Vol.2018-SE-198 No.34
2018/3/10

アプリケーションシステム開発における品質評価・改善モデルの構築

塩澤　仁[†1]　鈴木正人[†1]

概要：アプリケーションシステム開発では，プロダクトの品質確保が求められるが，実際にはリリース後に不具合が顕在化することが多い．その原因のひとつは開発の過程で行われる品質確保が不十分であることにある．従来の研究はプロダクトの設計・製造工程で品質を定量評価することに着目してきた．だが実際には，テスト工程で品質が確保されている場合が多いが，属人性が高い為，品質確保が不十分となることが問題となっている．
本稿では，このような属人性を排除したテスト工程の品質評価・改善モデルを構築する．モデルの構築にあたり，品質の定量化，及び評価・改善プロセスの形式化に着目する．またプロダクト間のトレーサビリティを応用し，3 次元化されたトレーサビリティマトリクスを用いて，品質低下の根本原因を特定する．開発過程における品質確保の十分性が向上するとともに，将来的にモデルの適用を自動化することで実用性の向上が期待される．

キーワード：アプリケーション，テスト工程，品質評価，品質改善，モデル，定量評価，メトリクス，形式化，根本原因，トレーサビリティ

1. はじめに

1.1 背景

アプリケーションシステム開発では，コスト，納期の遵守を前提として，プロダクトの品質確保が求められる．しかし現実は，システムのリリース後に不具合が顕在化することが多い．その原因として，システム開発の過程で行われる品質確保が不十分であることが挙げられる．

本稿では，開発プロセスとして現場で多く採用されている，図 1 に示すウォーターフォールモデル[1]，及び図 2 に示す V 字モデル[12]を前提とする．基本設計では主にビジネスフローの定義，詳細設計ではコンポーネントの詳細な定義を行う．V 字モデルは，実装・単体テストを中心に，左側に開発工程，右側にテスト工程が位置づけられる．双方向矢印による左右の工程の関付けは，同じ抽象度で開発仕様とテスト仕様を記述することを表す[11].

ウォーターフォールモデルにおいて，品質はプロダクトの設計・製造工程で作り込むことは，基本的な考え方である．しかし実際は，開発プロジェクトにおける時間，コスト，要員等の制約の為，十分な品質を作り込むことが困難であることが多い．そこで，テスト工程における品質確保が必要不可欠となる．

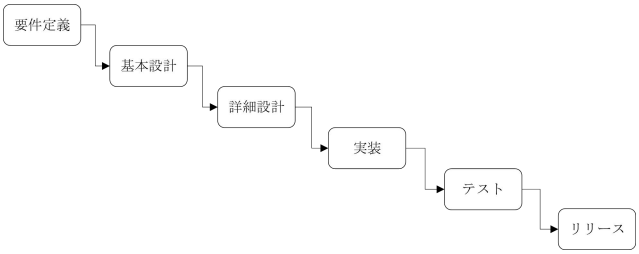

図 1　ウォーターフォールモデル

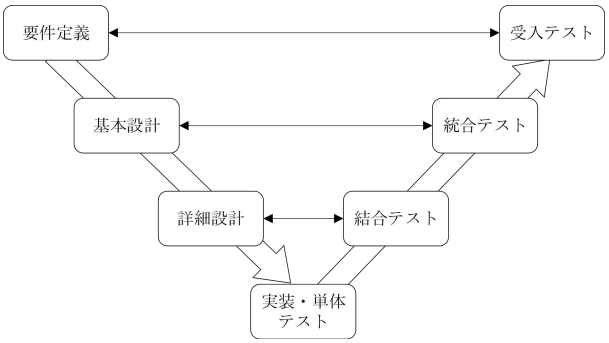

図 2　V 字モデル

1.2 課題

テスト工程では，事前に作成されたテスト項目に従い，テストを実施する．テストによってプロダクトの不具合が摘出された場合，当該不具合を修正し，修正箇所に対する再テストを行う．しかし，これだけでは十分な品質を確保できないことが多い．その理由は，プロジェクトの制約条件（時間，コスト，要員など）により，テスト項目の網羅性が不十分であることが多いからである．また，テスト項目の作成過程で，必要なテスト観点が抜けてしまうこともある．

そこで，テスト結果から品質上の弱点を分析し，テストで摘出した不具合以外の改善を行い，改善箇所に対して再テストする．このプロセスを品質が十分であると評価されるまで繰り返すことで，プロダクトの品質が向上すると同時に，テストの品質も向上する．しかし，分析，評価，改善の実施方法は，論理性が乏しく，属人性が高い傾向にある．そのため，品質確保が十分に行われず，不具合が次の工程に潜在することが多い．

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

ⓒ 2018 Information Processing Society of Japan

2. 関連研究

ソフトウェアの品質評価に関する従来の研究は，主に品質の特性（複雑さなど）に関するメトリクスを用いて，品質を定量的に表すことが目的とされてきた[2] - [6]．また，ISO/IEC25010[9]が定める品質特性に関するメトリクスを整理した活動報告も行われている[13][14]. メトリクスと品質の関係を示すために，設定したメトリクスの測定値と不具合の関係性を評価する研究も行われている[7][8].

