HUMAN WELFARE REVIEW Vol.3 (2000) CHOFU GAKUEN JUNIOR COLLEGE

Sakai, Tadamichi ; One Consideration of the Quality Control in Information System Development

2000年10月掲載承認

人間福祉研究
第3号/2000年度

情報システム開発における品質管理の一考察

― テスト工程におけるソフト品質向上手法の改善 ―

坂井忠通（さかい ただみち）

〈要　旨〉

本論文は情報システム開発におけるソフトウェアの品質管理手法の改善を実践的な立場から考察、提案するものである。情報システム開発における上流工程（設計、作成工程）で作り込まれたバグ（プログラムのミス等）を下流のテスト工程で確実に摘出／修正して品質向上を実現するために、広く採用されているＰＢ曲線活用方法の限界とその対策をテスト作業者のマインドに着目して改善する提案とその適用事例について述べる。

※本稿では「バグ」の意味合いを「プログラムのミス以外に一部仕様不備等も含むソフトの不具合」として使用する。

※マインドとは、担当者が作業を行う際の精神的な姿勢や価値観（心構え、考え方、意思、意欲etc.）を指す。

1．はじめに

コンピュータが社会的に重要な地位を占めるようになり、多くのアプリケーションソフトウエア（以下、ソフトと略す）が開発されている現在でも、開発ソフトの品質管理は相変わらず未解決の課題と言える。ソフト開発における品質管理は昔から３Ｋ（経験、勘、根性）あるいはＫＫＤ（経験、勘、度胸）と言われてきた。この原因の多くは「ソフトの不可視性」と「ソフトの属人性」にある。つまり作成されたソフトは目視チェックが難しく、動作させて初めて不具合が判る場合が多い。またその作成は人手に頼るため、開発に

携わる作業者（システムエンジニア〔ＳＥ〕やプログラマ）個々人のスキルとマインドに依存するところが大だからである。

それ故に実際のプロジェクトでは（優秀なリーダが居る場合を除き）問題の認識が甘いため対処が後手後手となり、納期が逼迫してくると最後は残業と徹夜作業で乗り切るやり方がしばしば見られる。結果として費用面や作業者の健康面に大きな負担を掛けるとともに、時間切れで潜在しているバグを残したまま納品してしまうケースも見られた。

このような実態に対処する各種の研究がなされてきたが、多くはプロジェクトが終了した後の統計的、確率論的なアプローチが中心であって、現実のプロジェクトに「いま従事している者」にとって、そのプロジェクトを成功させるために有益な実践的方法論を述べたものは少なかった。またソフト作成やテストの自動化ツール[1)]の開発も進められているがまだ充分な状態にあるとは言えない。

システム開発における品質管理は設計、作成、テストの各工程毎に論じられるが、本稿では最も重要なテスト工程に注目して品質管理を論ずる。テスト工程は上流工程で見落とした問題点や作業ミスを検出する、いわば品質確保の「最後の砦」として極めて重要な位置にある。作成工程では個人作業的な色彩が強く客観的な管理が難しいが、最終段階にあたるテスト工程では出来上がったソフトあるいはシステム全体の品質を客観的にチェックすることが可能となり、有効な措置を取りうるからでもある。

本稿では情報システム開発におけるテスト作業の実態と課題を品質管理の観点から明らかにし、ソフトのバグを摘出／修正する「テスト工程」に焦点を絞って、事例を示しながら品質向上の実践的な提案を行う。

以下の第２章ではシステム開発の全体像とテスト工程の重要性を、第３章ではテスト工程に於ける問題提起を、第４章では実践面での現実的な改善アプローチを、第５章では実践事例を、第６章でまとめを述べる。

２．システム開発の全体像とテスト工程の重要性

はじめにシステム開発の全体像を明らかにしておく。コンピュータの情報システム開発のプロジェクトは全体工程が設計、作成、テストに３分割される。**（図１参照）**これら工程の推移においては、上流工程の不備が下流工程に大きく影響を与える。

1）例えば、PURIFY や Bounds Checker はメモリリーク（使用メモリの解放忘れ）チェックに、QAC はソースコードの静的解析に効果があるが、いずれも限定されたものである。

＜図１．一般的な工程分割と流れ＞

●開始	業務設計		①プログラム設計		②作成	③テスト			●稼働
	基本設計	詳細設計	構造設計	ﾓｼﾞｭｰﾙ 設計	ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ & 単体ﾃｽﾄ ＊注２	結合ﾃｽﾄ	総合ﾃｽﾄ	運用ﾃｽﾄ	

