ソフトウェア品質は“ユーザー主導”の時代へ

― タイムリーかつ〝納得感〟のあるシステム構築に重要な「受入れテスト」フェーズ ―

マイナビニュース「ソフトウェア品質」サミット
2013/6/26（水）

電気通信大学　大学院情報理工学研究科
総合情報学専攻 経営情報学コース
西　康晴（Yasuharu.Nishi@uec.ac.jp）

自己紹介

- 身分
  - ソフトウェア工学の研究者
    - » 電気通信大学 大学院情報理工学研究科 総合情報学専攻 経営情報学コース
    - » ちょっと「生臭い」研究／ソフトウェアテストやプロセス改善など
  - 先日までソフトウェアのよろず品質コンサルタント
- 専門分野
  - ソフトウェアテスト／プロジェクトマネジメント／QA／ソフトウェア品質／TQM全般／教育
- 共訳書
  - 実践ソフトウェア・エンジニアリング／日科技連出版
  - 基本から学ぶソフトウェアテスト／日経BP
  - ソフトウェアテスト293の鉄則／日経BP
- もろもろ
  - TEF: テスト技術者交流会 / ASTER: テスト技術振興協会
  - WACATE: 若手テストエンジニア向けワークショップ
  - SESSAME: 組込みソフトウェア管理者技術者育成研究会
  - SQiP: 日科技連ソフトウェア品質委員会
  - 情報処理学会 ソフトウェア工学研究会 / SE教育委員会
  - ISO/IEC JTC1/SC7/WG26（ISO/IEC29119 ソフトウェアテスト）

ASTER

SESSAME

SQiP
Software Quality Profession

TEF: Software Testing Engineer's Forum

- ソフトウェアテスト技術者交流会
  - 1998年9月に活動開始
    - » 現在1800名強の登録
    - » MLベースの議論と、たまの会合
  - http://www.swtest.jp/forum.html
  - お金は無いけど熱意はあるテスト技術者を
    無償で応援する集まり
  - 「基本から学ぶソフトウェアテスト」や
    「ソフトウェアテスト293の鉄則」の翻訳も手がける
    - » ほぼMLとWebをインフラとした珍しいオンライン翻訳チーム
  - 技術別部会や地域別勉強会が実施されている
    - » プリンタ、Web、AVなど
    - » 東京、関西、九州、東海、札幌など
    - » TestLinkというオープンソースツールの日本語化部会もある

ASTER: Association of Software Test EngineeRing

- ソフトウェアテスト技術振興協会
  - テストを軸にして、ソフトウェア品質向上に関する教育や調査研究、普及振興を行うNPO法人
    - » 2006年4月に設立／理事・会員は無給
  - ソフトウェアテストシンポジウム（JaSST）を開催している
    - » 実行委員は手弁当／参加費は実費＋α
    - » 毎年4Qに東京で開催／今年はのべ約1,800名の参加者
    - » 今年は関西・四国・北海道・九州・東海・新潟・東北でも開催／会場はほぼ満席
  - ソフトウェアテストの資格試験（JSTQB）を運営している
    - » Foundation Levelは15,735名の受験者・8,227名の合格者
    - » Advanced Level（テストマネージャ）を2010年8月に開始・278/744名の合格
  - ソフトウェアテスト設計コンテストを開催している
    - » テスト設計の質の高さを競うコンテスト／今年は全国から21チームの参加
    - » 主要な成果物は無償で公開される／毎年テスト設計のレベルが向上している
  - 各地でソフトウェアテストの教育を行っている
    - » テストのスキル標準（test.SSF）をIVIAと共同で開発している
    - » セミナーや勉強会などを支援することで、地場の産業振興の定着を図る
  - アジア各国とテスト技術の交流（ASTA）を行っている
  - 先端技術を研究開発している：STE、智美塾・バグ分析・ツール・Wモデルなど

SQiP：Software Quality Profession

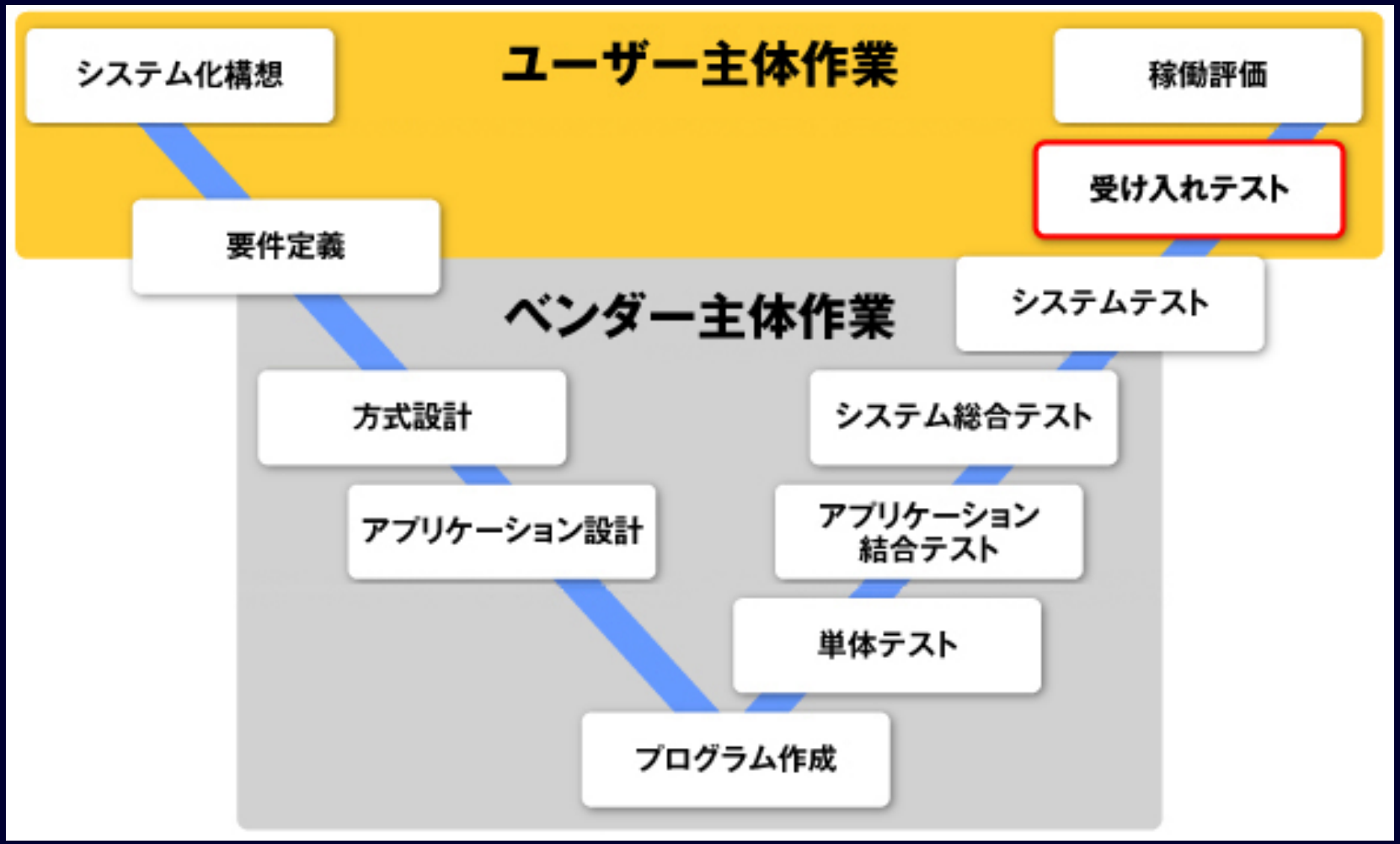
- 名称：
  - 日本科学技術連盟・ソフトウェア品質委員会
- 目的
  - SQiPは、ソフトウェア品質技術・施策の調査・研究・教育を通じて、実践的・実証的なソフトウェア品質方法論を確立・普及することにより、ソフトウェア品質の継続的な向上を目指す
- 3つの視点
  - ソフトウェア品質・実践・普及啓蒙
- 主軸とする活動
  - 1. 実践的・実証的なソフトウェア品質方法論の確立
  - 2. ソフトウェア品質方法論の普及促進・資格認定
  - 3. ソフトウェア品質向上のための国際協力の推進
- 活動方針
  - 1. ソフトウェア品質追究の重要性訴求
  - 2. 日本での実践的・実証的ソフトウェア品質方法論の形式知化
  - 3. グローバルな視野での活動
  - 4. 新しい課題へのチャレンジ

