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

VOL. 11
2017 AUTUMN

特別寄稿

ソフトウェア開発においてチームの品質や生産性に対するパフォーマンスを向上させるヒント

松尾谷　徹

特集

裁判例から見た情報システムの品質問題

松島　淳也

ベリサーブ サービス紹介

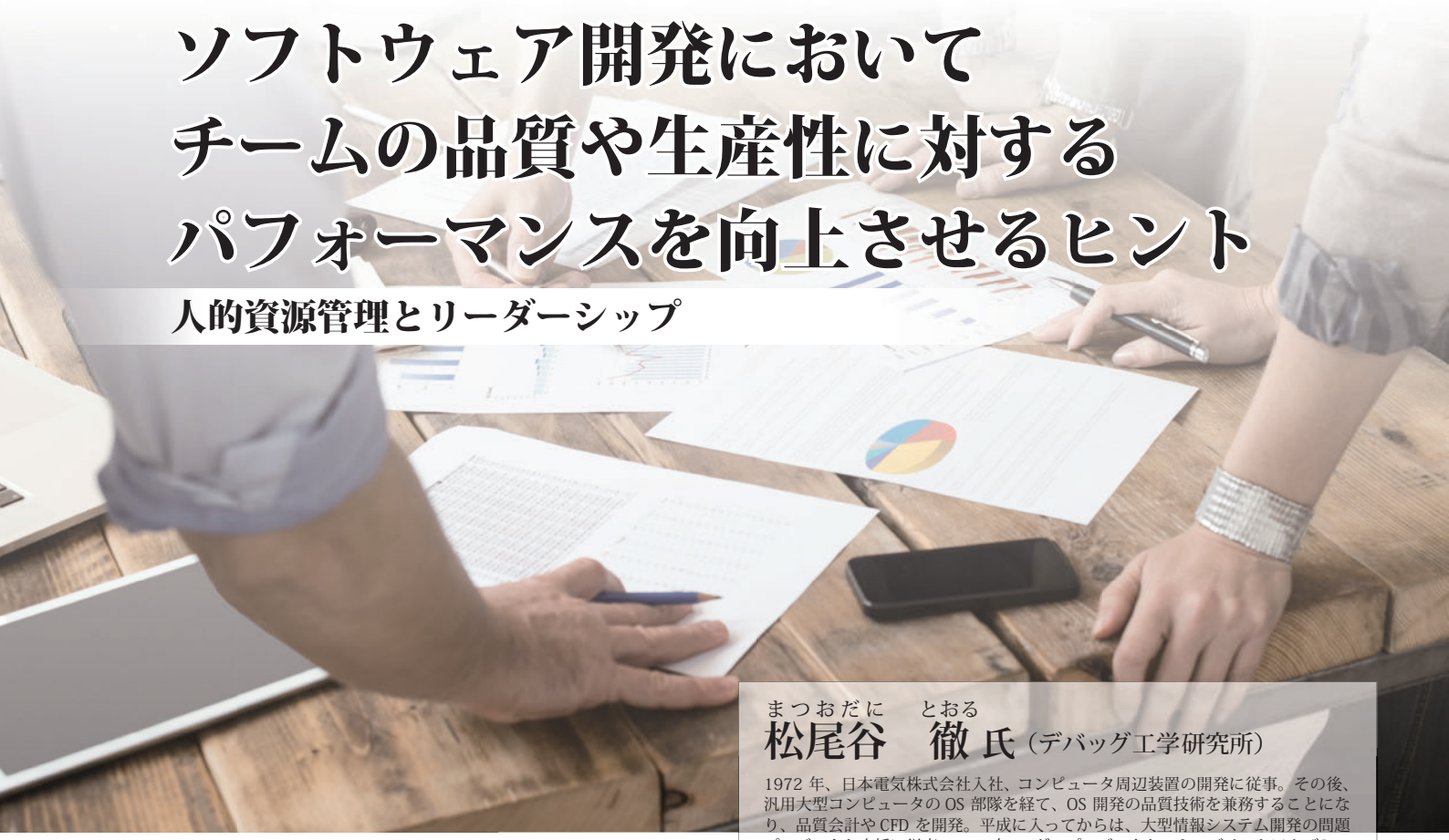
テストの方法論とテスト設計支援ツール「TESTRUCTURE」のご紹介

VERISERVE

特別寄稿

ソフトウェア開発において チームの品質や生産性に対する パフォーマンスを向上させるヒント

人的資源管理とリーダーシップ



まつおだに　とおる
松尾谷　徹 氏（デバッグ工学研究所）

1972 年、日本電気株式会社入社、コンピュータ周辺装置の開発に従事。その後、汎用大型コンピュータの OS 部隊を経て、OS 開発の品質技術を兼務することになり、品質会計や CFD を開発。平成に入ってからは、大型情報システム開発の問題プロジェクト支援に従事。2000 年、モダンプロジェクトマネージメントアカデミーを設立、高度人材育成に注力。2002 年に退職し、デバッグ工学研究所を設立、現代表。明治大学、東京理科大を経て、法政大学講師も務める。

1 はじめに

プロセス改善で有名なワッツ S. ハンフリー博士は、ソフトウェア開発を「組織」「チーム」「個人」の３つの観点から区別して論じています。この３つの特性は、全く異なったものであり、その進め方や改善にはそれぞれの対策が必要であることを彼は多くの著書で示しています。わが国では、大規模な調達における組織のパフォーマンスに焦点をあてたプロセス改善とその支援活動である SEPG（Software Engineering Process Group）については活発ですが「チーム」については関心が低いようです。

「組織」と「チーム」ではパフォーマンスを高めるアプローチがまったく異なります。多くのステークホルダがそれぞれの目的を持ち主張する大規模プロジェクトにおいては、プロジェクト遂行のために強い規範が必要です。その進め方がプロセス改善ですが、これをそのままチームに持ち込むと「百害あって一利なし」状態になってしまいます。IT 業界で多少の経験を積めば、多くのエンジニアが「立派な会社だがこのチームは…」「個々人は優秀だがチームとしては…」など、チームへの課題を認識・実感しているはずです。

ハンフリー博士もこのことから「組織」と「チーム」を明確に分離したのですが「チーム」に対する考え方が我が国では正しく伝わっていないようです。本稿では、その「チーム」に焦点を当てて、そのパフォーマンスを向上するための考え方や手法について簡単に説明します。

2 人的資源管理とホーソン実験

近年、経営科学において人的資源管理（Human Resource Management）についての研究が進んでいます。これは、いま社会的に話題になっている「働き方改革」に関する基礎理論でもあります。

