15. ソフトウェアの品質に関する計測の重要性

本章では、ソフトウェアの品質に関する計測法について整理する。フォーマルメソッドの適用前後で、ソフトウェアの品質がどのように向上したか、比較評価するための参考となる。

15.1. 計測の重要性

ソフトウェア開発に於ける計測と分析の主たる目的は、プロジェクトが目指している方向に向かっている事を監視し、問題があれば是正し、目標としている到達地点へと導く事である。

一般的なソフトウェア開発プロジェクトが目指すべき方向とは高品質なソフトウェアを短期間に低コストで開発する事である場合が多く、これらの目標を達成するためには、常にプロジェクトの動向を監視し、問題の前兆を早期に察知し是正する事が不可欠となる。

ある程度ソフトウェア開発を経験している企業であれば、ソフトウェア開発に関する何らかの数値は計測しているケースは多い。しかしながら、集めたデータをどのように使っているか？効果的な使い方ができているか？となると集めているだけで効果的な使い方ができていない場合も多いのではないだろうか。

そもそも計測とは(ソフトウェア開発に限らず)、本来あるべき姿や向かうべき先とそこにたどり着くまでの進捗を把握するために必要な情報が明確になった上で、具体的に何を計測するべきか？が決められるべきである。このように目的を明確にした上で、計測を行うフレームワークとしてはメリーランド大学の Basili 教授らによって提唱された GQM(Goal／Question／Metric)パラダイムが有名である。

GQM パラダイムをモデルとして計測を行う例を以下に示す。

Goal	・dd 日までに実装を完了したい
Question	・全体の実装量の内、どの程度実装完了しているのか？
	・このままで間に合うのか？
Metric	・残実装量
	・現在日付
	・1 日当たりの生産性
	・担当者の負荷状況

「dd 日までに実装を完了したい」という Goal に向けて、途中で進捗を把握するためには、「全体の実装量の内、どの程度実装完了しているのか？」「このままで間に合うのか？」といった事を確認する必要がある。これらの Question に回答するためには、「残実装量が nn キロステップで 1 日当たり n キロステップ実装しているので、あと何日で実装完了できる見込み」といった分析が必要になる。また、「このままだと間に合わないため、担当者を増やす必要がある」「担当者の負荷が集中しているため、他の作業を別の担当者に割り当てる」などといった是正処置が必要な場合もあるだろう。これらの分析や是正処置を行うためには、「残実装量」「現在日付」「1 日当たりの生産性」「担当者の負荷状況」といったメトリクスの必要性が導出される。

仮に、「残実装量」が計測されていないとすると、いくら「現在日付」や「1 日当たりの生産性」を計測していても Goal に向かっていることの分析は困難になってしまう。また、「担当者のコミット数」が計測されていたとしても「dd 日までに実装を完了したい」という Goal に対しては役に立たないデータであり、計測の手間が無駄になってしまう。

無駄なく必要最低限の計測を行う為には、的確な「Question」の設定がポイントとなる。上記の例で、Question が「全体の実装量の内、どの程度実装完了しているのか？」しか設定されていなければメトリクスとしては「残実装量」しか計測されていないだろう。この場合、「dd 日までに実装を完了したい」という Goal を達成するための分析や是正処置が行えなくなってしまう。

計測における「Question」を設定する際、Goal を満たすために必要な項目として Question を導き出すだけではなく、どのような事が起きると Goal が満たせなくなるのか？にも着目して Question を導き出すと良い。

15.2. 計測データの信頼性

ソフトウェア開発を計測し分析する上で、計測データの信頼性が非常に重要である。例えば「現在日付」として計測された値が 3 日遅れていればその後の分析や是正処置は誤ったものになってしまい、折角行った計測と分析全体が無意味なものとなってしまう。計測データの信頼性確保に当たっては、以下のポイントが重要となる。

- 計測方法（計測対象、計測タイミング、計測手順、分類など）
- 計測期間（計測の開始時期と終了時期）

ソフトウェア開発プロジェクトでデータを収集する場合、各担当者などの複数人がデータを収集する場合が多い。このようにして集められたデータを同じデータとして分析を行うのであれば、計測を行う担当者間で計測対象や計測するタイミング、計測の方法などが統一されている必要がある。例えば、ソースコードの量を計る場合、コメントを含めるか否かやヘッダファイルを含めるか否かの統一がされていない状況で各担当者が集めたソースコードの量を分析しても正確な結果を得る事はできないし、レビューの効果を分析するためにレビューの指摘件数を計測した場合、インスペクション形式で実施したレビューのデータとパスアラウンド方式で実施したレビューのデータを同一に扱うのにも無理がある。

