MDA ツールによるソフトウェア品質保証

- 開発ドキュメント削減による生産性向上と品質確保 -

MDA tool proto type Evaluation of SQiP Symposium

- Fisibility studies for Software Development improvement-

株式会社 IHI エスキューブ 品質保証部
Quality Assurance Dept. IHI Scube Co.,Ltd.
○平井宣
○Fusami Hirai

Abstract Software development have been used to make a lot of design documents as well as test evidences. We found that MDA tool can reduce the amount of these documents in development. As a result, this could realize software short delivery and keep heigh quality. And, this could also bring you new agile solution for huge project which many engineer participate in. This also reports the details of MDA tools for improvement of software quality.

1. はじめに

DX やクラウド型サービスなどに見られる新しいソフトウェア要求は，従来と比べ物にならないショートデリバリーとプロジェクトマネジメント力が求められるようになった．また，めまぐるしい設計変更への追従や，運用・保守における安定稼働を果たすために，ソフトウェア品質への期待はますます高まっている．

また，マネジメント，アプリ開発，セキュリティや高可用性などの非機能要件の実装など，高度技術者による分業は当たり前で，設計と製造で作業を分け並列化により開発期間を短縮するなど，とかくコミュニケーションやマネジメントのために，開発ドキュメントを重視する PM は多い．

一方で，発注者と受注者の間でドメイン知識が共有されており，中核メンバーが継続して内製化に取り組んでいるようなケースでは，開発ドキュメントを一部省略する事例も見られる．

今回考案した MDA ツール(Metamodel Driven Architecture tool)を使うと，開発ドキュメントを減らしても，品質を確実にコントロールできるようになる．加えて，テスト量も減ることが分かった．全体の作業量が減るので生産性が向上し，開発期間の短縮につながる．

以下，本論文の構成を述べる．まず 2 章で現状分析と課題提起を行う．3 章， 4 章で解決策の提案と評価結果を示す．5 章で評価結果に対する考察を行い， 6 章で成果と将来の展望で結ぶ．

2. 解決すべき課題

2.1 開発ドキュメントの検討範囲

IPA の開発手法ガイドブック[1]や CMMI[2]認定企業の開発ドキュメントを参考に，本論文で示す開発ドキュメントを表 2-1 に示す．訴求ポイントを絞るため，本論文で論ずるフェーズを，基本設計から結合テストまでとした（赤枠内）．

株式会社 IHI エスキューブ 品質保証部
Quality Assurance Dept. IHI Scube Co.,Ltd.

東京都江東区豊洲三丁目 1 番 1 号 豊洲 IHI ビル Tel: 03-6204-8053 e-mail:hirai9589@ihi-g.com
1-1, Toyosu 3-chome, Koto-ku, Tokyo Japan
1) 株式会社 IHI エスキューブ 品質保証部 技術主幹
Technical Manager, Quality Assurance Dept. IHI Scube Co.,Ltd.

