てすとづくり　～質の高いテスト設計の原理～

2012/11/3（土）
ASTERソフトウェアテスト設計コンテスト'13 関西地域予選会

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 / QuaSTom
  - 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）
  - IPA/SEC ソフトウェア品質監査制度部会 審査基準WG 委員

講演に際しての注意点

- この講演は、西康晴個人の立場でテスト設計の原理を説明するものであり、審査委員会全体の審査方針を表しているわけではありません

- この講演をお聞きの皆さんは、テスト設計コンテストに応募した方々および応募を考えておられる方々だと想定していますので、かなり難解な話になっています

皆さんはどのレベルでしょう？

- レベル1：コピー＆ペースト＆モディファイ法レベル
  - Excel上でコピペして、仕様書に合わせてちょっと単語を変える段階
- レベル2：機能一覧ぶら下げレベル
  - 機能一覧を仕様書からコピペして、その機能のパラメータをぶら下げる段階
- レベル3：テストレベル・テスト技法単純適用レベル
  - 単体・結合・システムテストなどの各テストレベルで境界値テスト・パステストなどのテスト技法を適用している段階
- レベル4：お仕着せテストタイプ
  - 標準的な／よく使うテストタイプが既にあって、それに当てはめていく段階
- レベル5：テストエンジニアリングプロセスレベル
  - テスト要求分析やテストアーキテクチャ設計でモデリングをする段階

技術が発展する順序

- 第１段階：癒着
  - KKDで質を確保している段階
    - » 全てのノウハウが頭の中にある
    - » 品質向上に関するコミュニケーションが難しい
  - 規模が大きくなるとニッチもサッチもいかなくなる
  - ノウハウの伝達が難しいので、人数も対象も増やせないし伝承できる世代も延ばせない
- 第２段階：剥離
  - 専門分化して経験を蓄積可能にすることで技術の質を高める段階
  - 分割統治原則に従うので、技術特性の列挙や技術の体系化、プロセスの整備が容易になる
- 第３段階：融合
  - 剥離した技術を同時に適用できるようにすることで質の高い技術を用いる段階
  - 剥離したままでは煩雑な技術群やプロセスをスッキリ見通しよく適用できる
  - 剥離したままでは悪いトレードオフになってしまう技術群に正しく方向付けできる

テスト設計技術を剥離する

- テストは何のために行うのか、の整理
  - テストで考えなくてはいけないことを露わにしやすい
- テストエンジニアリングプロセスの各工程への分割
  - 各工程で考慮すべき重要な事項を露わにしやすい

テストは何のために行うのか、を剥離する

- スタンス１：テストとは行動である
  - 「テストでは何も証明できない、ただバグを見つけるだけである」
    - » 行動の内容と結果だけを提示する
      - ・ 何時間テストしました、何千件テストしました
    - » 見つけたバグが全てだと思うあまり、探索型などという言葉を隠れ蓑にして“食い散らかす”だけのテストを導いてしまったりする
- スタンス２：テストとは説明である
  - 「テストとは仕様通り動くことを実証することである」
    - » 行動の根拠や行動の妥当性も提示する
      - ・ この仕様書の網羅性は100%です
    - » 自分達はこれに従ってこれをやりました、だから問題はないでしょう、という“説明無責任”のテストを導いてしまったりする
- スタンス３：テストとは納得してもらうことである
  - 納得してもらうために技術と知性（とまごころ）を尽くして説明する
    - » 相手に納得してもらいやすいように理解しやすく提示する
      - ・ これがテストの全体像です、一緒に検討して頂けませんか
      - ・ いくつか選択肢とメリットデメリットがあるので、一緒に検討していきましょう

