プログラムの品質と信頼性

株式会社ユートラム
永田　信行

はじめに

本章では、ソフトウェアの品質の問題を、プログラムミングの技法から、高品質のソフトウェアを安定供給する品質マネージメントシステムについてまで広げ考えてみる。

１．ソフトウェアの品質を考える

高品質なソフトウェアを作り出すコツは、プログラミング技法を学ぶことだけでない。品質が作り込まれるプロセスを知り、品質の特性を知り、品質マネージメントシステムを学ぶことが重要である。ここでは、ソフトウェアの品質について考えてみる。

１．１ 品質には特性がある

ソフトウェアの信頼性とは、ソフトウェアが典型的な使用状況の中でどのように稼動するかにかかわるものと定義される。分かりやすく言うと信頼性の高いソフトウェアとは、“バグ”がない、あるいは極端に少なくて、安定して稼動するということである。ソフトウェアの品質と信頼性は強い繋がりを持つが、それでは品質の高いソフトウェアとは“バグ”がないということだけであろうか。ソフトウェアの使いやすさとかメンテナンスのしやすさもその品質の一部を受け持つ。品質にはいくつかの特性があるということが古くから言われ、これまで多くの品質指標が提唱されてきた。今日ではソフトウェア品質を測る標準指標が示されている。

１．２ 品質は作り込むもの

信頼性が重視されるプログラムでは、“バグ”の存在は許されない。しかしながら、現在も過去においても、多くのプログラム技術者はプログラムの“バグ”に悩まされ、“デバグ”という作業に多くの時間を費やしてきた。この過程では多くの技術者が“バグ”を排除するにはどうすればよいかを考えてきたが、その結果、今日の認識ではそれは“バグ”がどのようにして作りだされるのかを正しく認識することであるということがわかっている。

では、“バグ”はどのようにして作りだされるのだろうか。“バグ”はプログラムの中にだけあるものではないことを体験した読者は少なくないであろう。プログラム仕様書や、設計書の中に“バグ”が潜み、これらは発見されずに、テストをする段階になって、一気に表面化する。では、“バグ”はどうして発見されずに次の工程へ受け継がれるのか。原因は、その「プロセス」にある。そこでは、設計の誤りが見つかりにくいプロセスになっていたり、レビュがうまく機能しないプロセスになっていたりしているはずである。

プロセスとは、作業手順だけでなく、設計者の「思考」や「行為」も含まれる。これらプロセス自

身に“バグ”があったり、プロセス間の連携に問題があったり、プロセスの品質をチェックするためのプロセスが設定されていないとき、“バグ”が入り込む。“プロセスの品質”が“バグ”の原因であるといえる。「品質は作り込むもの」といわれる所以である。

１．３ 品質保証の対象は３つある

品質には、次の３つの保証がある。

(1) 成果物そのものを保証する
(2) 品質プロセスを保証する
(3) 品質システムを保証する

(1)成果物そのものを保証する　とは、たとえばテストによって要求仕様を満足することを確認することなどである。医薬品開発の統計解析ではダブルプログラミングによって結果を保証することが行なわれるが、これも成果物そのものを保証する行為の一つである。しかしながら、成果物そのものを保証することに頼った品質管理は、“バグ”の除去は出来ても本来の原因の除去に寄与することはない。また、作業量が膨大になるだけで、次の機会にも同じような問題を再発することになる。プログラムの“バグ”をテストで見つけ出そうとすると膨大な工数が必要となることを体験した読者も多いだろう。

(2)品質プロセスを保証する　とは、ソフトウェア開発の各プロセスにおいて“バグ”が産み出されない仕組みを構築しそれを保証することである。成果物そのものを保証する行為もプロセスの一部と考えられるのでこれを含むと考えて良い。実はこれまでのソフトウェア開発では、この部分に重点をおいた品質管理が実施されてきた。

(3)品質システムの保証する　とは、品質プロセスのマネジメント・システムの質を保証することである。品質は作りこむものであるがために、時には“バグ”をも作りこむこともある。さらに外部環境の変化により折角構築した品質システムも劣化する。そのため、品質システムをいかに計画的に改善できるかが重要になる。品質プロセスの実施状況を常に監視し、それらの定期的な見直し PDCA(Plan Do See Act)サイクルを実施することが求められる。