これらの関連研究の手法が適用されるタイミングの多くは，設計書やプログラムといったプロダクトの設計・製造工程であることが想定される．一方，前節で述べたようにプロダクトの設計・製造工程で品質を作り込むことは重要であるが，現実的には時間，コスト，要員といったプロジェクトの制約により，十分な品質を作り込むことが困難であることが多い．よって，テスト工程での品質確保が必要不可欠となるが，内在する属人性を排除することを目的とした研究は少ない.

3. 品質評価・改善モデルの構築

3.1 アプローチ

前節の課題を受け，本稿ではテスト工程における品質評価・改善モデルを構築する．ここでモデルの構築とは，従来の品質確保・評価に内在する属人性を排除することを意味する．モデルを構築する際のポイントを以下に示す.

- 品質の定量的表現
- 品質評価・改善プロセスの形式化
- 品質低下の根本原因（root cause）の特定

3.2 品質評価・改善の対象

本稿が扱う品質評価・改善モデルが対象とするプロダクトと不具合について述べる.

3.2.1 プロダクト

不具合が作り込まれる工程と，そのプロダクトの対応を表 1 に示す.

表 1　工程とプロダクト

工程	プロダクト
要件定義	要件定義書
設計	システム仕様書，設計書

この中で，特に不具合が潜在する傾向が高いプロダクトは，システム仕様書，及び設計書である．その理由を以下に示す.

- 書式等の記述ルールが明確に定められていないことが多く，記述不足や記述ミスが生じやすい.
- テストのように記載内容の抜け漏れを動的に発見できない.

要件定義書については，誤りがないことを前提とするが，記述が不十分であることは想定しなければならない．要件定義書の記載が不十分な場合は，要求者と開発者間で不十分な点を認識し，システム仕様書にて補足する必要がある.

そこで本稿では，品質評価・改善の対象となるプロダクトを設計工程のプロダクトであるシステム仕様書，及び設計書とする．以降，システム仕様書を単に仕様書と記述する.

3.2.2 不具合

設計工程で不具合を作り込んだ原因を追究すると，プロダクトの性質（複雑さなど）によるものと，ヒューマンエラーによるものに大別される．ヒューマンエラーは，見間違い，勘違い，思い込み，書き間違い等，人的要因に起因するものである. 本稿ではヒューマンエラーは対象外とし，プロダクトの性質による不具合を対象とする.

3.3 品質評価・改善プロセスの詳細

テスト工程における品質評価・改善プロセスを図 3 に示す．テスト項目を消化したのち，テスト結果を基に品質評価を行う．品質評価では，プロダクト全体を対象として，テストで摘出した不具合から導出される改善点の有無を確認する. 品質評価の結果，品質改善が必要と判断した場合，プロダクトに対して改善を行う．その後，改善された部位に対して追加テストを行う．このサイクルを改善が不要と判断されるまで繰り返す.

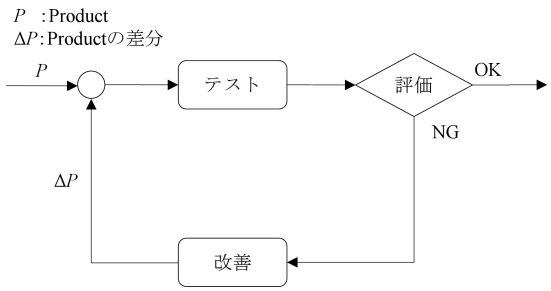

図 3　品質評価・改善プロセス

図 3 に示した品質評価・改善の繰り返しプロセスにおける，テストから改善までの 1 プロセスに着目し，その内容の詳細を図 4 に示す.