【キーワード：】メタモデル，MDA，Agile，開発ドキュメント，開発期間短縮，ソフトウェアテスト

表 2-1:主な開発ドキュメント

カテゴリ	フェーズ	主な開発ドキュメント
PJ 管理	立上げ，計画，実行，終結	実施決裁書，ﾌﾟﾛｼﾞｪｸﾄ計画書，開発品質計画書，ﾘｽｸﾚｼﾞｽﾀｰ，各フェーズ完了判定
開発	要件定義	現行稼働ｼｽﾃﾑ一覧，ｼｽﾃﾑ構成図，機能関連図，DFD，画帳一覧，画面ｲﾒｰｼﾞ，概念ﾃﾞｰﾀﾓﾃﾞﾙ，ｾｷｭﾘﾃｨ要件，メニュー体系
	基本設計	機能一覧，画帳一覧詳細，画帳ﾚｲｱｳﾄ，ﾊﾞｯﾁ一覧，ｼｽﾃﾑｲﾝﾀﾌｪｰｽ一覧，ｼｽﾃﾑ方式，ｱﾌﾟﾘ処理方式，論理ﾃﾞｰﾀﾍﾞｰｽ定義，CRUD 定義，権限定義，非機能定義，ﾃﾞｰﾀ移行計画書
	詳細設計～単体テスト	内部処理フロー，ﾌﾟﾛｸﾞﾗﾑ管理台帳，ﾘﾘｰｽ記録，単体ﾃｽﾄ仕様兼報告書
	結合テスト	結合ﾃｽﾄ仕様書兼報告書，品質分析報告書，障害票，変更通知
	総合テスト	総合ﾃｽﾄｼﾅﾘｵ，総合ﾃｽﾄ仕様書兼報告書，品質分析報告書，障害票，変更通知
	UAT～本稼働支援	UAT ｲﾝｼﾃﾞﾝﾄ管理，ﾕｰｻﾞﾏﾆｭｱﾙ，本番移行計画書，ﾘﾊ計画書，稼働判定表
運用保守	運用・監視	ｴｽｶﾚｰｼｮﾝ定義書，稼働ｼｽﾃﾑ一覧，NW 構成図，ｻｰﾊﾞ管理台帳，NW 機器管理台帳，PC 管理台帳，情報資産管理台帳，SW ﾗｲｾﾝｽ管理台帳，特権 ID 管理台帳，利用者管理台帳，障害管理，ｷｬﾊﾟﾌﾟﾗ，ﾘｿｰｽ計測，ｲﾝﾀｰﾈｯﾄ QoS
	ｾｷｭﾘﾃｨｱｯﾌﾟﾃﾞｰﾄ	実施計画書，ｼｽﾃﾑ停止連絡票，ｾｷｭﾘﾃｨﾊﾟｯﾁ管理台帳
	変更，改修	各種手順書(NW，ｻｰﾊﾞ，PC 配付，ﾊﾞｯｸｱｯﾌﾟ・ﾘｽﾄｱ，構成変更，ﾌﾟﾛｸﾞﾗﾑ変更，特権 ID 申請，利用者申請)

2.2 現状分析

IPA で公開されている開発ドキュメントのメトリクス[3]を表 2-2 に示す．基本設計の頁数や結合テストのテスト件数をソースコード量から求めることができる．例えば，開発コスト 1 億円で，開発期間が 12 ヶ月のプロジェクトだと，十数人の開発メンバーで構成され，プログラムの本数は 1,000 本(ソースコード 500KSLOC 相当)を超える．以下の式から頁数を求める．

・基本設計の頁数： メトリクス中央値×KSLOC

・結合テストの頁数： メトリクス中央値×KSLOC÷頁数比 ※頁数比：30 件/頁

すると，基本設計は 5,310 頁，結合テストは 862 頁となる．およそ 6,000 頁となり，10cm 箱ファイルで 6 箱である．一方ソースコードを印刷すると少なく見積もっても 7,000 頁あり，開発ドキュメントとほぼ同量となるため，やはり開発ドキュメントの量は多いように思う．

表 2-2 KSLOC 規模あたりの設計量とテスト件数

[ページ/KSLOC]

工程	N	最小	P25	中央	P75	最大	平均	標準偏差
基本設計	112	0.57	5.05	10.62	19.14	119.75	15.74	18.73
詳細設計	112	0.17	6.34	13.21	24.47	378.26	21.99	42.05

[件/KSLOC]

	N	最小	P25	中央	P75	最大	平均	標準偏差
結合テスト(テストケース)	665	0.0	22.2	51.7	120.1	100,000.0	484.9	4,821.2
総合テスト(テストケース)	620	0.0	6.4	16.0	40.7	219,000.0	653.9	9,799.1

また，運用・保守でこれらの開発ドキュメントは必要なのだろうか．改修要件に対するプログラムのあたりがつけば，設計書を読むより，設定値やソースコードを見て，動かしながら修正計画を立て，どこをどう直したのか後で分かるようにしておく方法もあると思う．

2.3 課題提起

ソフトウェアの品質保証において，開発ドキュメントが重要であることは変わらない．設計の品質が高まれば，プログラミングやテストの品質が確保できる．また，設計者とプログラマーで作業を分担できるので生産性も上がる．ISO12207 や IPA の共通フレームの「V 字モデル」の考え方である．[4]

これまで設計の品質を高めるには，設計書の量を増やせばよいという考え方が主流であった．今回，発想を変え，設計書をデータベース化するという考え方を提案する．データベース化すると情報が正規化されるため，ドキュメント量が相対的に減るはずである．また，「V 字モデル」によれば，設計書の量とテスト件数は相関性が高く，設計書の量が減ればテスト件数も減ることが