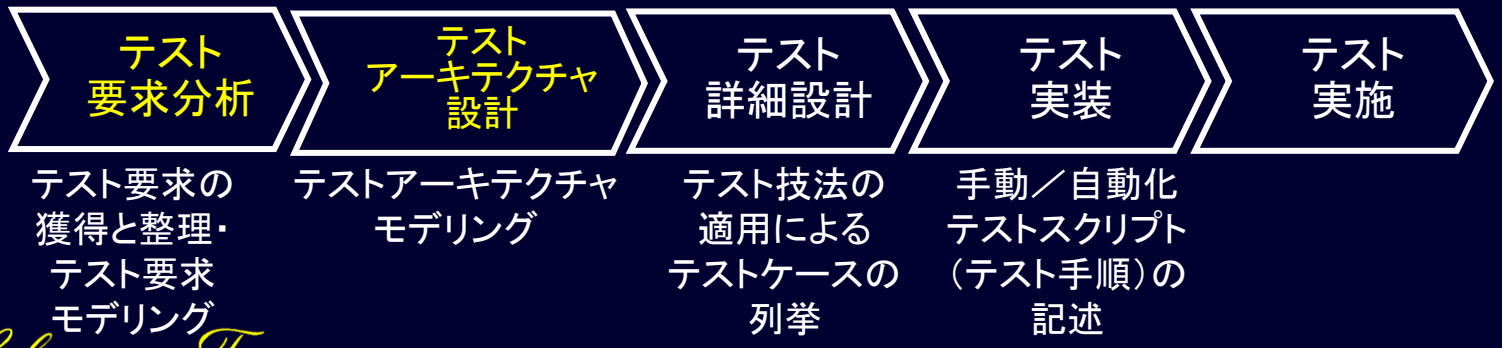
それぞれ考えなくてはいけないこと

- 行動するために考えなくてはいけないこと
  - 行動内容、行動手順、行動対象、品質ゴール・品質懸念点、トレードオフすべき制約・特性・行動内容、優先順位・工程終了条件、スコープ、フィードバック、バランス、保守性、・後工程作業容易性など
- 説明するために考えなくてはいけないこと
  - 完備性・網羅性、追跡性・根拠到達性・多重性、論理的整合性、表現統一性、詳細性、MECE性、多面性、優先順位明示性、トレードオフ適切性・許容範囲明示性など
- 納得してもらうために考えなくてはいけないこと
  - 根拠理解性、一目瞭然性、俯瞰性、ストーリー性、首尾一貫性(ブレないこと)、直感性、不安払拭性、選択可能性・選択肢明示性、説明多様性、上司説明性など

説明できたからといって
納得してくれるとは限らない

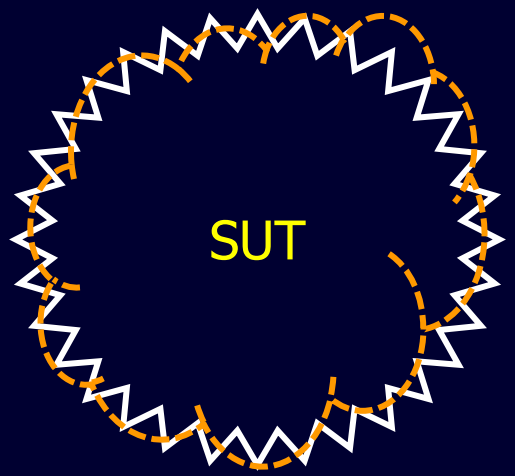
テストエンジニアリングプロセスとは？

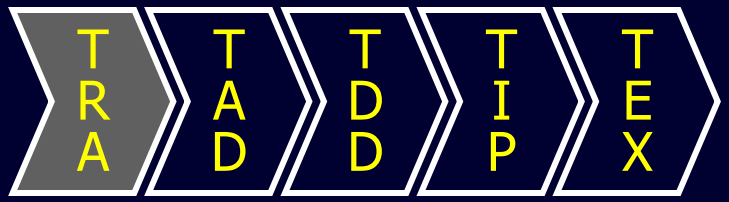
- テスト計画だのテスト戦略だのテスト方針だのと呼ばれている技術を剥離するために、分割した工程群
  - 技術があまり提案されておらず勘と経験でゴニョゴニョしている領域である
  - 工程を分割して剥離することで、重要な考慮事項が露わになりやすい
- テストケース群（テストスイート）を開発するという考え方をする
  - ソフトウェアや建築物のような人工物と捉える
- ソフトウェア開発プロセスと同様の工程から構成される
  - テスト要求分析（TRA）、テストアーキテクチャ設計（TAD）、テスト詳細設計（TDD）、テスト実装（TIP）、テスト実行（TEX）およびその後工程
    - » 理解、構想、論理立て、現実化、実証を行っていると考えてもよい