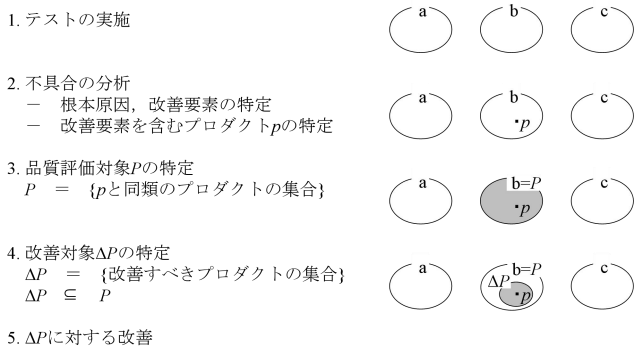

図 4　品質評価・改善の詳細

図 4 の No.1 のテストで摘出した不具合の分析を No.2 で行う．No.2 の改善要素とは，摘出した不具合の根本原因に対応する，プロダクトの改善すべき性質を表す．この改善要素を有するプロダクト p を特定する．No.3 では，品質評価対象 P を特定する．品質評価対象 P とは，テスト対象であり，かつ p と同じ種類（仕様書，設計書（クラス図，シーケンス図等））のプロダクトを表す．例として，3 種類のプロダクトがあるとし，各種のグループに a, b, c と名付ける．図 4 では，No.2 で特定した p がグループ b に属しており，No.3 で品質評価対象 P が b であると特定される．No.4 では，改善対象ΔP を特定する．改善対象ΔP は，P の要素のうち改善が必要なプロダクト（No.2 で特定した改善要素を有するプロダクト）の集合である．p はΔP に含まれる．No.5 でΔP の各要素に対する改善を行う．

3.4 品質評価・改善の問題点の明確化

本節では，不具合事例をもとに，テスト工程の品質評価・改善の問題点を明らかにする．

3.4.1 不具合事例

不具合事例の内容を以下に示す．

(1) 対象システム

金融商品（保険など）の加入システムである．加入者は商品を購入する際に申請し，審査を受け，審査結果が OK の場合のみ契約することができる．加入システムのユースケース図，及び加入者と商品のクラス図を図 5，図 6 に示す．

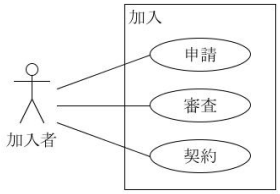

図 5　加入システムのユースケース図

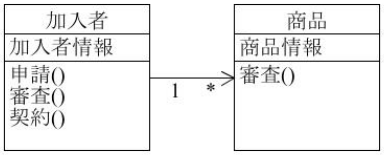

図 6　加入者と商品のクラス図

(2) プロダクト

加入システムの各機能（申請，審査，契約）に対するプロダクトの ID を表 2 のように定義する．

表 2　プロダクトの ID

機能	仕様書	設計書	プログラム
申請	s1	d1	c1
審査	s2	d2	c2
契約	s3	d3	c3

(3) 現象

ある加入者の審査を行ったところ，例外によりプログラムが異常終了した．

(4) 直接原因（immediate cause）

加入システムの 1 つのパッケージである審査機能では，審査結果が OK 時の処理（正常処理）と，審査が NG 時の処理（異常処理）が各々2 ケースある．各処理条件は，入力データの組み合わせパタンに対応している．審査機能の仕様書 s2 において，2 つの異常処理の条件の内，1 つの条件の記載が漏れていた．その結果，該当するデータパタンが入力された時に，例外によりプログラムが異常終了した．

審査処理のシーケンス図を図 7 に示す．判定条件 a214 が，記載が漏れた処理条件である．

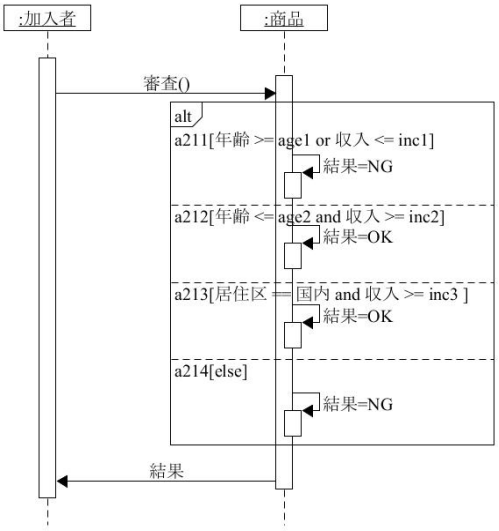

図 7　審査処理のシーケンス図

(5) 根本原因（root cause）

直接原因は，審査機能の設計書 d2，及びプログラム c2 における判定条件の記載漏れである．更にその原因は，表 3 に示す仕様書 s2 における判定条件に関する記載が不十分であることにある．つまり，仕様書 s2 において判定条件 a214 が，明確に記載されていないことが根本原因である．仕様書の記載が不十分となる原因として，要求者と開発者のレビュー不足が考えられるが，ここでは言及しない．

表 3　審査機能の仕様書

仕様書 ID：s2	仕様書名：審査
a211	年齢が age1 以上，または収入が inc1 以下の場合は NG
a212	年齢が age2 以下，かつ収入が inc2 以上の場合は OK
a213	居住区が国内，かつ収入が inc3 以上の場合は OK
a214	それ以外は NG