設計工程ではソフトの機能を規定するが、業務やソフトを良く理解していない人が設計すると後工程で不具合が多発することが多い。仕様書の「文学的表現」による曖昧さや「決め」の粗さが問題となる場合も多々ある。

作成工程では設計工程で作成された仕様書に基づいてプログラムを作成する。この工程では作業者つまりプログラマの能力個人差（理解力、プログラミング能力等）が顕著に現れる。個人の経験、センス、性格、メンバーとのコミュニケーション力などによっても生産性や品質が大きく左右される。またプログラミングは細かな命令を大量に書き綴り多人数で分業するので、どうしてもプログラム・バグが作り込まれる宿命にある。

これらの前工程に対して、下流のテスト工程では作成されたソフトが正しく動作するか（正しく作られているか、実際の運用上で問題無いか）が試験、確認される。上流工程で作り込まれた不具合（仕様のミスや漏れ、プログラミングのミスなど）をテスト工程では出来るかぎり摘出し、実際の運用に耐えられるレベルまで品質を高めなければならない。

一般的にソフトの品質管理はテスト工程から本格的に始まる。それ以前の作成工程はほとんど「作業者（プログラマ）個人」の管理に任されており、外部から他人がプログラムを覗くことが難しいからである。作成途中のソフトを第三者が動作確認しようとしても分割されたプログラムは単体での動作が難しく、作成工程での品質管理は作成者本人に任せきりになってしまい、個人のスキルでシステム品質が左右される危険が常に付きまとう。

プログラムが一式揃い、動作確認可能となる下流の「テスト工程」以降で外部から客観的な動作確認と品質管理が可能となる。しかもこの時期はプロジェクトの終盤にあたる納品間近な工程であり、僅かでも問題の認識と対処が遅れると大きな損失につながるので、如何に早く問題を認識して手を打つかが重要になる。

また、テスト工程の中でも、総合テストはテストと言うよりシステム全体が正常動作する事の「最終確認」なので、細部をチェックする単体テストと総合テストの中間に位置する「結合テスト」で一定品質を確保することが重要である。また複数人で分担作成したプログラム相互のインターフェースに係わる問題を摘出できるのも結合テスト工程である。

2）単体テストとはプログラムモジュールの作成と並行して行うモジュール単独の確認テストを言い、結合テストとはこれらを複数個結合してまとまった形で動作確認を行うテストを言う。

もっとも有効なテスト工程に時間とパワーを投入し、すでに作り込まれた潜在的不具合を徹底的に摘出、修正（場合によっては、別人による作り直しが必要）する事によってのみソフト品質は客観的に保証される。

3．テスト工程に関する問題提起

テスト実施における現場の悩みは、どこまでテストを実施すれば完了なのか判らない事である。つまり残存バグが何件有るか判らない。ソフトのバグは叩けば叩くだけ出てくる傾向があり、何時までやっても無くならない。ひたすら納期までテストを続けるのは過大テストで無駄なのか、止めれば過少テストで低品質を招くのか、出来るだけ早めにそれを把握して必要かつ最小限の体制と機材で効率よく納期を守ることがプロジェクト推進上では要求される。

そのためには客観的な指標管理が望まれる。実務一般ではＰＢ曲線が昔から使われている。これは横軸に日数を取り、縦軸にテスト項目の未消化残数（Ｐ：プログラムチェック項目）とバグ摘出の累積件数（Ｂ：バグ）をとってプロットした曲線である。日数の経過とともにテスト項目数の消化状況とバグ摘出累積数を示すもので、事前の予定との乖離状態や曲線の傾き度合いによってテスト項目の消化速度とバグ発生速度を一覧出来るので、テスト工程の進捗具合と当該ソフトの品質（残存する潜在バグ量がどのくらいか）の見通しがある程度可能となる。

ＰＢ曲線ではテスト項目数は過去に開発した事例から開発ステップ数あたりの標準テストケース数（テスト率）を算出して、これを必要充分なテスト量とみなす。同様にソフト品質もステップ数あたりのバグ摘出件数つまりバク率で現される。（バグはステップ数に比例して内在する、との考え方による。）過去に開発したシステムの平均値を標準とし、それとの乖離状態から品質を推測、管理しようとするもので、実務上の指標としては簡単で妥当なものとして多く使用されている。