ISO9001に見られるように、最近の品質管理では、「品質保証」の枠組みから一歩踏み出して、「品質マネジメント・システム」へシフトしてきている。

３つ保証の関係は、図１のようであると考える。

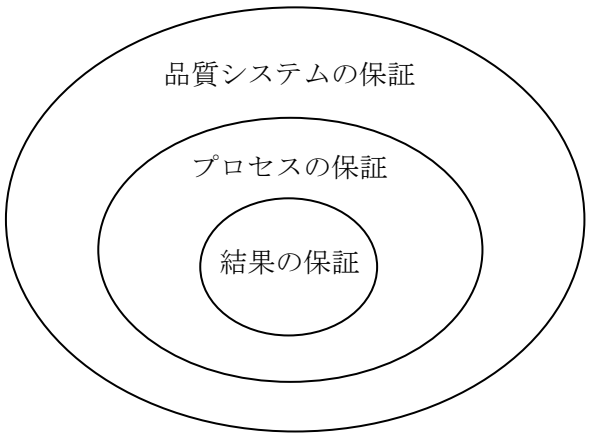

図１ 3つの保証の関係

２．ソフトウェアの品質特性

２．１ 品質特性と副特性

ソフトウェアの品質を測る指標が考え出されたのは、1970 年代初頭だが、その後、多くのの品質モデルが提唱され、世界共通の品質モデルとして 1991 年 ISO/IEC9126 が制定された。日本ではこの世界規格を 1994 年に JIS　X0129 として発行している。

ISO/IEC9126 では 6 個の品質特性が定義されており、またそれぞれに副特性が定義され合計 21 個の副特性がある。

（1）品質特性

ISO/IEC9126 JIS X0129 では以下の 6 個の品質特性を定義している。

1	機能性 (Functionality)	機能の集合の存在及びそれらの明示された性質の存在をもたらす属性の集合。機能は、明示的または暗示的な必要性を満たすものとする
2	信頼性 (Reliability)	明示された条件の下で、明示された期間、ソフトウェアの達成のレベルを維持するソフトウェアの能力をもたらす属性の集合
3	使用性 (Usability)	明示的または暗示的な利用者の集合が、使用するために必要とする労力、及び個々の使用結果による評価に影響する属性の集合
4	効率性 (Efficiency)	明示的な条件の下で、ソフトウェアの達成のレベルと使用する資源の量との間の関係に影響する属性の集合
5	保守性 (Maintainability)	仕様化された改訂を行うために必要な労力に影響する属性の集合
6	移植性 (Portability)	ソフトウェアをある環境から他の環境へ移す際のそのソフトウェアの能力をもたらす属性の集合

表1 品質特性

（2）品質副特性

品質特性にはそれぞれ副特性が定義されており合計 21 個の副特性がある。

機能性	1	合目的性	明示された仕事に対する機能が適切であること
	2	正確性	正しい結果・効果をもたらすこと
	3	相互運用性	明示されたシステムと相互運用ができること
	4	標準適合性	ソフトウェアの応用分野である規格、法律、規則等を遵守していること
	5	セキュリティ	プログラムとデータに関して不当なアクセスを排除できる能力
信頼性	6	成熟性	ソフトウェアに潜在する障害による故障頻度の影響度合
	7	障害許容性	ソフトウェアの障害部分を実行したとき、仕様化された達成のレベルを維持する能力
	8	回復性	故障時に回復させる能力、およびそれに要する時間・労力に関すること
使用性	9	理解性	ソフトウェアの論理的概念等についての利用者側の理解のしやすさ
	10	習得性	ソフトウェアの適用についての習得のしやすさ
	11	運用性	ソフトウェアの運用と運用管理を行うための、利用者側の労力に関すること
効率性	12	時間効率性	ソフトウェアを実行する際の処理時間・応答時間等に関する能力
	13	資源効率性	ソフトウェアを実行する際に使用する資源量等に関すること
保守性	14	解析性	故障等の原因診断、または改訂すべき部分の識別に必要な労力に関すること
	15	変更性	改訂など環境変更に関する対応能力
	16	安定性	改訂によって予期せぬ影響を与える危険性に関する能力
	17	試験性	改訂したソフトウェアの妥当性の確認に要する労力に関すること
移植性	18	環境適応性	特定の異なる環境にソフトウェアを適用する可能性に関する能力
	19	設置性	特定の環境に設置するために要する労力に関すること
	20	規格適合性	ソフトウェアの移植性に関する規格、規則等を遵守していること
	21	置換性	他のソフトウェアの代わりに置き換えて使用できる可能性に関する能力

表２ 品質副特性

３．ソフトウェア開発ライフサイクル

ソフトウェアの特徴の一つは可視的でないということがある。そのため、前工程の誤りが後工程に持ち込まれることが多い。後工程で発見されたときの手戻りは生産性を著しく阻害するだけでなく、そのことがさらに品質を低下させる要因にもなる。ソフトウェアの品質を考える場合、ソフトウェアがどのような手順を経て作成されるかを知っておくこと、またソフトウェア開発のそれぞれの作業における問題点とその重要性を知っておくことは重要である。

３．１ 共通フレーム

ソフトウェア開発ライフサイクルの標準として認知されているものとして、「共通フレーム 98 SLCP-JCF98」がある。1995 年に国際規格となったソフトウェアライフサイクルプロセス（ISO/IEC 12207）で定義された作業項目に日本のソフトウェア産業の特性を加味したものである。ソフトウェア開発を「共通の言葉」を用いて、作業範囲、作業内容、作業項目などを明確化することを狙って作成されたものである。