予想される．

本論文では，このデータベースをメタモデルと呼ぶ．さらに，このアプローチで品質がどう担保されるのかを説明するために MDA ツールを試作した．なお，ここでいう MDA は，OMG が提唱する MDA[5]と直接的な関係はない．

3. 解決策の提案

3.1 課題の解決方針

まず，開発ドキュメントのうち，基本設計，結合テストを例に，メタモデルとは何かを説明する．メタモデルはエンティティ，アイテム，プロパティ，アタッチメントで構成される．開発ドキュメントをエンティティに写像させ，エンティティを構成する行がアイテムにあたり，列がプロパティである．アイテムの補足はアタッチメントに登録する．また，いわゆるビジネスロジックなどアイテムの内部仕様は，プログラムのコメント行に追い出し，ドキュメント全体を減らす．全体の構成図を図 3-1 に示す．開発ドキュメントとの対応は表 3-1 に示す．また，MDA ツールによるメタモデルの管理イメージを図 3-2 に示す．

なお，コメントの表記方法において，Z 言語[6]などの仕様記述言語が知られているが，本論文の訴求ポイントと関係性が低いため，そこまで言及しないこととした．

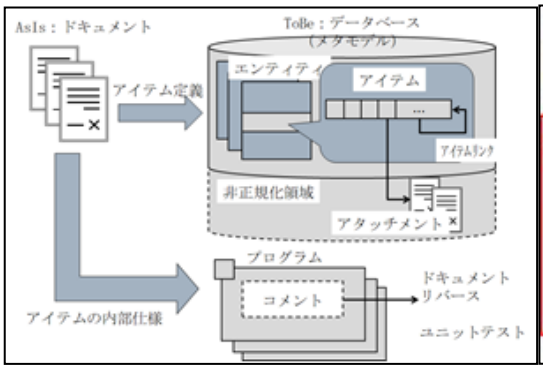

図 3-1:ﾒﾀﾓﾃﾞﾙの構成

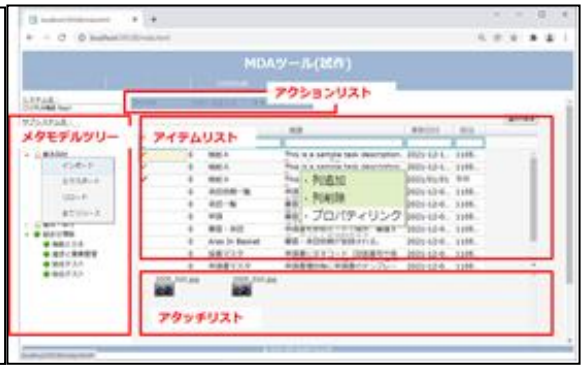

図 3-2:MDA ﾂｰﾙによるﾒﾀﾓﾃﾞﾙ管理ｲﾒｰｼﾞ

表 3-1:基本設計書と結合テスト仕様書のメタモデル

開発ﾄﾞｷｭﾒﾝﾄ	ｴﾝﾃｨﾃｨ名	ｱｲﾃﾑの登録例	主なﾌﾟﾛﾊﾟﾃｨ	主なｱﾀｯﾁﾒﾝﾄ
基本設計	機能	開発機能の一覧	機能名,概要	機能概要,DFD
	画面	開発画面の一覧	機能名,画面名,規模,難易度,ｲﾍﾞﾝﾄ,指摘件数	遷移図,画面ｲﾒｰｼﾞ,項目定義,ﾒｯｾｰｼﾞ一覧
	帳票	開発帳票の一覧	機能名,帳票名,規模,難易度,指摘件数	帳票ｲﾒｰｼﾞ,項目定義,指摘事項
	バッチ	開発バッチの一覧	機能名,ﾊﾞｯﾁ,規模,難易度,指摘件数	入出力 I/F(項目ﾚﾍﾞﾙ),ｸｴﾘｰ一覧,指摘事項
	外部 IF	他ｼｽﾃﾑ間のｲﾝﾀﾌｪｰｽ一覧	機能名,IF 名,ﾀｰｹﾞｯﾄ,入出力,連携方式,起動方法,難易度,指摘件数	入出力 I/F(項目ﾚﾍﾞﾙ),指摘事項