しかしこの方式で推進しているプロジェクトでも納品後に相当数のバグ発生が絶えない。その原因のひとつとして、ＰＢ曲線の限界を知らずにこれを使用していることが挙げられる。例えば、ＰＢ曲線のテスト項目数では個々のテスト項目の質、やり方（これは担当ＳＥのスキルやテスト環境によって異なるが）の判断は出来ない。テストの効果は「項目数×質」であり、従って項目数だけ目標を達成しても実質が伴わない場合がある。また摘出バグも件数だけでなくその「内容」の吟味が重要な意味を持つにもかかわらず、これらの

認識が薄いのが実態である。これはまさに作業者のマインドの問題である。

さらには、果たして過去のプロジェクトの平均値が当該システムに当てはまるのか、と言う疑問もある。実際のプロジェクトで顧客に納品するシステムは仕様、環境、要員スキルなど前提条件ひとつひとつが異なっており同一のものは無い。特にアプリケーションの特性や担当作業者のスキルなどは影響が大きいため、品質も単純な平均値との比較で善し悪しを論ずることは出来ないはずである。例えば、図２の例では、結合テスト段階ではＢプロジェクトがより高い品質評価を受けるが、実はこれは誤った判断なのである。

しかし、現実問題として、テスト工程で経験の浅い作業者はＰＢ曲線に沿ってテスト項目をすべて消化し標準バグ率のバグ件数を摘出すればテスト完了と判断する傾向が強い。特に問題なのは、当該工程でバグ率が標準より低いと「高品質である」と勝手に判断してしまうことである。そもそもテスト工程で発生した不具合の原因は上流工程の不備（仕様、フログラミング、単体テストの不備など）であって、テスト工程で多数のバグを摘出することは上流で作成されたソフトの品質が悪いことを証明すると同時に、当該工程が効果的に実行されている証のはずである。逆にバグが少ない場合はこの逆の可能性がある。**（図２参照）**

＜図２　工程別のバグ率の比較例＞数値は架空のもの。　○×は工程評価**

	テスト工程		運用段階
プロジェクト名	結合ﾃｽﾄ	総合ﾃｽﾄ	運用以降
Ａプロジェクト	× 5 KSTP	○ 0.2/KSTP	○ 0.1/KSTP
Ｂプロジェクト	○ 3/KSTP	× 1.0/KSTP	× 1.3/KSTP

結合テスト以降の数値から、仮に100Kステップとした場合、バグ総量は、
・Ａプロジェクトは　500＋20＋10＝530件
・Ｂプロジェクトは　300＋100＋130＝530件　となる。
つまりＢプロジェクトは結合テストでバグ率 3件/KS と一見品質良好に見えるが、実は下流にバグを流していた、と言うことが判る。

また、標準指標値は過去のデータの平均値に過ぎず、当該プロジェクトにとっての目標や理想の値では無い。プロジェクト管理上でテスト工程における品質管理として大切なのは、当該システムの保有バグ総量が本当はどれだけ有り、現在はどんな状況にあって今後はどうなるのか、を早く正確に推定、認識することである。テストの真の狙いは、これを出来るだけ正確に把握して早めに適切な対策を施し、すべてのバグを摘出／修正して「ノ

ーバグ」[3]にすることだからである。そして高い品質を実現するには、作業担当者の「かくあるべし」と言う高い品質意識、すなわちマインドが重要であり、これを如何に指導、徹底させるかが重要課題なのである。

4．実践面での現実的な改善アプローチ

以上の様な問題意識に基づき、テスト工程におけるＰＢ曲線利用の改善案をマインドに重点を置いてまとめてみた。まず、品質管理をする際に必要となるポイントを挙げる。

第一に「ノーバグ納品を目指す」と言うテスト目的をプロジェクトの全員に徹底させることである。予定された項目を消化すればそれで良しとする考えを改めさせ、担当者に真剣にテストさせることが肝心であり、具体的には以下の項目を徹底させることである。

・充分なテスト量を準備し、ノルマの消化に努めること。
・テスト仕様書と摘出バグ内容をリーダが充分吟味してテスト項目の質を高めること。
・全員の気力維持に気を配りテストにメリハリを付けさせること。そのために多段階、多重テストの形式を取る。（例えば第１次は基本確認、第２次は詳細機能確認、第３次はイレギュラー確認、第４次はインターフェース確認第５次は過負荷テスト、第６次はバグ修正再度確認、等々）

