エンピリカルソフトウェアエンジニアリングの勧め

独立行政法人 **情報処理推進機構**　ソフトウェア・エンジニアリング・センター　編著

SE
SHOEISHA

本書内容に関するお問い合わせについて

このたびは翔泳社の書籍をお買い上げいただき、誠にありがとうございます。弊社では、読者の皆様からのお問い合わせに適切に対応させていただくため、以下のガイドラインへのご協力をお願い致しております。下記項目をお読みいただき、手順に従ってお問い合わせください。

●お問い合わせの前に
弊社 Web サイトの「正誤表」や「出版物 Q&A」をご確認ください。これまでに判明した正誤や追加情報、過去のお問い合わせへの回答（FAQ）、的確なお問い合わせ方法などが掲載されています。

正誤表 http://www.seshop.com/book/errata/
出版物 Q&A http://www.seshop.com/book/qa/

●ご質問方法
弊社 Web サイトの書籍専用質問フォーム（http://www.seshop.com/book/qa/）をご利用ください（お電話や電子メールによるお問い合わせについては、原則としてお受けしておりません）。

※質問専用シートのお取り寄せについて
Web サイトにアクセスする手段をお持ちでない方は、ご氏名、ご送付先（ご住所／郵便番号／電話番号または FAX 番号／電子メールアドレス）および「質問専用シート送付希望」と明記のうえ、電子メール（qaform@shoeisha.com）、FAX、郵便（80 円切手をご同封願います）のいずれかにて“編集部読者サポート係”までお申し込みください。お申し込みの手段によって、折り返し質問シートをお送りいたします。シートに必要事項を漏れなく記入し、“編集部読者サポート係”まで FAX または郵便にてご返送ください。

●回答について
回答は、ご質問いただいた手段によってご返事申し上げます。ご質問の内容によっては、回答に数日ないしはそれ以上の期間を要する場合があります。

●ご質問に際してのご注意
本書の対象を越えるもの、記述個所を特定されないもの、また読者固有の環境に起因するご質問等にはお答えできませんので、予めご了承ください。

●郵便物送付先および FAX 番号
送付先住所 〒 160-0006　東京都新宿区舟町 5
FAX 番号 03-5362-3818
宛先（株）翔泳社編集部読者サポート係

目次

1 はじめに

1つのソフトウェア開発プロジェクトの失敗で、他の多くの成功プロジェクトが上げた利益を消してしまう場合があります。1つのプロジェクトで莫大な利益を上げることは難しいですが、プロジェクトの運営で間違った対応を繰り返していると、すぐに損失が大きくなってしまいます。

プロジェクトで問題が発生するのは当たり前ですが、その問題をできるだけ早期に発見し、適切に対処できるかどうかが鍵になります。問題を早期に発見するためには、プロジェクトの状況を常に正しく認識しておく必要があります。

本書では、「エンピリカルソフトウェアエンジニアリング」によって、プロジェクトの状況をリアルタイムに、定量的に把握し、状況の変化をいち早く捉え、対応できるようにすることを推奨します。また、そのための道具として「EPM ツール」を紹介します。

2 エンピリカルソフトウェアエンジニアリングの導入

エンピリカル Empirical とは、Experiment と Experience が融合した言葉で、実験や観察を通して実際に得られた知識・知見や現場での実績・経験、それに基づいた技術・手法を利用することを意味します。

エンピリカルソフトウェアエンジニアリングは、ソフトウェア開発プロジェクトの活動に関連するデータ（エンピリカルデータ）を収集し、実績のある手法で分析し、分析結果をプロジェクトに提示することで、プロジェクトが失敗しないよう支援するものです。