SQiP Software Quality Profession

講演の流れ

- ユーザー企業主導という考え方を身につける
- テストに必要な概念を自分たちなりに整理する
- テストの意図を把握する
- ユーザー企業ならではのノウハウを獲得する
- テスト自動化によってスピードを向上する

受け入れテストとは？



「ベンダー依存のシステム開発から脱却する“7つのポイント”」堀江弘志（豆蔵）
http://techtarget.itmedia.co.jp/tt/news/0906/04/news02.html

受け入れテストで困っているユーザー企業の悩み

- こんなことを悩んでいませんか？
  - 実のところ受け入れテストの丸投げをしてしまっており、その結果、問題が多発している
  - だからといって、自分たちで受け入れテストを全部遂行できる予算も人員も技術も無い
  - ベンダー企業にも問題は多々あるが、長い付き合いと暗黙の仕様理解があるため、すぐに発注先を変えるというわけにもいかない
- もう少し具体的にはこういう悩みだったりする
  - ベンダー企業が何を言ってるのか実は分かっていない
  - ベンダー企業が何を意図してテストしているのか分かっていない
  - 自分たちにユーザー企業としてのノウハウが無い
  - 受け入れテストに工数がかかりすぎて事業反映のスピードが遅い

悩んでいるユーザー企業は何をすればいいのだろうか？

- 世の中には予算にも人員にも技術にも恵まれたユーザー企業があるらしい
  - 海外の某ユーザー企業にはソフトウェア「品質」担当のVPがいる
    - 「情報システムの品質が企業の命運を左右するんだから、VPがいるのは当たり前じゃないかい？」
    - ユーザー企業として必要なテスト技術の技術開発をしている
  - 国内のミッションクリティカルなユーザー企業はガッチリテストを主導している
- しかし予算にも人員にも技術にも恵まれないユーザー企業はどうすればいいだろうか？
  - このまま丸投げをしてよく分からないままだと不幸が続くのは目に見えている
  - だからといって自分たちでテストを実施できるほどの予算も人員も技術も無い
  - 何かを変えなければいけない
- では、できるところから変えていきましょう
  - 考え方、テストの概念理解・知識獲得、テストの意図把握、ユーザー企業ならではのノウハウ獲得、テスト自動化によるスピード向上

丸投げとは何か

- 丸投げとは、何を委託しているかもよく分からず、
  委託先が何をしているかもよく分からずに、
  仕事を委託することである
  - 丸投げは、作業委託量の多さではなく、「納得度」の低さを意味している
  - ユーザー企業の納得度が低いと、
    残念ながらいつしかベンダー企業も質の低い仕事をするようになる
    - » 別にベンダー企業だってユーザー企業を出し抜いてやろうと意志があるわけではない
    - » ユーザー企業に納得させる必要の無い仕事は、
      ベンダー企業自身の納得度も低いため、質の低い仕事になりやすい
    - » 納得度の低い仕事でオカネをもらうには「頑張りました」というしかなく、
      その説明を一度始めてしまうと二度と納得度は上がらない
    - » 酷いベンダー企業に当たると、納得度の低さという足元を見られ、
      スキルの低いエンジニアを割り当てられ品質トラブルが多発したり、
      単価を引き上げられたりしてしまう

丸投げの本質は、納得感の欠如である

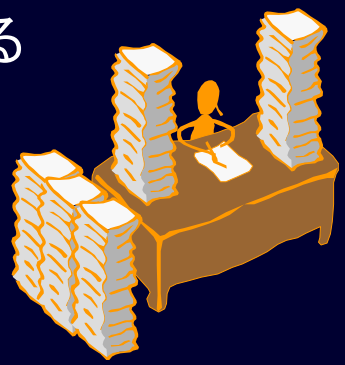
ユーザー企業が主導的に受け入れテストに関与する

- 受け入れテストの丸投げで発生している問題は、
  ユーザー企業が納得感を高める取り組みを進めていくと
  少しずつ解消されていく
  - すなわち、ユーザー企業が主導的に
    受け入れテストに関与していくことが必要である
    - » 受け入れテストの作業を全てユーザー企業が遂行せよ、と言っているわけではない
  - 受け入れテストの作業を委託するにせよ自分たちで遂行するにせよ、
    納得感が低ければ本質的に丸投げを同じ問題が発生してしまう
    - » 社外丸投げか社内丸投げか、の違いに過ぎない

- 何の納得感をどれくらいどう高めていくか、は
  ユーザー企業の予算や人員、技術に依存するので、
  大まかな構想をユーザー企業が主導的に考えなくてはならない
  - テストの概念理解・知識獲得、テストの意図把握、
    ユーザー企業ならではのノウハウ蓄積、テスト自動化によるスピード向上

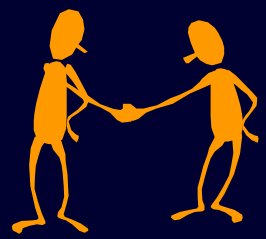
ソフトウェアテストの新しい国際標準：ISO/IEC/IEEE 29119

- UK提案の重量級のテストプロセスおよびテスト文書モデル
  - 5つのパートと1つの関連標準から構成される
    - » Part 1: 用語と概念、Part 2: テストプロセス、Part 3: ドキュメント、Part 4: 技法、Part 5: キーワード駆動テスト、ISO/IEC 33063: テストプロセスアセスメントモデル
  - テストプロセスが未成熟な組織が教科書として読む分にはよくできている
    - » きちんと適用するにはトレーサビリティ管理などが必要となる
  - IEEE829はもう更新されず、ISO/IEC/IEEE29119-3に一本化される
  - 現在FDIS（Final Draft International Standard）段階である
    - » 今年度中には国際標準として発行されるだろう
- 3つの特徴がある
  - 3階層モデル
    - » PJ横断的なテストポリシー・テスト戦略、PJごとのテスト計画、PJごとのテスト実行
  - リスクベースドアプローチ
    - » テストの意図や間引きのデメリットを明示的に検討する
  - 汎用的なテストプロセス記述
    - » 異なるテストタイプもテストレベルも同じ構造のプロセスモデルで記述している



ユーザー企業主導という考え方を身につける

- ユーザー企業にとってのISO/IEC/IEEE29119の考え方
  - ユーザー企業が全体的なテストポリシーやテスト戦略を定め、それをブレークダウンしていくテストのプロセスである
    - » PJ横断的なテストポリシーやテスト戦略に従って、プロジェクトごとのテスト計画を立て、テスト実行していく考え方のテストプロセスモデルである
    - » テストポリシーやテスト戦略からテスト計画、テスト設計へのトレーサビリティと共に、きちんとブレークダウンされているとユーザー企業が納得する必要がある
    - » テストケースの意図を把握し、間引きのデメリットを納得する必要がある
  - ユーザー企業がテストの実施をする必要はない（してもよい）
    - » あらゆることをガッチリ決めなくてはいけないわけではない
    - » 何でもかんでも網羅的にテストしなくてはいけないわけではない
  - ユーザー企業がベンダー企業を厳密に報告させて承認したり、単に監査して（見かけだけの）ガバナンスを達成するプロセスではない
    - » テストのゴールとして何を達成したいのか、どういうテストが必要になるのか、どういうリスクが発生すると（テストを間引くと）どういうまずいことになりどう対処するのか、などをユーザー企業とベンダー企業できちんと納得し合わないと上手くいかない
  - 受け入れテストで困っているようなユーザー企業は、テスト全体を自分たちが主導するという考え方をまず身につけるべきである
    - » 銀の弾丸（魔法の杖）のように29119を導入すれば上手くいく、と考えてはいけない



