークホルダーの定義、アーキテクチャの概要を作成します。また、ビジネス・シナリオの作成により、ビジネス・情報システム・技術についてのアーキテクチャ策定計画を行います。これらは、企業内のステークホルダーとの合意や、スポンサーの獲得などに利用します。

本章では、アーキテクチャ・ビジョンのアウトプットのうち、「品質保証サービス」についての「アーキテクチャ・ビジョン」と、「ビジネス・シナリオ」をご紹介します。

A-1.アーキテクチャ・ビジョン

<問題状況>

- 有識者のリスク分析知識を共有できていないため、全体効率が悪い。
- 品質評価手法は担当者によりレベルが異なり、評価結果にばらつきがある。
- 有識者の離任により知見が失われる可能性がある。

<目的>

- 品質確認業務効率と評価業務品質の向上を達成する。

<方針>

- 品質確認業務に保証ケースを利用したサービスを確立する。
- ITアーキテクチャ・リスク分析の観点の標準化とデータ化を行う。
- ITアーキテクチャから保証ケースを作成する新しい支援ツールを利用する。

<理由>

- 保証ケースによりITシステム品質を客観的に保証できる。
- 保証ケースによりリスク対策ができていることを説明できる。
- 品質保証結果を再利用することができる。
- 保証ケースによりコミュニケーションミスを防ぐことができる。
- データ化により担当者のスキルによらず、大幅に評価作業時間を短縮できる。
- データ化により経験が少ない担当者でもリスクを適切に検知することができる。
- ツール利用により評価報告書類を自動的に作成することができる。

A-2.ビジネス・シナリオ

<品質保証サービス活用の業務手順>

(A-2-1) ITシステム開発部門が、品質評価を品質評価部門に依頼する。
(A-2-2) 品質評価部門が「ITシステムが指定された品質特性を達成していること」に対するリスクを分析する。
(A-2-3) 品質評価部門がITシステム開発部門に、識別したリスクへの対策内容の提示を依頼する。
(A-2-4) ITシステム開発部門がリスク対策内容を品質評価部門に提示する。
(A-2-5) 品質評価部門がリスク対策によって、品質特性に対するITシステムのリスクが解消できることを確認する。
(A-2-6) 品質評価部門がすべてのITシステムのリスクが解消できることを説明する保証ケースを作成するとともに、ITシステム開発部門に説明する。
(A-2-7) ITシステム開発部門が評価部門から提示された保証ケースについて合意する。

品質保証サービスの将来像に対するビジネス・アーキテクチャ、アプリケーション・アーキテクチャ、テクノロジー・アーキテクチャをArchiMate（アーキテクチャモデリング言語）で記述すると図2のようになります。アーキテクチャ・ビジョンの策定フェーズではこのようなビジネスモデルを作成し、全体的な概要を検討します。

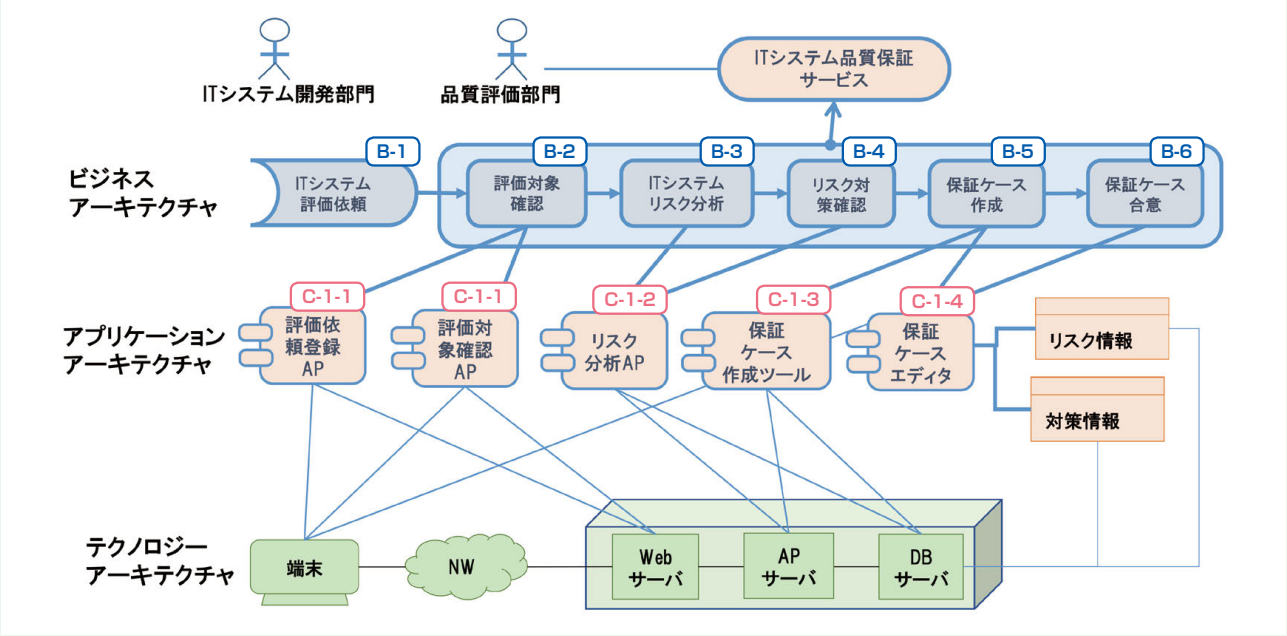


図 2 アーキテクチャ品質保証サービスの導入ビュウポイント 注）ArchiMate 記法

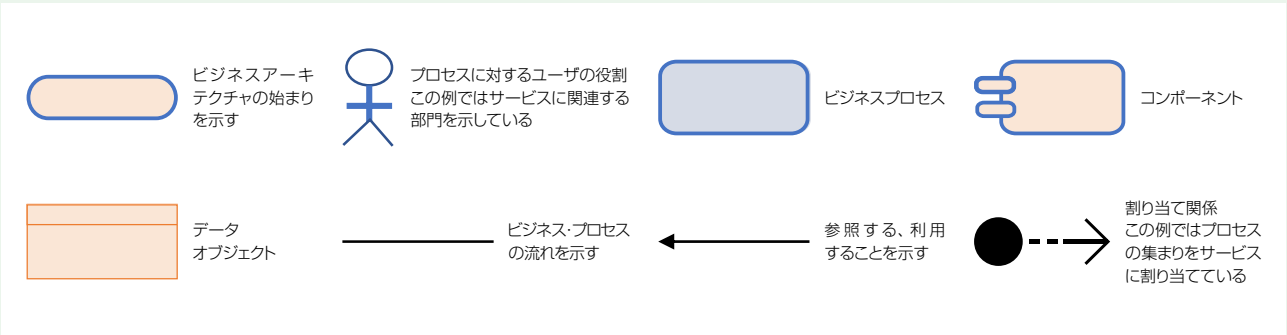


図 2 ArchiMate の凡例

フェーズB　ビジネス・アーキテクチャの策定

ビジネス・アーキテクチャ策定フェーズでは、フェーズAで作成したアーキテクチャの「目的」達成のために、企業がどのように行動するかを明らかにし、ビジネスプロセスや、プロセスの成果物、アーキテクチャに関連する外部のドメイン（この例の場合はサービスを利用する部門）との関係などを確定させます。プロセスを決定したのち、現行と目標とする状態とのギャップ分析を行い、検討要素を整理していきます。

新サービスでは、品質評価部門が「品質評価サービス」をビジネス・サービスとして提供します。このビジネス・サービスに対するビジネス・アーキテクチャは以下のようになります。

(B-1) 評価依頼イベント
(B-2) ITシステムと品質特性の確認
(B-3) 品質特性に対するITシステム・リスク分析
(B-4) ITシステム・リスクへの対策の確認
(B-5) 保証ケースによるITシステム・リスク対策の妥当性の説明
(B-6) 保証ケースによる合意形成

このとき、ビジネス・オブジェクト（ビジネス・アーキテクチャで用いる文書化された成果物）として、ITシステム、品質特性、リスク、リスク対策、保証ケースを用います。

これらビジネス・アーキテクチャのベースに、O-DAフレームワーク（Open Dependability through Assuredness Framework）を用いる場合、目的変化対応プロセスと、障害対応プロセスを考える必要がありました。

また、複数の部門で運用することをO-DAフレームワークは考慮していません。このため、ITシステム開発部門と、品質評価部門の担当範囲について検討した結果、品質評価部門の役割を以下のように定め、品質評価部門はITシステム自体の

リスクだけでなく、ITシステム運用のリスク分析も行うという位置づけにしています。

- ITシステムの品質要求に対するリスク分析を担当する。
- 問題検知、障害未然回避、迅速対応が必要となる障害など、運用上検討が必要となる課題についてリスクとして明確にし、対応を依頼する。
- リスクへの対策を確認する。

現行ビジネス・アーキテクチャと目標ビジネス・アーキテクチャをまとめると表3のようになります。

現行ビジネス・アーキテクチャ	目標ビジネス・アーキテクチャ
● IT システム開発部門からの IT システム品質評価依頼・結果を統一的に管理していない ● 有識者が IT システム品質を個別に評価 ● IT システム評価の観点を整理しているがリスク識別基準は属人的 ● IT システム品質保証基準はない	● IT システム開発部門からの IT システム品質評価依頼・結果を統一的に管理 ● 品質評価プロセスを標準化 ● 評価観点に基づくリスク識別基準を標準化 ● IT システム品質保証基準を保証ケースで標準化

表 3 現行ビジネス・アーキテクチャと目標ビジネス・アーキテクチャの比較

フェーズC　情報システム・アーキテクチャの策定

情報システム・アーキテクチャの策定フェーズでは、フェーズBで作成したビジネス・アーキテクチャを実現するためのアプリケーションについて検討を行います。「品質保証サービス」の場合、品質評価部門が利用するアプリケーション・アーキテクチャは、以下のようなアプリケーション・サービスを利用します。情報システム・アーキテクチャを決定したのち、現行と目標とするアーキテクチャとのギャップ分析を行い、検討要素を整理していきます。