第二に「目標バグ摘出件数」を明示し、これにチャレンジさせる「目標管理」を行うことである。過去の経験から想定されるバグ累計件数を上回る「高めの目標件数」を示してこれにチャレンジをさせる。人間の心理として、バグは「もう無い」と思えば努力しないが「まだあるはず」と思ったときは真剣に工夫してテストを行うことになる。そのためには潜在バグの残量予測と現実のギャップを担当者に認識させることが重要である。これによってテストのやり方の反省や今後取るべき体制、設備、期間、費用などの判断が決まってくるからである。バグが無いソフトを長期間テストするのは無駄であるが、逆にバグが多数残留しているのに「もう無いだろう」とテストを止めたり体制を縮小したりすれば、後々大きな問題となる。目標件数はこの残量予測を上回る数値で無ければ効果が無い。

第三に、新たに考案した「図で見るテスト実践ガイド」（**図３**）を適用することである。これはＰＢ曲線を典型的な５ケースにパターン化し、「テスト工程の妥当性」と「当該ソフトの品質レベル」について潜在している問題点を担当者に考えさせるもので、下記の２原

3）バグが全く無い状態を言う。現実問題として、プロジェクトの資源や期間の制約の中で完全にバグをゼロにすることは不可能に近い。しかし「運用上で殆ど問題無い」レベルまでは実現可能である。

則にもとづき簡単なチェックポイントを設定したものである。

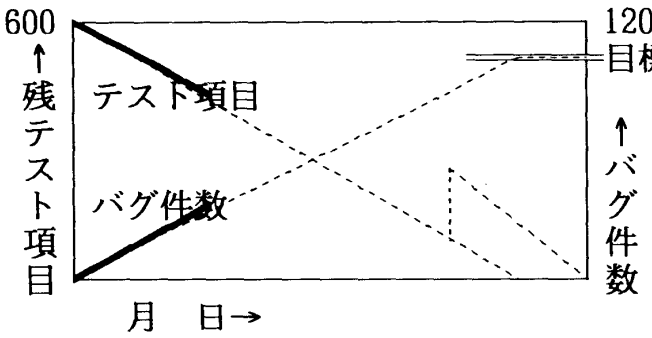
①バグが沢山出る⟶良くテストしているか、又はソフトが低品質である。

②バグが余り出ない⟶テストの仕方が悪いか、又はソフトが高品質である。

当該プロジェクトがどのケースに該当するかをビジュアルに作業担当者に理解させ、チェックの要点に従って仕事の進め方や工程進捗上の隠れた問題点を考えて見極めと評価をさせる事が目的である。要は担当者自身に当該プロジェクトの本質的な品質レベルを吟味させ、このままでノーバグを達成出来るかどうかを検討させるのである。

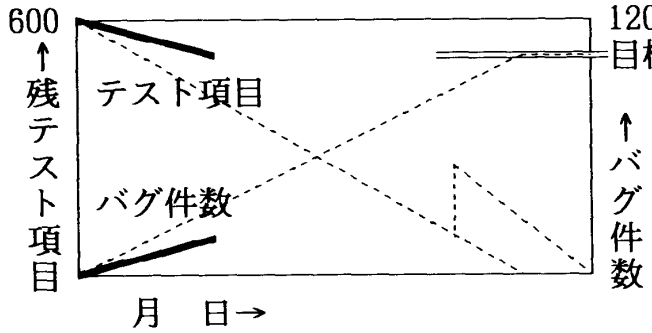
＜図３　図で見るテスト実践ガイド＞

【ケース１】「予定通り」------☆要注意☆　順調か？本当に？なぜ順調なのか？



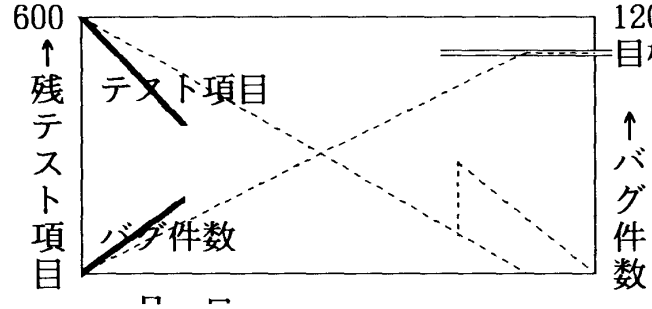
《チェックポイント》
- 本当に準備が良くて順調か？
- 過大体制のテスト計画ではないか？
- テスト内容に手抜きが無いか？
- バグ目標値が低すぎないか？
- この先に遅延要因（リスク）は？