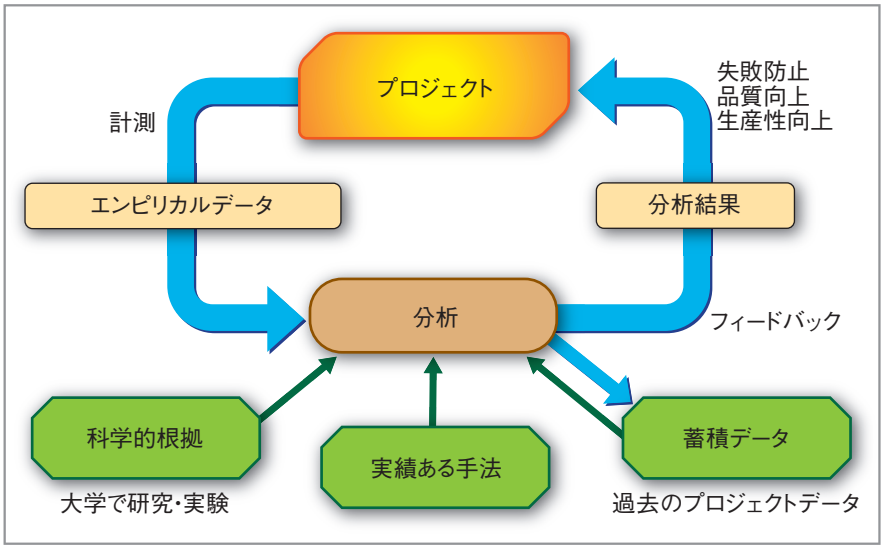

図 1　エンピリカルソフトウェアエンジニアリング

3 まずは実態を明らかに

プロジェクトの状況は日々刻々変化しています。その状況を的確に捉えるには、「計測」が必要です。天気図の作成や気象予報などを行うためには、日本国内の 1300 カ所から気象データを収集しています。

ソフトウェア開発プロジェクトにおいても計測を行い、できるだけ定量的にプロジェクトの実態を把握することが重要です。

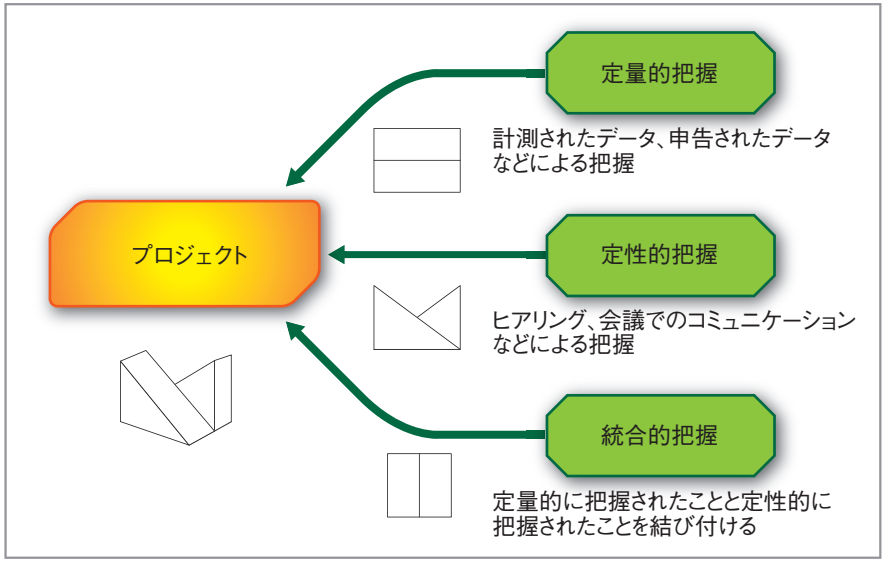

図 2　プロジェクトを把握する

4 負担をかけずにエンピリカルデータを収集し分析

データの収集に手間がかかってしまっては、ソフトウェア開発作業に支障をきたしてしまいます。できるだけ開発者への負担を少なくし、手間をかけずにデータを収集しようという発想で開発されたのが、EPM（Empirical Project Monitor）ツールです。

EPM は「EASE プロジェクト」（第 10 章参照）で開発されたオープンソースソフトウェアで、独立行政法人 情報処理推進機構（IPA）では、EPM を機能強化し、ソフトウェアエンジニアリングのプラットフォームとして普及させようと考えています（第 11 章参照）。

IPA のソフトウェア・エンジニアリング・センター（SEC）では、EASE プロジェクトの協力を得て、EPM をはじめ、さまざまなツール・手法を利用してエンピリカルデータを収集・分析する実証を「先進ソフトウェア開発プロジェクト」として実施しました。そのときのエンピリカルデータの流れを図 3 に示します。本書では、IPA で機能強化した「EPM ツール」を中心にエンピリカルソフトウェアエンジニアリングについて説明します。