また、テスト工程で発見した不具合件数を計測する場合、テスト工程の開始時期や終了時期がいつなのか？といった計測の開始時期や終了時期についても明確になっている必要がある。

効果的な計測を行う為には、計測対象のメトリクスに対して「誰が、いつ、何を、どのように計測して、どこに格納するのか」といった計測に関するルールを厳格に定義し、計測に携わる全てのメンバー間で共有する事が必須となる。

15.3. 計測情報と品質の関係について

ソフトウェア開発に於ける品質とは、大きく以下の 2 種類が考えられる。

- 要件定義／設計／実装工程における品質
- テスト工程における品質

前者は主にレビューによる品質確保を行い、後者は主にテストによって品質が確保される。だが、ここで「本当に十分な品質が確保されたのか？」という疑問が沸いてくる。この疑問に答える為には、レビューとテストのデータを計測し、定量的／定性的な分析を行う必要がある。

また、多くのソフトウェア開発現場では軽視されがちだが、テスト項目を設定する為の活動であるテスト設計も、前者と後者に関わる。要件定義や設計に曖昧性や欠陥があり品質が悪い場合、テスト設計の項目作成時に発見されることがよくある。一方、テスト設計の品質が悪く、テスト項目が必要以上に多い場合や、テスト項目が荒く十分に確認しきれない場合、テスト工程での進捗の滞りや、欠陥摘出漏れを発生させることとなる。そのため、「テスト設計における品質」も、定量的／定性的な分析を行う必要がある。

15.3.1. レビューによる品質確保の計測と分析

レビューによる品質確保が十分に行われたか？を分析するためには、個々のレビューが一定以上のパフォーマンスで問題を十分に抽出できたのか？といった定量的な分析と十分に問題が抽出できなかったレビューは特別な理由があるのか？その理由は許容可能な理由なのか？といった定性的な分析を実施する事が多い。

レビュー品質の計測における典型的な例は以下がある。

Goal	・レビューによって十分な品質確保を行う事
Question	・レビューによって不具合をどの程度抽出しているか？
	・レビューに掛けた時間は適当か？
Metric	・レビュー対象
	・レビュー時間(事前の準備時間も含む)
	・レビュー参加人数
	・指摘数
	・レビュー対象規模
	・レビューの種類

レビュー品質の分析を行う場合、「レビュー速度」や「レビュー指摘密度」といった評価尺度を用いる事が多い。レビュー速度とは レビュー規模／レビュー時間 で求められ、レビューの密度が適切かを分析する際に用いられる。組織の閾値と比べてレビュー速度が速い場合は、欠陥摘出が不十分である可能性があり、逆に遅い場合はレビュー効率が悪かった事が予想される。一方レビュー指摘密度とは、 指摘件数／レビュー規模 で求められ、レビューでの指摘件数が適切かを分析する際に用いられる。組織の閾値と比べて密度が高い場合は、前工程での品質確保が不十分であったなど、品質が悪い事が予想され、逆に密度が低い場合は、残存してしまっている欠陥がある事が予想される。このように定量的に組織の閾値と比較して標準から逸脱しているレビューを探しだし、それらに対して定性的な分析を行い、必要に応じて再レビューなどの是正処置を行う事が重要である。

組織の閾値は過去の実績から定義されることが理想的であるが、過去実績のデータが無い場合は直近のデータなどから設定する。閾値は定期的に見直しを行い、徐々に精度の高いものにしていく事が重要である。また、レビューのパフォーマンスはインスペクション／パスアラウンドなどのレビュー方法や初回レビュー／再レビューなどによって変わってくる為、閾値を設定する際にはこれらの要素も勘案すると良い。

15.3.2. テストによる品質確保の計測と分析

テストによる品質確保が十分に行われたか？を分析するための手法はゾーン分析、トレンド分析、信頼度成長曲線による分析など多岐に渡るがここでは信頼度成長曲線を活用した計測／分析の例を示す。