ユーザーの参画が必要

プロジェクト横断的なテストポリシー・テスト戦略

プロジェクトごとのテスト計画

受け入れテストのみユーザーの参画が必要

テストレベル・テストタイプごとのテスト計画

テストレベル・テストタイプごとのテスト設計

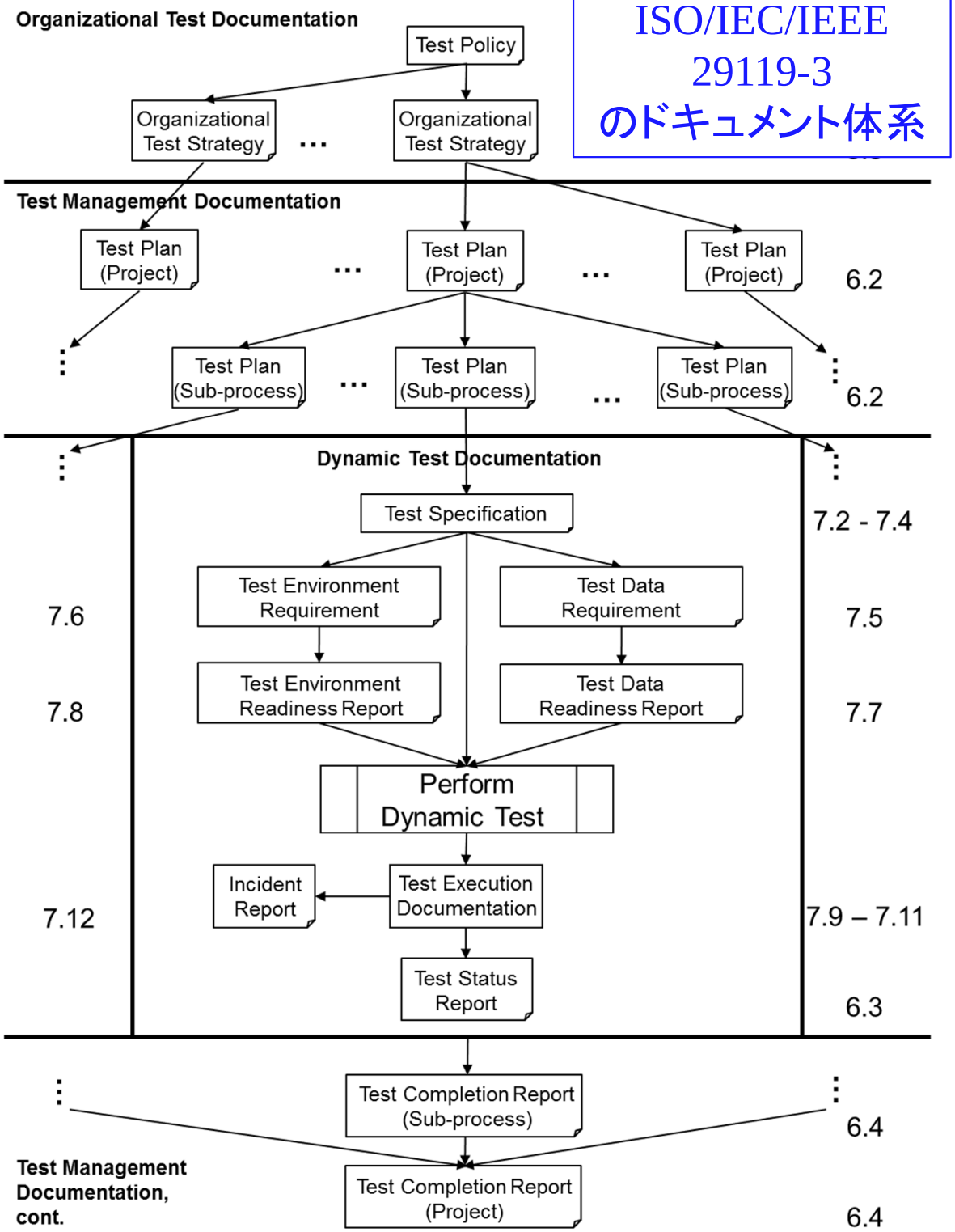
テストレベル・テストタイプごとのテスト実行

テストレベル・テストタイプごとのテスト報告

ユーザーの納得が必要

プロジェクトごとのテスト報告

ISO/IEC/IEEE 29119-3 のドキュメント体系



講演の流れ

- ユーザー企業主導という考え方を身につける
- テストに必要な概念を自分たちなりに整理する
- テストの意図を把握する
- ユーザー企業ならではのノウハウを獲得する
- テスト自動化によってスピードを向上する

テストに必要な概念を自分たちなりに整理する

- 自分たちで使っている用語の意味や違いが分かるだろうか？
  - テストケース、テスト項目、テストアイテム、テスト条件、テスト仕様、テストデータ
  - テスト対象、テストアイテム、テストターゲット、テスト環境、テストプログラム
  - テスト計画、テスト戦略、テスト仕様、テストレベル、テストタイプ、テスト技法
- どうやったら上手くテストできるかを学ぶ前に、
  どういう考え方や概念、用語があるのかを身につける
  - JSTQBテスト技術者認定制度（http://www.jstqb.jp/）のシラバスや用語集は無償なのによく整理されているので便利である
  - JSTQBで定められた用語名や用語定義を振り回すのではなく、自分たちやベンダー企業が馴染む言葉遣いを極力活かしつつ、混同している用語や新しい概念を整理するための土台としてJSTQBを使う
  - 用語定義や暗記にこだわりすぎないようにする

テストに必要な概念を自分たちで整理するのが
ユーザー主導の受け入れテストの第一歩である

コストをかけずにどうやって悩みを解決していくか

- まずWebサイトや書籍などでほとんどコストをかけずに勉強する
  - 詳しく説明を聞きたくなったら数千円～数万円のコストをかけてセミナーに参加する
    - » 正直なところ、高いセミナーだから良いとは限らない
- コストはかけずに熱意と(終業後の)時間を費やして、無償のオープンな勉強会に参加する
  - テストの方法論などを学ぶと、自組織向けの方法論をつくっていく助けになる
  - ちょうどよい勉強会が無ければ誰かに開催を頼んだり自分で主催してもよい
    - » ASTERがSTE(エンプラ部会)を開催している:query@aster.or.jp
  - 知り合いができると懇親会やTwitterなどでふんわり質問しやすくなる
    - » 友達になれば、ツールやコンサルの良し悪しも聞くことができる、かも

- コストはかけずに熱意とより多くの時間を費やして、オープンソースのツールを使って自動化にチャレンジしてみる
  - ある程度効果が見えてきたら、数十万円～数百万円のコストをかけて商用のツールにチャレンジするとありがたみが実感できる
- 組織としてガッツリ取り組みたくなったら、数十万円／月のコストをかけてコンサルタントに指導を依頼する
  - もちろん自分たちだけで進められるのなら、それに越したことはない

まずはどこから勉強するか

- Webサイトで無償で勉強するのが手軽である
  - JSTQBテスト技術者認定制度
    - » http://www.jstqb.jp/
    - » 受験料は決して安くない（¥21,000/回）
    - » シラバス（学習事項）や用語集が割と充実しているのに無償で配布している
  - JaSST（ソフトウェアテストシンポジウム）
    - » http://www.jasst.jp/
    - » 全国津々浦々で開催している（東京、関西、東海、北海道、九州、東北、四国、新潟）
    - » 各社の事例発表のスライドが無償でダウンロードできる
  - ASTERテストツールWG
    - » http://www.aster.or.jp/business/testtool_wg.html
    - » 「テストツール丸わかりガイド」という小冊子を無償で配布している
  - IPA/SEC（情報処理振興機構・旧ソフトウェア・エンジニアリング・センター）
    - » http://sec.ipa.go.jp/
    - » 様々なガイドをPDF版は無償で配布している
    - » 玉石混淆なので鵜呑みにしてはいけない

講演の流れ

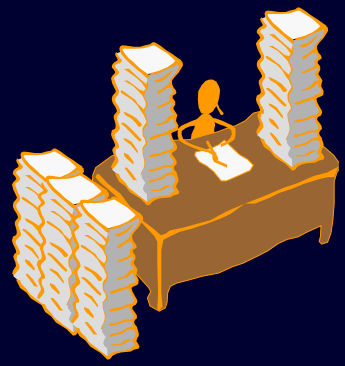
- ユーザー企業主導という考え方を身につける
- テストに必要な概念を自分たちなりに整理する
- テストの意図を把握する
- ユーザー企業ならではのノウハウを獲得する
- テスト自動化によってスピードを向上する