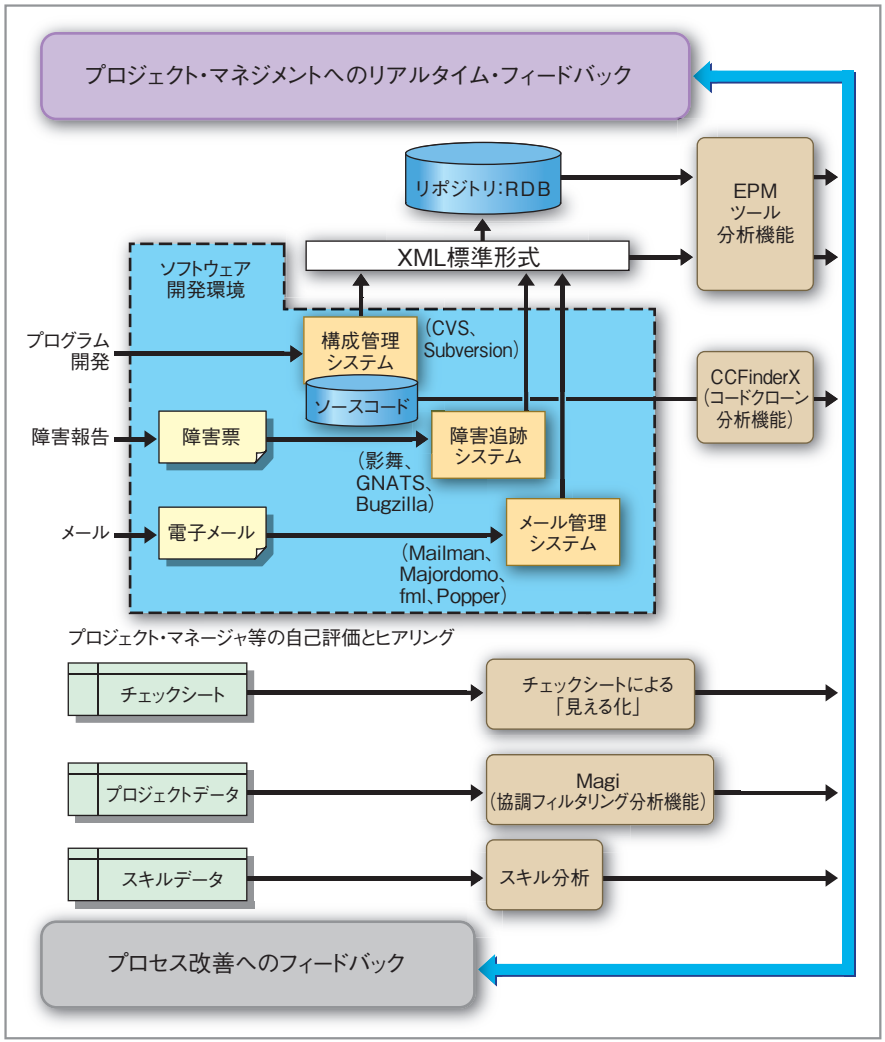

図3　エンピリカルソフトウェアエンジニアリングの実践

5 データ収集のポイント

ただ闇雲にデータを収集していたのでは無駄が多すぎます。データを収集する目的を明確にすることが重要になります。プロジェクトにおける問題点がある程度絞られていて、さらに原因を追究したいのであれば、問題に関係する詳細なデータを収集します。もし、プロジェクトの何処に問題があるのか良くわからない場合や、プロジェクトの異変をいち早く見出したい場合には、ある程度広範囲なデータを高い頻度で収集します。

収集したデータは蓄積してゆくことが重要です。できれば、同じデータをいくつものプロジェクトで、また、何年もに渡って収集しておくことで、プロジェクト間や過去のデータと比較することができます。特に、プロセスの改善効果を見たい場合には、改善前と改善後で同じデータを収集して、データがどう変化したかを確認することが重要です。

詳細な分析を行う上では、プロジェクトで利用している開発支援ツールの運用ルールを定める必要があります。狙ったデータが収集できるよう、ルールを工夫することが重要になります。