この研究が始まったきっかけは、1924 ～ 1932 年にかけて米国・シカゴで行われた「ホーソン実験」です。当時は世界恐慌によって経済が停滞しており、企業は少しでも安く製品を製造するために労働強化を進めていました。日本では「蟹工船」（小林多喜二）や「女工哀史」（細井和喜蔵）など、プロレタリア文学が誕生した時代です。

このような状況下で、良識的な経営者は労働強化に頼らずに生産性や品質を高める方法を探り、「テイラー理論 *1」を活用した生産性向上のための実証実験を行いました。それがホーソン実験で、ベル電話会社の製造部門であったウェスティングエレクトロニクス社において行われました。具体的には、工場の照明と作業能率の相関関係や、リレー組み立て作業における湿度、温度、賃金や休憩時間の影響などを実験し調査されました。

この実験では、「良い労働環境は生産性を高める」ということを実証することが目的であったため、労働環境を悪化させた場合に生産性がどれだけ低下するのかを測る手順で行われました。ところが、実験では労働環境を悪化させたにも関わらず、チームの生産性は大いに向上しました。想定とは真逆の結果に関係者は困惑し、当初の仮説に基づく 1924 年の実験は失敗に終わってしまいます。

この後、スタンフォード大やハーバード大の研究者が、この実験結果には何か未知の原因が潜んでいるのではないかと考え、約 10 年に及ぶ研究によって原因を明らかにしたのです。

直接的な原因は、実験に参加した女工たちの勘違いです。彼女たちは、それまで退屈な職場で日々の作業を行っていたのですが、ある日、実験のために特別な作業場（実験環境）に呼ばれ、経営者や新聞記者や学者に見られながら作業を行うことになりました。彼女たちは「私たちは選ばれたのだ」と感じ、休憩の合間など、観察者がいない非公式な場で「頑張ろう！」とチームの達成意欲（モチベーション）を高めていたのです。この勘違いから生じた意欲が、チームの生産性を高めたのです。

現代では一般的かもしれませんが、非公式なコミュニケーションから生まれたチームのモチベーションが、これほど顕著に生産性を高めるとは、当時の経営者や学者は誰も考えていませんでした。当時は、プロセスや技能が重要であり、属人的な特性は人事評価と給与で決まると考えていました。このホーソン実験後、このような考え方を「人間機械論」として葬り、ジョージ・エルトン・メイヨーによる「人間関係論」へと移行していきます。

さて、話を戻すと、日本におけるひと昔前の IT 業界は「人間機械論」的な状況にあり、新人に簡単な技術教育を行い、マニュアルやプロセスを整備して、要員を工数で捉え生産性と品質を追求しました。その結果、一部では「新 3K 職場 *2」として批判を受け、IT 業界のイメージを悪化させてしまいました。IT 業界は新興産業であり、その構成員の多くがエンジニア指向であることから「人間機械論」的な考え方が根強く、働く人やチームのモチベーションに対する系統的な対策が不足していると見受けられます。

*1　20 世紀初頭にフレデリック・テイラーが提唱した労働者管理の方法論で、労働を、1. 課業管理、2. 作業の標準化、3. 作業管理のために最適な組織形態、の 3 つに分けて考える手法です

*2　以前はいわゆるブルーカラーの仕事を指して「きつい・危険・汚い」だった 3K をもじって、「きつい・帰れない・給料が安い」という言葉が生まれました

3 リーダーシップ行動理論とオハイオ実験

チームのパフォーマンス向上において、もう一つ強い影響を与える要素があります。それがチームにおけるリーダーシップです。

ここで注意したいのは、政治や国家、企業経営など「社会」や「組織」に対するリーダーシップと、チームにおけるリーダーシップはまったく異なる特性を持っている点で、例えばナポレオンや織田信長のような「偉人型リーダーシップ」をチームに適応するのはあまり良い結果をもたらしません。チームにおける研究は、リーダーシップ行動論と呼ばれ、経営マネジメントの基礎となっています。この分野の研究も、ある実験がきっかけとしてそれまでの常識を覆すことから生まれました。それが「オハイオ実験」と「ミシガン実験」です。

オハイオ実験は、1940年代にオハイオ州立大学のシャートルらが行ったリーダーシップに関する調査研究で、米国空軍の戦略爆撃機におけるチームのパフォーマンスを対象として行われました。同じ機材、同じクルーでも、機長（准将クラス）が代わることによってチームのパフォーマンスに差が生じる事実について、リーダーの資質や経験ではなく、リーダーの行動がメンバーにどのような影響を与えるのか、その行動を尺度化し、統計解析を行いました。

膨大なデータ分析の結果は非常に単純で、リーダーシップ行動は2つの要素（因子）から構成されていることが明らかになりました。一つは「構造設定」という職務や課題に関するリーダーシップ行動であり、もう一つは「配慮」というメンバーとの関係に関するリーダーシップ行動です。

軍隊は、IT業界とか草野球チームとは異なり、チームを作りそのパフォーマンスを高めるため、訓練に多くの時間と資源を費やしています。特にこの実験対象である職業軍人は、厳しい訓練と選考によって選ばれたエリート集団であり、最強のチームと考えられていたのですが、未知のパフォーマンス要素がこの実験で明らかになりました。

続くミシガン実験はミシガン大学のリッカートらが行った実験で、軍隊のように特別な訓練によって作られた強いチームではなく、民間企業である大手生命保険会社の監督者と部下を対象に行われました。この研究も監督者の行動を部下の観点で尺度化し、統計分析を用いています。観測されたリーダーシップ行動はオハイオ実験同様に2軸で表され、「従業員重視」と「生産性重視」と名付けられました。これは、オハイオ実験の「配慮」と「構造設定」に対応しています。

なお、日本においては、経営学ではなく心理学者であった九州大学の三隅 二不二先生が公務員や企業の管理職の行動に対して尺度化と分析を行い「PM理論」として体系化しています。PはPerformanceであり「構造設定」「生産性重視」にあたり、MはMaintenanceであり「配慮」「従業員重視」に対応しています。このように、チームに影響を与えるリーダーシップ行動は、東西を問わず共通した特性を持っているようです。

日本のIT業界ではエンジニアのスキル分類や定義が先行していますが、どんなリーダーシップがどのように成果を上げているのかについて調べた実証型の研究はほとんど行われていません。そこで、我々デバック工学研究所では、「苦手な上司 VS 良かった上司」調査として、PM理論をベースに2002年から2017年までに、IT業界の実務者数百人に対して調査を行いました。