3.4.2 問題点の明確化

前節で述べた不具合事例をもとに，従来の品質評価・改善の問題点を明確化する．

(1) 改善要素の特定の問題

従来の品質評価・改善では，直接原因から改善要素を特定することが多い．その理由は主に 2 つある．一つは，プロダクト間の依存関係が明確でないために，改善要素を有するプロダクトを特定できないことである．もう一つは，プロジェクトの制限により，改善要素を特定する為の調査に必要な時間が費やせない，あるいは改善要素を特定することで将来の品質低下のリスクを低減する余裕がないことである．不具合事例の場合，直接原因は設計書 d2 の判定条件漏れである．したがって，設計書の判定条件の記載漏れが改善要素として特定されることが多い．しかし本来であれば, 仕様書 s2 の判定条件に関する記載の十分性が改善要素として特定されるべきである．

(2) 品質評価対象の特定の問題

改善要素の特定が適切でない場合，品質評価対象の特定も適切に行われない．不具合事例のように改善要素が設計書にあると特定されると，設計書 d1, d2, d3 が品質評価対象として特定される．しかし本来の改善要素は仕様書にある．よって，仕様書 s1, s2, s3 が品質評価対象として特定されるべきである．

(3) 改善対象の特定の問題

従来の品質評価・改善では，改善対象を特定した理由が定性的であることが多い．不具合事例の場合，品質評価対象が設計書 d1, d2, d3 と特定される場合，これらの中で改善が必要な設計書が改善対象となる. 例えば設計書 d1 を改善対象に特定した理由として, 設計書 d1 に記載されている判定条件が不十分であるといった定性表現が用いられる．この場合，どの程度不十分であるか分からない．本来であれば, 設計書 d1 に記載されている判定条件の十分性は基準値以下というように，定量的に表現されるべきである．

(4) 品質評価方法の問題

従来の品質評価・改善では，不具合の潜在件数を経験的に設定し，それを改善による不具合の摘出目標とすることが多い．本来であれば，判定条件の不十分性に関するメトリクスを設定し，それを用いて定量的な評価が行われるべきである．

3.5 品質評価・改善モデル

3.5.1 問題に対する解決策

前節で述べた問題点を解決するこが，品質評価・改善モデルを構築するアプローチとなる．以下に，各問題点に対する解決策を述べる．

(1) トレーサビリティ

改善要素を特定するためには，改善要素に関するプロダクト間の依存関係が追跡可能である必要がある[10]．一般的に，2 つのプロダクト間の追跡可能性はトレーサビリティマトリクスとして表現される．このトレーサビリティマトリクスを改善要素が特定できるように拡張する．トレーサビリティマトリクスは，必要に応じて特定すべき改善要素ごとに構築する必要がある．

今，不具合事例の改善要素を特定するためのトレーサビリティマトリクスを構築することを考える．不具合事例の不具合の直接原因は設計書 d2 の判定条件の記載漏れである．また，根本原因は仕様書 s2 の判定条件に関する記載不足であり，改善要素は判定条件の十分性である．よって，これらのトレーサビリティを表現するためには，仕様書，設計書，判定条件の 3 軸が必要となる．したがって，通常 2 次元であるトレーサビリティマトリクスを 3 次元に拡張する．不具合事例の場合，判定条件 a211, a212, a213, a214 が仕様書 s2，及び設計書 d2 に記載されるべきであるが，判定条件 a214 の記載が仕様書 s2，及び設計書 d2 から漏れた．この場合，トレーサビリティマトリクスは図 8 のように表現される．図 8 は，審査機能のみを抜粋した．ここで，破線の丸印は, 本来あるべき判定条件 a214 の記述がないことを意味する．

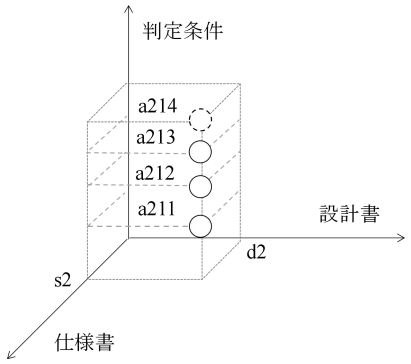

図 8　審査機能のトレーサビリティマトリクス

図 8 では判定条件 a214 のトレーサビリティを破線の丸印で記載した．しかし実際は，トレーサビリティマトリクスにおいて判定条件 a214 の項目体が存在しない. この場合, 改善要素を特定するためには，トレーサビリティマトリクス上，判定条件 a214 の項目がないことを確認する．判定条件 a214 の項目がないということは，仕様書 s2，及び設計

書 d2 に判定条件 a214 が定義されてないことを意味する．よって，改善要素が仕様書 s2 の判定条件の漏れであることが特定できる．改善要素と改善要素を有するプロダクトが正しく特定されれば，同類のプロダクトの集合である品質評価対象 P も正しく特定される．