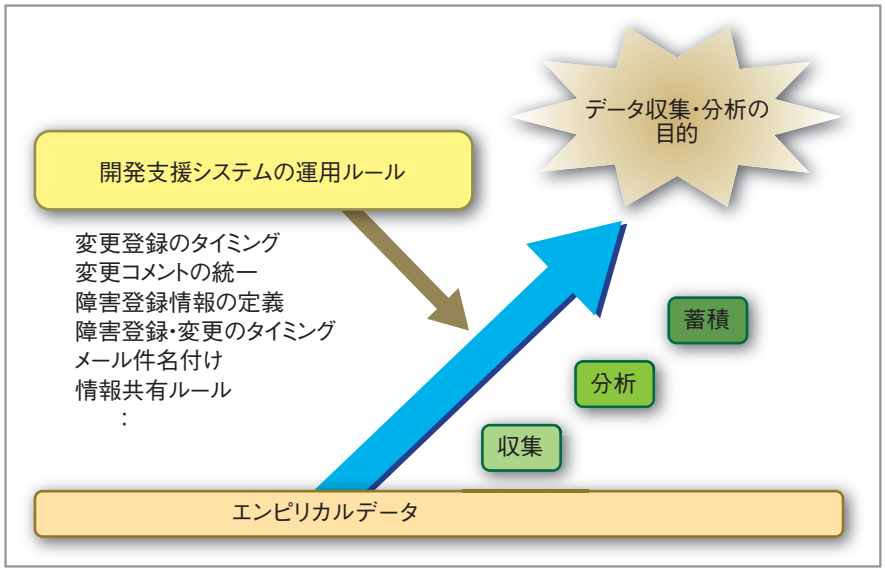

図４　エンピリカルデータの収集

6 EPM ツールのしくみ

EPM ツールは、Linux サーバ上で動作する「EPM サーバ」と Windows 上で動作する「EPM クライアント」で構成されています。

EPM ツールによるデータ収集と分析の環境を図５に示します。なお、下線は EPM ツール V1.0 の簡易インストーラでインストールされるものです。

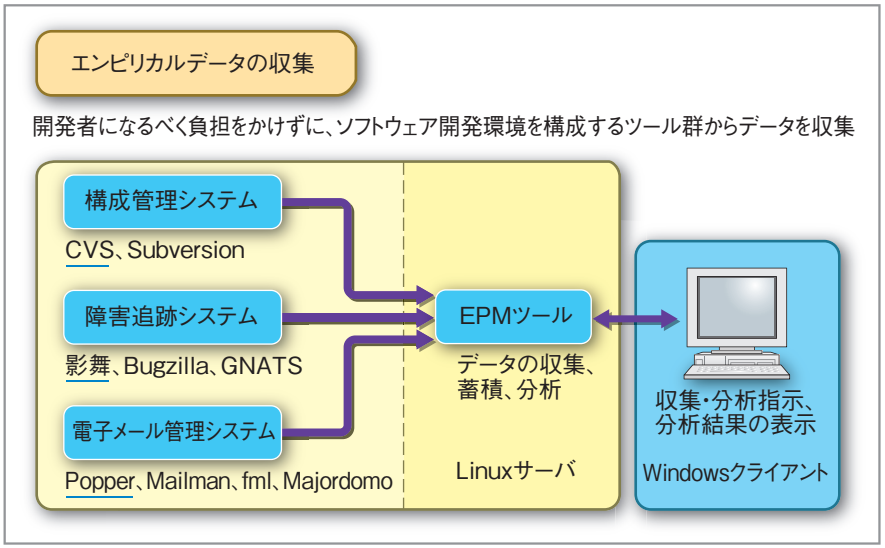

図５　EPM ツールの環境

EPM ツールは、ソフトウェア開発支援環境として利用されている構成管理システム、障害追跡システム、電子メール管理システムから情報を収集します。開発者はこれらのシステムを利用していれば、データ収集のための特別な入力を行う必要はありません。

データ収集の対象としているのは、表 1 に示すとおりですが、今後対象を広げてゆく予定です。

構成管理システム	CVS
	Subversion
障害追跡システム	影舞
	GNATS
	Bugzilla
電子メール管理システム	Popper
	Mailman
	fml
	Majordomo

表 1　EPM ツールのデータ収集対象

プロジェクト定量化の分類とエンピリカルソフトウェアエンジニアリング

プロジェクトの測定・定量化という分野で先駆者として知られるケイパー・ジョーンズは、その著書『ソフトウェア開発の定量化手法』(構造計画研究所刊) において、プロジェクトの定量化を次のように分類しています (解説は筆者による)。