この調査は、被験者に対して今までに経験した上司の影響を調べるもので、「苦手だった上司」＝再びチーム組みたくない上司と、「良かった上司」＝機会があれば一緒にチームを組みたい上司について、振返ってもらいます。そして、両者のリーダーシップ行動についてPM尺度をIT用に加工した質問紙で調べ、統計分析を行います。結果は単純で、メンバーから観た苦手な上司は、ほぼすべてが「配慮行動」「従業員重視行動」

の欠落であり、良かった上司は「配慮行動」が多いという共通点がありました。

この調査では、匿名で定性的なコメントも記録しており、その中で判明した最悪のリーダーシップ行動は、「配慮行動の欠落」＋「規範的な構造設定」でした。「規範的な構造設定」とは、マイクロマネジメントあるいは過干渉と呼ばれる行動です。具体的には、当該業務そのものの指導ができないので、コンプライアンス遵守や、上司自身のメンツや保身的な行動です。

この調査を通じて、どうやらIT業界のリーダーは他の産業界と比較して「配慮行動」が欠落していることが見えてきました。その原因としては、幾つか考えられますが、一番は対人スキルの経験不足と対人スキルを学び訓練する機会の欠如です。

オハイオ実験やミシガン実験からも「配慮行動」が欠落したリーダーシップ行動の原因は示されていて、それは、リーダー自身が育ったチームで「配慮行動」が欠落した場合です。小説やドラマで有名になった「下町ロケット」のように良いチーム文化は伝承されますが、逆も真なりで、悪いチーム文化も伝承されます。

4 チームのパフォーマンス向上における ダイクストラの分析と実例

ソフトウェア開発の場における生産性の問題は、個人→チーム→組織と規模が大きくなると、生産性の差が多少は縮まりますが、他の産業と比べると非常に大きなものです。

個人と比べると数は少ないのですが、チーム単位でもいくつかの研究があります。最初の実証研究はソフトウェア見積もりに使われているバリー・ベームのCOCOMO（Constructive Cost Model）です。ベーム博士は1981年に出版されたSoftware Engineering Economicsの中で64のプロジェクトについて生産性を計測し、モデル化を行っています。観測された生産性の違いは約10倍であり、一番の要因はチーム特性でした。日本においても総合通信メーカにおいて大規模な追跡分析が行われ、著者もこの調査分析に参加しましたが、同様の結果が得られました。

チーム特性の何が10倍に及ぶ生産性を左右するのか、ソフトウェア工学として最初にこの問題に取り組んだのは、1960年代後半に構造化プログラミングを提唱したエドガー・ダイクストラです。

以下の逸話は、日本におけるコンピュータのパイオニアとも言われる森口 繁一先生がダイクストラ博士から直接聞いた話で、その後著者が森口先生から聞きました。

ダイクストラ博士の構造化プログラミング発見の過程は、チームのパフォーマンスに関する調査から生まれました。博士は当時応用数学の研究者でしたが、ある企業からソフトウェア生産性を高めるための研究を依頼され、チーム別の生産性データを分析しました。その中でも顕著に生産性が高いチームを統計的に選び、対象チームがどのようなチーム運営を行っているのかを詳細に調べていったのです。

そのチームのリーダーは、メンバーに対してコーディング規約を課していました。その規約はプログラムの制御構造を解りやすくするための制約で、後の構造化プログラミングに通ずるものです。しかし、メンバーがコーディング制約を守るだけで生産性が上がることは考えられません。高生産性の秘密は、徹底的なコードレビューにありました。コードレビューを真剣に行うことは、レビューする側からすると、厄介で時間がかかる仕事です。特に、制御構造が滅茶苦茶だとレビューにとても時間がかかります。そこで、レビューの生産性を高めるために規範を作ったのです。その結果、リーダーのレビューによる指導効率が高まり、メンバーは早く成長し、チームのモチベーションや自己成長感などが高まり、生産性は飛躍的に高まりました。

なお、このチームでは、新人もベテランのコードをレビューします。絵描きに必要な活動として「良い絵を観る」ことがありますが、IT エンジニアにとっても同様に良いコードや構造を観ないことには成長しません。

ここでのポイントは、レビューの効率を高めるためにコーディング規約を課したという点です。日本の IT 業界は真面目な国民性から、構造化プログラミングの規範を特によく守っていると言われています。ところがその成果が実証されていません。有識者に構造化プログラミングの成果は何かと聞くと「読みやすくなる」と答えます。確かにその通りなのですが、読みやすくなっても未成熟なコードの品質は上がりませんし、エンジニアのスキルも向上しません。スキル向上にはリーダーによる指導的なレビューと自身が良いコードをたくさん読むことが必須なのです。

構造化プログラミングは、チーフプログラマ制を前提としています。この仕組みは、古くから続く棟梁を中心とする伝統的な技能集団（例えばマイスター制）と同様な仕組みであり、ソフトウェア開発においても欧米では基本的なチーム構成です。近年、開発スピードが重要視されると、コードが完成してからのレビューでは遅すぎる状況が生じています。アジャイル開発のペアプログラミングは、もっと早い段階でレビューを行い、エンジニア育成を加速することができます。

一方でチーフプログラマ制の問題は、師匠と弟子の濃密な関係を現代人が嫌う点にあります。欧米では日本人以上に古典的が主従関係を好まない傾向があります。GitHub に代表されるネットを介した OSS コミュニティは、この点を大きく進歩させています。先進的な多くの企業が、人材育成のため OSS コミュニティへの参加を推奨し支援しています。この世界のレビューは仕様、コード、マニュアル、テストを問わず３つの結果を返してくれます。

- いいね
- 自分だったらこのようにする
- ここが問題なので、ここを修正しないと却下

レビュー結果もオープンなので、レビュー者自身も評価されることから、改善が進む仕組みを内包しています。

5 まとめ

チームのパフォーマンスを決定する要素について経営科学とソフトウェア工学の知見を紹介しました。現代の IT 業界は昔の大規模な組織によるシステム開発、例えばバンキングや携帯電話から、Google や Facebook のように、プロダクトではなく新しいサービスを猛烈なスピードで開発し、試行錯誤の結果を市場の評価で選別するビジネスモデルへと変貌しています。その中心がクリエイティブなチームであることは明らかです。

チームのパフォーマンスに関する仕組みは、説明したように学術的には解明されつつありますが、日本における「実践」となると動きが鈍いと感じています。しかし日本人は、古くは宮大工から近年では下町ロケットまで、高いチームのパフォーマンスを実現していますので、残念ながら IT 業界が、まだまだ未成熟なだけです。

