2 テストプロセスとテストプロセス改善

大西建児・湯本　剛
（株）豆蔵

ソフトウェアの開発において，開発総工数のうちソフトウェアテストプロセスが占める割合が多いにもかかわらず，産業界の多くの開発現場ではソフトウェアテストのプロセスが最適化されているとはまだまだ言い難い状況である．昨今のソフトウェア開発で求められている，できるだけ短い開発サイクルでソフトウェア品質を維持しつつ，開発コストとのバランスをとるようにするためには，ソフトウェア開発全体のプロセス改善とともに，テストのプロセス改善をも同時に進めることが不可欠である．本稿では基本的なテストプロセスの解説およびテストプロセス改善手法について紹介する．

テストプロセスの定義の変遷

はじめにテストプロセスに対する定義の変遷に触れてみたい．1979年に出版された古典的なテストの文献によるとソフトウェアテスト（以降テストと記す）は[1]「エラーをみつけるつもりでプログラムを実行する過程である」と定義されていた．90年代に出版されたテストのバイブルとも呼べる文献2）でも「テストは不具合を発見するプロセスである」のように同様な定義がされている．これは暗にだが，90年代まではテストは不具合を見つけるための（単一の）プロセスであるように定義されていたことの反映と言える．

しかし，近年になってテストに対する定義は変わってきた．たとえば，1999年出版されたテストプロセス改善モデルを提唱した文献3）では，テストとは「システムの特性を理解し，実態と要求されている仕様との間の差異を明らかにすることを目的とした計画，準備，測定のプロセスである」と定義している．また，2000年以降に出版された別の文献では「ソフトウェアの品質を測定し改善するための，テストウェアを設計・活用・保守するライフサイクルプロセスである」と定義している[4]，☆1．

過去の2つの定義と近年の2つの定義からは，テストが単純な1つのプロセスとしての定義から，「計画，準備，測定」のような複数のプロセスからなるライフサイクルとして捉える定義へと大きく変化したことが理解できよう．

［開発モデルにおけるテストの位置づけ］

ソフトウェアの開発モデルにおけるテストの位置づけからも，定義の変化の流れを見てとることができる．

古典的なソフトウェア開発モデルである，ウォーターフォールモデル（**図-1**）では，要件分析→設計→コーディング→テストというように，テストが開発の最終工程として最後に実施するものと位置づけられている．

90年代までの定義であれば，テストは「不具合を発見する」ために行うので，ウォーターフォールモデル中においてテストの位置づけが開発としての最終工程に置かれていたとしても違和感は覚えないであろう．

ソフトウェア開発モデルとして代表的なSLCP（Software Life Cycle Processes）においても，テストの位置づけはウォーターフォールモデルと同様に扱われている．また，開発モデルの中にテストレベルの定義を加えたV字モデル（**図-2**）を見ても，テストはコーディングの後から始まるものになっているため，ウォーターフォールモデルの変形と言えよう．