共通フレームは、ソフトウェア開発を行う上で必要な作業を、まず大きな枠である「プロセス」で区切り、プロセス内の作業事項を「アクティビティ」で分割、さらに各アクティビティ内の作業内容を「タスク」として定義している。ISO9001 で要求されている内容を定義する際には、共通フレームを参照しながら行なわれることが多い。

（1）プロセス

共通フレームでは、ソフトウェアライフサイクルプロセスを 6 つの主プロセス、８つの支援プロセス、４つの組織に関するプロセスに分類している。図 2 に共通フレームの基本構成を示す。

主プロセスである開発プロセスは、V の形状をしているのが特徴である。支援プロセスには、品質管理の視点でのプロセスを定義している。プロセスはアクティビティの集合である。

（2）アクティビティ

タスクの集合で､プロセスの構成要素となる｡V の形状をもつ開発プロセスのアクティビティには、①プロセス開始の準備、②システム要求分析、③システム方式設計、④業務詳細設計、⑤ソフトウェア要求分析、⑥ソフトウェア方式設計、⑦ソフトウェア詳細設計、⑧ソフトウェアコード作成及びテスト、⑨ソフトウェア結合、⑩ソフトウェア適格性確認テスト、⑪システム結合、⑫システム適格性確認テスト、⑬ソフトウェア導入、⑭ソフトウェア受入れ実施　が定義されている（図 3)。