品質保証アプリケーション・サービス

品質保証アプリケーション・コンポーネント(C-1)と品質保証データベース(C-2)を用いて、品質評価部門によるアーキテクチャ品質評価のためのビジネス・プロセスを実現します。

C-1.品質保証アプリケーション・コンポーネント

品質管理サービスを実現するために、下記(C-1-1)～(C-1-4)のアプリケーションを用います。

(C-1-1) 品質保証依頼管理アプリケーション
- →ITシステム開発部門から受付けた評価依頼(評価対象アーキテクチャ・期待される品質特性)を登録して、ITシステム評価を開始する。
- →品質保証が完了したのち、依頼元から了承されたことを登録する。

(C-1-2) ITシステム・リスク分析アプリケーション
- →有識者はシステム構成要素に対する品質特性上のリスクを提示した情報を登録する。
- →品質評価部門がアーキテクチャ情報と期待される品質特性情報に基づいてリスクを抽出する。
- →ITシステム開発部門から入手したリスク対策情報を登録する。

(C-1-3) 保証ケース生成アプリケーション

品質保証依頼管理アプリケーション、ITシステム・リスク分析アプリケーションの情報を基に下記の情報を入力することで保証ケース情報を生成する。
- →ITシステム構成要素
- →ITシステムが満たすべき品質特性
- →ITシステム構成要素が品質特性を満たす上でのリスク
- →リスクへの対策

(C-1-4) 保証ケース・エディタ

保証ケース情報を基に保証ケースを作成する。作成した保証ケースにより、評価対象が期待される品質特性を持つことについて合意する。

C-2.品質保証データオブジェクト

アプリケーション・コンポーネントから利用する品質保証データベースでは、以下の情報を管理する。

- 品質保証依頼
- 品質特性情報
- 品質保証対象である IT アーキテクチャ情報
- IT システム・リスク情報
- リスク対策情報
- IT システムの品質保証ケース情報

現行アプリケーション・アーキテクチャと目標アプリケーション・アーキテクチャをまとめると次ページの表 4 のようになります。

現行アプリケーション・アーキテクチャ	目標アプリケーション・アーキテクチャ
●IT システム開発部門からの IT システム品質評価依頼・結果の管理が情報化されていない ●IT システムのリスク分析が情報化されていない ●リスク対策知識が情報化されていない ●IT システム品質保証が情報化されていない	●IT システム開発部門からの IT システム品質評価依頼・結果の管理を情報化 ●IT システム品質評価プロセスを情報化 ●IT システム評価観点に基づくリスク識別基準を情報化 ●IT システム品質保証基準を保証ケースで情報化

表 4 現行アプリケーション・アーキテクチャと目標アプリケーション・アーキテクチャの比較

フェーズD　テクノロジー・アーキテクチャの策定

テクノロジー・アーキテクチャの策定では、フェーズCで作成した情報システム・アーキテクチャを実現するための技術要素について検討を行います。テクノロジー・アーキテクチャを作成したのち、現行アーキテクチャとのギャップ分析を行い、検討要素を整理していきます。

「品質保証サービス」ではテクノロジー・アーキテクチャ実現のために以下の技術要件を満たすことが必要であることがわかりました。

●評価担当者がWebクライアントから、社内ネットワークを介して品質保証サービスのWebサーバにアクセスする。
●Webサーバは、サーバ上の評価依頼管理機能、リスク分析機能、評価結果登録機能を提供する。
●データベース・サーバでは、評価依頼情報、ITシステム情報、品質特性情報、リスク情報、リスク対策情報、ITシステム品質保証ケース情報を管理する。

現行テクノロジー・アーキテクチャと目標テクノロジー・アーキテクチャをまとめると表5のようになります。

これまでにご紹介したように、EA構築のフレームワークに従い、ビジネスアーキテクチャ、情報システムアーキテクチャ、テクノロジーアーキテクチャを体系立てて順に検討を積み重ねることで、ビジネスの思想に基づき、なおかつITシステムまでを考慮に入れたプロセスの全体像を構築していくことが可能になります。また、新サービスの例として取り上げた「品質保証サービス」は、品質保証業務にO-DAフレームワークを取り込むことで、サービスの品質向上と、効率化が実現できるという考え方を示しています。

4. おわりに

本稿では、テストと保証について、次の2方向を説明しました。

(1)テストによる保証範囲の明確化
　テスト参照フレームワークによりテスト項目を体系的に整理する方法により保証範囲を明確化できることを説明しました。
(2)保証ケースを用いたテスト業務の変革
　保証ケースを用いたエンタープライズアーキテクチャ開発手法であるO-DA標準を用いて品質保証サービスを変革できることを説明しました。

今後、本稿で紹介した手法がITシステム検証の実務で活用いただければ幸いです。

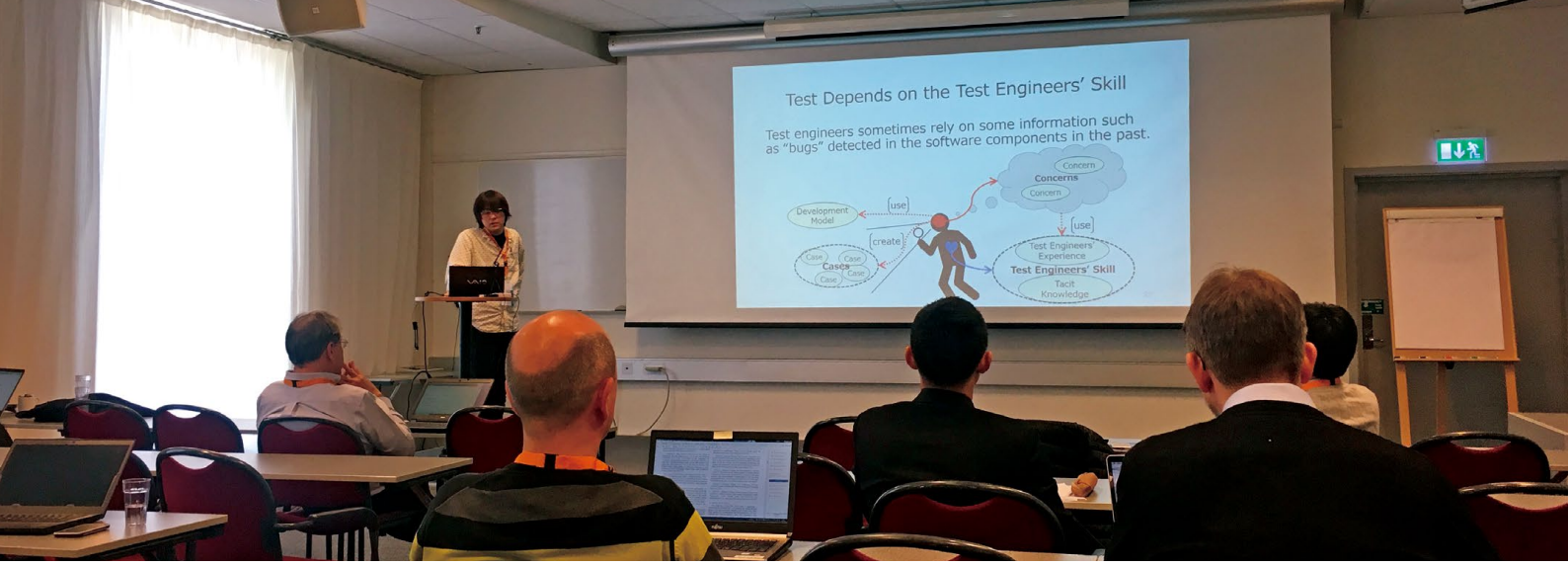
現行テクノロジー・アーキテクチャ	目標テクノロジー・アーキテクチャ
●担当者の PC で品質評価依頼と IT システム評価結果を作成して保存 ●品質評価の観点をワープロで文書化	●開発部門からの品質評価依頼から結果登録までを Web サーバで管理 ●リスク分析サーバ ●保証ケース生成支援ツール ●保証ケース・エディタ ●データベース・サーバで品質保証情報を管理

表 5 現行テクノロジー・アーキテクチャと目標テクノロジー・アーキテクチャの比較

参考文献

[1] TOGAF® Version 9.1, an Open Group Standard (2011).
[2] 山本修一郎,現代エンタープライズ・アーキテクチャ概論-ArchiMate入門-132ページ、出版社:デザインエッグ社;1版 (2016/7/20) ISBN-10:4865436804
[3] 主張と証拠 978-4-86293-095-8 (ダイテックオンデマンド出版,2013)
[4] 平山 雅之、中本 幸一,ソフトウェア工学の共通問題:4.組込みソフトウェア分野の共通問題の考え方,情報処理,vol.54, No.9, pp.890–863, 2013
[5] Open Group Standard, Real-Time and Embedded Systems, Dependability through Assuredness™ (O-DA) Framework,2013
[6] Shuichiro Yamamoto, Shuji Morisaki, A case study on architecture quality assurance service using O-DA, Book of industry papers, poster papers and abstracts of CENTERIS 2016, PROJMAN 2016, HCIST 2016, ISBN 978-989-97433-7-3, pp.185-193, 2016
[7] Yamamoto, S., Morisaki, S., An Evaluation of Assuring Test Case Sufficiency using A D-Case Pattern, WOSD2014, DOI: 10.1109/ISSREW.2014.10
[8] 大林英晶、森崎修司、渥美紀寿、山本修一郎、逸脱分析を用いた要求仕様からのテスト項目抽出法, 情報処理学会論文誌、Vol.57, No.4, pp.1262-1273, Apr. 2016

本記事の内容に関するご質問、お問合せは、**ベリサーブマーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。