では，ウォーターフォールモデルにテストの「計画，準備，測定のプロセス」を当てはめてみるとどうなるだろうか．ウォーターフォールモデルやV字モデルで

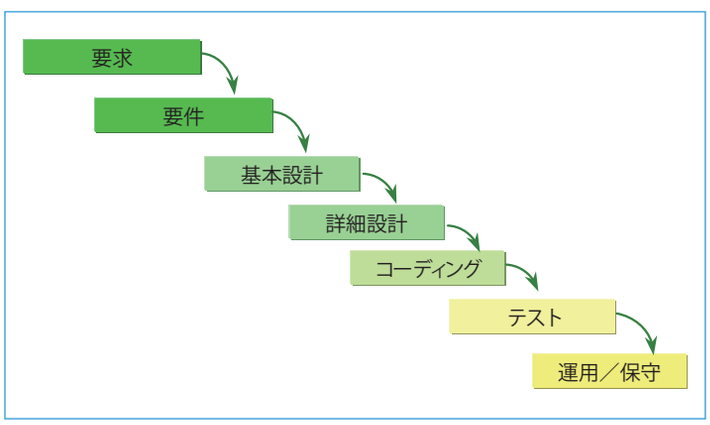

図-1　ウォーターフォールモデル

☆1 筆者（湯本）による翻訳．

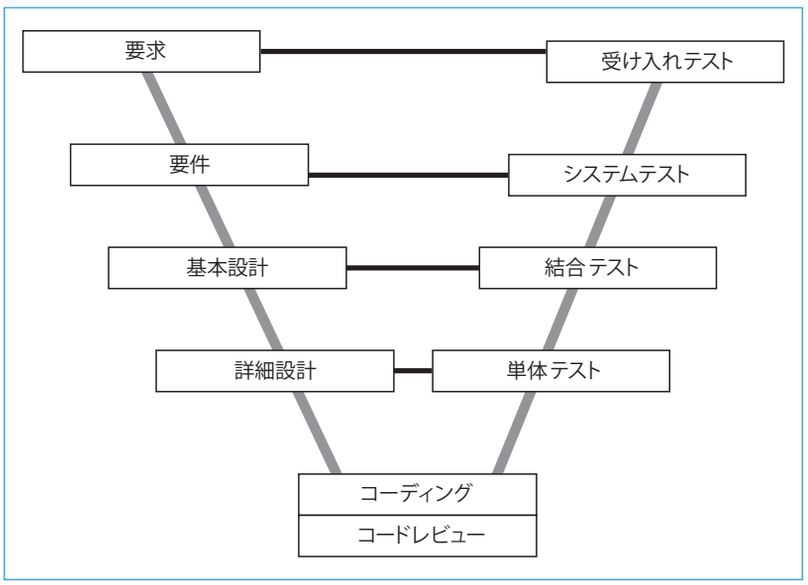

図 -2　V 字モデル

は，テストの「計画，準備，測定」のうちの，測定（テスト実行に該当する）の部分しかモデルの中では表現されていないため，測定以外は上手く収まらない．これに対して V 字モデルの後継のモデルとして 2002 年に Andreas Spillner によって W モデル（**図 -3**）が提案された．このモデルでは要件分析の段階でテスト活動を開始し，各設計フェーズに対応する各テストレベルの計画を，同時進行的に進めていくというプロセスを提案している．このアプローチであればテストをライフサイクルプロセスと捉えても違和感なく扱うことができる．

こういったテストプロセスの定義の変遷について，ソフトウェアエンジニアリングの知識体系を整理したものである SWEBOK（software engineering body of knowledge）[5] では「テスティングは故障を発見するアクティビティから，開発，保守プロセスにまたがる体系的なアクティビティへと進化している」と表している．

［ライフサイクルとして捉える理由］

では，なぜテストをライフサイクルとして捉える必要がでてきたのだろうか．それは開発プロセス全体に渡りテストプロセスが関与するため，ライフサイクルとして捉えないことには十分性が確保できないからである．

テストの総時間はデバッグを含めた場合，開発全体の 30％から，多いときには 90％を占めると報告されている [6]．ソフトウェア開発の規模が大きく複雑になり，かつ短納期を求められる中で，ただ単にテストで納品間近に不具合を取り除くというやり方だと，テストがコスト・納期に対するボトルネックになってしまう．だからといって十分にテストをしないまま開発を終了させてしまうと，市場に不具合を流出させる結果となり得る．

だからこそ，テストの十分性を考慮し，かつ効果的にテストを実行するためには，テストの計画・分析・設計・保守などを考慮しなくてはならない．つまりテストをライフサイクルとして捉え，各プロセスの役割を明確にしてソフトウェア開発を進めることが不可欠となる．実際，テストはテスト実行以外の作業のほうが作業ボリュームは多い．筆者（湯本）の経験だが，テストを予定通り進めることができたプロジェクトでは，テスト実行はテスト全体にかけた工数の半分以下であった．また，テスト工数全体の 45% 程度という報告をしている文献もある [3]．