日本の IT 業界でも、世界的なパフォーマンスを実現しているチームがたくさんありますが、それを実証的な立場から発表する例が少なく、業界の関心が低いことが、未成熟さの原因の一つではないかと考えています。Google や Amazon の事例だと現実とのギャップが大きすぎて参考にならないのでしょう。公表されている国内の事例では、「おサイフケータイ」を開発したフェリカネットワークスが参考になるかもしれません。著者自身も10年間に及びこのチームを支援しましたが、基本は単純で「チームビルディング」と「配慮行動」です。

裁判例から見た情報システムの品質問題

特　　集

松島　淳也 氏（まつしま じゅんや）

【経歴】
1995 年 早稲田大学理工学部卒業　富士通株式会社入社
マイクロプロセッサーの開発、電子商取引システムの開発等に従事
2006 年 弁護士登録
2017 年 松島総合法律事務所設立

【主な取り扱い分野】
システム開発・ソフトウェア開発、システムの運用・保守をめぐる各種紛争処理
（請負代金等の報酬請求や損害賠償請求）及び契約事務
インターネットビジネスに関する各種紛争処理及び契約事務
知的財産権（特許権、著作権、営業秘密）に関する各種紛争処理及び契約事務

第1 はじめに

ソフトウェア産業における技術は、日々、高度化、複雑化しています。これにより、システム開発プロジェクトが成功した場合、情報システムを発注して利用する企業（以下「ユーザ」といいます。）は、技術の恩恵を受け、業務の効率化を図ることができる一方で、開発業者（以下「ベンダ」といいます。）は、技術の高度化、複雑化に対応することを要求され、開発したシステムの不具合による品質問題に晒されるリスクは高まっていると言えます。

そして、このような品質問題を防止するためには、情報システムの場合、開発段階において各種テストを充実させる必要がありますが、不具合を完全に消滅させることを目標とすると、著しい量のテストを実施することになり、費用対効果の面で合理性を欠く結果になることも考えられます。

そのため、実施するテストの程度等は、開発規模や情報システムの性質（例えば銀行や証券取引のような公益性の高いシステムなのか、あるいは、個々の企業で利用するシステムであるか等）を考慮しながら、合理的な範囲で決定されることになり、ベンダもユーザも不具合と上手に付き合っていかなければならない状況になっています。

そこで、本稿では情報システムの品質問題が法律問題となるケースについてご説明します。

第2

情報システムの品質問題に関するベンダの民事責任

特に、品質問題として取り上げられやすい情報システムの開発工程は、民法上、請負契約であると評価されることが多いので、ここではベンダの責任を「請負契約における請負人の責任」として説明します。

情報システムの開発工程を請負契約と考えた場合、ベンダの責任は以下の表１のように、大きく分けて「債務不履行責任」と「不法行為責任」の２つが挙げられます。

（表１）

債務不履行責任	【仕事が未完成の場合】 契約の解除（民法 541 条等） 損害賠償請求（民法 415 条）
	【仕事が完成している場合】 瑕疵修補請求 （民法 634 条１項） 損害賠償請求 （民法 634 条２項） 契約の解除　（民法 635 条）
不法行為責任	損害賠償請求（民法709条）

大まかにいうと、債務不履行責任は契約上の責任であり、不法行為責任は契約が成立していなくても発生する責任となります。更に、ベンダの債務不履行責任は、請負契約に基づく「仕事」（情報システムの開発作業）が未完成の場合と完成している場合とに分け、未完成を前提とする責任と、完成を前提とした瑕疵担保責任の問題に整理することができます。

ベンダの責任としては、債務不履行責任の他に、不法行為責任も考えられます。「納品はしたが不具合が発生している」という品質問題が発生している状況の場合、債務不履行責任で考えると、仕事が未完成の場合、納期遅延に関する帰責事由の不存在についてベンダが立証責任を負い、完成している場合、瑕疵担保責任はベンダの無過失責任 *1 とされているのに対し、不法行為責任の場合には、ベンダの故意・過失についてユーザが立証責任を負うことになります（表２）。

このため、ユーザは立証責任の負担が軽くなる債務不履行責任の問題としてベンダの責任を追及することが多いので、本稿でもベンダの債務不履行責任について取り上げることとします。

*1 但し、契約上、特約が定められ、ベンダの帰責事由が要件とされている場合もあります。

（表２）

債務不履行責任	【仕事が未完成の場合】 納期遅延に関する帰責事由の不存在について、ベンダが立証責任を負う。
	【仕事が完成している場合】 ベンダの瑕疵担保責任は原則として、無過失責任である。
不法行為責任	ベンダの故意・過失についてユーザが立証責任を負う。

第3

納品した情報システムに不具合が存在した場合の法的責任

情報システムの品質問題が法律問題となるのは、多くの場合、「不具合等の品質問題により、ユーザがベンダに対して損害賠償請求をした上、契約解除の意思表示をして報酬の支払いを拒否する」場面ですので、以下の４点についてご説明します。

①不具合が発生すると、ベンダの報酬請求は直ちに否定されるのか

②ベンダが瑕疵担保責任を負うのは、どのような不具合か

③瑕疵担保責任に基づく契約の解除が認められ、ベンダが報酬請求できなくなるのはどのような不具合か

④システムの不具合にユーザも寄与している場合でも、ベンダは責任を負うのか

1. 不具合が発生すると、ベンダの報酬請求は直ちに否定されるのか

請負契約とは、約束した「仕事」（情報システムの開発作業）を「完成」させることを目的とした契約です。

> 民法 632 条
>
> 請負は、当事者の一方がある仕事を完成することを約し、相手方がその仕事の結果に対してその報酬を支払うことを約することによって、その効力を生ずる。

従って、原則として、契約上の特約がない限り、ベンダは「仕事」を「完成」することで、初めて報酬を請求できることになります。

そこで、まず考えられるユーザの言い分は、不具合等の品質問題が発生している以上、「完成」しているとは言えないから報酬の支払いを拒否するというものです。

裁判所は、不具合が発生した場合のベンダの報酬請求権について、以下のとおり判断しています。

> 東京地裁平成 14 年 4 月 22 日判決
>
> 民法の規定によれば、法は、仕事の結果が不完全な場合のうち仕事の目的物に瑕疵がある場合と仕事が完成していない場合とを区別し、仕事の目的物に瑕疵が存在しても、それが隠れたものであると顕れたものであるとを問わず、そのために仕事が完成していないものとはしない趣旨であると解される。
>
> よって、請負人が仕事を完成させたか否かについては、仕事が当初の請負契約で予定していた最後の工程まで終えているか否かを基準として判断すべきであり、注文者は、請負人が仕事の最後の工程まで終え目的物を引き渡したときには、単に、仕事の目的物に瑕疵があるというだけの理由で請負代金の支払を拒むことはできないものと解するのが相当である。
>
> ※下線は筆者が加筆