テストをしないと、テスト対象はよく分からない

テスト要求分析は理解の工程





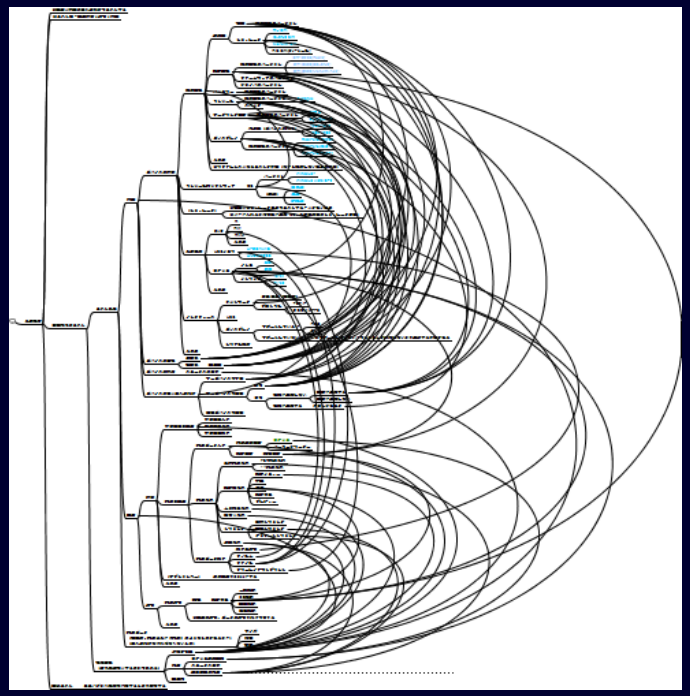
- テスト要求分析は、テスト対象がどうなっていて、何を実証すれば納得してもらえるのかを理解する工程
  - VSTePでは、テスト観点や関連、詳細化という考え方で整理をして理解していく
  - 多面性や詳細性、MECE性、抽象度妥当性などが特に重要になる

テストアーキテクチャ設計は構想の工程

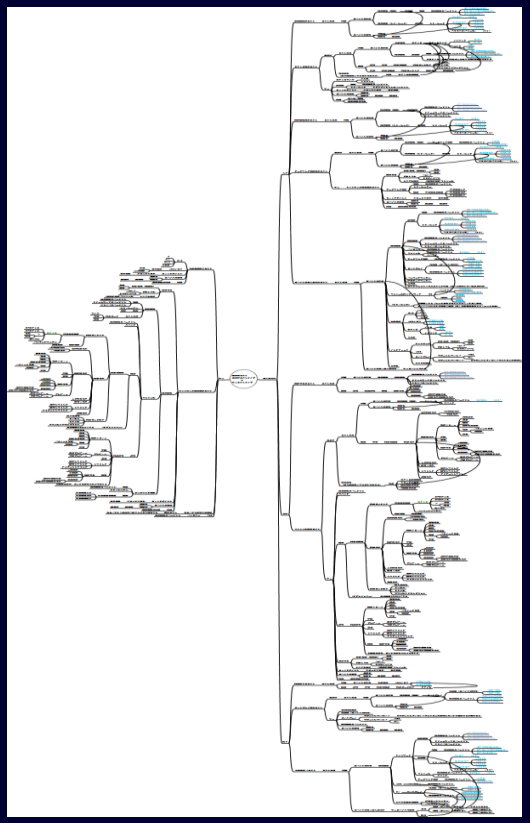
- テストアーキテクチャ設計は、論理立てや現実化、実証が容易になるように分割・整理するなど全体像を構想する工程
  - VSTePでは、テストコンテナという考え方で分割・整理して構想していく
  - 俯瞰性やバランス、凝集度、結合度、保守性などが特に重要になる