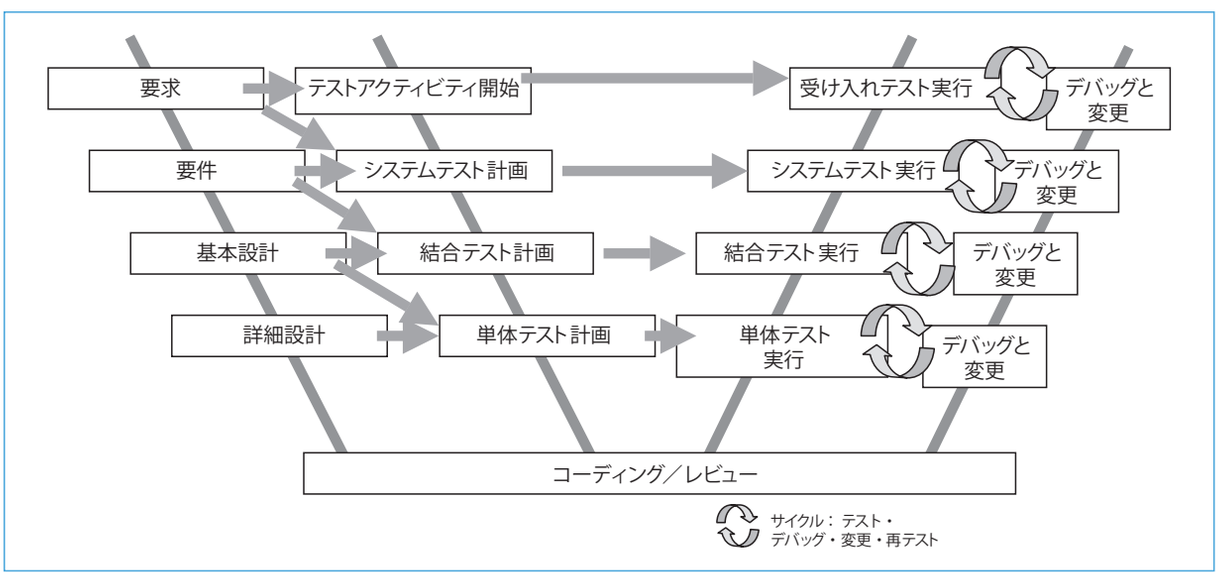

図 -3
W モデル

2 テストプロセスとテストプロセス改善

［3つの効果］

テストをライフサイクルとして捉えることによる具体的な効果として「欠陥予防」「テスト十分性確保」「手戻り削減」の3つが挙げられる．後述するテストプロセス改善では，テストを確実に実施することを目的にするだけでなく，これら3つの効果を最大限引き出し，開発プロジェクトにとって価値が高いテストプロセスとしていくことも目的としている．

1：欠陥予防

ソースコードに欠陥が混入することを予防することでテストを効率的にすすめることができる．このことをSWEBOKでは「問題が起きてから収拾するという見方から，問題が起きることを予防するという見方に変わってきている」と言及している．テストをベースにした欠陥予防の例としては，各設計フェーズ完了直後にテスト設計を実施して，テスト容易性を確認する方法が挙げられる．

2：テスト十分性の確保

市場での不具合発生による損失に対するリスクである，「品質リスク」とコスト・納期とのトレードオフでテスト十分性を考えることによって，コスト・納期の目標に合致したテストを行うことができる．一例としては，品質リスクの分析を計画立案の前に行い，テスト対象範囲や優先順位付け結果をテスト計画に反映する方法があり，この考え方はリスクベースドテスティングと呼ばれている．

3：手戻り削減

テストを最終工程に限定せず，計画的に適宜実施することで，デバッグや設計見直しなどの手戻りを最小限に抑えることができる．たとえば，システムの応答時間を評価するために行う性能テストは，一般的に開発の最終段階であるシステムテストで実施することが多い．しかし最終段階ではなく，システムとして統合する前の段階のコンポーネントやサブシステムといった段階から，それぞれが目標とする性能を実現できているか確認するテストを可能な限り早期に計画・実施すれば，問題の発見を早く行えるので，手戻りを最小限で済ませることができる．

［典型的なテストプロセスの定義］

それではテストをライフサイクルとして捉えた場合の典型的なテストの各プロセスの役割を，テストの国際資格認定を行う団体であるISTQB[7]（International Software Testing Qualifications Board）のシラバスでの定義をもとに紹介しよう．ISTQBでは，以下のプロセスを定義している．

- 「計画とコントロール」
- 「分析と設計」
- 「作成（implementation）と実行」
- 「終了条件の検証とレポート」
- 「終了作業」

1：計画とコントロール

計画は，どのようにテストを実行するかをあらかじめ決めた上で，スケジュールを立案するプロセスである．

前述したように，ただテストをしているだけでは，開発全体のコストや納期に対してテストがボトルネックになってしまう可能性や，テストの十分性が確保できない可能性が出てくる．このため，「3つの効果」を考慮したテストのポリシーや戦略からテストの目的を明らかにした上で，目的に対し十分性を確保するためのテスト方法やスケジュールを立案することが重要となる．

近年の考え方では，ポリシーや戦略，テストの目的をまとめたものを「マスターテスト計画書」としてプロジェクトの初期に作成して，各テストレベル（e.g. 統合テスト，システムテスト）での具体的な作業内容やスケジュールを記した計画書を「詳細テスト計画書」として別途作成する方法が推奨されている．