一方、品質保証プロセスには、①プロセス開始の準備、②製品の保証、③プロセスの保証、④品質システムの保証　のアクティビティが定義されている。

（3）タスク

具体的に遂行するための個々の作業である。アクティビティを構成する要素となる。

図２　共通フレームの基本構造

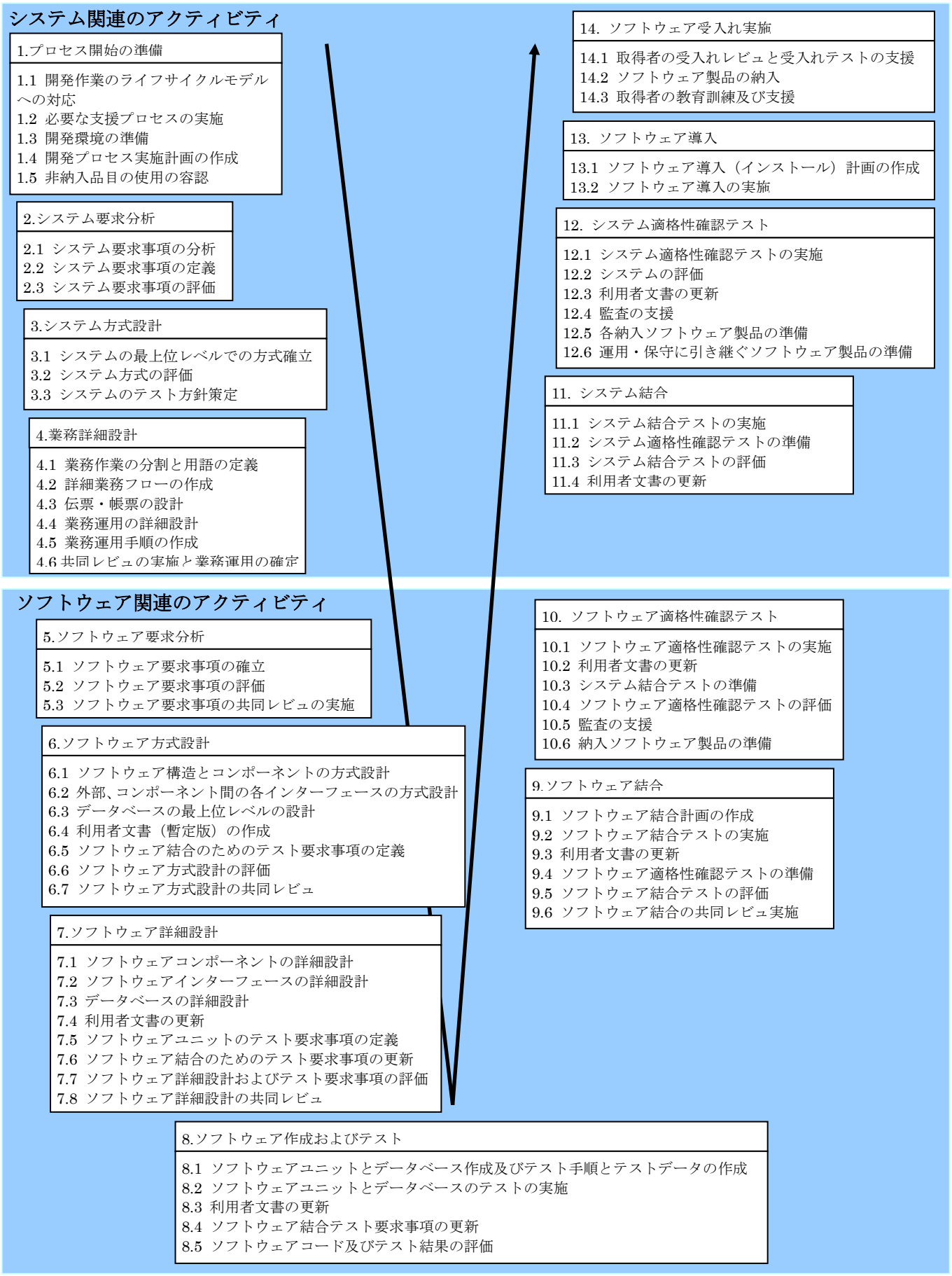

図３ 開発プロセスのアクティビティ・タスク構成図

３．２ 開発標準（プロセス）の改善

共通フレームは普遍性の高い作業項目や作業内容を定義している。医薬品開発の統計解析業務では似つかないアクティビティやタスクが存在するが、これらはテーラリングして自社で使用する開発標準や作業内容を作成することができる。またすでにある開発標準を共通フレームに照らし合わせて、検討し改良することに利用できる。

４． 品質マネージメントシステム

ソフトウェアの品質の問題は、アプリケーションが大規模で複雑になりだした 1970 年代以降急激に注目を浴びるようになり、その後さまざまなアプローチがとられてきた。今日では、ソフトウェアの品質管理基準である ISO9001（品質マネジメントシステム）や CMM（ソフトウェアプロセス能力成熟度モデル）などの品質管理基準が制定され、多くの企業によって採用されるに至っている。

ソフトウェアの品質を考えるとき、最も重要な事は高品質のソフトウェアを安定して供給し続けることである。このためには品質マネージメントシステムを確立する必要がある。以下に品質マネージメントシステムのいくつかを紹介する。

４．１ ISO9001（品質マネジメントシステム）

ソフトウェアの品質をチェックする体制はここ数年、ISO 9001 で根付いてきた。本規格は、国際標準化機構(ISO)から発行された品質マネジメントシステムの国際規格である。1987 年に初版が発行された後、1994 年に改定が行なわれ、さらに現在は 2000 年版として改定が行なわれた。2000 年版は従来の「品質保証」から「品質マネジメントシステム」へと軸足を移しており、成果物やプロセスの質だけではなくプロセスを管理するマネジメントシステムの質の保証という意味あいが強くなっている。

（１） プロセスモデル

本規格では品質システムに対する要求事項を規定している。本規格を適用する企業は、この要求事項をみたすように品質マネジメントシステムとして文書化し運営する。また、本規格では実施状況を確認し、それらの定期的な見直しを実施することを規定している。そのため、品質マネージメントシステムの確立後、それを PDCA(Plan Do See Act)サイクルに則って運用する必要がある。本規格に規定される品質マネージメント・システムの要求事項を「プロセスモデル」として表したものを図４に示す。