【ケース２】「末広がり」------★要警戒★立ち上がりが悪い？なぜか？



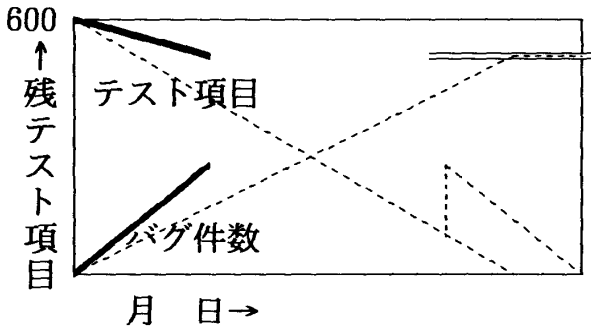
《チェックポイント》
- 準備不足？　体制不足？
- なぜ準備遅れたか？要員？環境？
- テスト仕様に問題はないのか？
- この先の見通しはどうなのか？
- 対策はあるのか？

【ケース３】「末閉じ」--------☆要注意☆立ち上がり好調？なぜ？　先の見通しは？



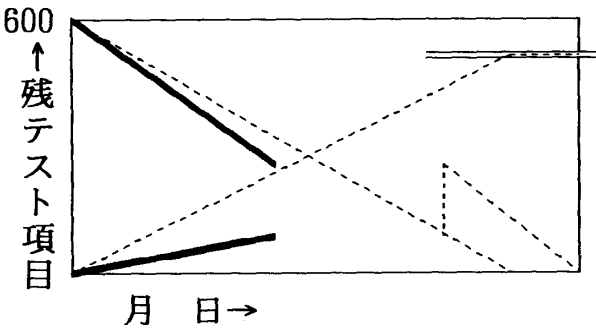
《チェックポイント》
- 準備良好？ガンバリ？体制過大？
- テスト仕様過少？テスト手抜き？
- 障害が早期に収束しつつある？
- バグ目標過少？
- 上流工程の品質が悪すぎた？
- この先の見通しは？
- 修正工数の確保は？

【ケース４】「上向き」----------★極めて危険★テスト消化不良？かつ、バグ多発？



《チェックポイント》
- 最悪？それともテスト内容が優秀？
- バグ多発対策は？（修正要員、他）
- バグ内容の分析は？（傾向、真因）
- バグ目標過少？
- 今後の見通しは？（リスクは？）
- 抜本的な対策は？（人、物、金）
- 上流工程の見直し？

【ケース５】「下向き」----------★要吟味★テスト消化良好？バグ検出低調？



《チェックポイント》
- テスト過少（質／量）？体制過大？
- テスト技量？（内容、要員レベル？）
- 本当に品質良好？後日上昇危険は？
- 本当に高品質ならその理由は何か？
- 次工程はどうなる？（見通し）

5．実　践　事　例

以上述べた改善アプローチを実際のプロジェクトのテスト工程に適用した事例を、特に重要である結合テスト工程について、開始時点での見通しから途中見直しの状況や対処および評価について述べる。