コントロールは，管理作業と同義であり，計画との乖離を監視して是正するプロセスである．ライフサイクルとしてテストを捉えた場合，複数のプロセスを適時に回すことで納期通りとなるようスケジューリングすることが多い．このためテストにおけるコントロールは，スケジュールの観点からも，重要なプロセスなのである．

2：分析と設計

分析と設計は，抽象度の高いテストの目的を具体的なテストケース（入力条件，テストデータ要件，期待結果の組合せを指す）へ変換するためのプロセスである．

テストの分析プロセスは，テスト設計のベースとするドキュメントである，テストベース（e.g. 要件定義書，アーキテクチャ設計書）をレビューし，仕様や構造を分析した上でテスト条件（テスト要件とも呼ぶ）や必要となるテストデータを洗い出していく．

テスト設計プロセスでは，テスト条件からテストケースやテスト実施環境の設計を行う．

3：作成（implementation）と実行

テストの作成（implementation）プロセスでは，テストケースに具体的なデータや手順を当てはめてテストケースを完成させ，テストケースの優先順位付けや，効率的にテストケースを実行していくためのテストケースをまとめなおしたテストスイート作成を行うことが該当する．またテストデータ準備，自動テスト用環境開発もテスト作成プロセスで行う．

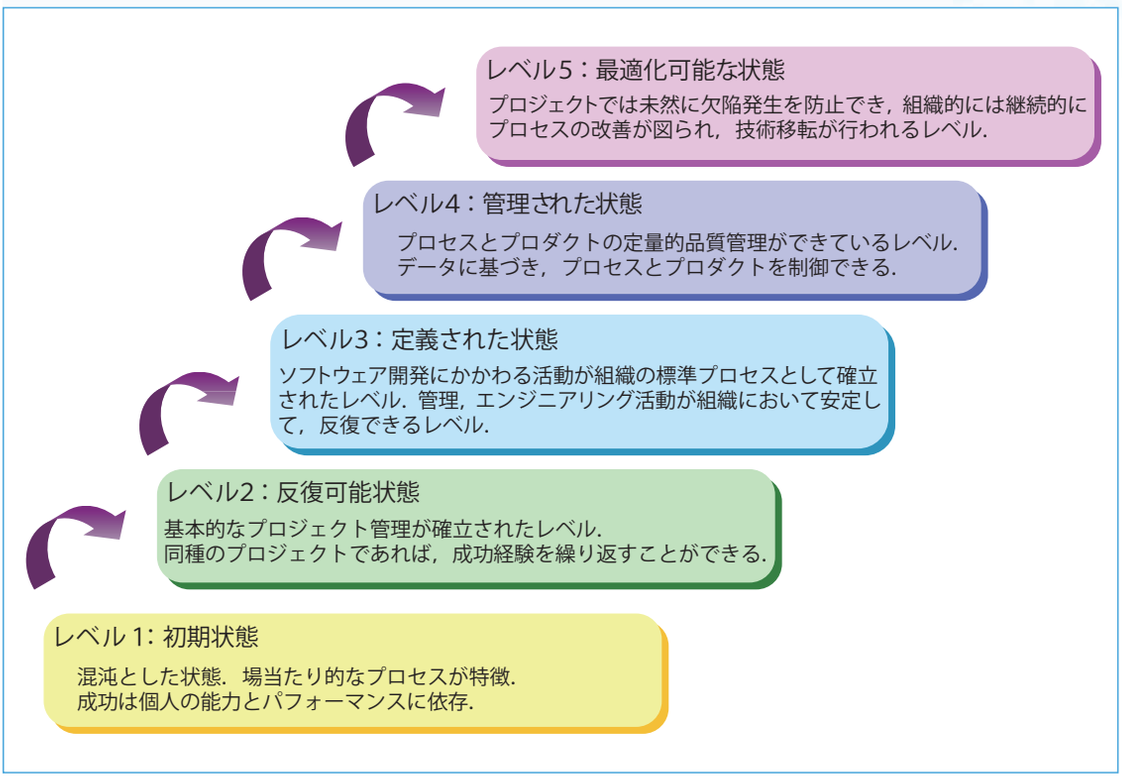

図 -4
CMM 5 段階レベルの定義

テスト実行プロセスでは，テスト環境のセットアップを行い，テストケースを実行し，実行結果のログとテストケースで定めた期待結果との比較を行う．両者が一致しない場合，不具合として報告する．