マネージメントシステムの基本構造は PDCA サイクルである。

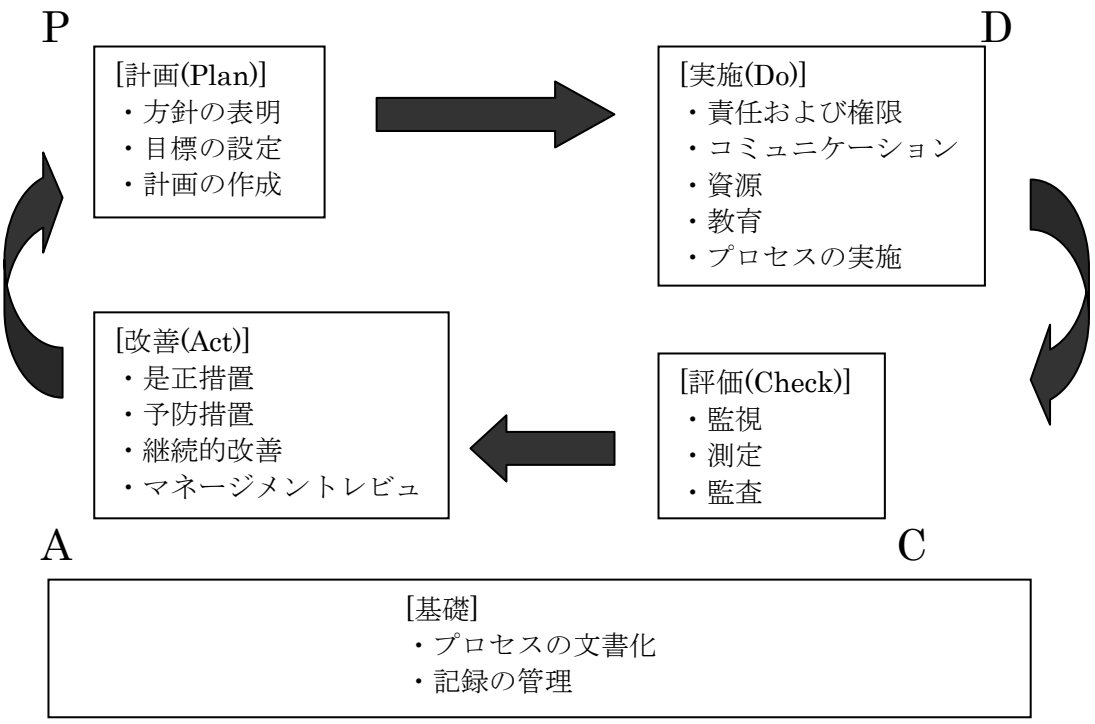

図４ 品質マネージメントシステムのプロセスモデル

（２） ８つの原則

本規格の各主要条項は、以下に示す ISO9000 の品質マネージメント・システム８原則に基づいている。品質マネージメントシステムの８つの原則は、本規格の真髄である。

1	顧客重視（Customer Focus）
2	リーダーシップ（Leaderchip）
3	人々の参画（Involvement of People）
4	プロセスアプローチ（Process Approach）
5	マネージメントへのシステムアプローチ（System Approach to Management）
6	継続的改善（Continual Improvement）
7	意思決定への事実に基づくアプローチ（Factual Approach to Decision Making）
8	供給者との互恵関係（Mutualy Benefical Supplier Relationship）

表３ 品質マネージメントシステムの８つの原則

４．２ CMM（ソフトウェア能力成熟度モデル）

日本では、品質管理の国際規格である ISO9001 が普及しているが、ソフトウェア開発の世界では、「CMM（Capability Maturity Model for Software）ソフトウェア能力成熟度モデル」が急速に注目を浴びている。政府・自治体がソフトウェアを発注するに当たって、発注先を決める要因の１つとして CMM を取り入れようとしようしている動きがあることも拍車を掛けている。

プロセスを重視しないで出来上がったソフトウェアの質は意図的な改善がなされない。このことに

注目してソフトウェアの改善活動を繰り返してできたのが CMM である。CMM はそのプロセス改善のためのテーマが体系化されているため、ソフトウェア開発の品質管理においてはもっとも有効な手法と言われている。

CMM は米カーネギーメロン大学ソフトウェア工学研究所（SEI）が開発したソフトウェア開発の能力成熟度を測定する品質管理基準である。開発プロセスを改善していないレベル１から、管理が最高位のレベル 5 まで 5 段階に分かれている（表４）。現在ではソフトウェア開発能力を測る世界的な基準として世界各国で採用されている。

ISO9001 と異なるのは ISO はソフトウェアだけでなくあらゆるジャンルを審査対象にするのに対して、CMM はソフトウェアのみを対象としていることである。CMM は「ISO9001 に比べソフトウェア開発に適しており、組織そのもののレベル向上につながる」といわれている。

ISO では、ISO15504 規格を進めており、これが、CMM に近い仕様となる予定である。

レベル５ （最適化管理）	プロセスが最適化された段階。定量的なデータによってプロセスを柔軟に改善できる
レベル４ （定量的管理）	プロセスが管理された段階。プロジェクトを定量的に管理でき、成果物の定量的な品質目標を設定可能
レベル３ （定性的管理）	プロセスが定義された段階。プロセスが文書化、標準化されており組織的なトレーニングが行なわれている
レベル２ （経験的管理）	開発のプロセスを反復することができる段階。プロジェクトを管理できる手法が確立されており、過去の経験に基づいた計画と管理が可能
レベル１ （混沌的管理）	初期段階。何もしていない状態でプロセスは場当たり的、技術者個人の能力に依存大

表４　CMMのソフトウェア開発五段階評価

４．３ PMBOK（プロジェクトマネージメント標準知識体系）

プロジェクトマネージメントの巧拙は「品質」としてあらわれる。スケジュールやコストの圧力は品質に大きな影響を与える。プロジェクトマネージメントの立場から、品質の問題を考えるアプローチも必要である。

「PMBOK（Project Management Body of Knowledge）プロジェクトマネージメント標準知識体系」はプロジェクトマネージメントの基礎プロセスを明確にして、プロジェクトマネージメントを成功に導くための知識体系を整備する目的で作成された。世界最大といわれているプロジェクト・マネジメント協会である Project Management Institute（PMI)は、1996 年 3 月に、A Guide to the Project Management Body of Knowledge（通称 PMBOK または PMBOK Guide）の改訂版を発行し、24,000 人の PMI 会員に配布すると共に、会員以外の実践家で、プロジェクト・マネジメントの知識体系の整理に関心を持つ人々に頒布を行っている。PMBOK は ISO10006（プロジェクトマネージメントにおける品質ガイド）の下敷きとなっており実質の国際基準として認知されている。