テストを“開発”しよう

～テストモデリング記法「UTP」とその拡張提案～

編集：株式会社ベリサーブ
マーケティング部

水野 昇幸氏
みずの のりゆき

TEF道（Testing Engineer's Forum:ソフトウェアテスト技術者交流会北海道部会）所属。
テスト設計コンテスト'17 OPENクラスでチーム「STUDIO IBURI」のメンバーとして参加し優勝。
国際カンファレンスへの参加実績としては、SpaceOps2014論文共著、6WCSQ2014論文発表、InSTA2017論文発表、InSTA2018論文共著がある。

はじめに

2018年4月、筆者はNPO法人ソフトウェアテスト技術振興協会（ASTER）のコミュニティ「智美塾」のメンバーとして、ソフトウェアテストの国際カンファレンスである「ICST[*1]」に参加しました。

カンファレンスに併設されたワークショップ「InSTA[*2]」にて、筆者らは、UML[*3]を拡張したテスト向けモデリング技法であるUTP[*4]を、さらに拡張する提案を行う論文を発表しました。

本稿では、「テストシステムアーキテクチャ」と「テストスイートアーキテクチャ」の2つのテストアーキテクチャという切り口から、UTPの概要と筆者らが提案した論文の内容をご紹介します。

1. ICSTとInSTAについて

論文の内容に入る前に、ICSTについてご紹介します。

ICSTは、通信・電子・情報工学を中心とした国際標準規格の策定や国際学会の運営を行っているIEEE[*5]が主催する、ソフトウェアテストに関する世界最高峰の国際カンファレンスです。世界中から研究者、エンジニアやマネージャが集まり、ソフトウェアテストの最新動向に触れることができます。

2017年3月に東京で開催（ICST2017）されたこともあり、聞いたことや参加されたことがある方もいらっしゃるかもしれません。

2018年にICST2018[*6]がスウェーデン郊外の都市「ヴェステロース（Västerås）」で開催されました。基調講演[*7]では、Facebook社のDulma氏による静的解析ツール「Infer[*8]」の紹介、Electronic Arts社のMagnus氏らによるゲーム開発におけるテストプロセスやAIにおけるテストの紹介、オックスフォード大学のMarta教授によるディープニューラルネット

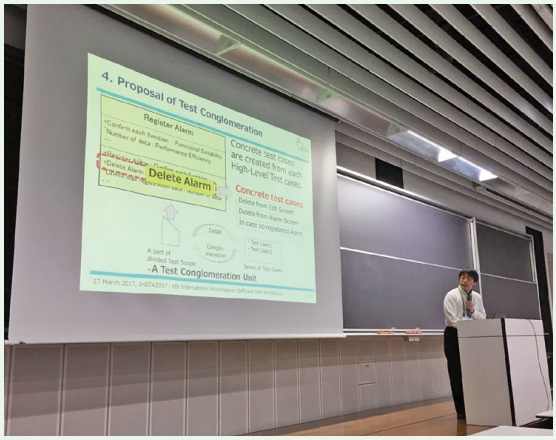
*1: ICST - IEEE Conference on Software Testing, Validation and Veification
*2: InSTA - International Workshop on Software Test Architecture
*3: UML - Unified Modeling Language:主にオブジェクト指向分析や設計のための、記法の統一がはかられたモデリング言語。
*4: UTP - UML Testing Profile
*5: IEEE - Institute of Electrical and Electronics Engineers

ワーク*9を用いた画像解析における検証技術の紹介などが行われました。

ICSTでは3日間で40件以上もの論文発表があり、業界のトレンドを知ることができます。昨年、今年とICSTに参加した限りでは、テスト自動化は当たり前という前提条件での事例や研究が多くみられます。非常に多くのテストケースが自動化された状況下で発生しやすい課題に対して、どのように対処すべきか?という発表がいくつもありました。このように、国際カンファレンスでは、世界と日本の状況の違いを感じることができます。海外での開催が多く、英語が必須ではありますが、ソフトウェアテストに関わる人であれば、一度はICSTに参加されることをお勧めします。

公式サイト http://aster.or.jp/workshops/insta2018/

ICSTでは、3日間の本会議前後の日程で、特定の興味分野に特化した併設ワークショップが開催されます。筆者らはICST併設ワークショップのInSTA2018*10への論文投稿と発表を行いました。InSTAでは、テスト対象の分析や、設計手法、テストの表記方法など「テストアーキテクチャ」と呼ばれる分野に特化したワークショップが開催されています。



InSTA2017 論文発表の様子

2. そもそもUTPとは?

2-1 UTPの得意エリア

UTPとは、前述の正式名称"UML Testing Profile"から想像ができるとおり、UMLをソフトウェアテスト向けに拡張したモデリング言語です。2018年5月段階ではUTP Ver.2.0 Beta版*11が公開されており、無料で仕様情報を参照することができます。

UTPはテスト実行用の仕組みである「テストシステム」の構造と、振る舞いを表現する場合に使うことができます。簡単な例をご紹介しましょう。

次ページの図1~3の例はUTP Ver.2.0 Beta版の「ATM Example(銀行のATMをテストするテストシステムの例)」に記載されているモデルの一部です。図1はテスト対象をUMLで記述したもので、図2、3はUTPで記述したテストの構造、振る舞いを表す例となります。UMLとUTPは親和性が強く、違和感なく連携できていることがわかると思います。

*6:ICST2018の公式サイトは右記を参照。http://www.es.mdh.se/icst2018/
*7:ICST基調講演概要は右記を参照。http://www.es.mdh.se/icst2018/keynotes/
*8:ツールは右記URLより取得が可能です。https://github.com/facebook/infer
*9:Deep Neural Networks:機械学習アルゴリズムの一つ。データのパターン認識によってグループ化や分類などに使用されます。
*10:InSTA2018 - 5th International Workshop on Software Test Architecture
*11:UTP Ver.2.0 Beta版リンク:https://www.omg.org/spec/UTP/About-UTP/

図1はテスト対象側のATMにおける加算処理について「お金(Money)」クラス(システム構成要素の一部)をモデル化してUMLのクラス図で描いた例です。簡単に説明しますと、iMoneyはUMLで使用する「お金(Money)」のインタフェースクラス[*12]です。それを実現するクラスとして、お金(Money)クラスとお金を入れる財布(MoneyBag)クラスが存在しています。お金クラスには、金額と通貨(ドルやユーロ、円などの種類)の属性、お金を追加するためのプログラムであるadd操作などが確認できます。

図2はテスト対象(Test Item)とテスト実行のためのテストコンポーネント(シミュレータ、ドライバ、スタブ等)との関連性を示す構造図です。UTPでは「Test Configuration」という名称で呼ばれます。図3は、テスト対象のadd操作に対するテストケースの振る舞いを示すシーケンス図です。

この図では、myMoney1インスタンスをドル、myMoney2インスタンスをユーロの通貨で生成しています。その後、myMoney1にmyMoney2を加算するという処理(add操作)を実施するテストケースの振る舞いを表現しています。こちらのシーケンス図では、ドルとユーロの異なる通貨を加算する処理を実施していることが分かります。また、<<CreateStimulusAction>>の部分で、テストケースの確認対象となるアクションを実施し、<<ExpectResponseAction>>の部分で結果確認を行っています。

特にUTP Ver.1.0では、このようなJUnit[*13]を想定したようなテスト対象とテストコンポーネントの構造や振る舞いの例をいくつも目にすることができます。初期のUTPでは、ユニットテストを中心とした構造や振る舞いをUMLベースで表現しようしていた様子がうかがえます。

最新のUTP Ver.2.0 Beta版ではシステムテストの検討にも活用できます。加えて、通信プロトコルのテスト等に使われるテストケース記述用の言語であるTTCN3[*14]と連携を行うことで、UTPの振る舞いモデルからのテスト自動実行スクリプト生成および実行も想定されています。

以上のように、UTPはテスト対象とテスト実行用の機能などを含めた環境、すなわちテストシステムを表現し、テスト自動実行まで議論できる「テストシステムアーキテクチャ」モデル[*18]の表現に適しています。

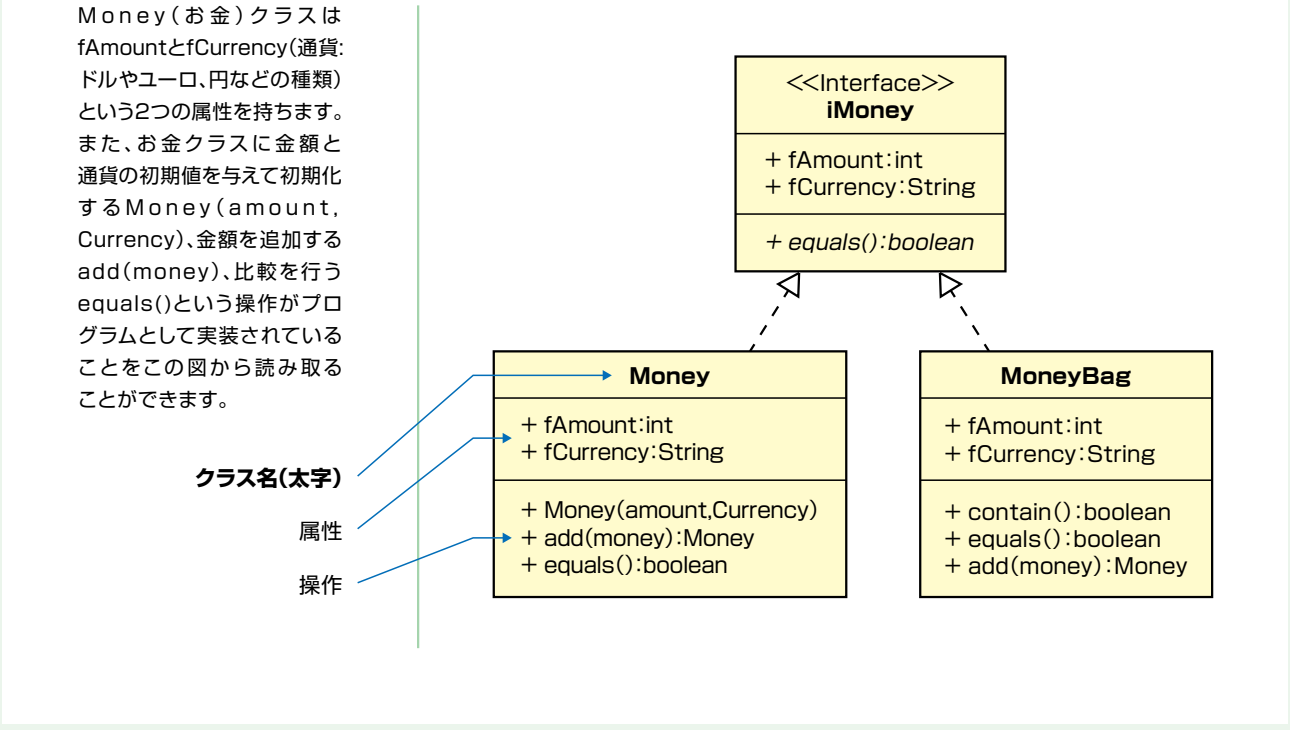
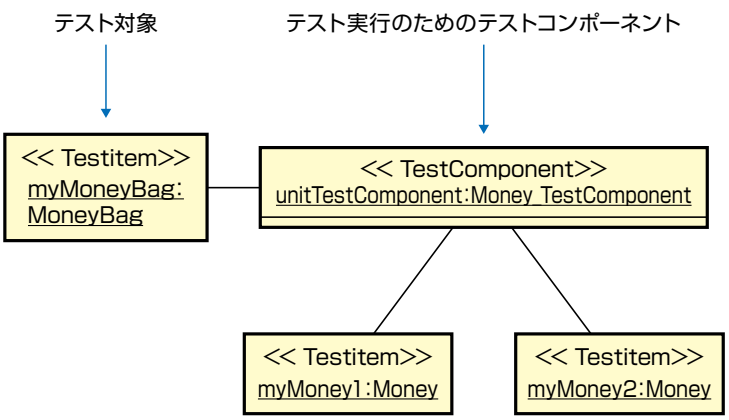