Goal	・テストによって欠陥を摘出し、品質を保証する事
Question	・テスト完了予定までに不具合が収束するか？
Metric	・テスト件数
	・消化件数
	・欠陥摘出数

テストで品質を確保するためには、そもそもテスト設計自体が妥当であるか？という検証が必要と

なるが、これについては「テスト設計自体の妥当性の検証の重要性」にて後述する。

信頼度成長曲線による分析では、以下の 4 種類の線を用いて分析を行う。(図 15-1 信頼度成長曲線 参照)

- テストケースの消化予定線
- テストケースの消化実績線
- 欠陥摘出累積予定線
- 欠陥摘出累積実績線

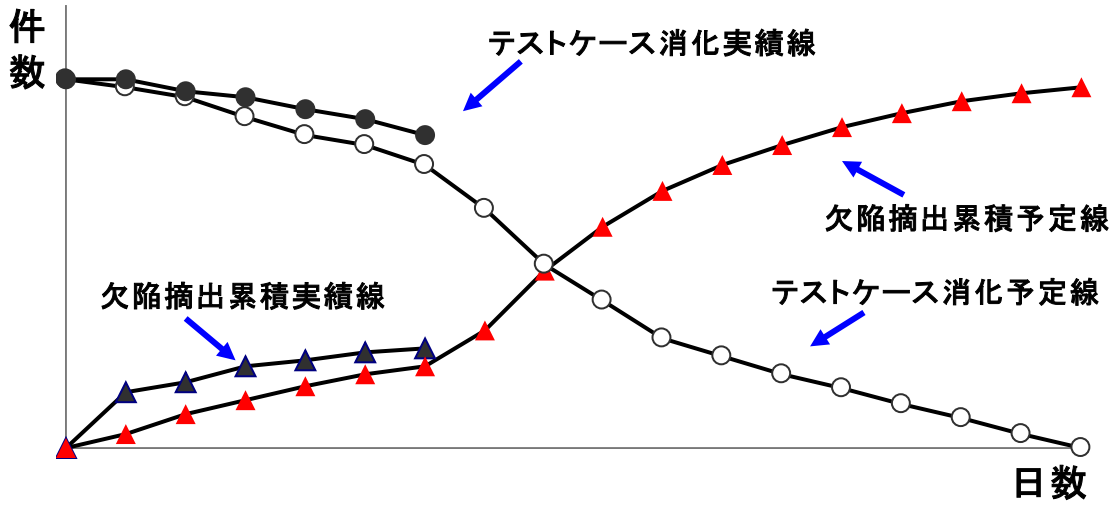


図 15-1:信頼度成長曲線

本分析手法では、テストの消化状況と欠陥発生状況のバランスを見ることによって、テスト進捗だけでなく品質の良し悪しが判断できる。テストの終盤段階では、欠陥摘出累積が収束傾向(グラフの線が水平になる)になると、品質が安定してきたと言える。信頼度成長曲線では、図 15-2 に示す傾向にある場合にテスト自体に発生している問題を早急に察知可能である。

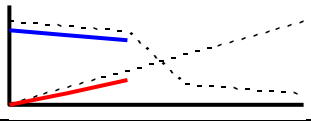
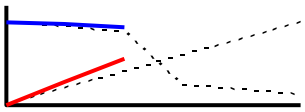
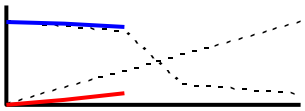
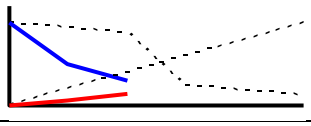
#	組み合わせ		信頼度成長曲線	考えられる問題
	消化実績	摘出実績		
1	正常	正常		特になし
2	消化 停滞	欠陥 多発		前工程における欠陥摘出不足 仕様変更、途中追加による欠陥の作りこみ
3	消化 停滞	摘出 不足		デバッグ／テスト環境の不備 テスト要員不足
4	急激 消化	摘出 不足		テスト項目の質が低い デバッグ実施優先度が単純機能に偏っている