(2) 改善要素の候補

改善要素を特定するためには，調査・分析が必要であり，時間を要する．一方，特定される改善要素は，ある程度パタンが決まっている．そこで，改善要素の特定を効率的に行うために，その候補を事前に洗い出す．

設計工程のプロダクトの性質に関する改善要素は，記載内容の曖昧さ，及び記載内容の複雑さに大別される．記載内容の曖昧さが問題視されやすい設計工程のプロダクトは，仕様書である．また，記載内容の複雑さは，仕様書，及び設計書の双方で問題視され得る．考えられる改善要素を列挙した表を表 4 に示す．以降，これを改善要素候補表と呼ぶ．種別は改善要素が内在し得る設計プロダクトを表す．

表 4　改善要素候補表

分類	種別
1.曖昧さ	
1.操作対象（目的語），または操作内容（動詞）が細分化されていない	仕様書
2.判定条件が網羅されていない	仕様書
2.複雑さ	
1.場合分けが多すぎる	仕様書 設計書
2.ユースケース（処理シーケンス）が長すぎる	
1.ユースケースの記述量が多い	仕様書
2.ユースケースの文章の情報量が多い	仕様書
3.ある情報が経由するパスが長い	設計書
3.クラスが大きすぎる	設計書
4.メソッドが大きすぎる	
1.メソッド内の独立したロジックが多い	設計書
5.クラス間の結合が強すぎる	
1.送信メッセージ（メソッド）の引数が多い	設計書
2.送信メッセージ（メソッド）が多い	設計書

(3) 改善要素評価関数

ここでは，改善対象ΔP を定量的に特定するための改善要素評価関数を導出する．

a) 改善要素のメトリクス

表 4 の改善要素に関する品質を定量的に評価するためには，各改善要素を定量的に表現するメトリクスを設定する必要がある．表 5 に，改善要素候補表の一部の改善要素に対するメトリクスの設定例を示す．表 5 の No.は，表 4 の各改善要素の番号に対応する．

表 5　改善要素に対するメトリクスの設定例

No.	メトリクス
1.2	if-else/case-default 設定率： ある条件（if）に対するその他の条件（else），複数の条件（case）に対するそれ以外の条件（default）の設定率．
2.5.1	送信メッセージ（メソッド）の引数の数
2.5.2	UCBO： コミュニケーション図において，あるクラスが他のクラスに送信するメッセージの数．ただし，同一メッセージの重複は排除する[7]．

メトリクスの測定値を評価するためには，基準値が必要である．今，プロダクト p_i のメトリクスの測定値を $m(p_i)$，その基準値を $m_0(p_i)$とおく．品質改善が必要である場合の一般的な m と m_0 の関係は，以下の 2 通りが考えられる．

$$m(p_i) > m_0(p_i),\ m(p_i) < m_0(p_i)$$

前者は m が大きいほど品質が劣化する場合であり，後者はその逆である．ここで，表現を統一するために，後者の場合，m を以下のように m'に変換する．

$$m'(p_i) = \begin{cases} 2m_0(p_i) - m(p_i) & if\ 0 \le m \le 2m_0 \\ 0 & if\ 2m_0 < m \end{cases} \qquad (1)$$

変換された m'を新たに m と置くと，品質改善が必要な条件は，$m(p_i) > m_0(p_i)$ に統一される．

補足 :品質改善を必要としないプロダクト p_i を $\overline{p_i}$ と置く．すると，$m(\overline{p_i}) \le m_0(p_i)$ が成立する．したがって，$m_0(p_i)$は $m(\overline{p_i})$ の最大値となる．

b) 改善要素評価関数

メトリクスとその基準値は，改善要素毎に様々な値をとり得る．一方，品質評価・改善を形式化するには，とり得る値を一定の範囲に収める方が都合が良い．そこで，そのような性質を持つ，p_i のメトリクス m に関する改善要素評価関数 e_i を定義する．

ここで，e_i を図 9 のように 0 以上 1 以下の値をとり，$m(p_i) > m_0$ のとき 1 から 0 へ曲線を描きながら降下する関数として定義する．

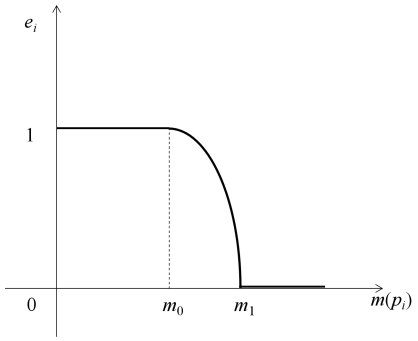

図 9　改善要素評価関数

e_i の曲線部分に 2 次関数を適用すると，e_i は以下のように表される．

$$e_i = f(m(p_i), m_0(p_i)) = \begin{cases} 1 & if\ 0 \le m < m_0 \\ g(m) & if\ m_0 \le m < m_1 \\ 0 & if\ m_1 \le m \end{cases} \quad (2)$$