テストアーキテクチャ設計の例（NGTによる例）

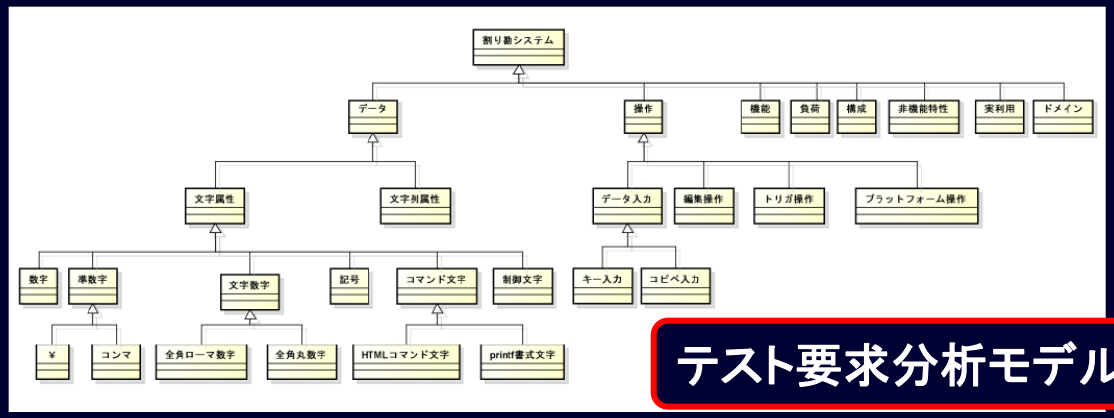
- 大規模複雑なテストでは、テスト要求分析モデルを分割／整理しないとテスト詳細設計が難しい