「仕事」の「完成」について、最後の工程まで完了しているか否かで判断する方法は多数の裁判例*2 で確認されているところであり、単に不具合が発生したというだけの理由では、ユーザは報酬の支払いを拒否することは困難と言ってよいでしょう。

*2 東京地裁平成 22 年 1 月 22 日判決、東京地裁平成 22 年 12 月 27 日判決、東京地裁平成 25 年 9 月 30 日判決等

2. ベンダが瑕疵担保責任を負うのは、どのような不具合か

では、請負契約において請負人となるベンダが、「仕事」を「完成」した場合でも、瑕疵担保責任に基づく損害賠償義務等を負担することになるのは、どのような場合でしょうか。この点について、裁判所は、本稼働体制となった後でも一定の不具合が発生するという、ソフトウェア開発の実情を踏まえ、以下のとおり判断しています。

> 東京地裁平成 9 年 2 月 18 日判決
>
> コンピューターソフトのプログラムには右のとおりバグが存在することがありうるものであるから、コンピューターシステムの構築後検収を終え、本稼働態勢となった後に、プログラムにいわゆるバグがあることが発見された場合においても、プログラム納入者が不具合発生の指摘を受けた後、遅滞なく補修を終え又はユーザーと協議の上相当と認める代替措置を講じたときは、右バグの存在をもってプログラムの欠陥（瑕疵）と評価することはできないものというべきである。
>
> これに対して、バグといえども、①システムの機能に軽微とはいえない支障を生じさせる上、遅滞なく補修することができないものであり、又は②その数が著しく多く、しかも順次発現してシステムの稼働に支障が生じるような場合には、プログラムに欠陥（瑕疵）があるものといわなければならない。
>
> ※下線①、②は筆者が加筆

つまり、上記の裁判例によると、不具合の発生後、直ちに、「瑕疵」と評価され、ベンダが瑕疵担保責任を負うのではなく、以下のような場合でなければならないとしています。

①不具合が、システムの機能に軽微とはいえない支障を生じさせる上、遅滞なく補修することができない場合（以下「類型 1」といいます。）

②不具合の数が著しく多く、しかも順次発現してシステムの稼働に支障が生じる場合（以下「類型 2」といいます。）

従って、上記の①、②に該当する場合、ベンダは瑕疵担保責任を負うことになり、ユーザはベンダに対し、瑕疵の修補請求（民法 634 条 1 項）、損害賠償請求（同法 634 条 2 項）をすることができます。

3. 瑕疵担保責任に基づく契約の解除が認められ、ベンダが報酬請求できなくなるのはどのような不具合か

では、ベンダの瑕疵修補や損害賠償義務のみならず、契約の解除まで認められてしまうのはどのような場合でしょうか。

例えば、請負契約において請負代金が 1 億円と定められ、瑕疵によりユーザに 1000 万円の損害が発生した場合を想定してみます。契約の解除ができない場合、ユーザはベンダの 1 億円の報酬債権と 1000 万円の損害賠償請求権とを相殺することしかできず、一方、ベンダはユーザから 9000 万円の報酬の支払いを受けることができます。これに対し、契約が解除されてし

まうと、ベンダは1億円の報酬を請求できないことに加え、更に、ユーザから1000万円の損害賠償請求を受けることになるため、ベンダとしては大打撃となってしまうのです。

この瑕疵担保責任に基づく契約の解除について、民法635条は、以下のとおり、単に「瑕疵」があるというだけではなく、「瑕疵」のために「契約をした目的を達することができない」状況になっていることが必要であると定めています。

> 民法635条
>
> 仕事の目的物に瑕疵があり、そのために契約をした目的を達することができないときは、注文者は、契約の解除をすることができる。

そして、裁判所は、前述の2つの類型の場合に、「瑕疵」と判断する傾向があるため、更に、各類型について「契約をした目的を達することができない」と判断されるのはどのような場合なのかを検討してみます。

（1）類型1について

不具合が、システムの機能に軽微とはいえない支障を生じさせる上、遅滞なく補修することができないことにより瑕疵担保責任に基づく契約解除の成否が問題となった事例として、東京地裁平成14年4月22日判決の事案があります。この事案の概要は以下のとおりです。

> 東京地裁平成14年4月22日判決の事案の概要
>
> 販売管理システムを納品したが、在庫照会のための検索処理等に30分以上の時間を要する点等が「重大な瑕疵」（契約の目的を達成できない瑕疵）に該当するか否かが争われた事案

この事案における「在庫照会の検索処理」とは、お客様から電話等で問合せを受けたオペレータが、在庫の有無を確認するためにシステムを利用して検索する処理のことですが、この処理に30分以上の時間を要するのでは、オペレータはシステムを利用することができないため、以下のとおり重大な不具合であるとして瑕疵担保責任に基づく契約の解除を肯定しています。

> 東京地裁平成14年4月22日判決の判断内容
>
> 本件システムは、（1）在庫照会の検索処理に三〇分以上の時間を要する場合があり、その間、画面が止まったような状態になること、（中略）、被告（ユーザ）の営業所では、検索に時間がかかるために、手書きの在庫台帳を作成して顧客からの問い合わせに応じていることによれば、本件本稼働後、本件システムに生じた処理速度に関する不具合は、被告が本件システムを用いて通常業務を行う上で、看過することができない重大な不具合であると認めるのが相当である。
>
> ※下線は筆者が加筆

ベンダは、不具合の指摘を受けると、すぐに修補しようとするのが通常です。また、プログラムの記述を修正するだけで簡単に対応することができればよいのですが、検索処理のように、設計工程に遡ってデータベースの設計を見直す必要がある場合等は、契約の解除が認められる可能性があります。

（2）類型2について

不具合の数が著しく多く、しかも順次発現してシステムの稼働に支障が生じることにより瑕疵担保責任による契約解除の成否が問題となった事例として、東京高裁平成26年1月15日判決の事案があります。この事案の概要は以下のとおりです。

> 東京高裁平成26年1月15日判決の事案の概要
>
> 出版社の販売管理システム（詳細設計～システムテストだけでも委託料約12億円をこえる大規模システム）を納品したが、以下のとおり不具合が発生していたため（但し、ベンダの認識）、ユーザがベンダの報酬請求を拒否し、契約解除した事案
>
期間Ⅰ	1月5日（納入日）～4月30日（検収不合格日）
不具合	159件（高7件、中78件、低74件） 4月30日時点で残8件
期間Ⅱ	5月17日（再納入日）～6月16日（解除日）
不具合	42件（高3件、中8件、低31件） 6月16日時点の残31件
期間Ⅲ	期間Ⅱ以後（第三者が、493画面のうち62画面をテスト）
不具合	29件（高1件、中6件、低22件）