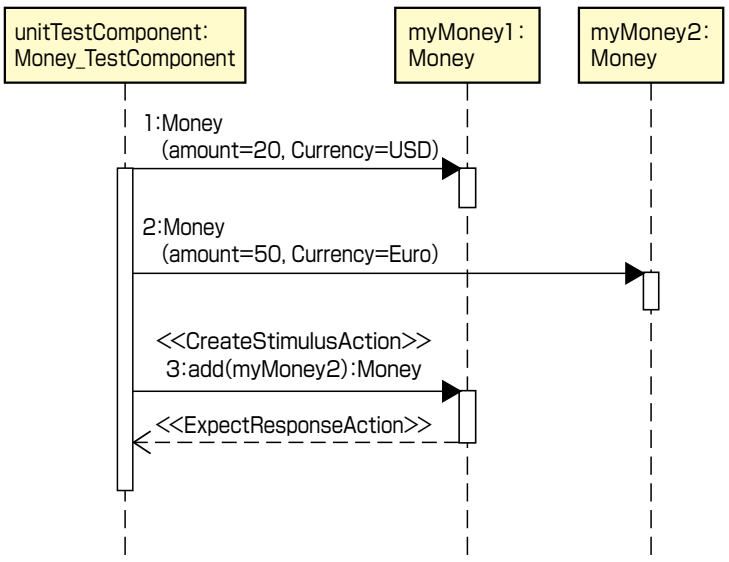
図 1 : テスト対象 UML クラス構成

*12:インタフェースクラスや実現に関しては、Java書籍を確認いただけると理解しやすいです。
https://www.amazon.co.jp/gp/bestsellers/books/515820
*13:JUnitはJavaで開発されたプログラムにおいてユニットテストの自動化を行うためのフレームワーク。
https://ja.wikipedia.org/wiki/JUnit
*14:TTCN3 - Testing and Test Control Notation Version3
TTCN3に関しては、下記リンクを参照。http://www.ttcn-3.org/



<<Testitem>>はテスト対象であることを示しています。
<<TestComponent>>はテスト実行のためのコンポーネント(シミュレータ、ドライバ、スタブ等)であることを示しています。
この図からは、Money_TestComponentというテスト用のコンポーネントから、MoneyBag(財布)と、Money(お金1)、Money(お金2)を使ったテストを実行することがわかります。

図 2 : UTP Test Configuration 図



この図から読み取ることができるテスト実行処理の流れ

1:1つ目のMoney(お金)を20ドルで初期化します。(myMoney1)

2:2つ目のMoney(お金)を50ユーロで初期化します。(myMoney2)

<<テストケースの確認対象となるアクション>>

3:myMoney1 にmyMoney2を加算する処理を実施します。

<<結果確認>>

加算処理が成功することを確認します。

図 3 : UTP 振る舞い図

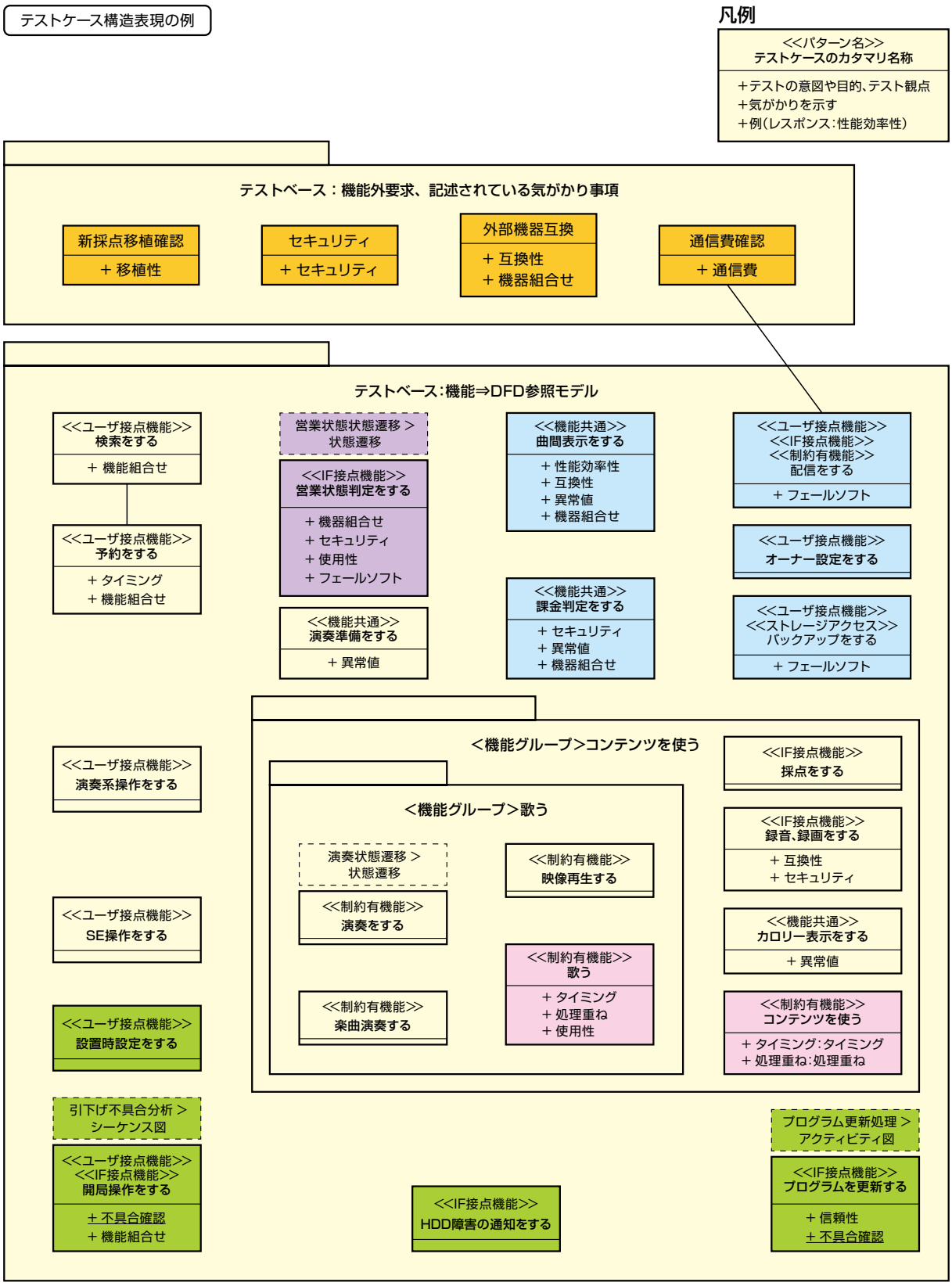


図４：テスト設計コンテストにおけるテストケース構造表現の例*17

*15:VSTeP - Viewpoint-based Test Engineering Process：テスト開発プロセスVSTePとテスト観点についての詳細は、次のリンクを参照。http://qualab.jp/research

*16:テスト設計コンテストは、NPO法人 ソフトウェアテスト技術振興協会（ASTER）が主催するソフトウェアテスト技術の向上と促進の機会を狙いとしたコンテスト、詳細は下記を参照。
http://aster.or.jp/business/contest.html

*17:発表内容の詳細は「テスト設計コンテスト'17オープンクラス 決勝戦レポート」を参照。
http://aster.or.jp/business/contest/contest2017/pdf/presentation_studio_iburi.pdf

*18:テストシステムアーキテクチャとテストスイートアーキテクチャの分類は、下記論文を参照。
http://qualab.jp/materials/MaSST2012.120620.paper.pdf

*19:ISO - International Organization for Standardization（国際標準化機構）

2-2 UTPの不得意エリア

UTP Ver.2.0 Beta版は「テストシステムアーキテクチャ」の表現に適しているのですが、テストケースの構造やテスト設計に相当する「テストスイートアーキテクチャ」を表現することは不得意です。UTP Ver.2.0 Beta版ではテスト戦略を考慮した図の作成や、テスト要求の表現が可能になっていますが、テストケースの表現にうまく活用できるかといえば悩ましい状況です。