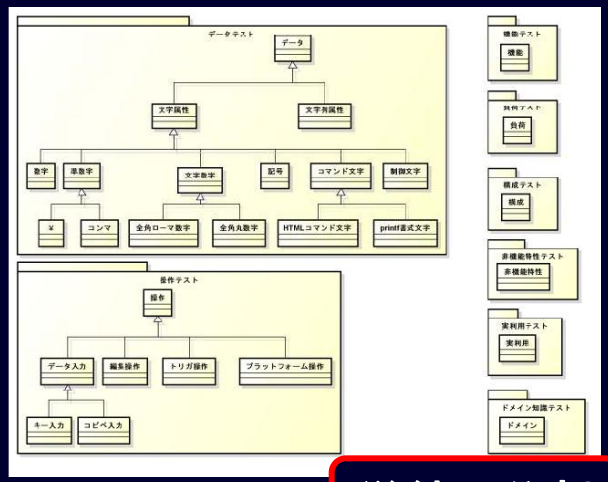
分割・整理



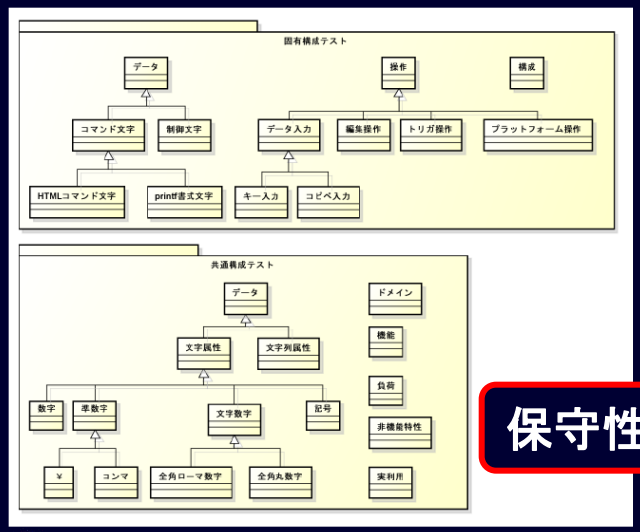
テストスイートに求められる品質特性によって異なるテストアーキテクチャの例



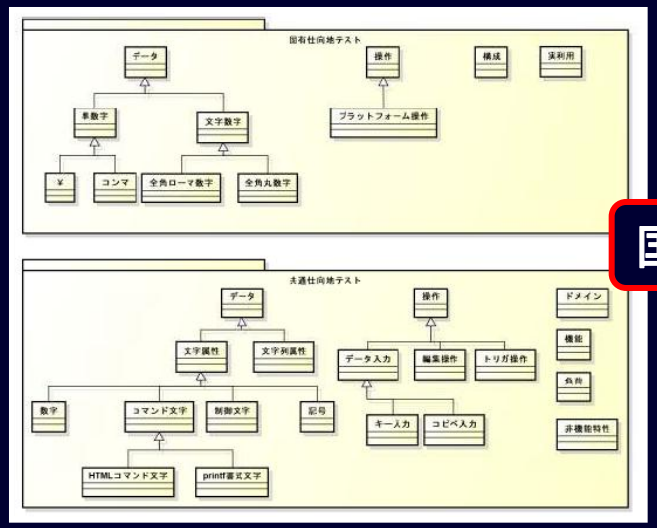
テスト要求分析モデル



単純に分割



保守性重視



国際化重視

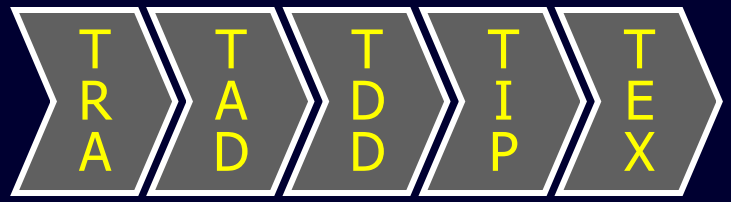
テスト詳細設計は論理立ての工程

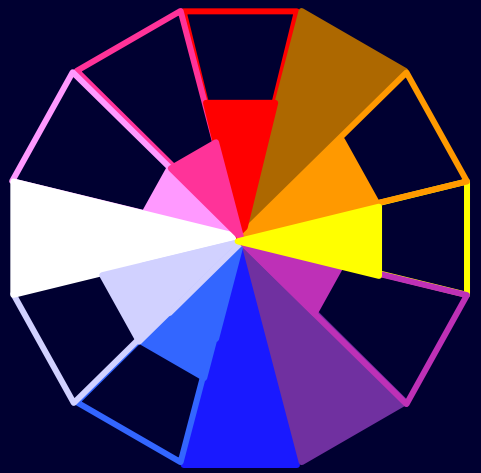
- テスト詳細設計は、どうやってテストしていけば実証したことになるのかを示せる論理立てを考案する工程
  - VSTePではテスト技法を用いて網羅していく
  - テスト詳細設計モデルや網羅アルゴリズム、一目瞭然性などが特に重要になる

テスト実装は現実化の工程

- テスト実装は、実際のテスト環境で実証を進めていくためにテストケースを現実化する工程
  - VSTePでは複数のテストケースを一緒に実行できるように集約したりもする
  - 詳細性、実行一意性、環境依存性、自動化可能性、実行円滑性、トラブル回避性などが特に重要になる
    - » 実行一意性：指示理解性や実行融通性、結果解釈性を考えないといけない

テスト実施は実証の工程





- テスト実施は、実際にテスト手順を実行して実証を進める工程
  - VSTePでは可能な限り自動化し、自動化しやすいように前工程を進める
  - 作業制約、作業速度、作業で得られる情報量などが特に重要になる

テスト設計の質を高めるための流れと原理群

- 行動→説明→納得の流れに沿って進化させる
  - 行動して終わり、説明して終わりのテスト設計ではなく、
    納得してもらえるような原理にしたがったテスト設計に質を高めることができる
- 理解→構想→論理立て→現実化→実証の
  流れに沿って進化させる
  - お仕着せのテストの進め方を適用して終わりのテスト設計ではなく、
    テスト対象や品質目標の理解・明示、全体像の提示をして保守性を高めるような
    原理にしたがったテスト設計に質を高めることができる
- 理解→構想の中身を充実させる
  - 貧弱なテスト設計から、充実させられる原理にしたがった
    テスト設計に質を高めることができる

理解・構想の中身を充実させる

- 記法によるメリットデメリットをきちんと把握し、デメリットをプロセスやルール、パターンでカバーできるようにしておく

- 自然言語記述型記法
  - 非常に詳細に内容や意図を書きやすい
  - 網羅性が非常に分かりにくい
  - 俯瞰性が非常に低く煩雑である
  - 構造化しにくいため構造が歪みやすい
  - 構造が過度に抽象化されているため（何を書いてもいいため）
  - 一貫性が保ちにくい

理解・構想の中身を充実させる

- 記法によるメリットデメリットをきちんと把握し、デメリットをプロセスやルール、パターンでカバーできるようにしておく

- 構造化リスト型記法（例：Excel大中小リスト、FV表）
  - 詳細に内容や意図を書きやすい
  - 上手に構造化をしないと網羅性は分かりにくい
  - 上手に構造化をしないと俯瞰性が低い
  - 構造化の段数を制約しやすいため構造が歪みやすい
  - 構造を過度に抽象化しやすいため一貫性が保ちにくい