不具合の報告を受けた開発者は，不具合の原因である開発ドキュメント，コード，テストデータやテストドキュメントの欠陥，テスト実行手法の誤りなどを解明するための調査・分析を行い，必要に応じてプログラムを修正する（この原因解明からプログラム修正のプロセスはデバッグと呼ばれている）．

プログラムを修正し，修正確認のための再テストを行うことで，不具合が発生しなくなることを確認する．また，変更していないプログラム部分に欠陥がないことや，欠陥修正により陰に隠れていた欠陥が現れないことを既存のテストケースを用いて確認する．これを回帰テストと呼ぶが，プログラムの修正では不具合の修正確認と回帰テストの両方を行うことで，はじめて修正の妥当性を確保できる．

4：終了条件の検証とレポート

終了条件の検証プロセスでは，テスト結果記録をテスト計画で定義した終了基準と比較し，テストを終了してよいか，もしくは追加テストが必要か，あるいは定義した終了基準を変更するべきかを判断する．終了基準としては，テストの合格率や不具合の発生の収束状況，不具合修正完了状況などといった複数のメトリクスを用いる．テストの終了判断は，これらメトリクスをもとに開発プロジェクトメンバや経営層の定性的な意見を含め総合的に検討して行うのが一般的である．

この判断結果をもとにテストのレポートを作成し，ステークホルダに公式にテストの終了を宣言する．

5：終了作業

テストの終了作業プロセスは，テストケースやテストデータなど開発時にテストで使用したものを再利用可能な状態に整理するプロセスである．再利用の形態としては，テストを行った同一プロダクトの保守フェーズでの再利用と別プロダクト（機種展開を含め）で流用する形態をとることがほとんどである．

［テストプロセス改善］

ここまで典型的なテストプロセスについて説明してきた．プロセスは開発プロセスもそうだが，いったん作ってしまえば終わりというものではなく，絶えず改善しなくては，日々変化する社会的状況や開発環境，組織的な変化などのさまざまな要因に適応することは難しい．

このためテストプロセス改善においても，改善サイクルを回していくという一般的な改善のアプローチをとることには変わりない．

そこでまず，テストプロセス改善手法が提案された背景について簡単に触れておきたい．テストプロセス改善手法はソフトウェアプロセス改善モデルを補完する形で提案されてきたという経緯を持つ．このため，ソフトウェア開発組織の成熟度を 5 段階で表現する SW-CMM：Software Capability Maturity Model（**図 -4**）や，ヨーロッパを中心に適応されているソフトウェア