例えば、テストケースをグルーピングしてうまく表現するための概念はほとんど用意されていません。テストケースに対する何らかの網羅性を表現することや、テストケースとその目的との関連性を表現することも考慮されていません。

この状況に対して、日本のテストコミュニティではVSTePにおけるテスト観点[*15]と呼ばれる概念が有名です。VSTePでは、テスト観点の構造化や、グルーピングにより全体像を表現することが可能です。

また、テスト設計コンテスト[*16]では、複数の参加チームにおけるテスト設計成果物として図4のようなテストケース構造が確認できます。

このようなテストの目的や意図とテストケースを結びつけた構造をモデルとして表現する方法が、効率的かつ信頼性の高いテストを行う上で必要とされています。

以後、これらのテストケースにつながるモデルを、「テストスイートアーキテクチャ」モデル[*18]と呼ぶことにします。

「テストスイートアーキテクチャ」モデルを上手に構築することで、次のような利点が得られます。

●段階的にテスト設計の思考過程を示すことができ、レビューがやりやすい。
●俯瞰することで大きなテストケースの抜けを発見しやすい。
●効率的なテスト実施順序を導出できる。
●類似プロダクトや派生開発に知見を流用することができる。

例えば、ISO9126やISO25000（SQuaRE）[*19]シリーズに記載されるソフトウェア品質特性から必要なテストケースを考える方法[*20]が有効です。

このような検討を行う際に、図4のように「テストスイートアーキテクチャ」モデル上で品質特性をテストケースと関連付けて表現できると役に立ちます。このような表現を実現するため、筆者らはUTPに新たな表記を加える取り組みに至りました。

3. UTPの課題を改善する「Test Aspect」の提案

前述したとおり、UTP Ver.2.0 Beta版の段階では「テストスイートアーキテクチャ」モデルの表現は不得意です。そのため、筆者らは、この課題を改善するために「Test Aspect」という概念をUTPへ追加することを、論文にて提案しました。

ここで、InSTA2018へ投稿した論文概要を簡単に紹介しましょう。テストケースを生成する際には、要件定義書や何らかの設計成果物（「Development Model」と呼ぶ）を基にテスト設計を行うことが一般的です。この際、「Development Model」作成担当者は厳しい期限に追われながら開発のためのモデルを作ります。その結果、目先の開発に必要な最小限の情報のみが「Development Model」に記載されてしまうことになります。

この最小限の情報ではシステムとしては不十分です。そのため、テスト分析やテスト設計時に、テストエンジニアが品質特性を考慮した検討を行う場合があります。他にも、パラメータを詳細設計することで情報を補完することもあるでしょう。

ここで筆者らは、「Development Model」に記載されていないがテストで必要な各情報を「テスト対象のテストベース（「Development Model」のこと）と異なる側面」と捉え、「Test Aspect」という概念として定義しました。この「Test Aspect」をモデル化すると役に立つということを論文でまとめています。

前述したVSTePにおけるテスト観点を知っている方は、本内容は理解しやすいはずです。具体的な「Test Aspect」を使用したモデルの例は次のようになります。テスト対象はATMのdeposit（入金）機能となります。

※論文からの引用ですので英語という点はご了承ください。

次ページの図5から、入金する紙幣の種類、金額、入金時のレスポンス時間、入金の繰り返しや異常時の手順を確認するテストを行う必要があることがわかります。

UTPへ追加提案するにあたって、記法を明らかにする必要もありました。図5に示す通り、クラス要素に<<TestAspect>>というステレオタイプをつけることで分類を明示しています。クラス図の関連表現、パッケージ要素を使用することで、複雑な要素も表現可能です。

*19:ISO9126 はISOが定めるソフトウェア製品の品質に関して規定している国際規格のことを指す。ISO25000は、システム／ソフトウェア製品の品質要求と評価に関して規定している国際規格で、ISO9126(Systems and software engineering)と、ソフトウェアの製品評価に関するISO14598(Systems and software Quality Requirements and Evaluation)シリーズを統合したもの。2つの規格の英語表記の頭文字をとってSQuaRE と呼ばれている。

*20:具体的な例については次のサイトを参照。http://snsk.hateblo.jp/entry/20120227/p1

以上で紹介した筆者らのInSTA論文に関しては、JaSST'16東京[*21]の基調講演を行ったJon Hagar氏から「その提案をUTP提唱者のMarc-Florian氏に伝えておくよ」と言われています。

今後、UTPの新しいバージョンに「Test Aspect」の概念を追加できるように活動する予定です。

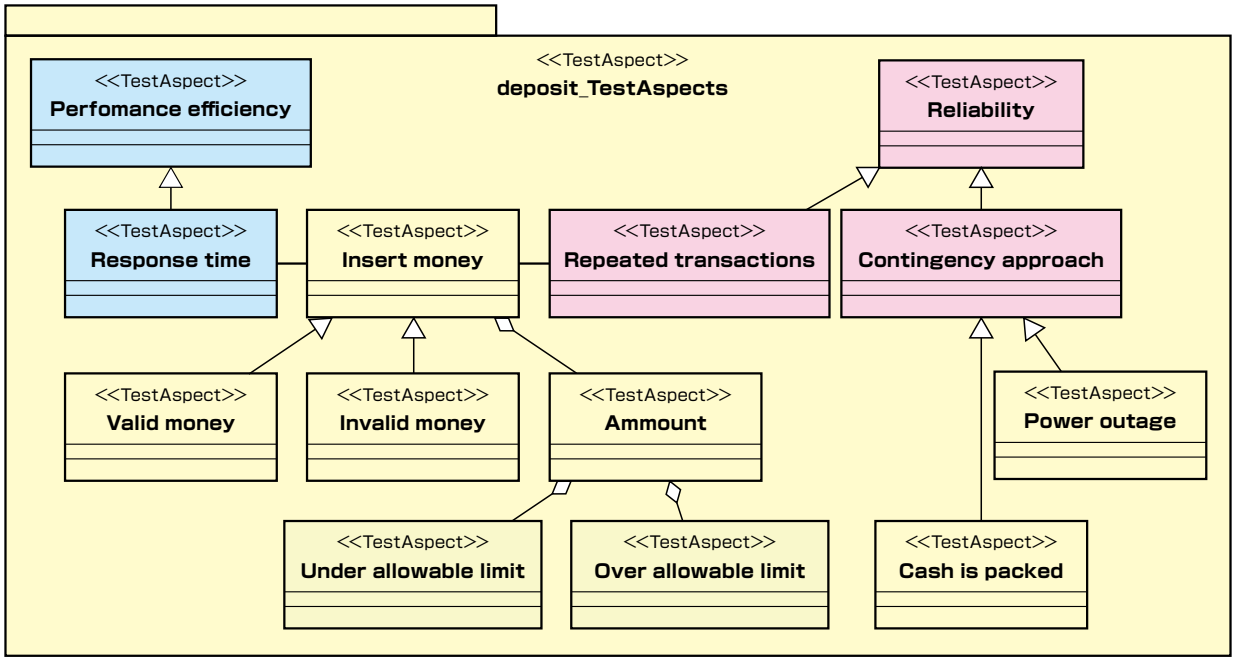


図5：Test Aspectモデルの例

4. テストを「開発」する

UTPで表現できる「テストシステムアーキテクチャ」、InSTA2018で提案した「Test Aspect」で表現する「テストスイートアーキテクチャ」、紹介した2つのテストアーキテクチャはテストベースからの単純な引用というような検討のみでは構築することができません。効果的な「テストシステムアーキテクチャ」を構築する場合には、開発するシステムの構造を知るだけでなく、テストをやりやすいように開発側へ設計提案できるべきでしょう。

さらに、「テストスイートアーキテクチャ」を作り上げる際には、品質を確保することを考えるだけではなく、「テストシステムアーキテクチャ」の構成、逐次リリースされるシステム状況における効果的なテスト実施も考える必要があります。

例えば、VSTePの話題で取り上げたカラオケのような組込みシステム[*22]を考えてみましょう。

カラオケシステム本体をテストする際、マイクやスピーカ、リモコンなどの周辺機器は実物を使うのか、それともシミュレータを使用するのか考える必要があります。さらに、システムテスト自動化を行う場合には、どの部分をどのように自動化するかを考える必要もあります。また、自動化の際には、どのシステム構成でどのテストケースを実施すると効率的に実施できるかを考慮しながらテストを整理することが有効です。この際、図6のようなモデルを用いることで、複数名での議論をやりやすく、短時間で共有ができるようになります。

図6は、上側に「テストシステムアーキテクチャ」（テストを実施するための環境）と下側に「テストスイートアーキテクャ」（実施するテストケース構造）を構築し、対応付けて記述しています。この図を用いることで、テスト実施環境とテストケースの関連を明らかにできます。また、テスト実施の環境やアーキテクチャ毎にグルーピングされているため、テスト自動化範囲や、自動化可能なテストケースを関連付けて議論する材料として有効です。

このような2つのテストアーキテクチャを考える際には単純にテストケースを作るのではなく、テストを「開発」するという言葉

*21:JaSST - Japan Symposium on Software Testing：NPO法人ソフトウェアテスト技術振興協会（ASTER）が主催するソフトウェアテストに関するシンポジウム。

*22:題材は「テスト設計コンテスト」におけるASTERカラオケシステムを参照。
http://aster.or.jp/business/contest.html
※【テストベースのダウンロード】（最新版）からDLできます。

が適しているでしょう。高い技術で効果的な価値をもたらすテストを「開発」することで、より高い価値を生み出していきましょう。

おわりに

ソフトウェアテスト向けモデリング記法のUTPは、テストシステムの構成を明らかにする「テストシステムアーキテクチャ」モデルの表現に適しています。対して、筆者らがInSTA2018論文で提案した「Test Aspect」を用いた「テストスイートアーキテクチャ」モデルは、テストケースの構造や意図を整理し、効果的な実施順番などを表現することができます。今回紹介したこれら2つのテストアーキテクチャを考えることで、より高い価値のテスト設計ができると考えています。現場でのテスト分析やテスト設計を行う場合に試してみてはいかがでしょうか。

なお、来年のICST2019は中国の西安で開催の予定です。古い歴史を持つ都市へ最新のテスト動向を探りに行くということも楽しいですよ!