	DB	ﾃｰﾌﾞﾙ一覧	ﾄﾞﾒｲﾝ,ｴﾝﾃｨﾃｨ,項目名,英字名,型桁,PKEY,備考,指摘件数	
	権限	開発画面イベント別権限一覧	ﾛｰﾙ,ｸﾞﾙｰﾌﾟ,備考,指摘件数	権限内容の捕捉,職制情報,現行仕様書
	機能と DB	開発機能と DB の関係	機能名,画面名,ｲﾍﾞﾝﾄ,ﾃｰﾌﾞﾙ,CRUD	-
	機能と担当者	開発機能と担当者，納期の関係	機能名,画面名,帳票名,ﾊﾞｯﾁ,IF,付帯作業,予定,実績,工数,担当者	-
	機能と PG	開発機能とプログラムの関係	機能名,画面名,ｲﾍﾞﾝﾄ,処理ﾀｲﾌﾟ,PG 名,担当者	ﾌﾟﾛｸﾞﾗﾑ処理方式設計書
結合ﾃｽﾄ	結合ﾃｽﾄ	開発画面別イベントの一覧	機能名,画面名,ｲﾍﾞﾝﾄ,ｹｰｽ,内容,ﾃｰﾌﾞﾙ,更新区分,ﾃﾞｨﾌｪｸﾄ,完了日,担当者	ﾃｽﾄ証跡,横展ﾒﾓ

3.2 3 つのアイテム

メタモデルを理解するにあたり重要となる 3 つのアイテムを説明する．設計アイテム，テストアイテム，プログラムアイテムである．

設計アイテムは，表 3-1 の基本設計書をカバーする．例えば，機能，画面，帳票，バッチ，DB 等のエンティティに登録されたアイテムである．プロパティで管理できない非正規データは，アイテム毎のアタッチメントに登録する．

テストアイテム，プログラムアイテムはリンクアイテムと呼び，設計アイテムのプロパティが参照できる．テストアイテムは，表 3-1 の結合テストをカバーする．プログラムアイテムは，表 3-1 のエンティティ名が機能と PG のアイテムである．

3.3　プログラムアイテムの役割

課題の解決方針で述べたように，コメント行に追い出す設計情報は，ハッシュタグのようなアノテーションと呼ばれるものや，テスト判定に必要なアサーションと呼ばれるものを駆使して記述する．これは，図 3-2 に示すドキュメントリバース[7]や，図 3-3 にしめすユニットテスト[8]で利用される．

全てのプログラムは，プログラムアイテムに登録され，設計アイテムと関連付けられる．プログラムアイテムには，プログラム名や関連付けの分類も登録される．関連付けの分類とは，１つの設計アイテムに関連付く複数のプログラムアイテムを識別するものである．この分類は，プログラム処理方式の検討によって導かれるものである．検討のイメージを図 3-4 に示す．