※この事例におけるバグ率は、標準値と比べて約10％低いものであるが、期間に無理があったため、作業的には苦しいものとなった。

（1）プロジェクト概要と実践内容

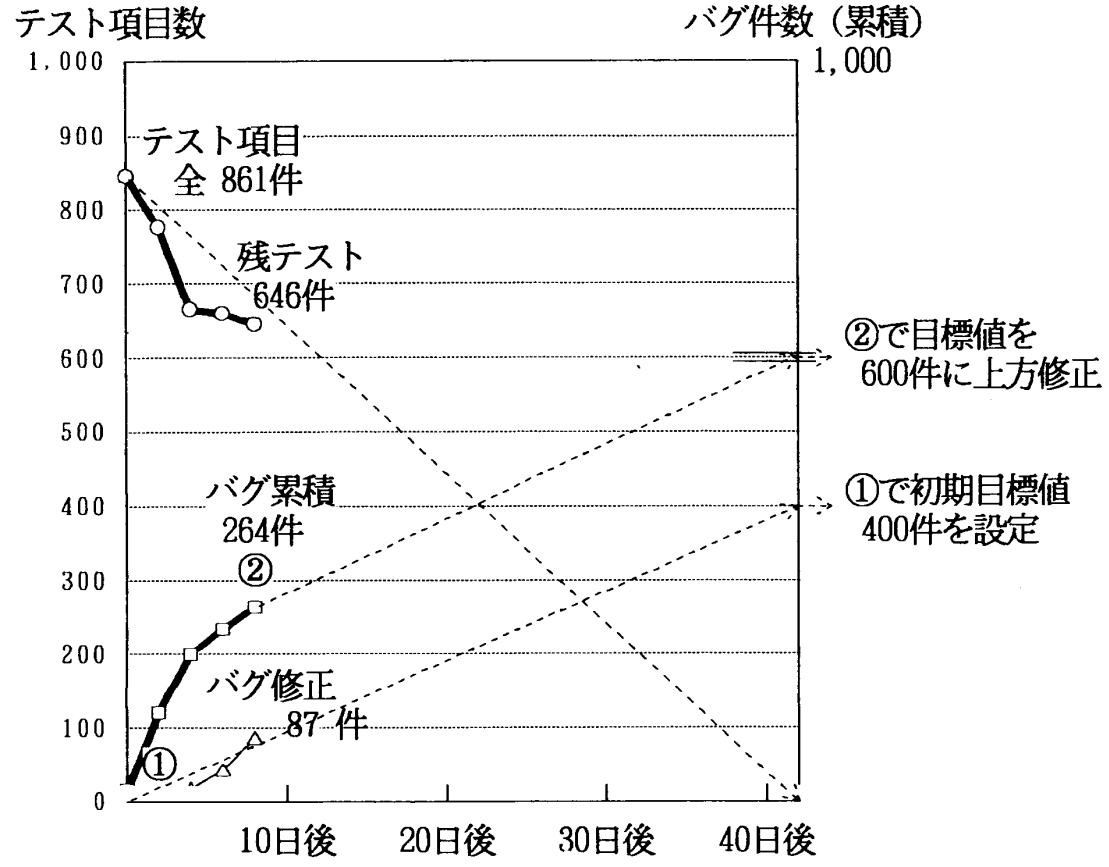
このプロジェクトは航空機のフライト訓練の計画／実績管理を行うシステムで、Windows NTと ORACLE を使用したサーバ／クライアントシステムである。使用言語はVisualBasic、開発期間は実質10ケ月とかなり短納期（そのうちテスト期間はおおよそ結合テスト1.5ケ月、総合テスト２ケ月）で、プログラム規模は約300Kステップ（コメント含む）のやや大きめの中規模プロジェクトである。当初見積もりでは200Kステップだったが、開発途中での仕様増大などで結果的にプログラムステップ数が1.5倍に膨れ、結合テストの最盛期は22名もの要員が投入された。

①開始時点の状況

諸般の事情による作業の立ち上がり遅れから開発期間が２月ほど短くなり、かなり厳しい作業となった。担当者は結合テスト開始時点ではバグ予測を過去の経験から、目標値400件と設定して作業開始した。