PMBOK の構造は、①統合マネージメント、②スコープマネージメント、③タイムマネージメント、④コストマネージメント、⑤品質マネージメント、⑥組織マネージメント、⑦コミュニケーションマ

ネージメント、⑧リスクマネージメント、⑨調達マネージメント として体系化されている。このなかの、⑤品質マネージメントでは 3 つプロセス（品質計画、品質保証、品質管理）を紹介しているが、品質マネージメントは、ISO9001 の規格やガイドラインと整合できるように書かれている。

品質管理とプロジェクト管理は相互補完の関係にあることを認識しておきたい。

４．４ IEEE 品質標準

IEEE（Institute of Electrical and Electronic Engineers）は世界で最大のNPO（非営利団体）の一つである。IEEEの標準の多くは、ANSI（American National Standards Institute）と共同で出版されており、「ANSI/IEEE Std nnnn」として呼ばれる。FDAの21 CFR Part11（電子記録・電子署名に関する規制条例）もIEEEを参考にした部分が多いと伝えられている。

以下にIEEE標準の一部を示す。詳細は、http://www.computer.org　から参照されたい。

- ANSI/IEEE Std 0983-1986（ソフトウェア品質保証基準）
- ANSI/IEEE Std 1012-1987（ソフトウェア確認および検証基準）
- ANSI/IEEE Std 1016-1987（ソフトウェア設計の推奨プラクティス）
- ANSI/IEEE Std 1063-1987（ソフトウェア文書化基準）
- ANSI/IEEE Std 1058-1987（ソフトウェアプロジェクト管理基準）
- ANSI/IEEE Std 1028-1988（ソフトウェアレビュおよび監査基準）
- ANSI/IEEE Std 1074-1991（ソフトウェアライフサイクルプロセス基準）

５． 品質管理に盛り込まれる技法

これまではソフトウェアの品質について考え、品質を作りこむ品質管理システムについて考えてきた。ここではソフトウェアの品質を向上させるための代表的な技法を紹介し、それぞれが品質向上に果たす意味や役割を考える。

（１）教育

ソフトウェアの“バグ”やその他の問題は技術者のスキル不足に起因することも多い。品質の善し悪しは、技術者のもつ技量にも大きく影響を受ける。ソフトウェアは基本的に人間が作る。技術者が変われば、結果が変化するということがあるが、それはその企業で行われているプロセスは変わらなくても、技術者のなかにあるプロセスが変化するからである。「この程度でいいだろう」というプロセスと、「もう少し考えて見よう」というプロセスの違いが結果の差になってくる。これは技術者の資質に依存する部分も多いが、定期的かつ継続性した教育によって改善することができる。

教育カリキュラムには、設計、プログラミング、テストに関するものだけでなく、品質管理、開発プロセス、標準化、文書化なども加えたい。

ソフトウェアの開発は教育を受けたものが携わっている事が必要である。

（2）標準化

名前の付け方が統一されていない、変数名が１，２文字で何の目的で使っているか想像できない、プログラムがやたら長い、無駄なコーディングが多い、コメントがない、こんなプログラムを見たり書いたりした経験はないだろうか。きれいなプログラム（理解しやすい、論理的であるなど）には“バグ”が少ないという定説がある。わかりにくいプログラムは、エラーを多発しデバッグをやりにくくし、またテストケースの設計をやりにくくする。

企業で作成されるプログラムは、本来は企業の共有財産になるべきものである。しかし、多分に個人の財産になっているきらいがある。プログラムの作成がすべてプログラム技術者個人にゆだねられ、結果として、属人性の強いプログラムになってしまうからである。プログラムの論理の作成、コーディングは各人各様の方法で行われ、作成者以外がプログラムを理解するのは大変困難であるのが普通である。従って、作成者以外の人間がそのプログラムを保守することが困難になり、作成者がいつまでたってもそのプログラムと縁がきれなくなる。このような現象が生じるのは、プログラム作成の方法が標準化されていないことが原因となっている場合が多い。標準化されたものが整理されていれば、新人はそれを学ぶことによって標準化された技術を身に付けることもできる。

標準化の対象となるものには、プログラムだけでなく、文書、作業手順、設計書、プログラム仕様書などがある。おなじプログラム仕様書を作成しても、作成者によって異なった形式のものができあがってしまったり、おなじ事柄を表現するのに異なった用語が使用されていて、思わぬ誤解の原因になることはよく経験する。

標準化の推進方法としては、①テンプレートを準備する、②レビュにより相互チェックする、③コンピューター（チェッカー）により標準をチェックする。などが考えられる。

標準化はプログラミング技術の継承にも貢献する。

（3）レビュ

設計上の誤りを早期に発見することを目的として、各設計の終了時点で作成者と複数の関係者が設計書を会議形式で検証する方法である。設計の仕様を曖昧なままにしていたり、“バグ”を次の工程に持ち越すと、次の工程での対応に掛かるコストは、より大きくなる。レビュは、仕様やプログラムの“バグ”を発見し改善する効果が期待できる。