$$g(m) = -a(m - m_0)^2 + 1, \quad a > 0 \quad (3)$$

すると，m_1 は以下のように導出される．

$$m_1 = m_0 + a^{-\frac{1}{2}} \quad (4)$$

また暫定的に，$m_1 = 1.5m_0$ と設定すると，a は以下のように導出される．

$$a = \frac{4}{m_0^{\ 2}} \quad (5)$$

上記で定義した改善要素評価関数 e_i は，評価対象 p_i に対して次の性質を有する．

- e_i=1 の場合，品質は基準を満たしており，改善は不要
- e_i<1 の場合，品質は基準未満であり，改善を要する
- e_i<e_j の場合，p_i は p_j よりも品質が劣っている

(4)　品質評価関数

ここでは，改善要素に関する品質評価対象 P の品質を定量的に評価する品質評価関数 E_m を以下のように定義する（$n = |P|$）．

$$E_m = h(m(p_i), m_0(p_i)) = \sum_{i=1}^{n} w_i e_i \quad (6)$$

E_m は e_i の加重平均を表し，0 以上 1 以下の値をとるように p_i に対する重み w_i を設定する．p_i が重要であるほど，w_i の値を大きく設定する．品質評価関数 E_m は，P に対して以下の性質を有する．

- E_m=1 の場合，品質は基準を満たしており，改善は不要
- E_m の値が小さいほど品質改善の必要性が高い

ここで p_i の重要度の例を以下に示す．

例 1）業務上の処理の重要度（金額計算，合否判定，個人情報取り扱い等）

例 2）システムの内部構成上の重要度（メッセー送受信数，依存度等）

表 6 に，例 1）業務上の処理区分とその重要度の具体例を示す．この具体例では重要度を 3 段階評価している．

表 6　処理区分と重要度

処理区分	重要度
金額計算	3
合否判定	3
個人情報取り扱い	3
統計処理	2
情報参照	1

プロダクト p_i の重要度を s_i とすると，w_i は以下のように表される．

$$w_i = \frac{s_i}{\sum_{j=1}^{n} s_j} \quad (7)$$

3.5.2 品質評価・改善の形式化

ここでは, 前節で述べた解決策の実施手順を示すことで, 品質評価・改善を形式化する．

ステップ 1：改善要素を特定する

プロダクト間のトレーサビリティマトリクスを利用して, 直接原因から根本原因に対応する改善要素を特定する．改善要素は，表 4 の改善用要素候補表から選択する．

ステップ 2：品質評価対象 P を特定する

不具合の根本原因と改善要素を有するプロダクトと同類のプロダクト p_i の集合 P を特定する（$n = |P|$）．

ステップ 3：改善要素に関するメトリクス m を決定する

改善要素に関するメトリクス m を決定し，その基準値 m_0 を定める．

ステップ 4：改善要素評価関数 e_i を決定する

式(2)に従い，改善要素評価関数 e_i を決定する．

ステップ 5：品質評価関数 E_m を決定する

式(6)に従い，品質評価関数 E_m を決定する．

ステップ 6：改善対象ΔP を特定し，高優先順に改善する

品質評価関数 E_m の性質より，$E_m < 1$ の場合，P は品質改善が必要である．しかし現実的には，プロジェクトの状況により，品質改善を途中で終了することを判断することも多い．この場合，品質改善を要する条件は $E_m < c\ (c \le 1)$ となる．このような状況下では，品質改善が必要な P の各要素 p_i を改善する場合，品質改善の必要性が高い順に改善を行うことが求められる．ここで，p_i の品質評価値の最大値は w_i（$e_i = 1$ としたときの $w_i e_i$ の値）である．よって，P において最大値 w_i と実測値 $w_i e_i$ の差 $w_i - w_i e_i$ が最大となる p_i が, 最も品質改善を必要とするプロダクトであると解釈することができる．

本ステップは，図 10 のようにアルゴリズム形式で表現できる．

Input　：w_i（p_i の重み），e_i（p_i の改善要素評価値），
　　P（品質評価対象）

Output：ΔP（改善対象）

while ($E_m < c$) do {
　select p_i from P where max{w_i-$w_i e_i$, $1 \le i \le n$ ($n = |P|$)}
　improve p_i
　add p_i to ΔP
}

図 10　ΔP を特定するアルゴリズム

4. 品質評価・改善モデルの適用例

前節で述べた不具合事例に対して品質評価・改善モデルを適用する．

ステップ 1：改善要素を特定する

不具合の直接原因を元に，判定条件に関するプロダクト間のトレーサビリティマトリクスを追跡する．トレーサビリティマトリクスを図 11 に示す．追跡の結果，本来あるべき判定条件 a214 がマトリクスにないことが分かる．よって，根本原因は仕様書 s2 における判定条件 a214 の記載不足であり，表 4 の改善要素候補表 No.1.2「判定条件が網羅されていない」が，改善要素であると特定される．