①運用環境の測定
　ソフトウェア開発用コンピュータの延べ使用時間など
②開発中のプロジェクトの測定
　プロジェクトの計画とその達成状況、開発中のプログラムの行数、テスト中の障害発生件数など
③ソフトウェア資産とバックログの測定
　ソフトウェアへの投資額など
④顧客満足度の測定
　ユーザーへのインタビューやアンケート結果など
⑤完了プロジェクトの測定
　完了したプロジェクトのファンクションポイント数や開発稼働日数、工期など
⑥ソフトウェア要因の測定
　ソフトウェア開発プロジェクトに影響を与えるさまざまな要因。開発方法、開発ツール、メンバの技能、組織構成など
⑦ソフトウェアの障害の測定
　検出された障害の数、影響度・原因などで分類した障害数、開発中・開発後など期間別の障害検出数、障害除去に要した期間など
⑧企業内従業員統計の測定
　従事するエンジニアのスキル、人数など
⑨企業内の意識調査
　エンジニアあるいはソフトウェア開発にかかわる従業員のソフトウェア開発に対する意欲や不満、要望など

ソフトウェア開発における測定・定量化というとき、一般的には「完了プロジェクトの測定」を指すことが多く、完了プロジェクトのデータを収集してさまざまな分析が行われてきました。

しかし、本書で対象とするソフトウェアエンジニアリングでは、これまでの完了プロジェクトだけでなく、「開発中のプロジェクトの測定」も含めています。現場のプロジェクトマネージャやリーダーは、納期遅延や予算超過、品質不良など多くの問題を抱えています。エンピリカルソフトウェアエンジニアリングは、進行中のプロジェクトのさまざまな指標を計測することにより、プロジェクトマネージャやリーダーを直接支援することを目的としています。

7 データの分析例

収集したデータをわかりやすく「見える化」することが重要になります。EPM ツールにおける分析は、収集したデータの推移をグラフに表示することが特徴です。EPM ツールの主な分析機能としては、以下が用意されています。

①ソースコードの規模推移
②更新時期とチェックアウト数の関連
③累積・未解決障害件数および平均障害滞留時間
④更新と障害件数
⑤メール投稿数と更新時期
⑥パレート図
⑦クロス分析
⑧ロジカルカップリング分析
⑨SRGM（信頼度成長曲線）

①から⑤は日次や週次など定期的に分析を行い、必要に応じてさらに細かな周期で追跡します。⑥から⑨はある程度データを収集してから、詳細に分析を行う場合に利用します。

EPM ツールを使って収集したデータでどのような分析ができ、何がわかるのか、以下に例を示して説明します。

ソースコードの規模推移で大規模な修正を検出

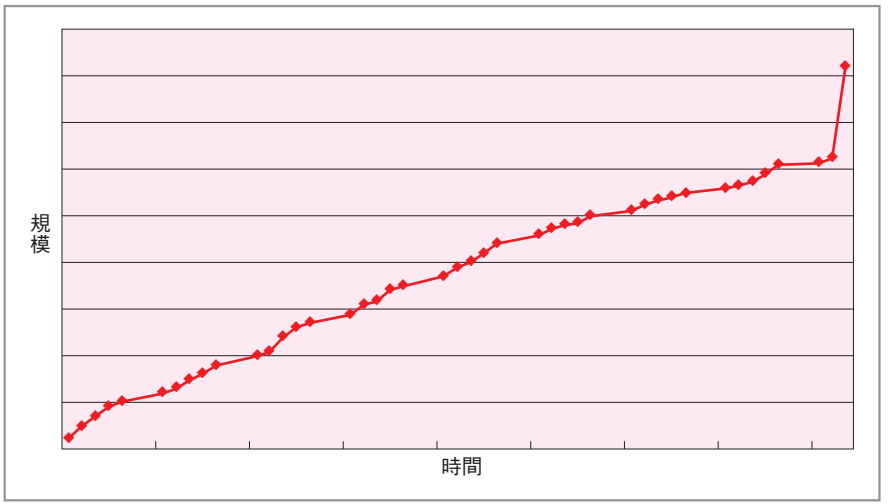

図６　ソースコードの規模推移（急増）

図６はソースコードの規模の変化を示したグラフで、急に折れ線グラフが立ち上がっています。これは多くのソースコードが追加されたことを意味しており、たとえば、障害を修正するときに、広範囲な修正か大規模な追加が行われた可能性を示しています。もし、広範囲な修正があったとしたら、修正したソースコードのレビューが確実に行われているか確認する必要があります。