レビュは各工程で作成される設計書などの文書やプログラムのソースコードなどを対象とする。レビュ結果は記録としてまとめておくことも重要である。

（4）ウォークスルー

品質管理や“バグ”の除去をレビュのみに頼ってもうまくいかない。工程完了時のレビュで“バグ”が多数発見されたとしても、手遅れになる場合も多い。そこで早期の“バグ”発見と除去を目的とした組織的な方法としてウォークスルーが考え出された。

ウォークスルーは、設計者やプログラム技術者がそれぞれの成果物を他のメンバーに詳細に説明しながら机上検証を行なう非公式なミーティングである。開発の各段階で複数の関係者が集まって、技術的問題に焦点をあてながら、異なった角度から共同で机上検証を行い、“バグ”や誤りなどの問題点を検出する。

ウォークスルーは、公式のレビュでの不充分な点を補完することを目的としている。

（5）コードインスペクション

デバグの一つでコーディング終了時に，複数の人間でプログラムの検証を行うことである。プログラムのソースを査読し、誤りを検出する。実際にプログラムを動かす事はないため静的テストと呼ばれることもある。インスペクションが有効なのは、自分で書いたプログラムの“バグ”を自分で見つけることは容易ではないというところにある。作成した人と違った考え方や発想の人が参加することで、思い違いや認識不足の箇所に気付く事が出来る。

インスペクションはウォークスルーとよく似ているが、インスペクションは発見されたエラーを文書化するだけでなく、それを管理用のデータとして用いるところが異なる。“バグ”や工数のデータは統計的に処理し品質改善に用いる。

（6）モジュール化

モジュール化とはプログラムの機能ごとに処理単位をまとめるという作業。プログラムの処理単位をまとめると、プログラムの開発やデバッグなどをわかりやすく行う事ができる。

プログラムの複雑度と“バグ”の発生率との間には、強い相関関係がある。要素が増えれば複雑さは急激に増すことは古くから知られている。プログラムの複雑さを最小にし、独立性を最大にすることにより、プログラムの可視性を良くし、また単体テストも容易にすることができる。モジュールの大きさは小さいほど良いと言われているが、その主な理由は

- 理解しやすい
- テストしやすい
- 独立性が高まる

の３点があげられる。

モジュール化で留意すべきは、モジュールの独立性を高めることである。モジュールの独立性を高めるには２つのアプローチがある。１つは、それぞれのモジュール内部での関連性が最大になるようにすることであり、他はモジュール間の関連性が最小になるようにすることである。同じモジュール内の命令は密接に関連性をもち、異なったモジュール間の命令は相互にできるだけ関連性をもたないようにすることである。

モジュール化は古くから生産の原理として実施されてきたが、現在では、経済学や経営学の分野でも注目を帯びているテーマの一つである。

（7）再利用

「作らなければ、“バグ”は入らない」は真理である。一般には誰しも欲しいと思われるものは、かなり用意されているものである。あるいは用意すべきである。自分で新たなロジックを作ると、デバッグが必要になるが、すでに用意されたものを使えばその必要もなくなる。プログラムも短くなり、“バグ”も入りにくい。経験の深いプログラマは、経験から蓄積された私的な再利用可能なプログラムを保有している。これを全社的なレベルに引き上げれば良い。

再利用成功のポイントは、再利用モジュールの品質保証と管理組織の明確化である。再利用はプログラムだけでなく文書についても同様である。テンプレートを準備することにより再利用に近い効果がえられる。

（8）テスト

テストはプログラムが設計者の意図どおりに機能することを確認する重要な行為である。テストを実施する際にはテスト計画書を作成し、それに基づいて実施する。モジュール毎のテストを実施し、さらにモジュールをいくつか組み合わせた機能をテストする結合テストのように段階的にテストを実施すると、効率良く漏れのないテストが実施できる。

テストは品質管理技法の中でも重要な意味を持つが、要求仕様に満たされていないことを発見することは困難であることも十分認識しておく必要がある。

テストケースの設計方式としては、ホワイトボックステスト、ブラックボックステストがある。

・ホワイトボックステスト

プログラムの内部構造が妥当か否かをテストする方式である。内部構造を解析し、プログラムのすべての処理ルーチンについてテストを行うのが基本である。プログラムの内部構造や論理が記述された内部仕様書に基づくテストでありプログラム技術者自身が実施する。

ホワイトボックステストはユニット（モジュール）テストでよく用いられる。

・ブラックボックステスト

プログラムの内部構造とは無関係に、外部から見た機能を検証するプログラムのテスト方法である。入力データと出力データの関係が仕様書とおりの結果であるかどうかを調べるテスト方法ともいえる。入力と出力だけに着目し、様々な入力に対して仕様書通りの出力が得られるかどうかを確認する。その間、システム内部でどういった処理が行われているかは一切問題としない。テストケースは入力と出力の関係に着目して設定する。運用上で想定できるケースについては必ず確認しなければならない。