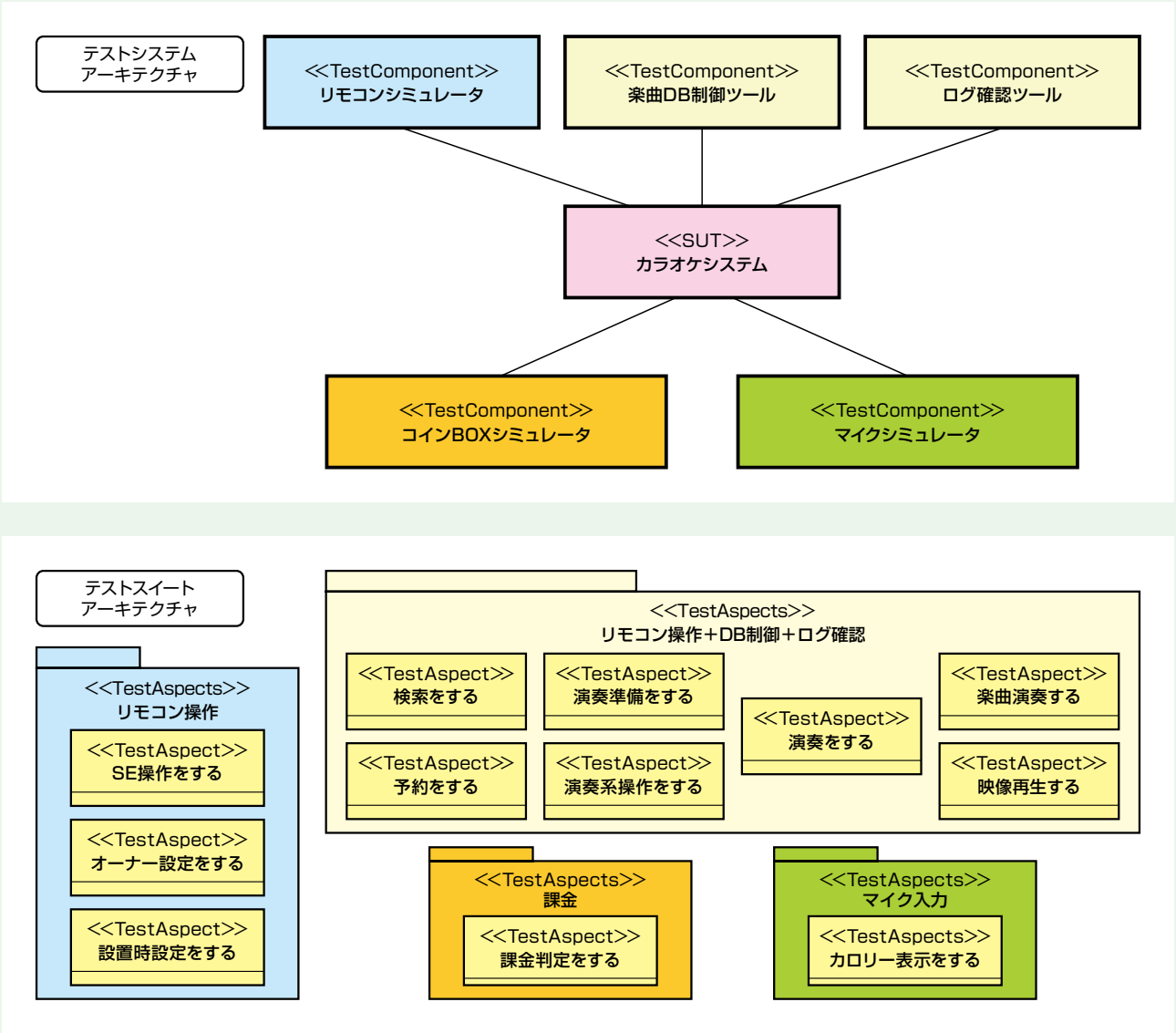


図 6 : UTP テストシステムアーキテクチャ（上）、TestAspect テストスイートアーキテクチャ（下）例

本記事の内容に関するご質問、お問合せは、**ベリサーブマーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。

飛田 基氏
とびた もとい

・ライフコーチ／ゴールドラットコンサルティング・プリンシパル／NPO法人教育のためのTOCマスターリードファシリテータ。
・1974年生まれ。早稲田大学理工学部卒業後、米国フロリダ大学にて化学物理の博士号を取得。日立製作所基礎研究所に勤めたのち、エリヤフ・ゴールドラット博士のTOC（制約理論）に感銘を受け、TOCを適用する経営コンサルタントおよびライフコーチに転身。産業界の既成概念を打ち破り、未解決問題へのブレークスルー実現に取り組む。「世界で800万人が実践! 考える力の育て方」（ダイヤモンド社）が高く評価されている。

担当者があなたで良かった！

「想定外」を減らし、期待を超えるコミュニケーションスキル

ソフトウェア開発のさまざまな場面で、自動化が進んで来ています。では、自動化によって、QCDの満足度が上がったでしょうか?自動化可能な作業は、全体のほんの一部です。ソフトウェア開発の不確実性と、それに起因する「想定外」の発生という本質的な問題は、長年変わっていないのではないかと思います。

失敗案件を経験したメンバーからはこんな声をよく耳にします。「できる人にばかり仕事が集中してしまう。」「若手がなかなか育たない。」「残業体質から抜けられない。」「疲弊してますますミスが増える。」一方、成功プロジェクトの裏には、「○○さんがいたおかげ」と大抵名前があがる人が存在します。自動化すべき内容を設計するのは“人”、作業結果を分析し活用するのも“人”。成功が「できる人」に依存しているのもうなずけますね。

では、プロジェクトの成功のカギを握る「できる人」をどう育てたら良いでしょうか?仮説はこうです。「できる人」は、コミュニケーションを通して、本当に目指すべきこと、本当にやるべきことをはっきりさせ、関係者の理解とモチベーションを高めることに大きな役割を果たしている、のではないでしょうか?

そこで、本稿では、目標設定と、目標実現のための要素を効果的に言葉で表現するための4つの視点*を紹介することにします。

*これらの視点は、イスラエルの物理学者エリヤフ・M・ゴールドラット博士により開発されたものである。

目標設定の大切さについては、あえて説明するまでもないと思います。ですが、それを関係者が誤解なく、他の人に説明しても同じように理解されるようにするためには、言葉を磨き上げなければなりません。そのために使うのが、以下の4つの視点です。

1. 文や単語の意味は明瞭か?
2. 言葉通りの事実が存在するか?
3. 論理（因果）が通っているか?
4. 結果を出すための、要素（原因）が揃っているか?

この4つを順番に確認していくことで、コミュニケーションの質を改善できます。目標設定と、言葉を磨き上げるステップを、商談の会話例を通して見てみましょう。以下、Aはソフトウェア開発会社の幹部、Bはソフトウェアテストを請け負う会社の営業という設定での会話です。

目標の明確化

A：テストの自動化を導入したのだけれど、プロジェクトがなかなかうまくいかないんだよね。何とかならないのかなぁ?
B：なるほど、うまくいかないのですね。確かに、最近、色々な企業様から同じような話をお伺いしています。まず、お伺いしたいのですが、御社では、「プロジェクトがうまくいく」とは、具体的にはどのようなことを想定されていますか?
A：自動化すれば、より早く顧客の要望を取り込んだプロダクトを提供できると思っていたのだが、実際はそれほど納期も変わっていないし、自動化について余計なコストばかりかかっているよ。
B：ということは、「うまくいっている」状態は、テストを自動化することで、顧客にプロダクトをより早く、コストを抑えて提供できている状態ということですね?
A：うん、そんな感じかな。

誰でもプロジェクトを成功させたいと思っています。でも、それだけでは、目標が明確な形で表現されていないということがよくあります。そこで、「成功」とはどういう状態なのかを聞き出すことで、目標を明確にすることができます。

この時に使う質問にはいくつかのバリエーションが考えられます。

●「うまくいく」とは、具体的にはどのようなイメージでしょうか?
●何ができたら「うまくいった」と言えるのですか?
●導入完了時に検証できるような、成功基準はどのようなものですか?

1. 文や単語の意味は明瞭か?

A：ところで「自動テスト」という言葉がはやっているけど、本当のところは、何が自動化されるのかな?
私が「テストをやっといて」と言えば、テスト済みのプロダクトがポンと出てきたら最高なのだけど、さすがにそういうものではないよね?
B：はい、将来そのような日が来るかもしれませんが、現在の「テスト自動化」とは、そのようなものではありません。大きく分けると、ソフトウェアのテストには、計画、分析・設計、実行、評価と報告、終了という大きなフェーズがあります。テスト自動化という場合は、実行フェーズが記述1つで自動的に実施できるようになっていることを指すのが一般的です。評価と報告フェーズの一部が自動化されていることも最近では珍しくありません。しかし、分析・設計フェーズと、終了フェーズは人がやる仕事になっています。テストの実施以外に時間がかかっている箇所がないか、全体を見直す必要がありそうですね。
A：なるほど、よくわかったよ。大体想像していた通りだね。

文や単語の意味がそもそも分からなければ、コミュニケーションは成立しません。

また、業界によっては、使っている言葉は同じでも、意味が異なるということもあり得ます。

さらにやっかいなのは、やたら難しい言葉をあえて使うことで、自分の能力を示そうとしているのでは?と感じるような人もいるということです。
そんなときは、

●「テスト自動化」とはどういう意味ですか?
●「テスト自動化」の「自動」とは何を意味しているのですか?