構造化リストの例：Excel大中小リスト

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	状態	イベント	状態＋イベント
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

理解・構想の中身を充実させる

- 記法によるメリットデメリットをきちんと把握し、デメリットを
  プロセスやルール、パターンでカバーできるようにしておく

- 組み合わせマトリクス型記法（例：状態遷移表、競合マトリクス）
  - 詳細に内容や意図を書きにくい
  - 組み合わせの網羅性は分かりやすい
  - 規模の増大に従って俯瞰性が急激に下がっていく
  - 巨大なマトリクスで空白のセルが多いと俯瞰性が低い
  - 組み合わせマトリクスだけでは完結しないので
    他の記法と組み合わせねばならず俯瞰性が下がる
  - 構造を過度に抽象化しにくいため一貫性が保ちやすい

理解・構想の中身を充実させる

- 記法によるメリットデメリットをきちんと把握し、デメリットをプロセスやルール、パターンでカバーできるようにしておく

- フレームベースマトリクス型記法
（例：DT、直交表、ゆもつよマトリクス、スープカレー表）
  - 詳細に内容や意図を書きにくい
  - 上手に構造化をしないと網羅性は分かりにくい
  - 全体マトリクスと詳細マトリクスを分けないと俯瞰性が下がりがちになる
  - トリガ側を熱心に詳細化しやすいので俯瞰性が下がりがちになる
  - 構造化の段数を制約しなければ構造は歪みにくい
  - 構造を過度に抽象化しにくいため一貫性が保ちやすい

フレームベースマトリクスの例：ゆもつよマトリックス

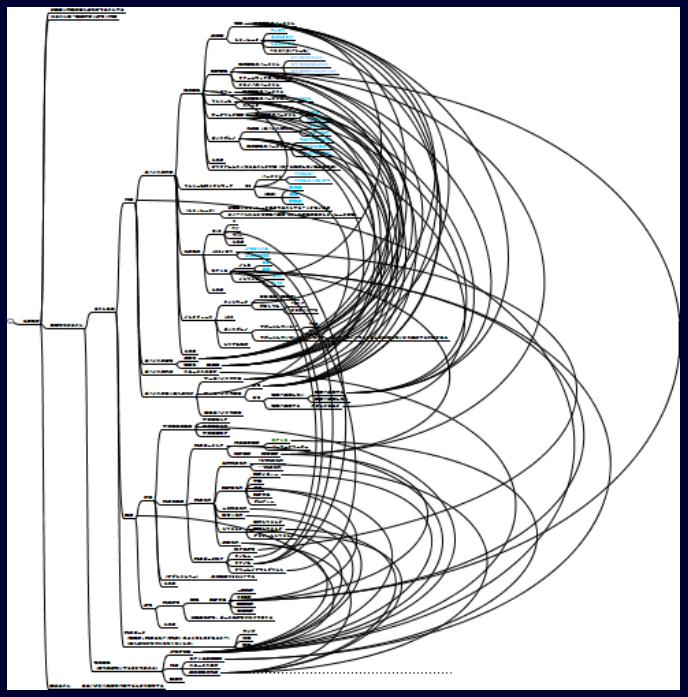
テストカテゴリ / 機能（群）		入力・表示		計算処理			エラー処理	排他処理	インタフェース		状態	
		アドレスの入力	アドレスの変換	暗号化	エンコード	文字数カウント	タイムアウト	既受信メールの編集	SMTP	IMAP4	サーバ未接続	サーバ接続中
メール送受信系機能群	メール送信											
	メール返信											
	メール転送											
メール編集系機能群	文字の入力											
	文字の削除											
メール管理系機能群	メールの削除											
	フォルダの作成											
	フォルダの移動											