テストの意図を把握できているだろうか？

- ベンダー企業の作ってきたテスト計画やテスト仕様、テスト設計、テストケースなどが、どういう意図で作られたのかをきちんと理解しているだろうか？
  - テスト技術の低いベンダー企業のよくあるセリフ①:「要件は網羅しています」
  - テスト技術の低いベンダー企業のよくあるセリフ②:「シナリオは網羅しています」
  - テスト技術の低いベンダー企業のよくあるセリフ③:「機能は網羅しています」
  - テスト技術の低いベンダー企業のよくあるセリフ④:「納期が来たから終わりです」
- テスト技術の低い組織は、CPM法や固定3レイヤー法を駆使してかなりの工数を費やして大量のテストケースを作成するが、実はきちんと意図を把握しているわけではない
  - ベンダー企業に「意図をきちんと把握しろ」と言ってもダメなことは目に見えている
    - » ベンダー企業自身が意図を把握しておらず、把握していないことも分かっていない
  - ユーザー企業が意図を把握しようとしない限り、ベンダー企業は変わらない
  - だからといって自分たちで全て受け入れテストができるわけでもないし、テストケースを全て解釈しレビューできるわけもない

CPM法（コピー＆ペースト＆モディファイ法）

- CPM法とは
  - 仕様書の文章をコピーしてExcelにペーストし、
    語尾を「～する」から「～すること」に変更することで
    テストケースを作成する方法
  - 複数のテストケースをコピー＆ペーストし、
    置換コマンドを使って機能名などをまとめて変更することで
    テストケースを増殖する方法



- CPM法の問題点
  - テストケースがどんどん増殖していき、収集がつかなくなる
  - テストケースの趣旨が分からないためテストの終了判定ができず、
    頑張って徹夜して消化できたところまでで勘弁してもらうことが常態化する
  - 何のために存在しているのか分からないテストケースが
    保守や機能追加の度に増えていき、怖くて誰も減らせない

固定3レイヤー法

- 固定3レイヤー法とは
  - テストケースを大項目・中項目・小項目のフォーマットに従って設計していく方法

大項目	中項目	小項目	テストケース
機能レベル1	機能レベル2	機能レベル3	機能レベル10
機能レベル1	機能レベル8	機能レベル9	機能レベル10
機能	環境	データ	機能＋環境＋データ
機能	データ	GUI	機能＋データ＋GUI
機能	データ	前提条件	機能＋データ
機能	画面	操作	画面＋操作
テストカテゴリ	機能	データ	機能＋データ
組み合わせ	機能①	機能②	機能①×機能②

固定3レイヤー法の問題点

- 詳細化のレベルが飛んでいる・揃わないので網羅できない
  - レイヤー間の距離が大きくなるほど漏れが発生しやすい
  - エンジニアによって偏った詳細化をしてしまう
  - テストケース群ごとに詳細化の偏りにばらつきがある
- 異なるテスト観点の組み合わせを詳細化だと考えてしまう
  - 機能＋環境＋データ、機能＋データ＋GUI
- テスト設計で考慮する必要の無いものが入ってしまう
  - 機能＋データ＋前提条件
- 異なるテスト観点にぶら下げているので網羅できない
  - 機能＋画面＋操作
    - » 機能ごとに画面を操作してしまうので、画面遷移が網羅できない
- テスト観点の詳細化を行わない
  - テストカテゴリ＋機能＋データ
    - » 負荷にも色々あるはず．．．。
- 組み合わせテストを押し込んでしまう
  - n元表を使うべきである

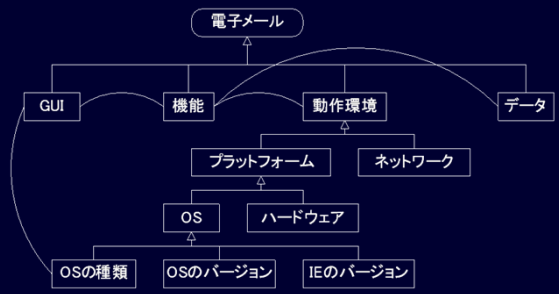
テストの意図を把握する

- CPM法は要件網羅や機能網羅と言いながら、
  実のところ特に意図無く単に書き写しているだけである
  - なのでCPM法と固定3レイヤー法はしばしば併用される
- 固定3レイヤー法の大項目・中項目・小項目も
  きちんと意図を表しているとは言い難い
  - 本来、固定3レイヤー法の大項目・中項目・小項目とは、
    そのテストケースの意図を表しているはずである
  - しかし、「大・中・小」という区分けが余りにも便利なため、
    いつしか意図がよく分からない区分になってしまっていることが多い
  - Excelの場合、結局どんな大項目（と中項目）があるのかを一覧しにくい
  - 大項目から中項目、中項目から小項目が網羅されたのかどうか分からない

手間をそんなにかけずに
もう少しきちんとテストの意図を把握したい

マインドマップでテストの意図を把握する

- マインドマップのツールはフリーのものがたくさんある
  - FreeMind: http://sourceforge.jp/projects/freemind/
  - Xmind: http://jp.xmind.net/



- いきなり込み入ったことは難しいので、
  まずテスト対象を中心に置き、
  固定3レイヤー法の大中小項目をそのままノードにしてみる
  - 個々のテストケースは書かない
  - 納得いく図にはなっていないだろう
- マインドマップのノードにあたるものを「テスト観点」と呼び、
  マインドマップ全体を「テスト観点図」と呼ぶ
  - 我々はNGTという記法とVSTePというプロセスモデルを定義しています
    - » NGT（Notation for Generic Testing）：テスト開発のための記法
    - » VSTeP（Viewpoint-based Test Engineering Process）：テスト開発プロセス
    - » 詳しく知りたい方は http://qualab.jp/vstep/ に資料があります
    - » Twitterなどで希望して賛同者がいれば無償の勉強会も可能です

テスト観点図でテストの意図を整理する

- 次に、テスト観点図をもう少し整理してみる
  - ルール1：親と子の関係がおおむね納得できるまで、ノードを移動させる
    - » 子は親の一種（継承／is-a関係）、子は親の一部（合成集約／has-a関係）、子は親の性質（属性化関係）が代表的である
      - ・ 継承関係の例：環境（親）←OS（子）、合成集約関係の例：サーバ（親）←メモリ（子）、属性化関係の例：OS（親）←OSのバージョン（子）
    - » なるべく、親を評価するために子という手段が必要（目的手段関係）、親の条件で動かすと子というふるまいをする（原因結果関係）、の関係を減らす
      - ・ 目的手段関係の例：セキュリティ（親）←ログイン（子）
      - ・ 原因結果関係の例：負荷（親）←性能（子）
    - » 親子が離れていると感じたら、その間にノードを増やす
      - ・ 大中小の3階層ではなく、必要に応じて階層を増やして構わない
  - ルール2：親との関係が同じ子を兄弟と考える
    - » 親と子の間の線に、親子関係を示す名前をつけると便利である
      - ・ 親子関係の名前は<<親子関係>>のように書く
      - ・ 同じ親に異なる兄弟群がぶら下がることになることも多い
  - ルール3：兄弟はなるべく漏れなくダブりないようにする
    - » 漏れてるな、と思ったらノードを追加してよい
    - » 漏れがありそうなところは「その他」という兄弟をつくったりして、そのノードに漏れがありそうなことを明示する
    - » 絶対に漏れがないと確信できるノードは囲みをつけたりマーカーをつけるとよい

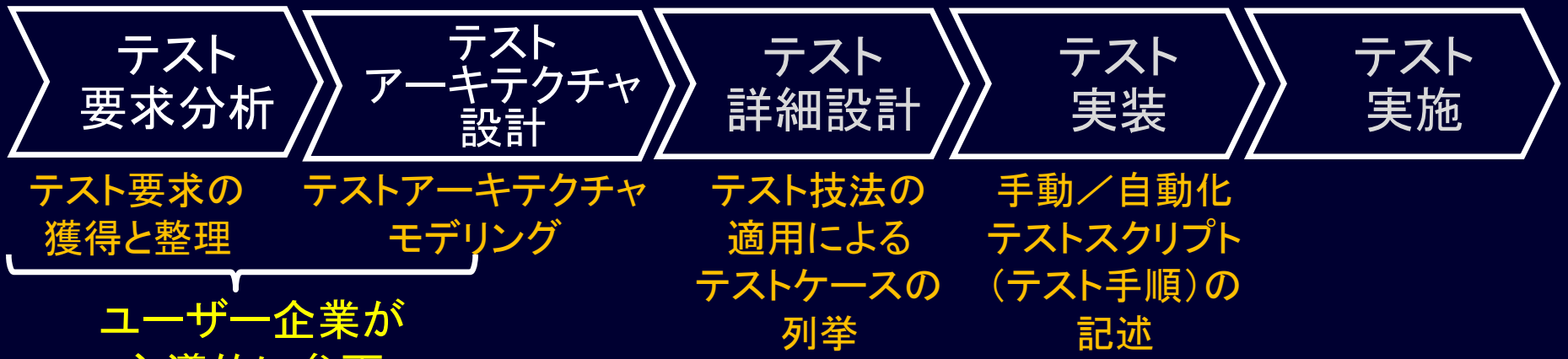
テストの意図を整理してみると色々なことに気付く

- 固定3レイヤー法をそのまま描き直しただけのテスト観点図と整理したあとのテスト観点図とを比較してみると、色々なことに気付くだろう
  - 機能やデータ、正常系業務フローなどに偏ったテスト観点図になっているだろう
  - 固定3レイヤー法では大中小項目に表れないが実は必要なテスト観点がある
  - 親子関係がきちんとつながっていないため網羅しているかどうかが分からなかった
- 受け入れテストでテスト観点図が描けたら、システムテスト・結合テスト・単体テストとテスト観点図を増やしていき、テスト全体の意図をきちんと把握し納得できるようにする