2 テストプロセスとテストプロセス改善

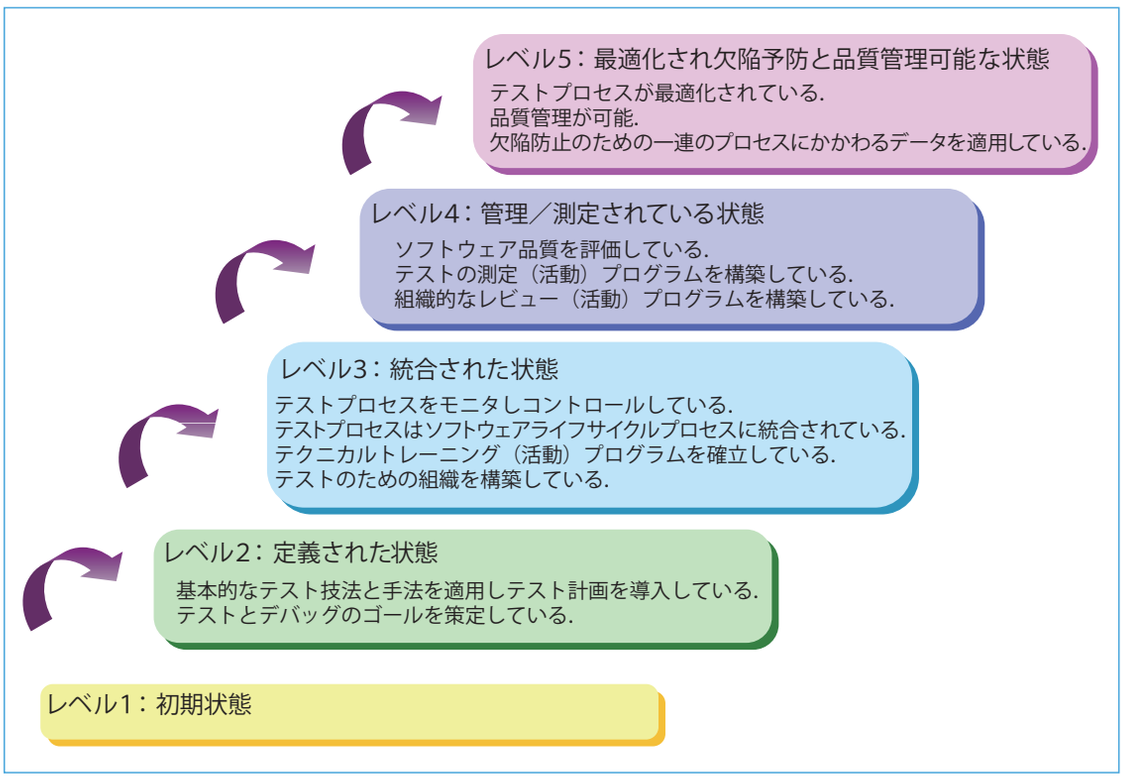

図 -5
TMM 5 段階レベルの定義 [8)]

プロセス改善手法である SPICE：Software Process Improvement and Capability dEtermination といったものや，テストプロセス改善手法どうしで参照しあっているという関係にある．

テストプロセス改善手法としては，SW-TMM[8)]：Software Testing Maturity Model（単に TMM と表記することも多いため以降は TMM と記す）と TPI[3)]：Test Process Improvement の 2 つが代表的である．

■ TMM

TMM は，CMM を導入する上でテストプロセスの改善を補完するためのモデルとして，イリノイ大学の Ilene Burnstein 博士を中心に提案されたモデルである．TMM が提案された理由として，CMM そのものはソフトウェアプロセス全般を取り扱っているため，テストプロセスの具体的な記述に乏しいことから，実際にテストプロセス改善を行うためのノウハウに対するニーズが CMM を導入する組織からあったことが大きい．このため，TMM を用いたテストプロセス改善は，CMM と連動させることが想定されており，5 段階表現も CMM に合わせたものとなっている．（**図 -5** 参照：筆者 大西による翻訳）つまり，CMM がレベル 2 ならば TMM もレベル 2 という対応関係にある．しかし，レベル定義の名称は，組織の全体的な能力を捉えた CMM とテストプロセスに特化した TMM では異なっている．

この各段階（レベル）は次の 3 つの要素で構成されている．

1）成熟度ゴールのセット：図 -5 中の各段階に箇条書きされている事項が成熟度ゴールとなる．それぞれの成熟度ゴールを満たしているかどうかは，サブゴールの達成度合いで判断する．ただしレベル 1 についてはプロセスとしては初期，つまり確たるプロセスが存在しない状態であるため特にゴールは設けられていない．

2）成熟度ゴールをサポートするサブゴール：それぞれの成熟度ゴールに複数のサブゴールが存在する．たとえばレベル 2 のサブゴールには次の 4 つがある．

①デバッグを行えるグループが組織として構成され予算とサポートが得られる

②テストとデバッグのポリシーやゴールがプロジェクトやテスト計画に反映されている

③不具合を分類するスキームが確立されていて，リポジトリによる基本的な管理体制が構築されている

④テストとデバッグのためのいくつかのメトリクスがあり計測／収集されている

3）アクティビティとタスクと責務：段階的にレベルを上げながらテストプロセスを改善していくために，各レベルで実施すべきアクションを定義したものである．実際にアクションを実行に移すためには，アクションを行う組織のコミットメントが必要となるため，その責務範囲も併せて定義する．

テストプロセスの成熟度を判定する手法として TMM ではアセスメントモデルである TMM-AM（Assessment Model）を用意している．TMM のアセスメントではア

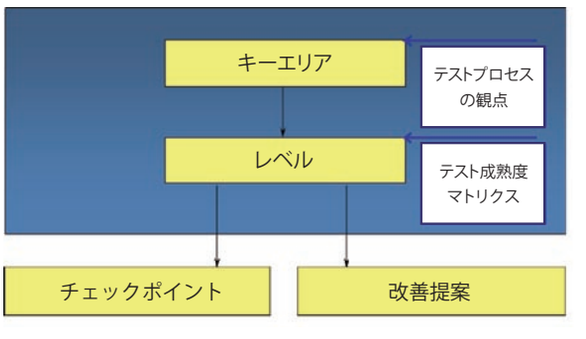

図 -6　TPI によるテストプロセス判定の仕組み [3)]