理解・構想の中身を充実させる

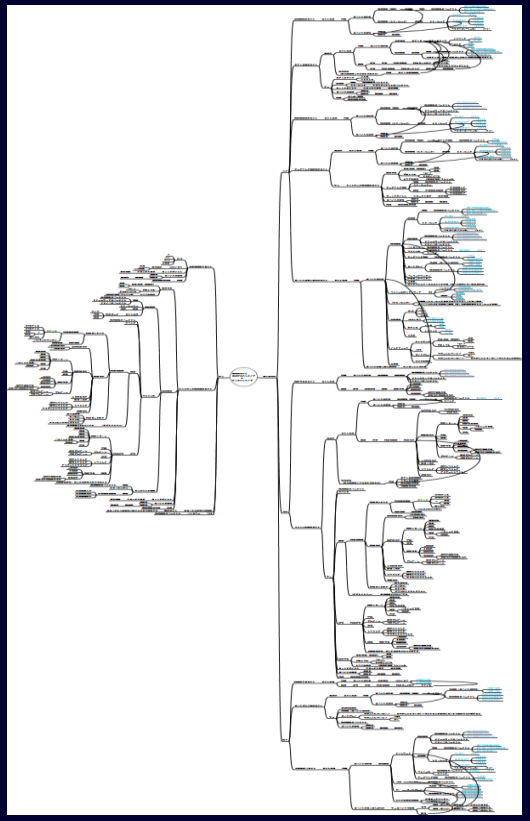
- 記法によるメリットデメリットをきちんと把握し、デメリットをプロセスやルール、パターンでカバーできるようにしておく

- ツリー・ネットワーク型記法
（例：マインドマップ、クラシフィケーションツリー、NGT）
  - 詳細に内容や意図を書きにくい
  - ノード間の関係を明示しないと網羅性は分かりにくい
  - 詳細ノードを折りたためれば俯瞰性は高い
  - 構造化の段数を制約しないため構造が歪みにくい
  - 特定のサブツリーばかり熱心に詳細化すると構造は歪みやすい
  - 構造を過度に抽象化しにくいため一貫性が保ちやすい
  - 同じ構造が複数の箇所に表れやすいため煩雑になることがある
  - ネットワーク構造の場合、上手に整理しないとゴチャゴチャになる

ツリー・ネットワーク型記法の例：テスト観点図（NGT）



分割・整理



テスト設計のレベルの読み解き

- レベル1：コピー＆ペースト＆モディファイ法レベル
  - テスト対象を理解していない
  - 【TRA&TAD&TDD&TIP】→【TEX】だが、実際は転記のみ
- レベル2：機能一覧ぶら下げレベル
  - テスト対象を一面的にしか理解できていない
  - 【TRA&TAD&TDD&TIP】→【TEX】
- レベル3：テストレベル・テスト技法単純適用レベル
  - テスト対象を理解しているようでいて、実は自分達の作業しか理解していない
  - 【TRA&TAD&TDD】→【TIP】→【TEX】
- レベル4：お仕着せテストタイプ
  - テスト対象を多面的に理解しているようでいて、
    実は自分達で想定している標準テスト対象（＝ルール）しか理解していない
  - 【TRAもどき＆TADもどき】→【TDD】→【TIP】→【TEX】
- レベル5：テストエンジニアリングプロセスレベル
  - テスト対象を多面的に理解した上で　きちんと設計に落とし込んでいる
  - 【TRA】→【TAD】→【TDD】→【TIP】→【TEX】

テスト設計のレベルごとの問題点の例

- レベル1：コピー＆ペースト＆モディファイ法レベル
  - どれだけテストをして何を実証したかが全く説明できない
    - » 結局のところ、頑張りました、としか言えない
- レベル2：機能一覧ぶら下げレベル
  - 品質特性や状態遷移などが漏れてしまう
- レベル3：テストレベル・テスト技法単純適用レベル
  - テスト技法の使いどころが分からず現場では上手にテストできない場合が多い
- レベル4：お仕着せテストタイプ
  - テストタイプが重複したり漏れたり、
    タイプ同士をどう組み合わせて良いのかが分からない場合が多い
    - » お仕着せテストタイプは少しずつ蓄積され、上手く整理されていないことがある
    - » 負荷テストとストレステスト、低メモリテストの位置づけなどの解釈で困る
- レベル5：テストエンジニアリングプロセスレベル
  - 剥離した技術を統合するテスト開発方法論の構築を目指すことが必要となる
  - 抽象化能力などのモデリングスキルが必要となる

自分たちで工夫して技術を融合して テスト設計の質を高めていきましょう

電気通信大学　西　康晴
Yasuharu.Nishi@uec.ac.jp