  - テスト観点図が大きくなったら、テストレベル（受け入れテスト、システムテストなど）やテストタイプ（負荷テストや環境変更テスト）ごとに図を分割すると分かりやすい
  - テストレベルやテストタイプをまたいで同じテスト観点が出てきても構わないが、多くの場合意味合いが異なるのでよく検討してみる必要がある

テストの意図の把握にはそれほどコストがかからない

- テスト全体を遂行することに比べると、
  テストの意図の把握にはコストがかからない
  - 一度テスト観点を整理しておくと、ユーザー企業側で使い回しがきくことが多い
  - 細かい日本語の文章を書かないので楽だし、テスト観点の整理に集中できる
  - ACC分析は分かりやすく整理する方法の一つである
    - » A(システムが達成したい性質)、C(構成要素や機能、業務)、C(変化／網羅できるもの)の3つの視座からテスト観点を挙げる方法
    - » 「テストから見えてくるgoogleのソフトウェア開発」に紹介されている
  - VSTePや29119-2のように、テスト観点からテストケース、テスト手順までトレーサブルに設計・実装する仕組みを作っておくと、
    ユーザー企業が納得したテストの意図が確実に反映される



講演の流れ

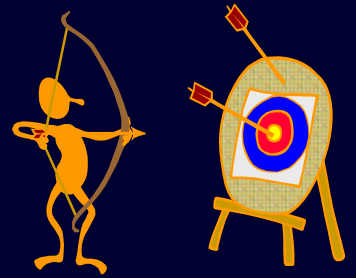
- ユーザー企業主導という考え方を身につける
- テストに必要な概念を自分たちなりに整理する
- テストの意図を把握する
- ユーザー企業ならではのノウハウを獲得する
- テスト自動化によってスピードを向上する

ユーザー企業ならではのノウハウを獲得する

- ユーザー企業ならではのノウハウを蓄積できているか
  - 業務知識、業務知識、業務知識！
    - » しかし正常系の業務フローは要件化されているし、異常系もそれなりに要件化されている
    - » とはいえ、なんだか似たようなところに不具合が多発している気がする
  - 「悪さの知識」を体系的に蓄積している組織は少ない
    - » 悪さの知識：そうやると上手くいかないよ、という知識
    - » ものごとは一見、良さの知識だけでやれてしまうような気がするが、実のところ、良さの知識と悪さの知識の両方が揃っていないと上手くいかない
    - » システム開発の場合、気をつけてはいても嵌まってしまうような仕様や設計、実装、言語仕様、環境などの罠というものがある
  - 「悪さの知識」をうまくパターン化して蓄積し、受け入れテストの質を上げたい
    - » 発生頻度は低いものの種類が多い例外的な業務に対する検討は漏れやすい
    - » 複雑だったり入り組んでいたり、思い込みを誘発するような業務上の「いやらしいところ」というのは、どの業務ドメインにも存在する
      - 業務だけでなく関連法規や制度にも「いやらしい」ところがある
    - » 業務上の「いやらしいところ」は正しく仕様化しても、設計や実装でバグが入り込みやすい
    - » 我々は不具合モードとして体系化を進めている

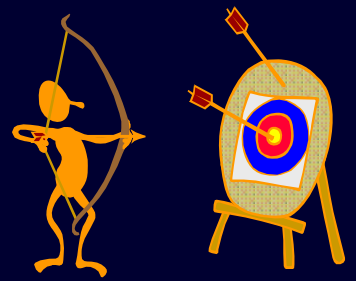
不具合モードによるピンポイントテストの設計

- 不具合が作り込まれそうな「弱点」（不具合モード）を狙って
  ピンポイントでテストを設計する方法
  - 過去の不具合を蓄積・分析し、不具合が作り込まれそうな箇所を推測することで、
    ピンポイントにテスト設計を行う
    - » よく発生する、同じようなメカニズムで作り込まれたと思われる
      不具合を集めて分析することで、そのメカニズムを推定し、
      仕様や設計、コードのパターンを導き出してテストを設計する
    - » 当該工程だけでなく、後工程でバグを作り込む原因となりうる部分もパターン化する
  - 開発者がどう間違えるかに着目して開発者が陥りやすい「罠」をパターン化する
    - » 人間の思考の誤謬と、誤謬を引き起こしやすいパターンをセットで明らかにする
      - 例）後から追加された例外的な仕様は、設計漏れを引き起こしやすい
      - 例）優先順位の表現で1（高い）～5（低い）と　5（高い）～1（低い）が混在すると混同しやすい
      - 例）異なるサブシステムでリソースの確保・解放を行うとリークしやすい
      - 例）「備考」にはバグが多い
    - » 同じパターンのバリエーションをツリー状に整理する
    - » 我々は誤謬を引き起こしやすいパターンを
      「ロジカル・アフォーダンス」と名付けている
      - 認知科学・認知心理学分野との学際領域になる



不具合モードによるピンポイントテストの設計

- 不具合が作り込まれそうな「弱点」(不具合モード)を狙ってピンポイントでテストを設計する方法
  - テスト設計だけでなく、レビューの指摘項目の設計や開発ガイドラインの改善にも活かすことができる
    - » テストと開発、ユーザー企業とベンダー企業のコミュニケーションを密にするツールにもなる
    - » 自分たちの業務の弱点を得られるので、業務プロセスの改善にも活かすことができる
    - » テストケースあたりの不具合の検出率は向上するが、網羅するわけではないので品質保証には適さない
    - » 「またやっちゃった」という不具合を防ぐことができる
    - » 限られた時間で多数の不具合を検出するための手法に過ぎず、テスト漏れを防ぐ手法ではない
  - 不具合分析(なぜなぜ分析)のレベルが低いと、効果的な不具合モードを得ることができない



業務やテスト観点とガイドワードを組み合わせて検討する

強度	強い	弱い		時期	早く	遅く	
数量	多い	少ない		時間	長く	短く	
	増加	減少		間隔	広く	狭く	
	余分	不足		期間	長く	短く	
	ある	ない	ゼロ	順序	さきに	あとで	
種類	多種	小種			同時に	並行に	
規模	大きい	小さい			早く	遅く	
距離	遠い	近い			逆転	反復	挿入
速度	速い	遅い		タイミング	早く	遅く	
	高速	低速		拡張	広く	狭く	
	緩	急			多く	少なく	
設定	許容	禁止		頻度	多く	少なく	
	必須	任意			高く	低く	
範囲	長い	短い		程度	いつも	たまに	
	上限	下限		回数	多く	少なく	
	広い	狭い		接続	つなぐ	切り離し	切り替え
	最大限	最小限		方向	上へ	下へ	
頻度	高い	低い		負荷	高い	低い	
	多い	少ない			超過	不足	限界
金額	大きい	小さい		位置	高く	低く	
					遠く	近く	