図 15-2:信頼度成長曲線の傾きとテストに発生している問題の例

信頼度成長曲線による計測／分析は結合テスト、システムテストなどのテストレベル毎に行う。テスト全体の有効性や品質の安定度合いは各テストレベルで摘出した欠陥数の分布を見ることで評価する。図 15-3 は、各テストレベルで摘出した欠陥数と、そこから導き出される品質評価の例である。

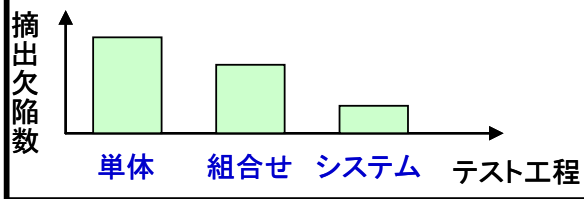
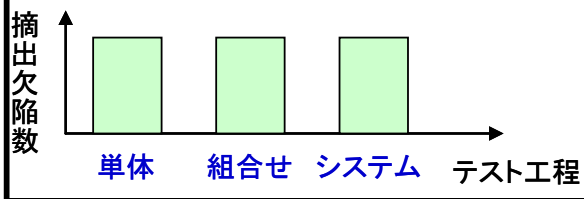
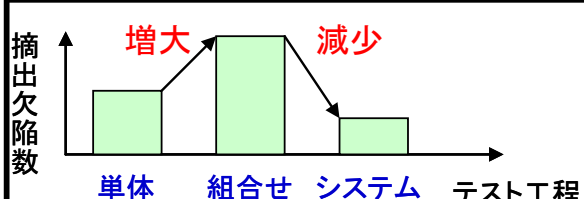
工程別摘出欠陥数	品質の評価
	①摘出欠陥数が単調減少(指数型) － 品質は安定傾向
	②摘出欠陥数が単調非減少(一様) － 品質は非常に悪いことが多い － 前工程で摘出すべき結果を次工程に持ち越している。
	③摘出欠陥数が増大から減少に － 組合せテスト工程において結果を大部分摘出 － ただし、各工程の欠陥数の目標に占める割合の再評価が必要。

図 15-3:欠陥の摘出工程分布と品質評価

これらの分析を行い、テストにおける問題が発見された場合は、テスト項目の追加や前工程に戻

って品質を再確保するなどの是正措置を早急に行う事が重要となる。

15.3.3. テスト設計の品質の検証

一般的にテストとは、ほぼ無限にあるテスト項目に対して限られた時間とコストで臨む事になる。従って、いかに効率よく欠陥を摘出できるかが肝であり、より少ないテスト項目でより多くの欠陥を摘出するために行うのがテスト設計である。通常テスト設計では何に対してどのようにテストを行い、どのような結果を期待するのか？を設定する。つまり、テスト設計の品質がテストの成否を左右する。

また、テストとは設計された通りにソフトウェアが実装されていること、または顧客が要求したとおりにシステムが動作する事を確認する行為であり、テスト設計とはこれらを確認するためのテスト項目を設定する為の活動であるため、設計内容を要求仕様やシステム要件の視点で見直す事にもなる。この為、テスト設計を行うと仕様や設計の欠陥や曖昧である部分が発見されることが良くある。

以上よりテスト設計工程では、テスト設計そのものの品質と、要件定義／設計の品質の 2 つの測定／分析を行う。

テスト設計の計測における例を以下に示す。

Goal	・より少ない項目でより多くの欠陥を摘出するテスト項目を作成する事
	・要件定義／設計の品質を検証する事
Question	・テスト項目は妥当か？
	・要件定義／設計の不具合をどの程度摘出しているか？
Metric	・要件定義／設計の仕様書の規模
	・テスト項目数
	・要件定義／設計の欠陥／曖昧性摘出数

仕様書の規模に対してテスト項目数が少ない場合、テスト設計が荒く、テスト工程で十分に不良を摘出しきれない事が想定される。一方、仕様書の規模に対してテスト項目数が多い場合、必要以上のテスト項目が作成され、テスト工程で必要以上の時間を浪費してしまうことが想定される。

欠陥／曖昧性摘出数が多い場合、要件定義／設計の品質が悪いことが想定される。このように、テスト設計工程で要件定義／設計の品質が悪いことがわかると、前工程への手戻りが発生する事になってしまう。この点に注目し、設計が終了したタイミングでテスト設計を行うW字モデル開発（図15-4）という考えがある。