この事案の発生状況を説明すると以下のとおりです。

納品後4ヶ月の検収期間（概要で「期間Ⅰ」と記載した期間）が設定され、この4ヶ月間で、毎月約40件の不具合（合計159件の不具合）が発生し、期間Ⅰの終了時点でユーザは検収不合格の判断をします。その後、ベンダは再納品しますが、再納品された後の1ヶ月の検査期間（概要で「期間Ⅱ」と記載した期間）で、更に42件の不具合が発生したため、ユーザは再度検収不合格の判断をし、契約解除の意思表示をしました。ユーザは契約を解除した後もテストを実施し、29件の不具合が確認されています。

他方で、この事案は開発工程のみで委託料が約12億円を超える大規模なシステム開発であるため、一定の不具合の発生自体はやむを得ないように思います。

契約の解除が認められると、前述のように、ベンダは損害賠償請求を受けるだけではなく、報酬の請求すらできないことになりますが、ベンダにそのようなペナルティを課すこともやむを得ない事案であるという視点で、裁判所は、この事案について、以下のとおりユーザによる契約解除の意思表示が有効であると判断をしたものと考えられます。

東京高裁平成 26 年 1 月 15 日判決の要旨

（結論）解除は有効である。
（理由）
1．今後どの程度の障害・不具合が生じ、その①補修にどの程度掛かるのかについて、その目途が立たない状態にあったものと認められるのである。
2．本件新基幹システムが検収され、現行システムとの並行稼働が可能となる状態になるのは、②現行システムのホストコンピュータの保守期間満了から少なくとも半年以上経過した後になると認められる。
※下線は筆者が加筆

開発工程のみで委託料が約 12 億円を超える大規模な開発であることも考慮すると、裁判所の判断はベンダに厳しい判断であるとも言えます。類型２において、ユーザによる解除の意思表示の有効性を検討する上で、是非とも参照していただきたい事案です

4. システムの不具合にユーザも寄与している場合でも、ベンダは責任を負うのか

ここまでは、ベンダの責任ばかりを取り上げてきましたが、ユーザ側の行動に起因して不具合等の品質問題が発生する場合もあります。設計工程を完了し、開発工程に着手した後の段階で、ユーザが仕様変更を申し出て、ベンダがこれを受け入れたがために不具合が発生するケースは、その典型例と言えます。

開発工程で仕様変更が発生すると、ベンダは開発作業のやり直しをすることになりますが、納期に間に合わせる必要もありますので、時間的な余裕がない中で、開発作業のやり直しを実施することとなり、その結果、品質問題が発生することがあります。このような場合を前提として、裁判所は以下のとおり判断し、ユーザの責任を肯定しています。

東京高裁平成 26 年 1 月 15 日判決(ユーザの責任について)

外部設計後に多数の変更を行えば、本件新基幹システムにおける不具合・障害の発生の可能性を増加させ、その検収完了が遅延するおそれが生じ得ることに照らせば、控訴人（ユーザ）が被控訴人（ベンダ）に対し本件新基幹システムについて多数の変更を申し入れたことは、本件ソフトウェア開発個別契約の目的を達成できなくなった原因の一つであると認められ、その点において控訴人に過失のあることを否定できないのである。
※本事案では「控訴人」はユーザ、「被控訴人」はベンダをいいます。
※下線は筆者が加筆

もっとも、この判決では、ユーザだけに責任があるとしているわけではありません。ユーザの責任を肯定し、ベンダの責任についても以下のとおり判断した上、ベンダとユーザの過失割合を６対４としています。

東京高裁平成 26 年 1 月 15 日判決(ベンダの責任について)

控訴人（ユーザ）がシステム開発等についての専門的知見を備えているとは認められない顧客であるのに対し、被控訴人（ベンダ）は、システム開発等の専門的知見や経験を備えた専門業者であって、控訴人ユーザからの変更の申入れに応じることが、本件新基幹システムにおける不具合・障害の発生の可能性を増加させ、そのために検収終了時期を大幅に遅延させ、本件ソフトウェア開発個別契約の目的を達成できなくなる場合においては、本件プロジェクトの業務委託基本契約に基づく善管注意義務及び本件ソフトウェア開発個別契約における付随的義務として、その専門的知見、経験に照らして、これを予見した上、ユーザに対し、これを告知して説明すべき義務を負うものであって、なお、ユーザが変更を求めるときは、これを拒絶する契約上の義務があると認められるのである。
※本事案では「控訴人」はユーザ、「被控訴人」はベンダをいいます。
※下線は筆者が加筆

この判決の趣旨からすると、ベンダはシステム開発の専門家であるから、素人であるユーザが、仕様変更の申し入れをした場合であっても、ベンダ側で、要求を撤回させる、納期を変更する、仕様変更部分を二次開発にまわす等の交通整理をする必要があるように思います。仕様変更に関する上記のような判断については、顧客であるユーザが取引上の強者であり、請負人であるベンダは取引上の弱者であるため、ユーザからの申し出に対し、これを拒否することは事実上困難であるという現場のエンジニアからの意見を聞くことがありますが、現在のところ、裁判所はベンダに厳しい判断することがあります *3 。

*3　東京地裁平成 16 年 3 月 10 日判決、旭川地方裁判所平成 28 年 3 月 29 日判決等。

第4 おわりに

情報システムの開発に関する裁判所の判断傾向を把握しておかないと、ベンダの担当者としては、ユーザからの仕様変更の要望を、むやみに受け入れてしまうことになりかねません。逆に、ユーザの担当者としては、誤って報酬の支払いを拒否したり、契約解除の意思表示をしてしまうことになりかねません。

情報システムの開発に関する契約金額が高額になっている今日においては、紛争になった場合の損害賠償請求や報酬請求の金額も高額になる傾向があるので、法務部門のみならず、ベンダの開発担当者やユーザの情報システム部門の担当者も一定の法律的な知識を身に着けておく必要があると言えます。