という質問で、まずは、言葉の意味をハッキリさせると、その後のコミュニケーションがスムーズに進むようになります。

2. 言葉通りの事実が存在するか?

B：ところで、「コストを抑えたい」ということですが、コストにもいろいろございます。具体的にいうと、ソフトウェアの開発と保守に関わる費用を抑えたいという意味ですか?
A：もちろんそのつもりなんだけど、テストの結果が素早くフィードバックされて、結果としてより多くの顧客に使ってもらえるプロダクトを素早く作っていきたいんだよね。
B：では、本当の目的は、テスト自動化で時間短縮効果の高いテスト実施作業の効率をアップすることで、より多くの顧客に使ってもらえるプロダクトを素早く出していくということですね。
A：それ、いいね。たくさん使ってもらえれば、売上も増えるし、結果としてコストも下がりそうだしね。

我々は、「一般的すぎる言葉」や「抽象的な言葉」を、深く意識することなく使ってしまいがちです。ここでは、「コストを抑える」という言葉がそれに相当します。これを具体化したり、限定したりすることで、より頭の中に持っているイメージと、

言葉で表現されたものが一致するようになります。

●「コストを抑える」というのは、具体的にいうとどのようなコストですか?
●テスト自動化で成し遂げたい一番重要なことは「コストを抑える」ことですか?

相手が言うことに対して、「本当ですか?」という投げかけを、表現を工夫しながら進めていく本ステップは、相手に気づきを与え、新しいものの見方があることを知ってもらう強力な手段になるはずです。

3. 論理(因果)が通っているか?

B:では、もし、テストの実行と分析の一部を自動化すれば、結果として、より多くの顧客に使ってもらえるソフトをより早く提供することにつながりそうでしょうか?
A:うーん、どうかなぁ。自動化すれば、テスト実施の効率がアップして、テストが早く終わるのは間違いないと思うけど、それだけだと何とも言えないなー。
B:ですよね。でも、テストの実施が早く終わるようになれば、テスト結果を開発者に素早くフィードバックすることができますし、そうすれば工期の中で開発をもう1サイクル回すこともできるかもしれません。
そうしたら、顧客により使ってもらえるソフトに近づいていくのではないでしょうか?
A:それが理想だね。ということは、テストメンバーと開発メンバーの密接な連携がこれからは重要になるってことか。そうでないと、せっかく捻出した時間が活かせないしね。なんとなく理解していたけど、あらためて納得したよ。

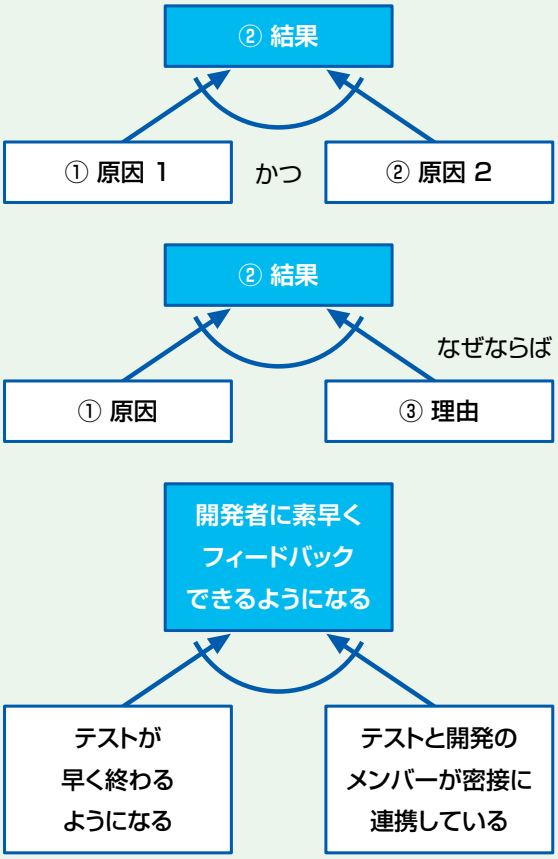
話を聞いたときは納得できても、後になって「あれ?」となることありませんか?論理的な形で話を再構築できなかったときに起こる現象ですね。逆に情報間のつながりがきちんとできれば、コミュニケーションの相手が腹落ちし、さらに理解が深まることも珍しくありません。因果がきちんとつながっているかどうかは、「もし」「結果として」の2つの言葉を補いながら確認してみると良いでしょう。

●もし、テストの実行と分析の一部を自動化すれば、結果として、テスト実施の効率がアップする。
●もし、テスト実施の効率がアップすれば、結果として、テストが早く終わるようになる。

2つ(一般的には「複数」)の原因から、結果がもたらされる場合は、「かつ」または「なぜならば」という言葉を補って確認してみましょう。図中では、これを曲線型の記号で表しています。

●もし、テストが早く終わるようになる、かつ、テストと開発のメンバーが密接に連携しているならば、結果として、開発者に素早くフィードできるようになる。
●もし、テストが早くおわるようになるならば、結果として、開発者に素早くフィードバックできるようになる、なぜならば、テストと開発のメンバーが密接に連携しているから。

このように読んでみて、しっくり来れば、因果がきちんと通った表現、つまり誰にとっても納得性が高い表現になったと言えるでしょう。



4. 結果を出すための、要素(原因)が揃っているか?

B:確認なのですが、テストの実行と分析の一部を自動化するだけで、より多くの顧客に使ってもらえるプロダクトをより早く提供することができそうでしょうか?
他に、必要だと感じることはないでしょうか?
A:確かにテスト、テストといっても、仕様通りに動いているかを検証するだけで、顧客の実際の業務と離れていると使ってもらえないよなー。

B：ありがとうございます。ということは、テストの設計段階で、仕様を確認することだけでなく、顧客の実際の業務が改善することを確かめる必要があるのですね?
A：そう、その通り。あとは、もちろん、プロダクトを妥当な価格に抑えることができれば、使ってもらえそうだな。
B：そうですね。自動化を進めると費用を大きく抑えられる部分と、そうでない部分がございます。自動化の対象と費用、期待される効果について、次回、弊社のエンジニアもつれてきますので、また相談させてください。

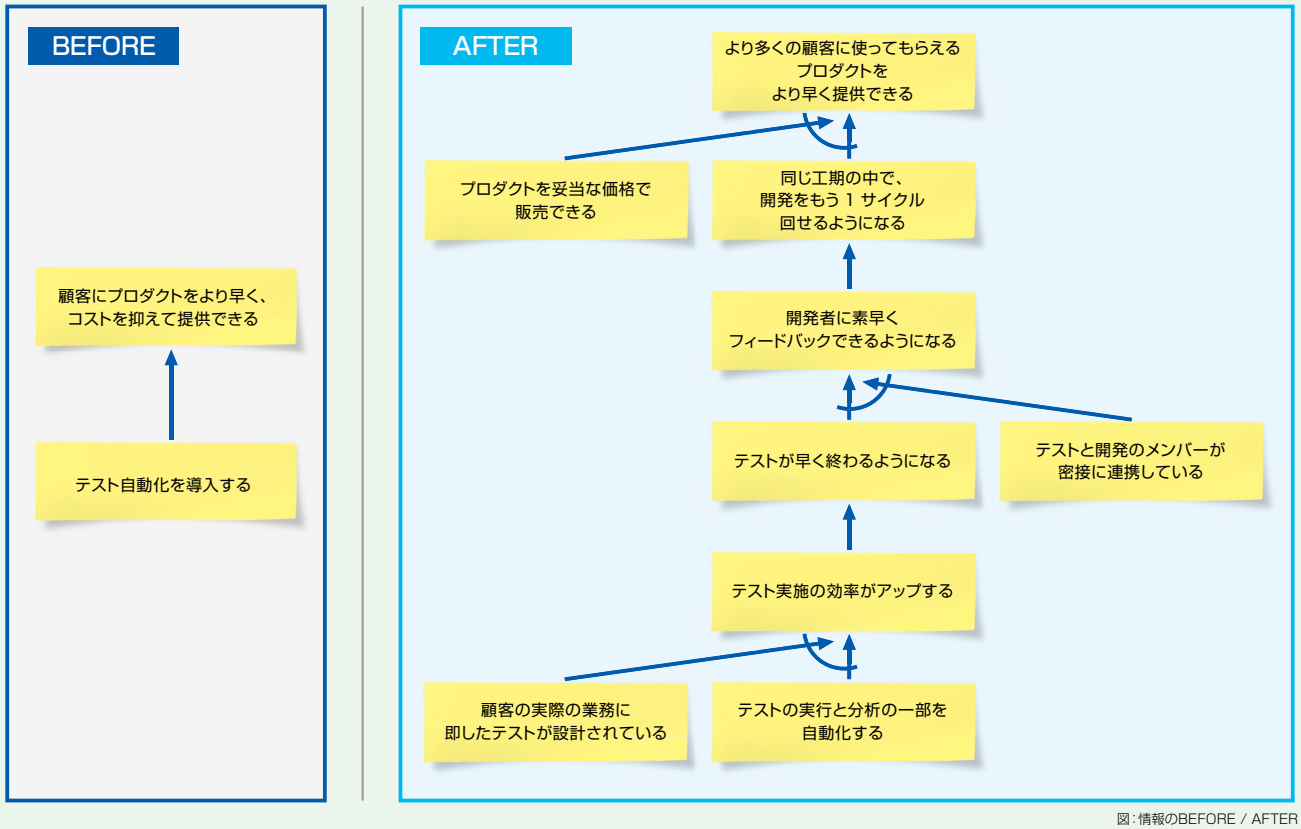
「必要十分条件」という言葉は、中学の数学の授業で聞いたことがある人が多いはずです。ですが、このコンセプトを日常生活で使っている人はどれだけいるでしょう?