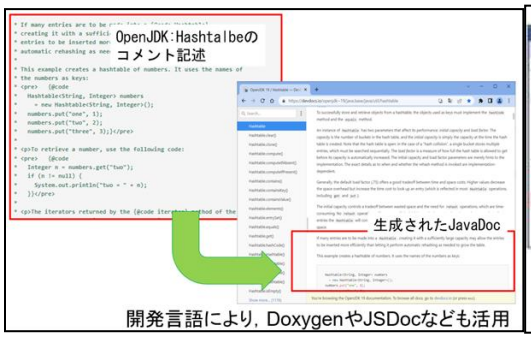

図 3-2:注釈からのドキュメントリバース

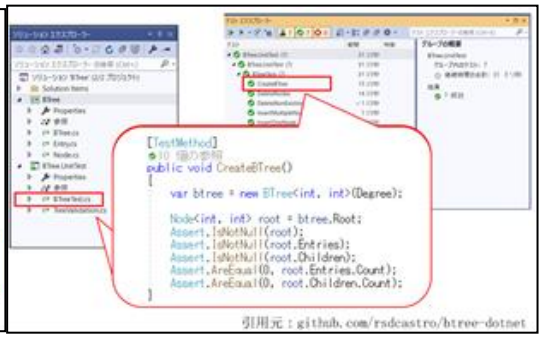

図 3-3:タグによるユニットテスト

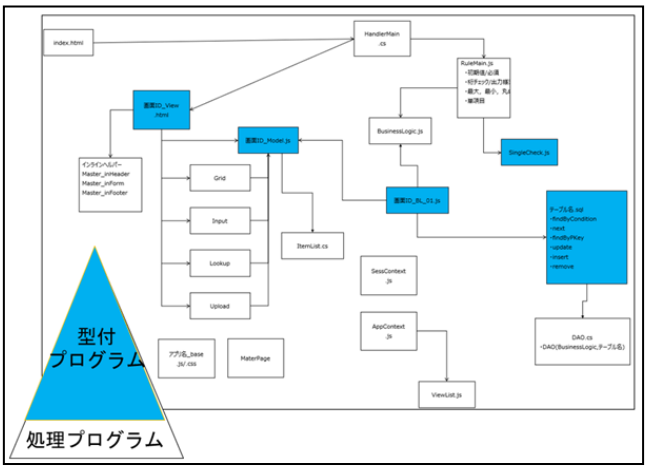

図 3-4:プログラム処理方式設計書（抜粋）

3.4　テストアイテムによる品質保証の考え方

テストアイテムは，１アイテムがテスト１件に相当する．シナリオやデータバリエーションが複雑な場合は，アイテムを増やす．もしくは，アイテムに詳細アイテムを持たせ，まとめておく．設計アイテムと関連付けて登録することで，全ての設計アイテムのテストが可能になる．

例えば途中でテストを追加するなど，追加理由やテストのレビュー記録は，アイテムのアタッチメントに登録しておくことで，テストアイテムの粒度や内容が収斂されていく．

設計アイテムから追い出した仕様のテストは，プログラムアイテム毎の単体テストで行う．また，後工程でテストの組合せやデータバリエーションを増やす場合，テストアイテムを増やすだけでよいので，同じ手順で増やすことができる．

4.　解決策の評価

4.1　評価方針

過去のプロジェクトから評価モデルを作成し品質データを抽出した．また，評価モデルを元に

メタモデルを構築し，各アイテムから品質データが算出できるので，この２つを比較する．
また，結合テストにおけるテストの網羅性と設計変更に対する追従性を評価する．

4.2　評価モデルの品質データ

プロジェクトの概要は以下の通り．開発実績の内訳を表 4-1 に示す．

・対象システム名：ワークフローシステム
・開発コスト：¥5,000,000（基本設計～総合テスト）
・開発期間：5 ケ月
・成果物:
　ﾌﾟﾛｸﾞﾗﾑ:画面(7 本)，帳票(0 本)，ﾊﾞｯﾁ(1 本)，ｲﾝﾀﾌｪｰｽ(1 本),ﾃｰﾌﾞﾙ(6 ﾃｰﾌﾞﾙ)
　開発ﾄﾞｷｭﾒﾝﾄ:基本設計書，結合ﾃｽﾄ仕様書兼報告書，総合ﾃｽﾄ仕様書兼報告書

表 4-1:開発実績の内訳

工程	基本設計	詳細設計～単体ﾃｽﾄ	結合テスト	総合テスト
工数(h)	160.0	600.0	80.0	80.0
設計書(頁)	20	5	10	10
テスト件数	-	40	30	5
バグ件数	-	20	7	2
指摘件数	20	20	5	2
KSLOC	-	4.6	-	-