W 字モデルでは、要件定義／設計工程が終了した段階で、その工程に対応するテスト設計を行う。このため、テスト設計工程で発見された要件定義／設計工程での欠陥や曖昧性を、実装終了後にテスト設計を行う場合に比べて早い段階で摘出可能となる。したがって、W字モデルを適用することで、前工程への手戻りにかかる工数を削減できる。

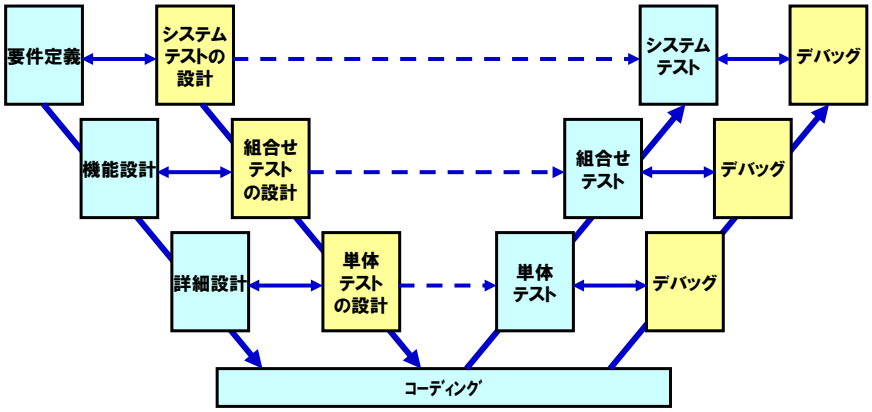


図 15-4:W 字モデル開発

15.4. プロセス管理での利用上の効果

計測や分析は品質に寄与するのみではなく、ソフトウェア開発プロセスに従った開発を推進する上でも非常に重要な役割を担う。組織で定められたプロセスがある場合、実際に行われている開発作業の計測／監視を行う事で、どの程度プロセスに沿った開発が行われているかを評価する事ができる。ここで計測対象となるのは、先に述べた数値的な部分のみではなく開発の進め方そのものも計測対象とする必要がある。このようにプロセスがどれだけ遵守されているかを監視する考え方は「プロセス QA」と呼ばれる。プロセス QA については、CMMI のプロセスエリア「PPQA」を参照の事。

プロセス QA の目的は、プロセスの不遵守を発見し是正させる事のみではなく、プロセス自体の問題点を発見する上でも役立つ。「不遵守＝守らない側が悪い」とするのではなく、なぜ不遵守なのかを分析した上で必要に応じてプロセス自体を改善する事も重要である。

15.5. 簡便な計測方法

そもそも計測とはソフトウェア開発を成功に導くために行う活動であって、計測活動そのものが目的となってしまわないように注意する必要がある。過度な計測やあまりにも高度な分析はそれ自体に多大なコストがかかるため、計測そのものがソフトウェア開発の足枷となってしまうリスクを孕んでいる。計測活動はあくまでもソフトウェア開発のサポートに位置する活動である事を認識し、計測や分析に掛かるコストを算出した上で活動の範囲が決定されるべきである。

効果的に計測活動を行う為のポイントとしては以下がある。

- 計測対象の絞込み
- データ収集の自動化
- 分析基準の数値化

15.5.1. 計測対象の絞込み

事業目標や業界動向などからコアコンピタンスとなる部分を定め、計測を行う対象を絞り込む。また、既存のメトリクスを複数掛け合わせる事によって関数的に導出可能なメトリクスは新たなメトリクスとして定義しないなどの工夫も重要である。

15.5.2. データ収集の自動化

ツールの活用によって、データ収集の自動化を行うようにする。例えばソースコードの量であれば、構成管理ツールにコミットしたタイミングで自動的にステップカウンタに掛ける事で自動的に収

集したり、作業工数であれば出退勤のシステムから情報を自動的に取得する事もできるかもしれない。データ収集の自動化は、計測コストの削減のみならず人間系の計測による作業ミスの発生を回避する効果もある。

15.5.3. 分析基準の数値化

標準からの逸脱を判断するための基準を数値化し、誰でも同じ判断が瞬時に行えるようにする。