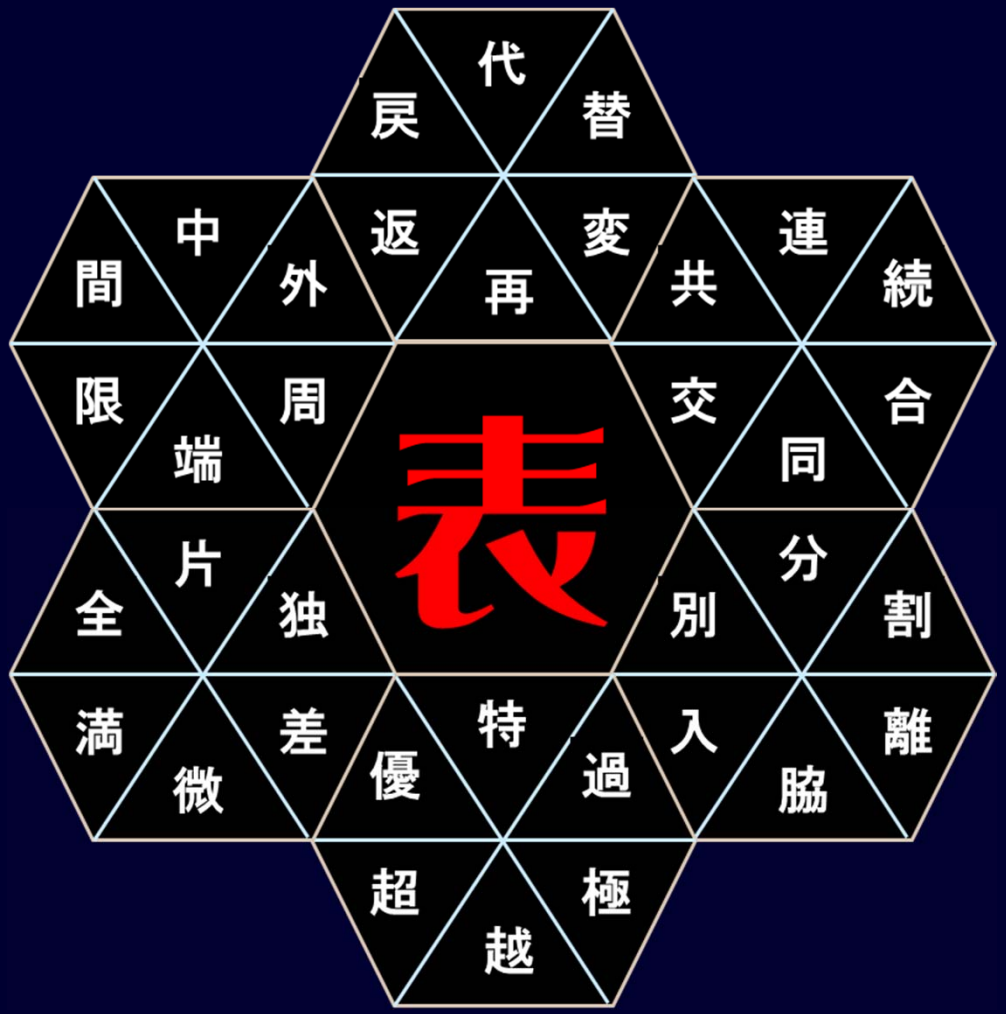
鈴木三紀夫さん（MRTコンサルティング）、秋山浩一さん（富士ゼロックス）がご作成

業務やテスト観点とガイドワードを組み合わせて検討する

	原則	例外			正常	異常	準正常
	全体	部分			通常	非通常	
	全部	一部			通常	例外	代替
	主	従			定期	臨時	
	正	誤			通例	異例	常例
	無事	有事			平時	非常時	緊急時
	公正	不正			定常	可変	
	任意	強制			尋常	非常	
	基本	詳細	主要		一般	特別	
	完了	未完			普通	特殊	
	有効	無効			並	特異	
	起動	停止					

鈴木三紀夫さん
（MRTコンサルティング）、
秋山浩一さん
（富士ゼロックス）
がご作成

業務やテスト観点と「意地悪漢字」を組み合わせて検討する



鈴木三紀夫さん（MRTコンサルティング/Twitter:@mkoszk）がご作成

美しくない仕様は「罠」が多い：富士通の7つの設計原理

①単純原理
- 可能な限りシンプルにすること

②同型原理
- 同じものは同じように実現すること
- 例外を設けないこと

③対称原理
- 対称にすること
- 対になるものは対にすること
- バランスを崩さないこと

④階層原理
- 階層関係、主従関係、前後関係、本末関係などを崩さないこと
- 層に穴を空けないこと

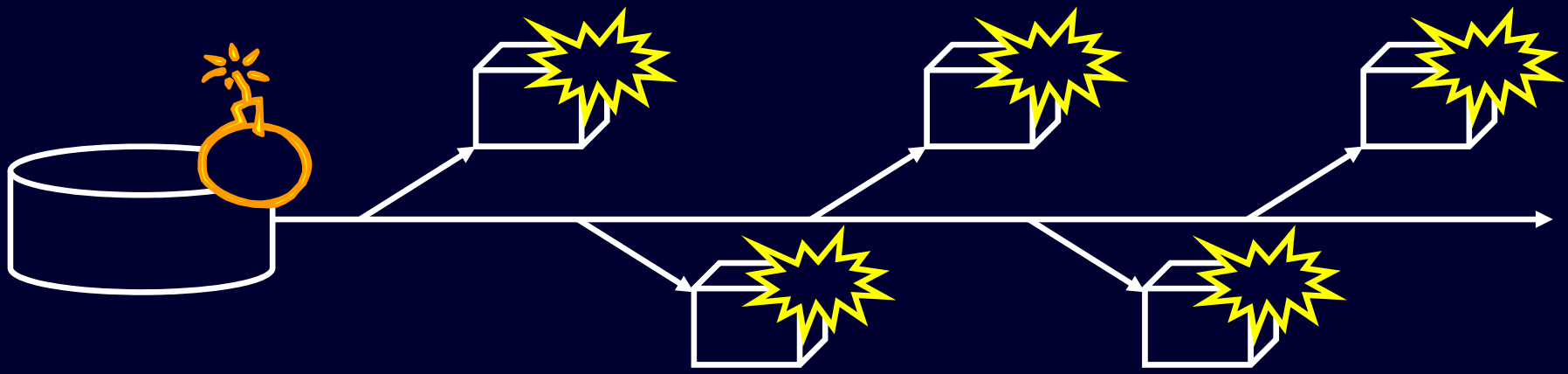
⑤線形原理
- 特異点や閾値、組み合わせによる特殊な振る舞いが無いこと

⑥明証原理
- ロジックは直感的に分かりやすくすること
- 分からないものやふるまいがないこと
- おかしなものを受け取らない、先に送らない、無視しない、勝手に処理しないこと

⑦安全原理
- 必然性のないところや曖昧なところは安全サイドに設計しておくこと

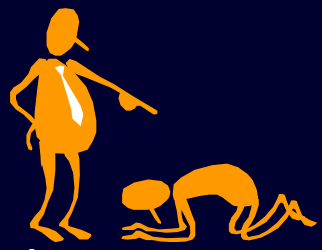
保守・機能追加における頻発バグを防ぐ

- 保守や機能追加では特定の種類のバグが
  保守や機能追加の度に発生することがある
  - 関連法規や制度、業務、流用母体に不具合モードもしくは
    不具合モードの芽があるため、保守や機能追加の度にバグを作り込んでしまう
    - » 不具合モード「例外」：法規対応、多仕向地、機能複合型製品
    - » 流用母体の設計にそもそも難がある場合も多い
    - » 仕様書群の構造そのものに不具合モードがあったりもする
  - 関連法規や制度、業務、流用母体の不具合モードに着目することで
    再発バグを効率的に防ぐことができる



不具合分析のアンチパターンと効果の薄い対策

- Vaporization（その場限りの簡単なミスとして片付けてしまう）
  - コーディングミス：スキルを向上させるよう教育を行う
  - 考慮不足：しっかり考慮したかどうかレビュー時間を増やし確認する
  - うっかりミス：うっかりしていないかどうか気を引き締めてダブルチェックする
- Exhaustion（不完全な実施や単なる不足を原因としてしまう）
  - ちゃんとやってない：レビュー時間を増やし確認したり書類や承認印を増やす
  - テスト不足・レビュー不足：名ばかりの工程を追加したり単に時間を増やす
  - スキル不足：スキルを向上させるよう教育を行う
  - 経験不足：経験を補うためにスキルを向上させるよう教育を行う
- Escalation（上位層に責任を転嫁してしまう）
  - マネジメントの責任：トップを含めた品質文化を醸成する
- Imposition（外部組織に責任を転嫁してしまう）
  - パートナー企業の責任：パートナー企業を含めた品質文化を醸成する
- Culturalization（文化的問題に帰着してしまう）
  - 品質意識／品質文化の欠如：全社的な品質文化を醸成する