顧客に使ってもらえるソフトを素早く出したい。そのためには、
「テストの自動化が必要だ!」
多くの人は、このように必要なことだけを考えてしまうものです。
「テストの自動化さえすれば、顧客に使ってもらえるソフトを素早く出すために十分か?」とはなかなか考えないものですね。だからこそ、

- **テストの自動化さえすれば、顧客に使ってもらえるソフトを素早く出すことができそうでしょうか?**
- **テストの自動化の他に、顧客に使ってもらえるソフトを素早く出すために必要なことはないでしょうか?**
- **テストの自動化だけで、顧客に使ってもらえるソフトを素早く出すのに十分でしょうか?**

このような問いかけによって、十分条件を考えてみましょう。
よりモレなく考えることができるようになるので、目先のコミュニケーションが良くなるだけでなく、仕事の質も上がっていくでしょう。

会話を始める前と終了した後の情報を、BEFORE / AFTER形式でまとめてみました。会話を通して、相手の考えややりたいこと、その効果が格段にクリアになったのではないでしょうか?



図：情報のBEFORE / AFTER

会話例を通して、4つの視点を見てきましたたが、実は会話をしながら図解ツールを併用するとさらにパワフルになります。発言1つ1つをふせん紙に書き留め、因果関係でつながった情報を矢印でつなげていくのです。コンピュータにとって代わられる仕事がますます増えていくだろうと予想される今だからこそ、コミュニケーション能力の向上を通して、なくてはならない人になりたい。私自身もそう思いながら、日々精進しています。

本記事の内容に関するご質問、お問合せは、**ベリサーブマーケティング部** Email:verinavi@veriserve.co.jp までお寄せください。

OSSリスク管理ソリューションの紹介

はじめに

今日、オープンソースソフトウェア(以下OSS)はソフトウェアの開発において無くてはならない存在となっています。非常に数多くのOSSを活用した開発プロジェクトが存在し、高機能・高品質なOSSが多く見られるようになりました。近年では積極的・戦略的にOSSを活用する企業が増えており、自社のソースコードをオープンソース化する流れができています。

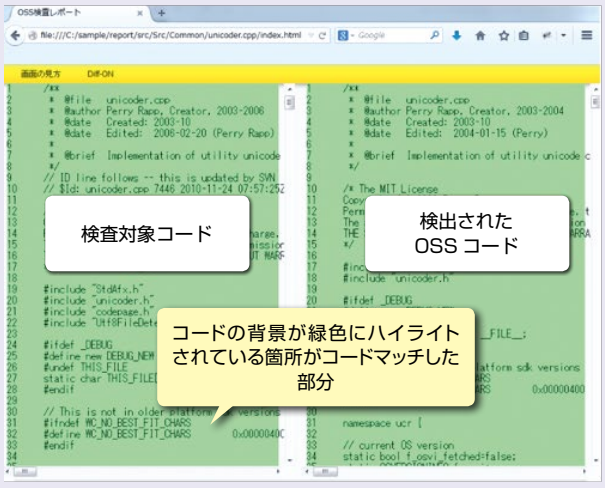
OSSは上手く活用することで様々な恩恵を受けることができますが、一方で、正しい知識を持ち、適切な使い方をしないと思わぬリスクを抱えることになります。ひとつは、ライセンスの条項違反に関連するコンプライアンス上のリスク、もうひとつは、脆弱性に起因するセキュリティ上のリスクです。

OSSには、必ずOSSライセンスというものが存在し、そのOSSの使用を許諾する範囲や、頒布するための責務となる条項が定められています。OSSを使用する際は、このライセンスの条項を遵守しなければなりません。遵守を怠ると著作権侵害として訴訟をされるリスクがあります。

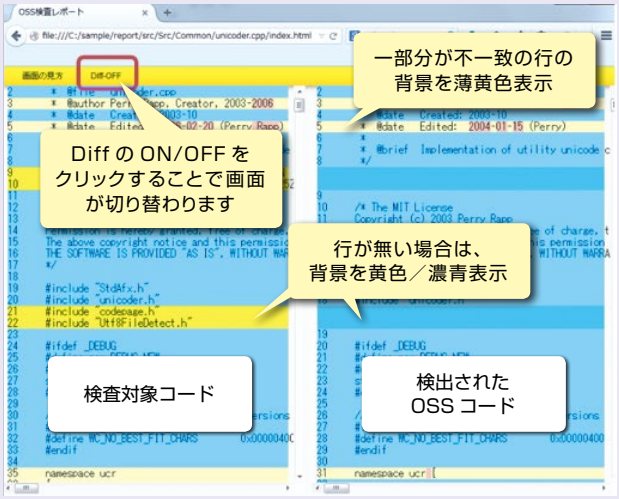
またOSSは、ソースコードが公開されているというその特性上、脆弱性に対して攻撃の標的になりやすいという実態があります。一言で脆弱性と言っても様々ですが、例えばそれが権限掌握や任意のプログラムを実行されてしまうような不具合であった場合、非常に大きなリスクに直結します。実際に、近年でもいくつかのOSSにおける脆弱性の問題により、個人情報の流出やサービス停止などの被害が出たというニュースを見かけることがあります。

(図 1) 対象のソースコードに含まれている OSS を検出、可視化した結果

画面 1：Diff off の状態



画面 2：Diff on の状態



*各画面中の左側が検査対象コード、右側が検出された OSS コード。
*画面 1：コード内の一致した部分の背景を緑色表示
*画面 2：行単位で検出し、一致した部分の背景を水色、内容不一致部分を薄黄色、コードがない部分を黄色表示

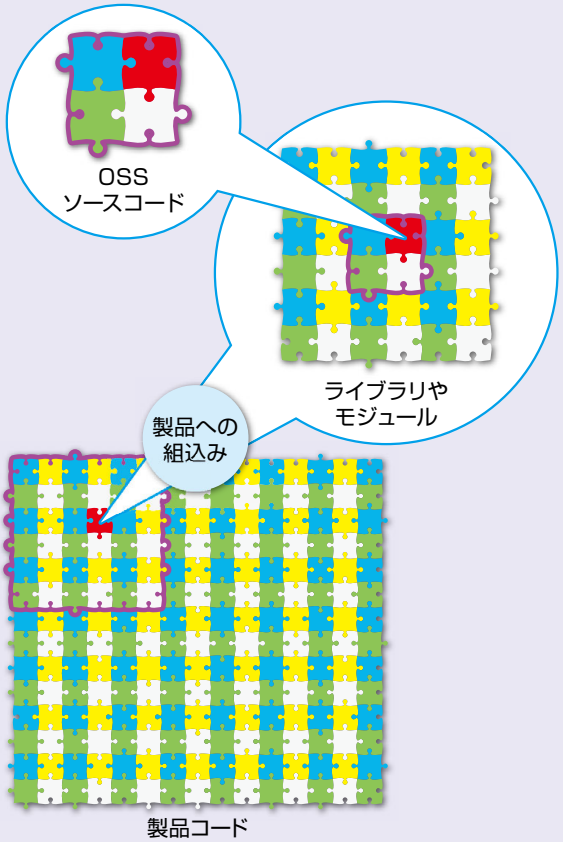
当ソリューションの概要

このような状況において、当社では、安心してOSSを利用するための総合的なソリューション・サービスを提供しています。主軸としているのが、ソースコードをスキャンすることでOSSを検出するツールを用いて、お客様の製品に含まれているOSSを可視化するサービスです（図1）。この可視化は、検出したOSSのライセンスの情報、脆弱性を持っている場合にはその情報も含みます。

ソフトウェア開発の現場において、適切に使用しているOSSを管理していくことは、それなりのコストが掛かります。開発者が開発中、意図せずOSSのソースコードをコピー&ペーストしてしまい、OSSが混入してしまうという問題や、OSSのコミュニティサイトで宣言されているライセンスと実際のソースコードに書かれているライセンスが異なる、などといった問題が起きてしまう可能性もあります。混入したOSSを、例えば目視のコードレビューで検出するのは不可能に近く、または莫大なコストが掛かってしまいます（図2）。脆弱性についても、使用しているOSSの情報を日々追っていくことは多大な労力が必要です。

しかし当ソリューションが提供するツールを利用していただくことによって、大きなコストを掛けることなく、正確で、かつ迅速にOSSの管理を行うことが可能となります。

（図2）混入したOSSの確認は難しい



当ソリューションの特長

当ソリューションでは単純なツールの販売・サポートを行うだけでなく、特長としては個々のお客様のニーズに合わせた以下のようなソリューションを提供しております。

6つのソリューション

①OSSを取り扱うための企業としてのポリシーおよびプロセスの策定に関するコンサルティング
②検出したOSSのライセンスを遵守するためのQA対応
③ツールを利用するためのインフラの構築支援
④OSS検査業務の代行
⑤検査業務の一部自動化
⑥OSS利用推進・啓蒙活動のためのセミナー開催など

ご要望に応じて柔軟な対応ができる体制を整備しております。また内容によっては、お客様のご要望を満たすためのツールを自社で開発することが可能です。

当ソリューションでは年間230プロジェクトのOSSライセンス検査実績があります。組込系、Web系、エンタープライズ系、および車載系など開発分野を問わず対応が可能であり、これまでに蓄積した各分野固有のノウハウを活用することで更なる検査効率化を図り、お客様へ最適なサービスの提供を行っています。

今後の展開

OSSの活用は、今後ますます加速・拡大していくことが容易に想像され、OSSの関わらない開発は、近い将来において特定の一部を除き存在しなくなるのではないかと考えられます。

当ソリューションでは拡大するOSSの活用に対して技術面のキャッチアップとサービスの改善を継続し、今後もお客様がOSSの管理に大きなコストをかけず、安心してOSSを利用していただくための環境構築のご支援を行って参りますので、導入のご検討をよろしくお願いいたします。

当ソリューションに関する問合せ先:
TEL：03-5909-5702　E-mail：sol@veriserve.co.jp

VERISERVE NAVIGATION（ベリサーブナビゲーション）

2018年7月号

編集・発行	株式会社ベリサーブ 東京都新宿区西新宿6-24-1 西新宿三井ビル14F
お問い合わせ	マーケティング部：03-6302-0707 verinavi@veriserve.co.jp