以上

テスト設計支援ツール

特許第6148362号

TESTRUCTURE

テストラクチャー

――高品質テスト設計の決定版！

効果的なテストを効率的に設計するための業界初のブレークスルー

テストの方法論とテスト設計支援ツール「TESTRUCTURE」のご紹介

IT 企画開発部
谷﨑 浩一

はじめに

ソフトウェアの複雑化・大規模化、開発の短納期化に伴い、テストを効果的・効率的に行うことで、不具合の市場流出を防ぐことの重要性が高まっています。

効果的・効率的なテストを行うには、勘と経験だけでテストするのではなく、きちんとテストを設計することが必要です。その際、テスト対象の全体を把握し、テストに必要な観点を構造的に整理し、仕様書などのテストベースとのトレーサビリティを保ちながらテスト設計を行うことが肝要です。また、エンジニア個人の経験やスキルに過度に存することなく効率的にテスト設計を行えるようにするため、方法論やツールの開発も重要です。

本稿ではテストの方法論とテスト設計支援ツール「TESTRUCTURE（テストラクチャー）」について解説します。

1. テストの方法論とテスト設計の関係

「テストの方法論」というものをご存知でしょうか。複数の方法論（富士ゼロックス　秋山浩一氏「HAYST 法」、電気通信大学　西康晴氏「VSTeP」、ソフトウェアテスト技術振興協会　湯本剛氏「ゆもつよメソッド」など）が知られていますが、そもそもなぜ方法論が必要になるのでしょうか?

図1はベリサーブの標準的なテストのプロセスを表す、「Veriserve Standard Method（略称：VSM）」というもので、「テストをする」場合にどんな作業をどういう順番で行ったら良いかが規定されています。このように各プロセスによってテストの作業をどう進めるかを規定していますが、各プロセスの中で成果物をどのように作るかは人それぞれで、成果物の内容は技術者のスキルに依存しがちです。また、成果物の作り方を技術者が毎回一から考えるのは効率が悪いという問題もあります。

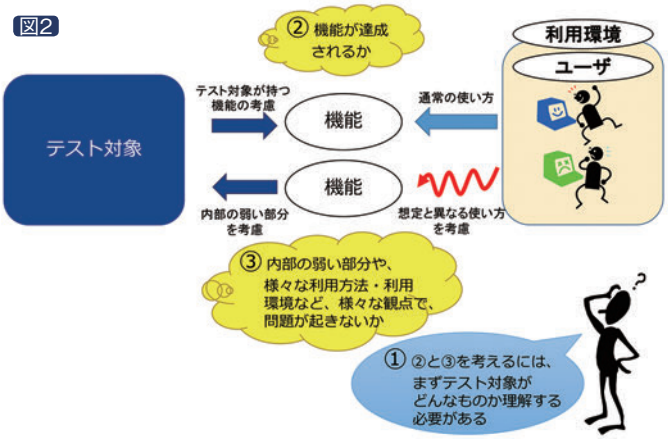
そこで、技術者のスキルに過度に依存せず、一からではない効率的な成果物の作り方を実現するために、「どうのように作るか」を具体化した方法論が必要になります。方法論によって、成果物をどう作るかとそのための考え方が具体化されていれば、その考え方に沿って成果物を作成することで、効率よく質の高い成果物を作成することができます。

これからご紹介するテスト設計支援ツール「TESTRUCTURE」は、図2に示す思考の流れを方法論のベースとしています。①テスト対象がどんなものかを理解し、②テスト対象の機能が達成されるかを確認するために、③様々なテストの観点を考慮して、テスト設計を行います。

図1

ベリサーブ標準検証手法
「Veriserve Standard Method」

図2



2. テスト設計支援ツール「TESTRUCTURE」の概要

「TESTRUCTURE」は、エンジニア個人の経験やスキルに過度に依存することなく効率的にテスト設計を行うことを目的として開発しました。

「TESTRUCTURE」はトレーサビリティを確保しながらテスト設計を行うための支援ツールです。図3に示すように、「TESTRUCTURE」にインポートしたテスト対象の開発仕様書・取扱説明書等のテストベースを参照しながら、「TESTRUCTURE」上でテスト設計を行い、テスト設計の成果物をExcelで取扱い可能な形式で出力することができます。テストベースのファイル形式はWordおよびExcel形式に対応しており、WordかExcelをHTML形式で保存したファイルをインポートすることができます。

図3

*1 Word・Excelからhtml形式に変換したファイルをインポート可能
*2 エクスポートするファイルはExcelで編集可能なxml形式

3. テスト設計の流れと対応する「TESTRUCTURE」の機能

「TESTRUCTURE」を利用したテスト設計の流れは図4のとおりです。この流れは、図2で示した方法論のベースとなる思考の流れに沿ったものにしています。

インポートした仕様書等のテストベースからテスト設計者が読み取った内容を、テストベース上にタグ情報として残しておくことができます。アナログの世界だと、文書から読み取った内容や考えたことを色ペンなどで書き込むようなイメージ（図5）です。

たとえば、テスト設計者がテストベースの内容を読んで「これはテスト対象の機能を意味している」とか「ここはテスト観点として利用可能だ」「ここは意味が分からないので質問しないといけない」といった形で考えた内容を、タグ情報としてテストベースに付与します。

次に機能一覧、テスト観点一覧を作成します。どちらも階層構造で作成できるようにしています。また、過去に作成したテスト観点一覧などを事前に登録しておくことで、ノウハウの再利用につながります。

テストベースに機能タグ、テスト観点タグを付与していた場合、機能一覧・テスト観点一覧にタグと同じ内容が白色で表示されます。各一覧画面において手動で追加した項目は黄色で表示されるため、テストベースとのトレーサビリティが取れている項目とそうでない項目が区別できます（図6）。

図4

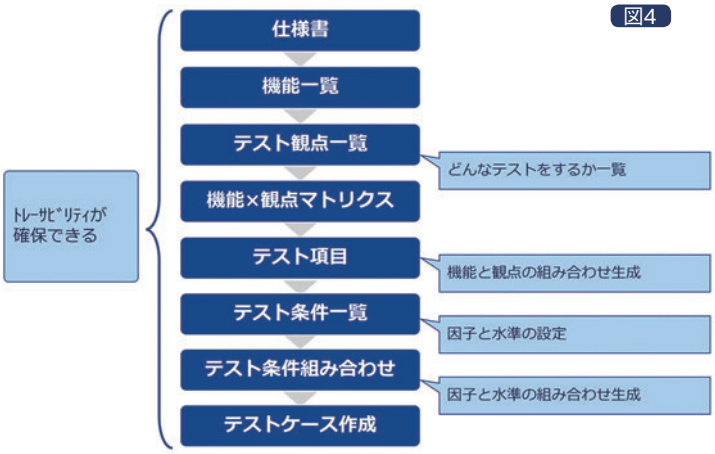
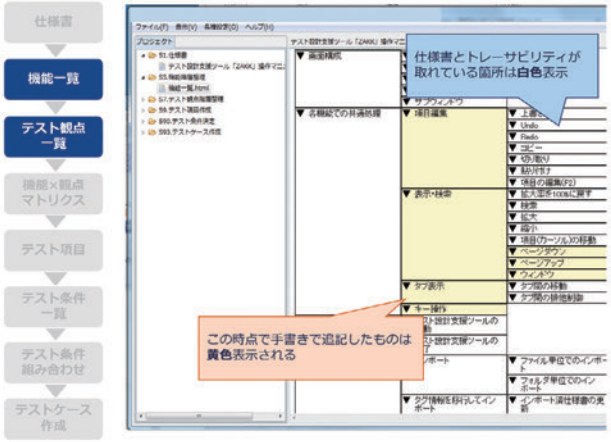