講演の流れ

- ユーザー企業主導という考え方を身につける
- テストに必要な概念を自分たちなりに整理する
- テストの意図を把握する
- ユーザー企業ならではのノウハウを獲得する
- テスト自動化によってスピードを向上する

テスト自動化によってスピードを向上する

- テストが原因となってビジネススピードに追従できない
  - ビジネススピードに合わせた機能追加や仕様変更に追従できない
    - » システムを作るのに時間がかかるのはある程度仕方が無い
    - » テストで改めて時間がかかるという状況を何とかしたい
- どのテストを変更すればいいかが分からない
  - テストの意図を明確化し、トレーサビリティを確保するとかなり改善される
- 過去に発生したものと似たような不具合はさっと見つけたい
  - 不具合モードなどでテストケースの質を高めるとよい
- 単なる機械的作業のように実施しているテストがある
  - ツールを使おうとしてもキャプチャ＆リプライが必要なので
    機能追加や仕様変更に対して時間がかかってしまう
- それなりの金銭面・努力面の先行投資をすると、
  機械的作業のようなテストは自動化できる可能性が高い
  - 自動化して効果がでると納得できるまでは自動化しない方がよい
    - » まずはオープンソースのツールを使い、DDTやKDTについて
      具体的な活用場面や効果、コストをイメージできるようになるとよい
  - ベンダー企業と協調して自動化できると色々と楽になる

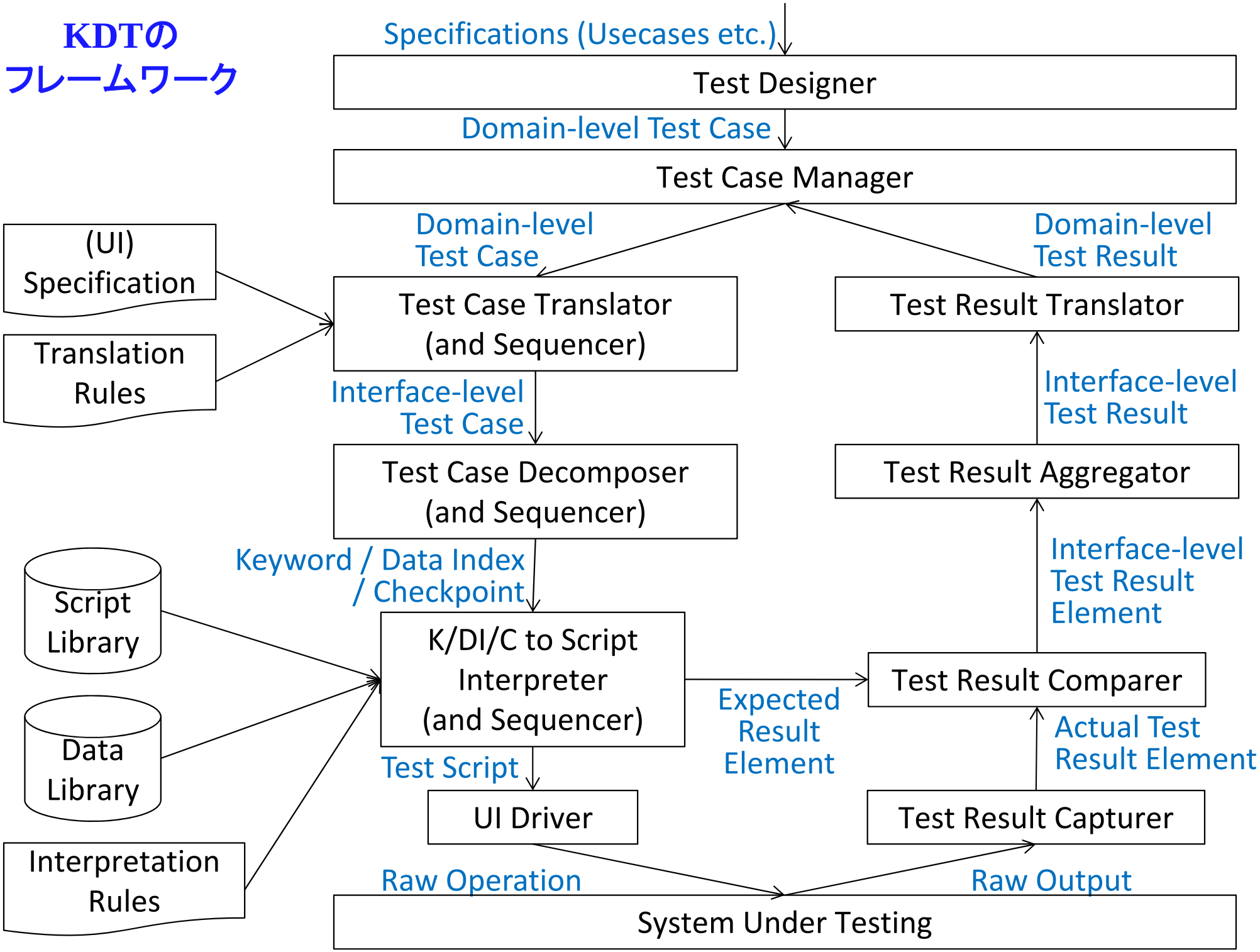
データ駆動テスト・キーワード駆動テスト

- 自動テストツール（自動操作ツール）では、
  テストケースをデータも操作も全てそのまま
  キャプチャしてプレイバックしなくてはならなかった
  - キャプチャの手間が必要となるため、一般的には、
    全く同じテストケースを4回以上実行しないと元が取れないと言われていた
  - 多くのバリエーションをテストするには手間がかかりすぎた
- そこで「データ駆動テスト（DDT）」が生み出された
  - 全く同じ操作でデータだけが異なる場合は、
    ツールが自動でデータの書かれたExcelファイルを読み込んで
    テストケースのバリエーションを自動で作成し実行してくれるようになった
  - しかし単純な操作の組み合わせのようなバリエーションは、自動化テストスクリプト
    そのものを変更しなくてはならないため依然として手間がかかる
- そして「キーワード駆動テスト（KDT）」が生み出された
  - 単純な操作（キーワード）をスクリプトライブラリと関連づけておくと、
    テストケースをキーワード名で書くだけで、自動化テストスクリプトを意識しなくても
    ツールがテストケースのバリエーションを自動で作成し実行してくれるようになった
  - しかしテストケースを操作レベルまで詳しく書かなくてはいけないため、
    機能追加や仕様変更にすぐに追従できるというわけではない