ブラックボックステストは結合テストなどでよく用いられる。

（9）外注管理

開発プロジェクトにおける要員不足により、プログラミングを外注依存する割合は高くなっている。外注に委託する理由として、要員の確保、技術力の確保、開発リスクの低減、ソフトウェア品質の向上などが挙げられる。プログラムを外部に発注する場合、外注先の企業が常に品質の高いプログラムを開発できる体制、能力を持っているかを判断する必要がある。また、発注側の品質管理に関する要求事項を示し実施を義務つける事も重要である。

発注側では適切な監査と牽制を実施する必要がある。

６．開発支援環境（ワークベンチ）の活用

プログラム技術者はプログラム設計、コーディング、テストの直接作業の他に、文書作成、テストデータの維持管理、プログラムの保管、テスト結果の整備などの間接作業に多くの時間を費やしている。時間を費やしているだけでなく、そのためにミスを犯すこともある。

「このマクロを呼んでいるプログラムはどこだろう」というときなど、直ぐに分かるような仕掛けになっていなかったり、その要求を入れるためにどのマクロを修正すべきかをすぐに判断する材料が用意されていなければ、イージーミスは簡単に起きてしまう。

開発中のプログラムは頻繁に更新されるので、最新状態を維持することに常に気をつけていなければならない。古いプログラムをテスト用に誤用するなどの不注意やミスは致命的な結果を招く。

プログラムが完成したのち、そのプログラムに関する文書化はプログラム技術者にとってはやっかいなものであり、スケジュールに追われているときは、ついつい手を抜いてしまうことになる。

統計解析プログラムなどは、後で再実行したり修正したりする可能性があるので、ドキュメントやデータ、それにプログラムなどは対で管理しておくことも必要となる。

変更を防止するためのプログラムを固定する機能もときには必要になってくる。プログラム仕様をプログラムからリバースエンジニアリグで作成してくれるソフトウェアがあれば便利だ。

プログラムが資格のないものに誤用・盗用されたり、破壊されないようにするためのセキュリティ保護システムも必要である。機密保護を強化するためには、ソフトウェアが具備されていなければならない。

設計者やプログラム技術者の作業負荷を軽減し、イージーミスを防止できる開発支援環境の活用は、ソフトウェア開発の信頼性、経済効果を高めることに貢献する。

おわりに

品質の悪さは,後工程での手戻りの発生となり、結果的にコスト増加の要因となる。この部分では、低品質は低生産性（高コスト）という関係が成り立つ。高品質は，高生産性をもたらすともいえる。しかし一定レベルを越えて高品質を追求すると、今度はそれに伴って急激にコストが増加する。前述の関係は崩れる。だからといってコストを押さえようとすると品質に対するリスクが発生する。

今日ではソフトウェアの品質の良し悪しが、社会的に大きな影響を与えることは多くの人に知られている。しかし一方でビジネス社会の構成要素であるソフトウェアは常にコストを意識されなければならない一面も持つ。ソフトウェアの品質が要求レベル以下であれば良くないのは当然だが、一方で、品質目標を大幅に上回る品質を実現した場合には、それは不必要にコストを掛けたということにもなる。過剰な品質もビジネスでは問題となる。

セキュリティの問題も品質の場合とよく似ているが、この問題を考えるときいつも一体どこまでやれば良いのかという壁に突き当たる。はたして答えはあるのか。それはYES&NOである。現実的にはリスクの程度を認識した上で、うまくバランスさせることになる。

重要なのは、アプリケーションに求められる品質特性を知り、それに応じた品質目標を設定する。そして目標達成への施策を講じる。そして何よりも重要なのは、常に改善のサイクルを回し続ける事である。これこそが最適な時間・コストで目標とする品質のソフトウェアを安定して供給できる方策ではないだろうか。

文　献

1) 共通フレーム 98　―SLCP - JCF98―　通産資料調査会(1998)
2) 成功するソフトウェア開発―CMM によるガイドライン　オーム社(2001)
3) CMM レベル３はこうやって取得する　NIKKEI SYSTEMS PROVIDER(2002.2.15)
4) PMBOK による IT プロジェクトマネージメント実践法　ソフト・リサーチ・センター(2002)
5) ソフトウェア品質のガイドライン　共立出版(1999)
6) モジュール化：新しい産業アーキテクチャーの本質　東洋経済新報社(2002)
7) 効果的プログラム開発技法　近代科学社(1984)
8) ソフトウェアインスペクション　共立出版(1999)
9) C プログラミング診断室　技術評論社(1993)
10) ソフトウェアのテスト技術：品質保証へのアプローチ　企画センター(1984)