4.3　評価結果

(1)設計ドキュメントの比較

メタモデルの場合，以下の計算式で，頁数を算出した．評価モデルとの比較を図 4-1 に示す．また，IPA の品質指標から求めた頁数も付記した．

メタモデルによる頁数＝エンティティ数＋Σアタッチメントに登録されたドキュメント頁数

(2)テスト件数の比較

メタモデルの場合のテスト件数と，評価モデルの比較を図 4-2 に示す．また，IPA の品質指標から求めた件数も付記した．

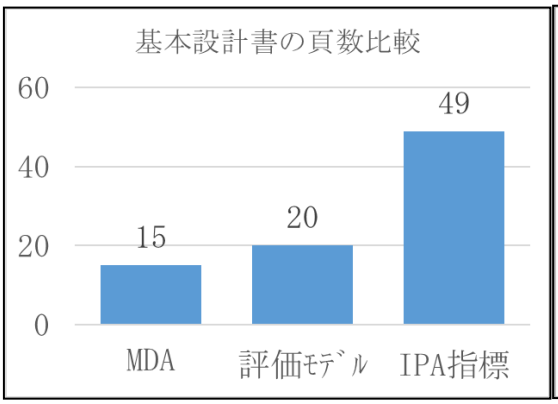

図 4-1:設計アイテムの評価

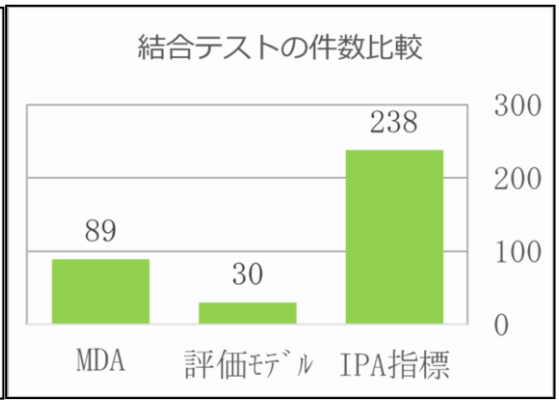

図 4-2:テストアイテムの評価

(3)テストアイテムによる網羅性

図 4-3 にテストの網羅性を示す．グラフは構築した設計アイテム同士の相関性を 3D グラフ[9]で表したものである．3D グラフは，構成するノードに電磁力やバネ弾性力を持たせることで，エッジでつながったノード同士を見やすく 3D で表示できる．ツールは MorcegoBrowser[10]を使用した．左図は設計アイテムの関連付けを点線の無向グラフで表したものである．右図はテストアイテムの関連付けを実線で表したものである．点線が漏れなく実線になっていることが分かる．

登録した設計アイテムとテストアイテムのデータを別紙に示す．

(4)設計変更の追従性評価

構築したメタモデルに対し，設計アイテムの追加，変更，削除を行い，テストアイテムの影響有無（テスト漏れ，テストすべきアイテム，テスト不要など）を図 4-4 に示す．なお，アイテムのマージは，メタモデルのデータベース（SQLServer)に対し，SQL を発行して確認した．

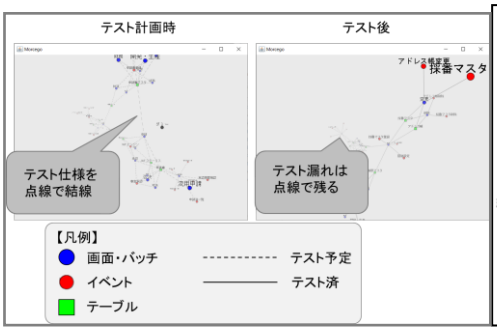

図 4-3:テストアイテムのテスト網羅性

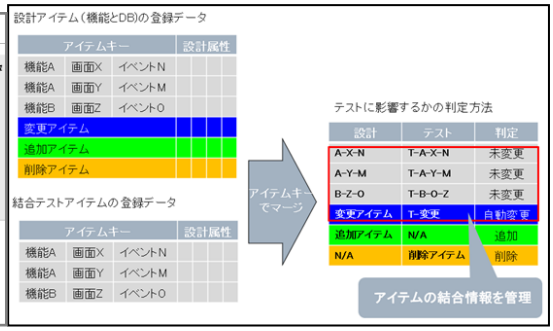

図 4-4:設計変更の追従性

5. 考察

5.1 得られた知見

(1)設計ドキュメントの削減

図 4-1 によると設計書をメタモデルで構築すると頁数が削減されることが分かった．特に画面仕様のドキュメントが少ない．データ項目レベルの仕様やボタンなどのイベントに関連付けられた内部仕様がプログラムのコメント欄に追い出されている．

今回は 7 画面と規模が小さいが，100 画面であっても 1,000 画面であっても，このアプローチは有効と考える．なぜなら，画面，担当者，担当者が受け持つプログラムの関連付けが保持され，設計情報量が減っているわけではないので，設計品質が確保されるからである．