あるいは、発注側からの強い意向があって、製造工程終盤に入ってから新たな機能を追加することになり、追加機能に対応するソースコードを登録した可能性も考えられます。

ソースコードの規模推移で試行錯誤を検出

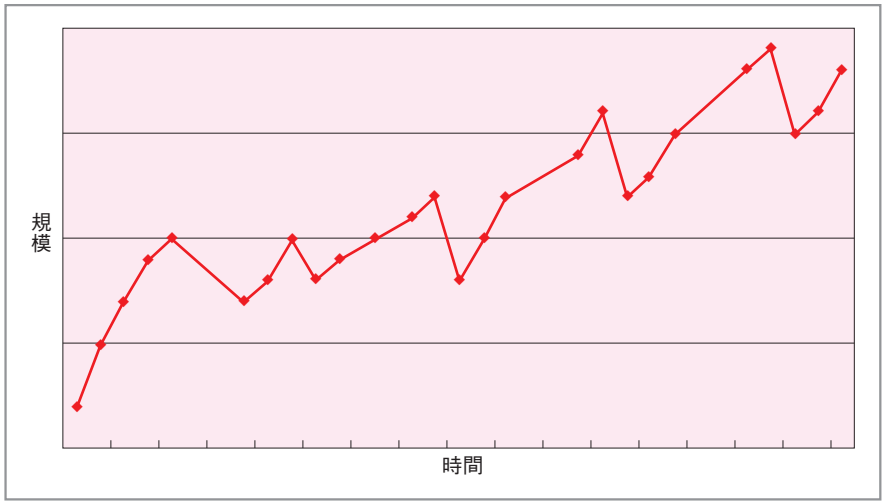

図７　ソースコードの規模推移（増減）

図７は、ソースコードの規模の変化を示したグラフで、増減を繰り返していることがわかります。これは、試行錯誤を繰り返しながらプログラミングを行っていることが予想されます。プログラム設計を実施してからコーディングを行えば、通常はこのような増減は起こりません。作り直しが繰り返されているのであれば、その原因を明確にして対策を講じる必要があります。

もし、プログラム設計を行っていない場合には、その原因を明確にして対策を講じます。プログラム設計が行えない事情がある場合には、コーディングの完了基準を明確にし、品質を確保するための対策を講じます。

ソースコードの規模推移で技術者の不調を検出

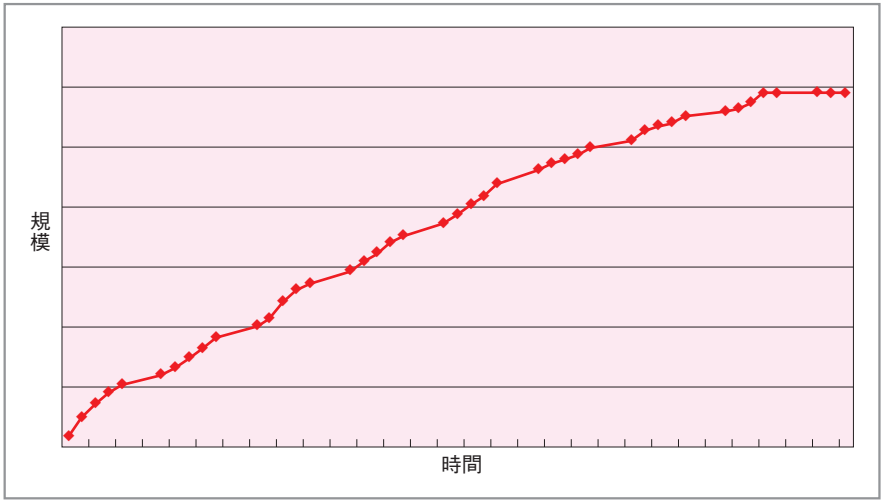

図８　ソースコードの規模推移（変化なし）

図８は、ソースコードの規模の変化を示すグラフで、しばらく規模の増減がないことを示しています。コーディング作業中にこのような状況が現れる場合は注意が必要です。担当の技術者が体調を崩して休んでいるかもしれません。何らかの理由で技術者のモチベーションが低くなり、作業が止まっているのかもしれません。あるいは、プログラム設計に重大な誤りが見つかり、対応を検討している最中である可能性もあります。いずれにせよ、原因を明確にし、適切に対処する必要があります。

コーディングが終了し、単体テストの準備中である、あるいは、単体テストで障害が検出されていないため、ソースコードの修正がないのであれば問題ありません。問題ないと判断できる根拠があれば対策は不要ですが、なぜソースコードの規模が変化していないかを把握しておく必要があります。