ンケートやインタビュー，ドキュメントレビューなどを通してテストプロセスの成熟度を判断するとともに，改善するためのアクションプランを策定するという流れで改善を進める．

TMM は日本語による情報が乏しいこともあり，国内ではあまり普及していないのが現状である．しかし CMM と TMM は親和性が高いため，CMM の導入を検討しているか，実際に展開している組織では TMM が大いに参考となろう．なお，現在 TMM は CMM が CMMI（Capability Maturity Model Integration）としてソフトウェア以外の領域へも適用できるように進化したのと合わせ，TMMI として進化し，現在はアメリカを中心に適用されている．

■ TPI

TPI は TMM のように直接特定のソフトウェアプロセス改善モデルと連携するというよりは，テストプロセス改善に特化したモデルと言える．

TPI はオランダの IQUIP 社（現 SOGETI 社）の Tim Koomen と Martin Pol によって提案された手法であり，同じく IQUIP 社が構築したテストプロセスを構造化したモデルである TMap：Test Management approach for structured testing という独自のテストのための方法論をベースにしている．

では TPI を用いたテストプロセス改善アプローチについて説明しよう．TPI も TMM 同様，テストプロセスの状況を評価し，段階的に改善していく手法である．

ただし，評価手法は独自のものを持つ．

TPI ではテストプロセス全体を 20 のキーエリアという形にブレークダウンして表現している．つまりテストプロセスの観点で整理したものがキーエリアである．

各キーエリアにおける状態をレベルとして A から D の 4 段階（一部キーエリアでは 2 ～ 3 段階）で評価する（**図 -6，表 -1** 参照）．

この各キーエリアにおけるレベル判定を行うためのツールとして，表 -1 に示すようなテスト成熟度マトリクスを用いる．レベル自体の判定には，さらに詳細化した 0 ～ 13 までの 14 段階で表現したスケールで行う．

スケールの判定はチェックポイントという，TMM のサブゴールに該当するものにより行う．タスクやアクティビティを組織として持っているかどうか，メトリクスを収集しているかどうかといった，Yes/No で判断できる事項をスケールごとに用意しておき，確認するというアプローチをとる．このため，たとえばあるキーエリアのスケールが 7 だとしたら，6 以下のチェックポイントで定められた事項もすべてできているという，いわば積み上げ的な性質を持っている．このためスケールを上げるための活動としては，現在より一段階上のスケールのチェックポイントを満たすための改善提案を行い，計画立てて改善を進めるという流れになる．

TPI のユニークな点は，各キーエリアのレベルとスケールの対応を一律に扱うのではなくキーエリアごとに分けていることである．

キーエリア	スケール													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		制御されたレベル					効率的なレベル					最適化レベル		
1 テスト戦略		A					B				C		D	
2 ライフサイクルモデル		A			B									
3 関与の時点			A				B				C		D	
4 見積もりと計画				A							B			
5 テスト仕様化技法		A		B										
6 静的テスト技法					A		B							
7 尺度						A			B			C		D
8 テストツール					A			B			C			
9 テスト環境				A				B						C
10 オフィス環境				A										
11 コミットメントと意欲		A				B						C		
…														

表 -1
TPI のテスト成熟度マトリクス [3)]

2 テストプロセスとテストプロセス改善

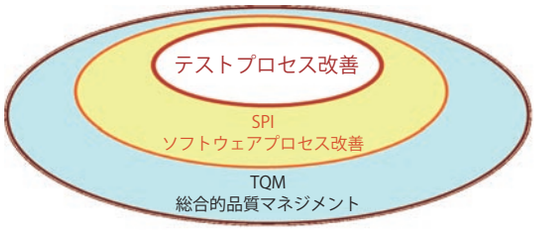

図 -7　テストプロセス改善の位置づけ

たとえば「テスト戦略」の A におけるスケールは 1 であるが，「テストツール」の A はスケール 4 といったようにである．このようなキーエリアごとの違いは，定性的なテストプロセスの成熟度を，定量的なスケールへ変換する際に，性質の異なるキーエリアの重みづけを一律に合わせることが合理的ではないからである．

このスケールはキーエリアの成熟状態を知るためにも用い，次のような基準を TPI では示している．

- 制御されたレベル：スケール 1 ～ 5
- 効率的なレベル：スケール 6 ～ 10
- 最適化したレベル：スケール 11 ～ 13

もう 1 つ TPI のユニークな点は，キーエリアのレベルを各キーエリア単独ではなく，キーエリア間のレベルごとの依存関係を含めて判定していることにある．

たとえば表 -1 中では，キーエリア 5「テスト仕様化技法」のレベル A の依存性があるキーエリアとして，キーエリア 11「コミットメントと意欲」のレベル A が定められている．これはキーエリアの 5 と 11 をレベル A と判定するためには，各々のキーエリアのスケールのチェックポイントを単独で満たすだけではなく，両方ともに満たさなければならないという意味を持つ．