図5

図6



機能一覧とテスト観点一覧の作成が終わったら、自動で作成されるテスト観点マトリクス上で設計者が機能とテスト観点の対応付けを行うことで、テスト項目が自動で作成されます（図7）。

テスト項目に対してテスト条件を選定し、組み合わせ方法を指定すると、テストケースが自動生成されます（図8）。テスト項目一覧・マトリクス、テストケースは、Excel で編集可能な xml 形式でエクスポートすることができます。

図7

図8

4.「TESTRUCTURE」の利用メリット

「TESTRUCTURE」を利用することによるメリットをまとめると以下になります。

① 方法論に基づき順序立ててテスト設計を行えるように各機能が実装されています。テスト設計のやり方、成果物の形式が標準化され、担当者によるバラつきを抑制します。

② テスト対象の機能およびテスト観点を構造的に整理・可視化することで、テスト設計段階での考慮漏れを防止します。機能とテスト観点をマトリクスで整理でき、レビュー時にも理解しやすい形にしています。

③ テストベースにタグ情報を付与してテスト設計の作業を進めることで、テストベースとテスト設計成果物のトレーサビリティが自然と確保されます。テストベース更新時には変化点をツールが自動的に抽出し、該当する項目が強調表示されます。強調表示を参考にすることで、テストベース変更による影響箇所の見落としを防ぎます。

④ テスト対象製品のテスト設計で汎用的に利用される機能一覧やテスト観点一覧はテンプレートとして登録でき、必要なときに再利用できます。エンジニアの個人スキル依存度を低減し、テスト設計の品質向上・作業効率向上に繋がります。

5. ここが業界初!

業界初と銘打つポイントは、「テストベースと成果物間のトレーサビリティを確保しながら、テスト対象の仕様の理解からテストケース作成までを一気通貫に行うことができる」という点です。

要件管理ツールなど、トレーサビリティを確保する機能を持つツールはすでに存在しますが、テスト設計の作業を行うことで自然とテストベースと成果物とのトレーサビリティが取れるという点は、他のツールにはない大きな特徴です。

また、テストベースに色ペンでメモ書きするという方法、テスト観点を階層構造で整理するという方法、機能とテスト観点のマトリクスでテスト設計を行うという方法は、既存の方法論などで使われていますが、それらを統合して、仕様の理解からテストケース作成まで行えるテスト設計専用の支援ツールは、調べた限りですが世界にも見当たりません。

これらの新規性を根拠に、「TESTRUCTURE」の各機能について特許を取得しています。
（特許第 6148362 号）

6. むすび

方法論の必要性と、効率的にテスト設計を行うための支援ツールについてご紹介させて頂きました。「TESTRUCTURE」の裏には方法論によって整理された思考の流れがあることをご理解頂けたかと思います。方法論もツールも決して「銀の弾丸 *1」ではありませんが、方法論のベースとして示した考え方やツールを利用することで、より良いテストを実施しテスト対象の品質を高めるための取り組みを今後も進めてまいります。

最後に、今回ご紹介した「TESTRUCTURE」にご興味を持たれた方は、まずはトライアルをしてみませんか?お手数ですが当社までご連絡頂くか、ホームページより是非お申込みください。

*1 銀の弾丸：ソフトウェアの生産性において格段の向上を一気にもたらすようなプログラミング技法や理論のたとえ。

広告

テスト設計支援ツール

TESTRUCTURE テストラクチャー

（特許第6148362号）

VERISERVE

―― 高品質テスト設計の決定版！

業界初！テスト分析／設計支援プラットフォーム

1 プロセスの標準化

- ツールが規定するプロセスに従って作業することで、プロセスや成果物が標準化され、テスト設計の品質のバラつきを抑制

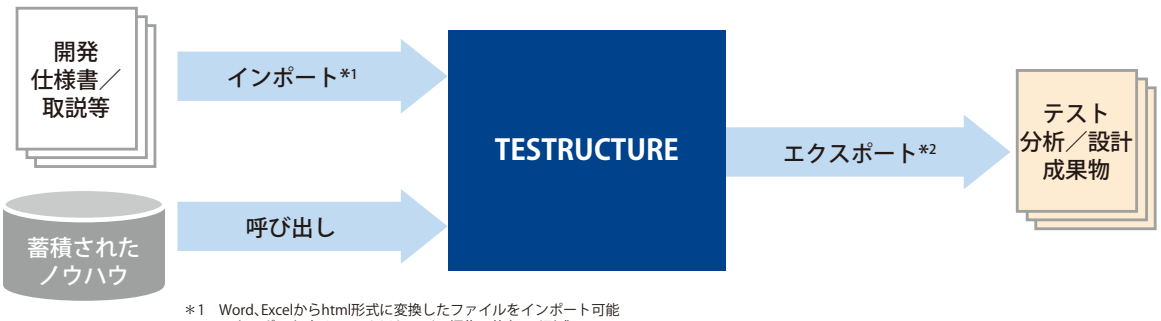
2 ノウハウの可視化

- 汎用的に利用される機能やテスト観点をノウハウとして蓄積し、いつでも参照可能に
- テスト観点を可視化することで、各エンジニアのスキルへの依存を低減し、テスト設計の品質向上を実現

3 高品質化

- 蓄積した情報を活用し、機能やテスト観点の抽出漏れを防止
- 仕様書と設計成果物のトレーサビリティを確保。仕様変更時の影響範囲の修正漏れを未然に防止
- 標準化・可視化により、成果物のレビューが容易に

高品質なテスト分析／設計を実現



＊1　Word、Excelからhtml形式に変換したファイルをインポート可能
＊2　エクスポートするファイルはExcelで編集可能なxml形式

エンジニアの思考に沿った使い勝手

テスト分析／設計に特化した機能

ノウハウの蓄積と活用

ライセンス形態

年間ライセンス契約

※PC1台ごとに1ライセンスが必要

VERISERVE NAVIGATION（ベリサーブナビゲーション）
2017年10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