(2)結合テストの件数削減

図 4-2 から評価モデル:MDA≒1:3 とテスト件数が増えている．評価モデルは別紙の登録した設計アイテム数とほぼ等しかった．これに対し MDA は別紙の登録したテストアイテムに示されているように，データバリエーションでテスト件数が増えている．メタモデルは，テスト件数の数え方が分かりやすく，テスト件数が無駄に多いのか，設計に対して少ないのかが，定量的に判断しやすい．

(3)テストの網羅性

図 4-3 からテストアイテムの過不足は視覚的に判断できる．増やすにしても，増やす場所とその影響有無が分かりやすい．3D グラフを見て，増やす経路やデータバリエーションを 3D グラフを見ながら立体的に検討できるからである．テストシナリオを巡る認識齟齬も防ぐことができるだろう．

なお，テストアイテム単体の品質は，つまり単体テストが終わっているかは，プログラムアイテムのプロパティで確認できる．

(4)設計変更に対する追従性

アイテムはユニークキーが割り当てられ，設計アイテムとテストアイテムの関連付けが登録されている．このためリンク元のエンティティで追加，変更，削除があると，以下のことが分かる．

・設計変更があれば，影響範囲が事前にわかる．

・改修する場合，必要なテストが事前にわかる．

基本設計から始まって，作るプログラムや担当者の割当までブレイクダウンしたあと，作ったプログラムを積上げていくのが至難の業で，このツールがあれば見通しを立てることも，軌道修正することも前倒しで取組みやすい．

5.2 MDA ツールの課題

今回は，訴求ポイントを絞るため，開発フェーズ，開発事例を限定した．MDA ツールの効果が得られる条件として，開発プロジェクト全員がメタモデル構築に参加する必要がある．構築には，プログラムの標準化や，内部仕様のコメント表記の指導が必要となる．このため設計者は，プログラミングもできる体制が望ましい．

また，メタモデル構築に向かないドメインとしては，標準化の効果が出にくい，組込みシステムやゲーム開発などが挙げられる．

6. まとめ

6.1 成果

基本設計をコンパクトにすることで，結合テストの件数が減り，付帯作業が減ることで生産性が上がる手法「ＭＤＡ」の有効性に見通しをつけた．生産性の向上は，開発期間の短縮や開発コストの削減につながる．

また，テストアイテムによる結合テストやプログラムアイテムによる単体テストにより，従来と変わらない品質を確保できることが分かった．

さらに，MDA ツールは，設計情報やテスト仕様をデータベース化することで，抜け漏れを防止できるので，QCD の見える化に有効であると思われる．

6.2. 将来の展望

設計情報が正規化されることで，効果が期待できるのがクラウドサービスやパッケージ領域である．Fit&Gap を標準化しやすい．アイテムの登録や属性の変更，追加は簡単で，エンドユーザも巻き込みやすい．

また，QCD の見える化は，DX でも期待が高い大規模 Agile への対応や大規模 PJ の QCD 対策(ﾘｽｸ早期発見,未然防止)に不可欠な技術である．当社品質保証の活動の中で確実に育てていきたい．

7. 謝辞

最後に，本シンポジウム発表にあたり，査読，審査くださった論文審査委員会の皆様，また発表活動をバックアップいただいた，IHI ｴｽｷｭｰﾌﾞの岡﨑 栄一取締役，益田 衡品質保証部長に厚く感謝申し上げたい．また，この「MDA ツール」のアイデアは，IBM Rational Doors をプロジェクトマネジメントに積極的に活用していた PM との面会が着想につながった．貴重な経験ができたことに感謝したい．

8. 参考文献

[1] IPA/SEC 高信頼化ソフトウェアのための開発手法ガイドブック Jun. 2011
[2] 橋本隆成 はじめての「開発のための CMMI」とプロセス改善 Aug. 2013
[3] IPA/SEC ソフトウェア開発分析データ集 2022 https://www.ipa.go.jp/digital/chousa/metrics/metrics2022.html
[4] IPA/SEC 共通フレーム 2013 Mar. 2013
[5] OMG MDA https://www.omg.org/mda/
[6] Information Technology - Z Formal Specification Notation ISO/IEC 13568:2002
[7] OpenJDK API reference https://openjdk.org/
[8] Simple BTree implemention in C# https://github.com/rsdcastro/btree-dotnet
[9] 藤澤誠 CG のための物理シミュレーションの基礎
[10] Morcego 3D Network Browser の fork サイト https://github.com/hacklabr/morcego