図 11 に示されるように，不具合事例では申請機能，契約機能にそれぞれ複数の判定条件が設定されており，判定条件 a122 が定義されていない．

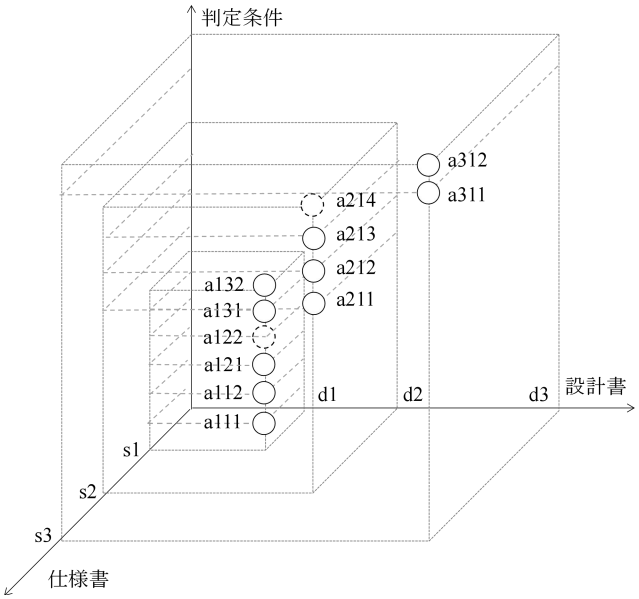

図 11　判定条件のトレーサビリティマトリクス

申請機能と契約機能の判定条件を抜粋した仕様書を表 7，表 8 に示す．

表 7　申請機能の仕様書

仕様書 ID：s1	仕様書名：申請
a111	新規加入者の場合，情報が設定されていない入力フォームを表示する．
a112	それ以外（申請履歴のある加入者）の場合，個人情報の入力フォームに前回入力した情報を設定して表示する．
a121	有職者を選択した場合，勤続年数，年収の入力項目を表示する．
a122	それ以外（無職者）を選択した場合，退職年月日の入力項目を表示する．
a131	住居が賃貸を選択した場合，家賃の入力項目を表示する．
a132	それ以外（持ち家）を選択した場合，購入額の入力項目を表示する．

表 8　契約機能の仕様書

仕様書 ID：s3	仕様書名：契約
a311	審査結果が OK の場合, 契約ボタンを活性化する．
a312	それ以外の場合，契約ボタンを非活性化する．

ステップ 2：品質評価対象 P を特定する

改善要素を有するプロダクトは仕様書 s2 である．よって，仕様書 s1, s2, s3 が品質評価対象 P と特定される．

ステップ 3：改善要素に関するメトリクス m を決定する

表 5 より，改善要素に関するメトリクス m は，No.1.2「if-else/case-default 設定率」と決定される．

ここで，品質改善が必要である m と m_0 の関係は $m(p_i) < m_0(p_i)$である．したがって式(1)に従い，$m(p_i)$を変換する．

また，ある条件に対するそれ以外の条件は，必ず設定されなければならない．したがって，メトリクスの基準値 m_0 は，1 である．

ステップ 4：改善要素評価関数 e_i を決定する

$m_0 = 1$ であるから，式(2)の改善要素評価関数 e_i の m_1, a は式(4), (5)より次のように算出される．

$$m_1 = 1 + 1/2 = 1.5,\ a = 4/1 = 4$$

ステップ 5：品質評価関数 E_m を決定する

E_m は，式(6)に従う．ここで不具合事例の場合，プロダクト p_i の重要度 s_i はいずれの i についても一定である．したがって式(7)より，$w_i = s_i \Big/ \sum_{j=1}^{3} s_j = 1/3$ となる．

ステップ 6：改善対象ΔP を特定し，高優先順に改善する

P の要素である s1, s2, s3 の各々について，m と変換後の m である m', e_i, 及び w_i-$w_i e_i$ の計算結果を表 9 に示す．

表 9　計算結果

仕様書 ID	m	m'	e_i	w_i-w_ie_i
s1	0.67(2/3)	1.33	0.56	0.15
s2	0 (0/1)	2	0	0.33
s3	1 (1/1)	1	1	0

表 9 より，w_2-w_2e_2 > w_1-w_1e_1 > 0 であることが分かる．よって改善対象はΔP={s1, s2}として特定され，改善の優先順は，s2, s1 となる．