②10日後の状況（**図４－１参照**）

＜図４－１　結合テスト見直し時点②のＰＢ曲線＞

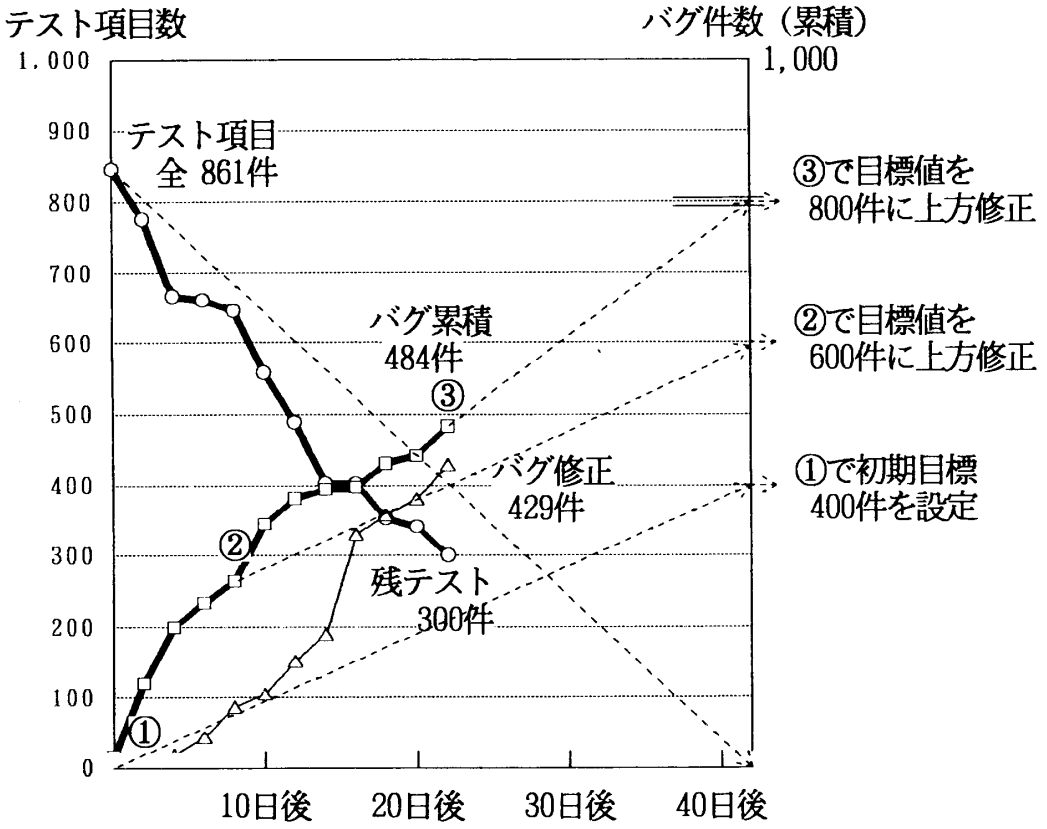


しかし結合テスト開始10日後にはＰＢ曲線が「図で見るテスト実践ガイド」の【ケース３】「未閉じ：要注意」の形態を示したため見直しを行った。この時点では、テスト項目の消化実績は予定以上だったがバグ発生件数が予定を大幅に超過し、しかもバグ修正は発生件数に追いつかないと言うギャップも拡大傾向にあった。そこで目標値の当初数値は非妥当と判断して目標件数を600件に上方修正し、併せて要員手当てと体制強化を行った。原因は短期間で無理をしたため、上流工程の品質が悪すぎたと判断され、継続して厳しい進捗管理を行った。

③さらに２週間後の状況（**図４－２参照**）

その後も状況は改善せず、ＰＢ曲線が一層顕著に【ケース３】「未閉じ：要注意」の形を示し、バグは特定のプログラム（担当者）に集中していた。しかし①の時点と違ってバグ修正が発生件数に追いついてきたので、その点は好転したものと判断出来た。そこで特定