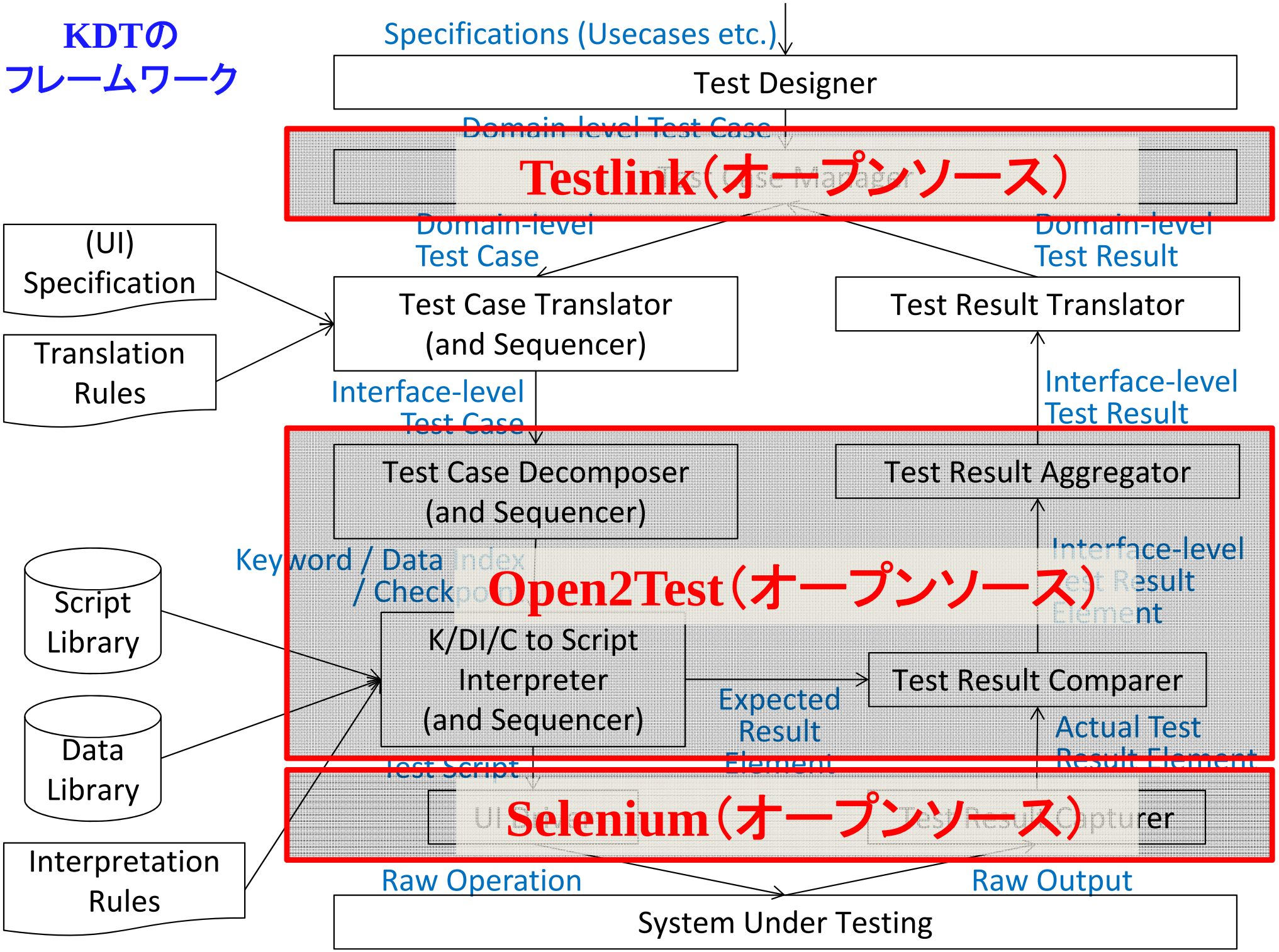
ビジネスレベルキーワード駆動テスト

- さらに「ビジネスレベルキーワード駆動テスト」が生み出された
  - 操作レベルのキーワードをビジネスレベルのキーワードと関連づけておき、
    テストケースをビジネスレベルのキーワードで書くと、
    ビジネスレベルのキーワードを操作レベルのキーワード群に変換し、
    さらに自動で自動化テストスクリプトに変換し実行してくれるようになっている
    - » ビジネスレベルキーワードの例:「予約する」/操作レベルキーワードの例:「クリックする」
  - 要するに、ユーザー企業が書いたテストケースが
    そのまま自動化テストスクリプトに変換され、自動で実行してくれる技術である
- もちろん、銀の弾丸(魔法の杖)ではない
  - 全てのテストケースのうち、良くて30%程度しか自動化できないと言われている
    - » 最初から全てを自動化せず、シンプルな自動化を少しずつ積み重ねていくと安全
  - 機能追加や仕様変更からビジネスレベルテストケースに
    スムーズに落とし込めるテスト設計方法論やトレーサビリティ管理ツールが必要
  - ビジネスレベルのテストケースから操作レベルのテスト手順に
    きちんと落とし込むようなテストプロセスと変換ツールが必要
  - UIが変わっていないことが大前提なので、ベンダー企業と協調することが必要
    - » ユーザー企業で自動化が進むとベンダー企業も自動化を進めやすくなる

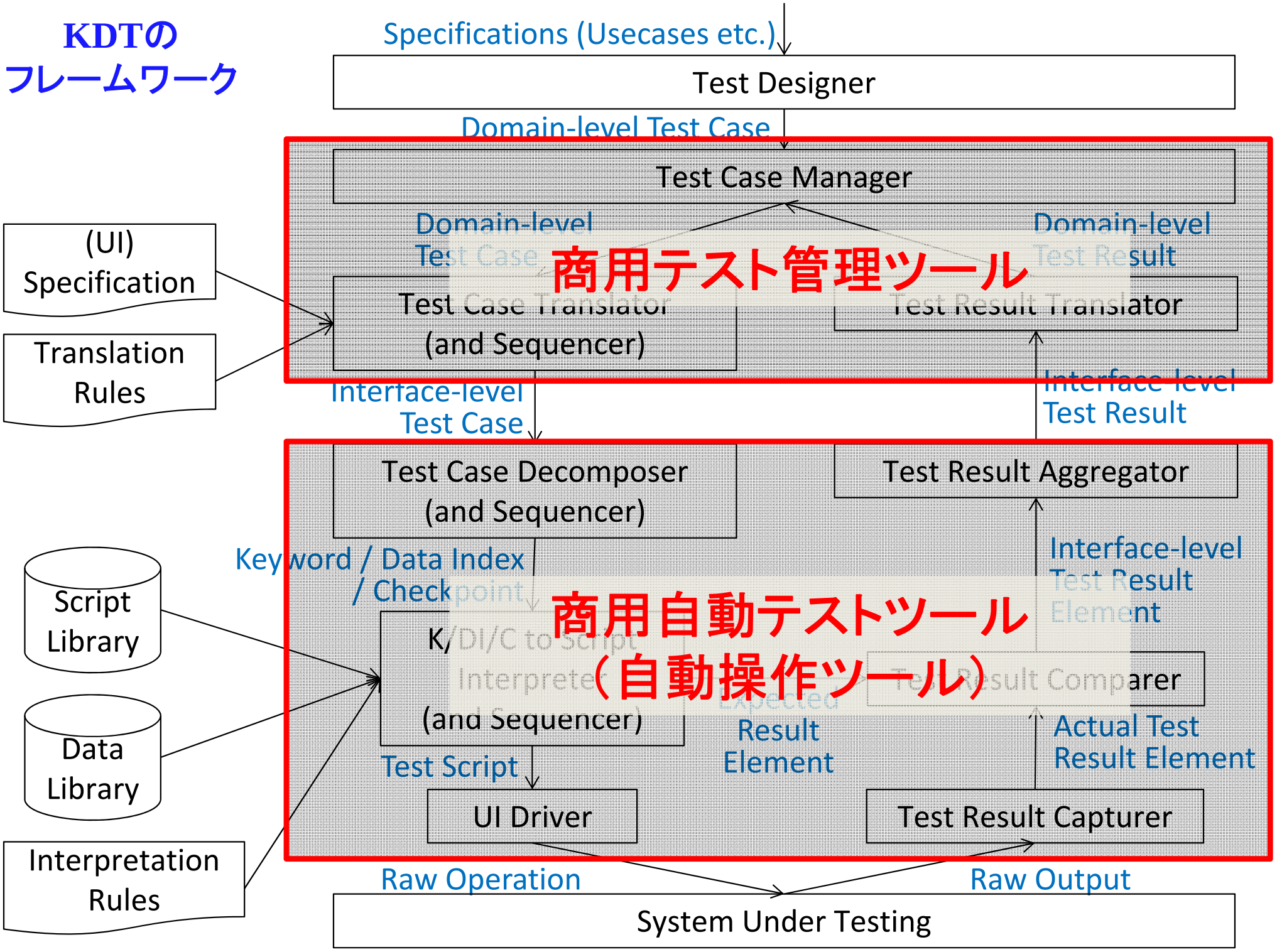
KDTの
フレームワーク



KDTの
フレームワーク



KDTの
フレームワーク



ユーザーとベンダーのWin-Winな開発を目指して

- 受け入れテストで悩んでいるユーザー企業は納得感が低い
  - このまま丸投げをしてよく分からないままだと不幸が続くのは目に見えている
- 受け入れテストの丸投げで発生している問題は、ユーザー企業が納得感を高める取り組みを進めていくと少しずつ解消されていく
  - 何の納得感をどれくらいどう高めていくか、はユーザー企業の予算や人員、技術に依存するので、大まかな構想をユーザー企業が主導的に考えなくてはならない
- 納得感を高めタイムリーにシステムを構築するための具体的な取り組みを紹介した
  - テストの概念理解・知識獲得、テストの意図把握、ユーザー企業ならではのノウハウ蓄積、テスト自動化によるスピード向上

納得感を高めユーザーが主導することで
ユーザーとベンダーのWin-Winな開発が可能になる

納得感を高めると技術が向上し
一体感を持って開発を進められる

電気通信大学　西　康晴
Yasuharu.Nishi@uec.ac.jp
http://qualab.jp/