ここまで説明したことから分かるように，TPI ではチェックポイントにより明示的なレベル判定ができるため，改善の進捗度合いを把握しやすく，レベルを上げるために次に何をすればよいのかが分かりやすい．

TPI の国内での適用例は徐々に出てきている状況である．これは書籍や SOGETI 社の Web ページなどから日本語で情報を得ることができることが理由として挙げられる．現在の TPI の動向では，SOGETI 社と欧州の自動車および部品メーカと共同で TPI Automotive を策定するといった取り組みが挙げられよう．

［テストプロセス改善が目指すところ］

ソフトウェア品質の改善や開発自体の最適化や効率化を図る上で，テストプロセスの改善を行うことは大いに効果が期待できよう．これは冒頭に述べたようにソフトウェア開発のライフサイクル全体におけるテスト工程の占める割合が他の工程と比べて多いことからも言える．

TPI のようなアプローチをとれば，まずテストプロセスの改善から行うことができるし，TMM であればソフトウェアプロセス改善と連動した改善活動が可能となる．組織の内部リソースだけで改善を実施するならば TPI から始めることができるが，TMM は CMM 同様，外部リソースによる評価を前提にしているという違いがある．このため現場主導のボトムアップ的なアプローチなら TPI が向いており，経営・管理層が主導するなら TMM のトップダウン的なアプローチが適しているとも言えよう．

ただし，テストプロセスのみを単独で改善するのではなく，ソフトウェアプロセス改善活動の一環として取り組むべきであることを言及しておきたい（**図 -7**）．また，ソフトウェアプロセス改善自体も，組織全体の改善，いわゆる統合的品質マネジメントのような組織全体の活動と切り離すことなく推進していくことで，真に競争力を持ち，品質・コスト・納期それぞれのバランスがとれるソフトウェア開発組織へと成長できるのである．

参考文献

1) Myers, G. J. : The Art of Software Testing (1979) : ソフトウェア・テストの技法，近代科学社 (1980)．
2) Kaner, C., Falk, J. and Nguyen, H. Q. : Testing Computer Software (1993) : 基本から学ぶソフトウェアテスト，日経 BP（2001）．
3) Koomen, T. and Pol, M. : Test Process Improvement (1999) : テストプロセス改善，共立出版 (2002)．
4) Craig, R. D. and Jaskiel, S. P. : Systematic Software Testing (2002) : 体系的ソフトウェアテスト入門，日経 BP（2004）．※本文中は原著を直接参照した
5) A Project of the IEEE Computer Society Professional Practices Committee, Guide to the Software Engineering Body of Knowledge 2004 Version (2004) : ソフトウェアエンジニアリング基礎知識体系 —SWEBOK2004, オーム社 (2005)．
6) Beizer, B. : Software Testing Techniques (1983) : ソフトウェアテスト技法，日経 BP（1994）．
7) International Software Testing Qualifications Board (2005) Certified Tester Foundation Level Syllabus Version 2005 : テスト技術者資格制度 Foundation Level シラバス　日本語版 Ver1.0.1（Version 2005）．http://www.jstqb.jp/syllabus.html
8) Burnstein, I.：Practical Software Testing, Springer, NewYork (2003)．

（平成 19 年 12 月 27 日受付）

大西建児（正会員）
ohnishi@mamezou.com
国内電機メーカ，外資系通信機器ベンダーで培ったテストや品質保証などの経験を活かし，主にソフトウェアのテストや品質改善の分野でコンサルティング活動に従事．IEEE-CS，ACM，日本品質管理学会各会員．JSTQB 技術委員，NPO 法人ソフトウェアテスト技術振興協会 副理事長．

湯本　剛
yumoto@mamezou.com
1991 年製造メーカに就職し，原価管理，製品管理システム構築プロジェクトに参画．その後，ソフトハウスにてパッケージソフト，プリンタドライバ，C/S システム，Web システムなどソフトウェアテスト業務に携わる．現在ソフトウェアテストのコンサルタントとして活動中．JSTQB 技術委員，NPO 法人ソフトウェアテスト技術振興協会 理事．