5. 考察

表 5 のメトリクスは, 表 4 の改善要素の一部に対する設定例である. 例えば, 不具合事例ではメトリクスとして表 5 の No.1.2「if-else/case-default 設定率」を採用した．しかし，一般的には，if や case に対応する判定条件の記載が漏れているケースが考えられる．よって，これに対応するメトリクスの検討が必要である．このように，各改善要素に対するメトリクスは引き続き検討し，設定する必要がある．また，メトリクスの基準値は，決定的なものとそうでないものがある．例えば，if-else/case-default 設定率の基準値は 1 であり，決定的である．一方それ以外の基準値は，決定的でないものもある．したがって，これらについてはメトリクスの基準値を適切に設定する方法を検討する必要がある．

本稿では，品質評価・改善モデルの適用例を示した．これにより，手順は明らかに示されたが，他の改善要素に対する適用例を示すことで，本モデルの実用性を示す必要があると考える．また，本モデルの効果を示すことも必要と考える．この効果は，本モデルを適用した結果，どの程度品質が改善されたかを示すことで表現される. 具体的には，モデルを適用して品質改善した結果，摘出される潜在的な不具合の量を示すことが望ましい．

本稿はウォーターフォールモデルを前提としたが，テスト工程の品質評価・改善は，アジャイル開発でも重要である．アジャイル開発では，テストファーストが採用されることが多い．テストファーストは，最初にテストケースを記述することによって仕様の曖昧性を排除するといったメリットがある．一方，テストの網羅性の観点から，テストファーストではテストの十分性の確保が重要であり，その十分性を評価することが課題である．本稿で取り上げた品質評価 ･改善モデルは, テスト品質の改善の側面を持つ. よって，テストファーストの課題解決の糸口となり得ると考える．

6. おわりに

本稿では，アプリケーションシステム開発のテスト工程における品質評価・改善モデルを構築した．これにより，従来の品質評価・改善プロセスの課題であった，属人性が排除されたものと考える．本モデルを適用することで，開発過程における品質確保の十分性が向上するものと考える．将来的には，構築したモデルの適用が自動化されることで，実用性が向上することが期待される．

参考文献

[1] W. W. Royce. Managing the development of large software systems: concepts and techniques. Proc. of ICSE 9th Int'l Conference on Software Eng., 1987, pp. 328-338.
[2] T. J. McCabe. A complexity measure. IEEE Trans. on Software Eng., 1976, Vol. SE-2, No.4, pp.308-320.
[3] Chidanber, S.R. and Kemercer, C.F.. A Metrics Suite for Object Oriented Design. IEEE Trans. Softw. Eng., 1994, Vol.20, No.5, pp.476-493.
[4] L. C. Briand, P. Devanbu, and W. Mélo. An Investigation into Coupling Measures for C++. Proc. of The IEEE 19th Int'l Conference on Software Eng., pp.412-421, Boston, USA (May, 1997).
[5] Bansiya, J. and Davis, C.G.. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Trans. Sft. Eng., 2002, Vol.28, No.1, pp.4-17.
[6] 佐藤美穂，田村慎吾，上田賀一. UML 設計を対象とした品質評価モデルの検討. 情報処理学会論文誌. 2007, Vol.49, No.7, pp.2319-2327.
[7] Ana Erika Camargo Cruz and Koichiro Ochimizu. A UML approximation of three chidamber-kemerer metrics and their ability to predict faulty code across software projects. IEICE Trans. on Information and Systems, 2010, Vol.E93-D, No.11, pp.3038-3050.
[8] 田村慎吾，佐藤美穂，上田賀一. UML モデルを対象にした設計品質評価のためのメトリクスの提案. 電子情報通信学会技術報告（KBSE）. 2008, Vol.107, No.540, pp.25-30.
[9] ISO/IEC 25010:2011 System and software engineering - System and software Quality Requirements and Evaluation(SQuaRE) - System and software quality models.
[10] CMMI Product Team. CMMI for Development, Version 1.3. Carnegie Mellon University, Software Engineering Institute, CMU/SEI-2010-TR(2010).
[11] マレー・カンター著，村上雅章訳．ソフトウェア開発管理の要．ピアソン・エデュケーション，2002.
[12] 上村和夫．ソフトウェア品質知識体系ガイド（第２版）－SQuBOK GuideV2－．東京，株式会社オーム社，2014.
[13] METI ソフトウェアメトリクス高度化プロジェクト．“情報システム／ソフトウェアの品質メトリクスセット”．経済産業省．http://www.meti.go.jp/policy/mono_info_service/joho/cloud/2011/11_03.pdf, (参照 2018-01).
[14] ソフトウェアメトリクス高度化プロジェクトプロダクト品質メトリクス WG．“システム／ソフトウェア製品の品質要求定義と品質評価のためのメトリクスに関する調査報告書”. 経済産業省．http://www.meti.go.jp/policy/it_policy/softseibi/metrics/20110324product_metrics2010.pdf , (参照 2018-01).