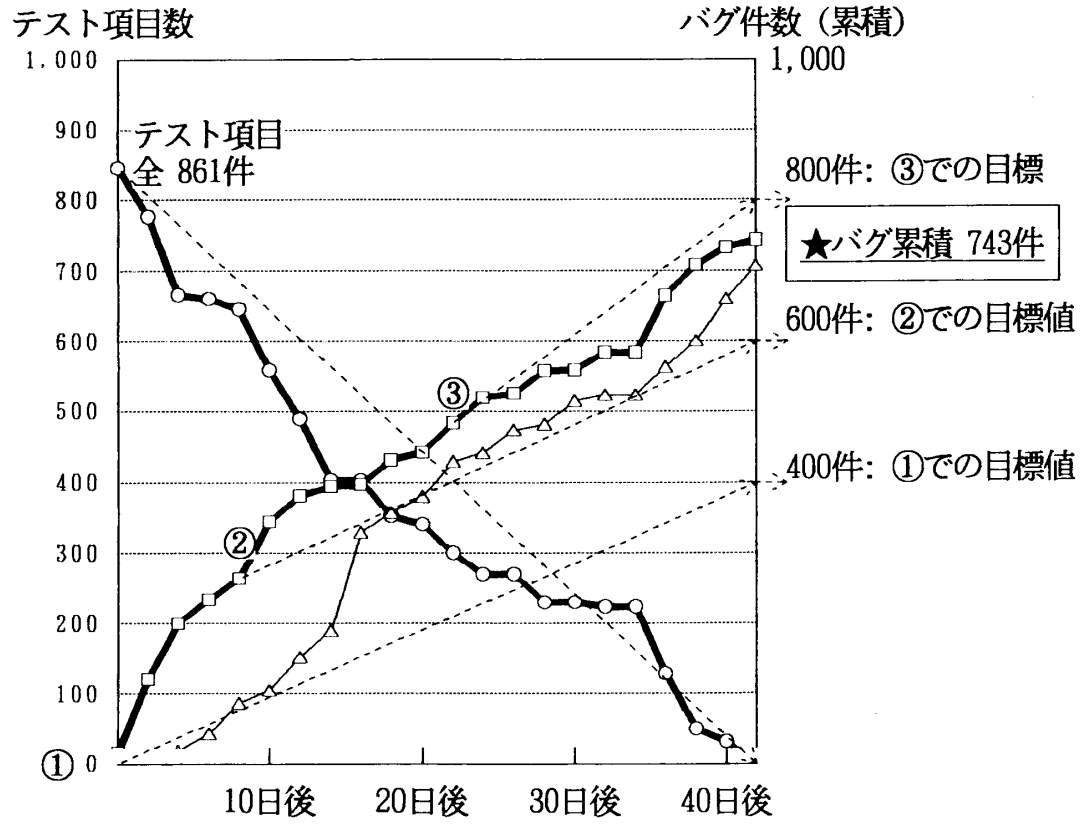
＜図４－２　結合テスト見直し時点③のＰＢ曲線＞



プログラム部分に要員補強を行うと共に、再度、バグ摘出目標値を900件に上方修正した。

④さらに３週間後の状況（**図４－３参照**）

＜図４－３　結合テスト完了時点のＰＢ曲線＞



引き続き３週間の結合テストを継続して、ＰＢ曲線が概ね【ケース５】「下向き：要吟味」の形を示しながら品質はようやく安定してきたと判断出来る状況になり、結合テストを終了として、次の総合テスト工程に進んだ。

２）評価

この事例では結合テストにおけるＰＢ曲線が初期には【ケース３】「末閉じ：要注意」の典型的なパターンとなったが、早めに要員確保など措置を講ずることによって後半からは【ケース５】「下向き：要吟味」の状態となり品質が安定化して来た[4)]。（＊注４）

さらにその後、約１ケ月間の総合テストを実施し、そこで約100件のバグを摘出してテスト完了となり、無事顧客に納品することが出来た。今回、ＰＢ曲線を活用するうえで「バグ目標値の設定」と「図で見るテスト実践ガイド」を適用することによって下記の効果が得られた。

・テスト工程の進展に沿って品質把握が出来たため、迅速、的確な対処が出来た。
・バグ摘出目標値の設定により担当者のテスト作業目的が具体化され、全員の励みとなってテストの進展やチームワークに好影響を与えた。
・ＰＢ曲線と「図で見るテスト実践ガイド」との比較がビジュアルに出来るため、経験の浅い担当者でもソフトの品質と潜在しうる問題点を容易に実感して理解出来た。
・経験の浅いリーダの初期の目標設定値はかなり楽観的である点は注意を要する。
・統計的には信頼度成長曲線（バグ累積曲線）はＳ字型を示すが、品質が悪いシステムの場合は、初期の発生状況をリニアに延長しての目標予測がより適切と言える。

このほか、実際のプロジェクトへの適用を通じて、経験の浅い担当者でもテスト工程における問題点がビジュアルに理解出来たので、ソフト品質や作業見通し判断の訓練や品質マインドの向上と言う、教育面での効果も大きかった。

６．ま と め

本稿ではソフト品質を改善するため、テスト工程におけるマインド重視の観点からの具体案を提供し、これを実際のプロジェクトに適用して効果を確認した。具体案とは、第一に「ノーバグ納品の意識付け」を行って作業者の基本姿勢を正すこと、第二に、個別のプ

4）この事例では期間、体制ともに余裕が無かった為、結合テスト工程で「追加テスト項目」を付加出来なかった。

ロジェクトに過去の統計値や経験を適用するための実践的アプローチ（方法論）として「バグ摘出の目標管理」で作業ノルマ的にやる気を起こさせ、「図で見るテスト実践ガイド」で客観的、直観的に品質実態を評価してアラーム検出を早期に行わせることである。

ここで述べたアプローチは「品質も最後は担当者のマインドの問題」と言う考え方による。ソフト品質は作業担当者、特にプロジェクトリーダの品質マインド指導によって大きく影響される。通りいっぺんのテストをやり、バグが出たら修正し納期が来たら納入するような作業者は品質マインドが高いとは言い難い。品質評価ツール、改善ツールを使用してもマインドの低い作業者の場合は「ツールを使う事」だけが目的となり、必ずしも品質向上にはつながらないケースも多い。マインドが低いと目の前で発生しているバグをさえ見落としてしまうのである。

「納入ソフトの品質は高いレベルを保証すべきである」と考える高い品質マインドを持った作業者は、上流工程から引き継いだソフトの品質レベルを常に考えながら現在のテスト項目消化状況とバグ発生状況を客観的な指標に基づいて吟味し、それらによって残った潜在バグ総量を推定して今後の工程、体制、やり方などに必要な対策を施すことが出来るはずである。

情報システム開発においてはこのような人材が必要とされる。その為には本稿で述べた作業方法によって、リーダが実作業を通じて作業メンバーを教育、指導することが効果的であると考える。