変更時期とチェックアウト数の関連でコミュニケーションミスを検出

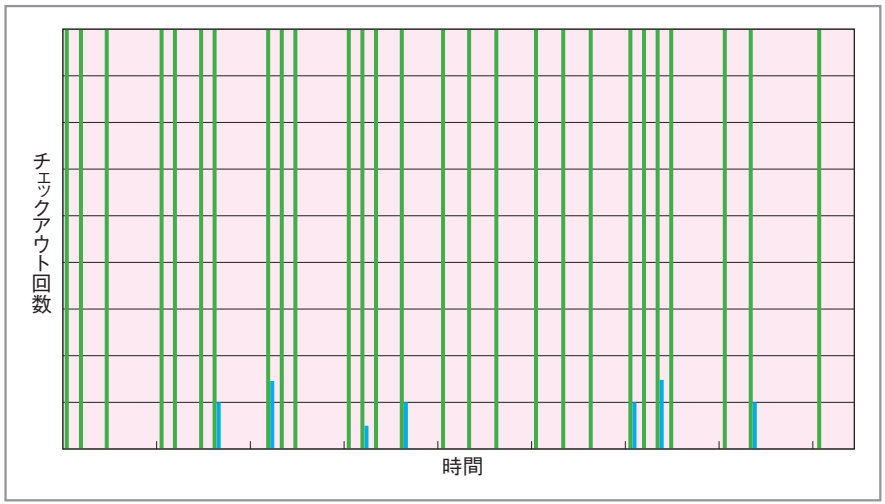

図 9　変更時期とチェックアウト数の関連（少ない）

図 9 は、構成管理システムとして CVS を使ったときの変更時期（縦の線）とチェックアウト数（棒グラフ）の関連を示したグラフです。

このプロジェクトでは、ソフトウェアを複数拠点で分散して開発して、相互に関連のある部分を開発していたとします。開発拠点が複数あっても、構成管理システムである CVS を一本化し、EPM ツールで一元管理していれば、ソースコードの作成状況や更新状況をひと目で見ることができます。

ソースコードに重要な更新が行われているにもかかわらず、関連する部分の開発拠点からのチェックアウトがなかった場合には、重要な更新が行われたことが周知されていない可能性があります。重要な更新が行われた場合の周知方法の確認と、それが実際に行われているかを確認する必要があります。

変更時期とチェックアウト数の関連で影響範囲の大きい修正を検出

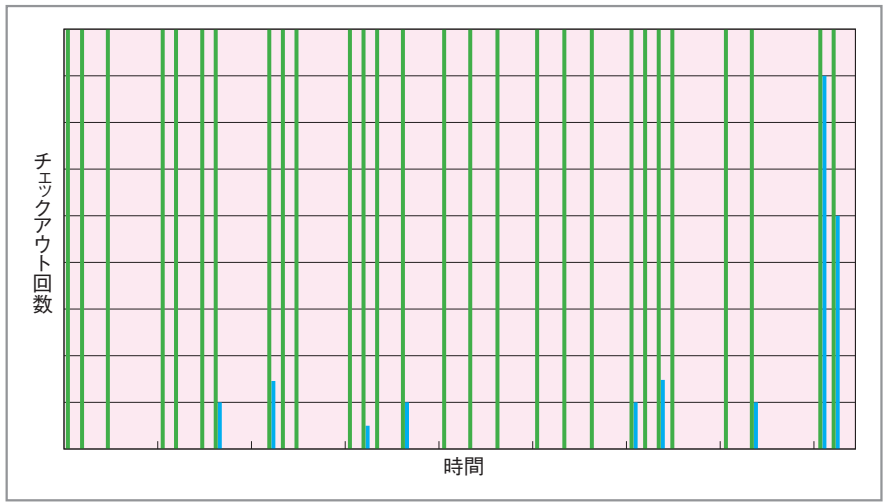

図 10　更新時期とチェックアウト数（急増）

図 10 は、構成管理システムへのソースコードの変更時期（縦の線）とチェックアウト数（棒グラフ）を示したグラフですが、急にチェックアウト数が増えています。

チェックアウト数が急増する直前の更新が、非常に影響範囲の広い修正で、この修正によって影響を受ける人がチェックアウトして修正内容や影響を確認しようとしている可能性があります。この場合には、登録された内容の影響範囲が明確になっていることと、すべての関係者にレビューされていることが重要です。

累積障害件数の推移で低品質モジュールを検出

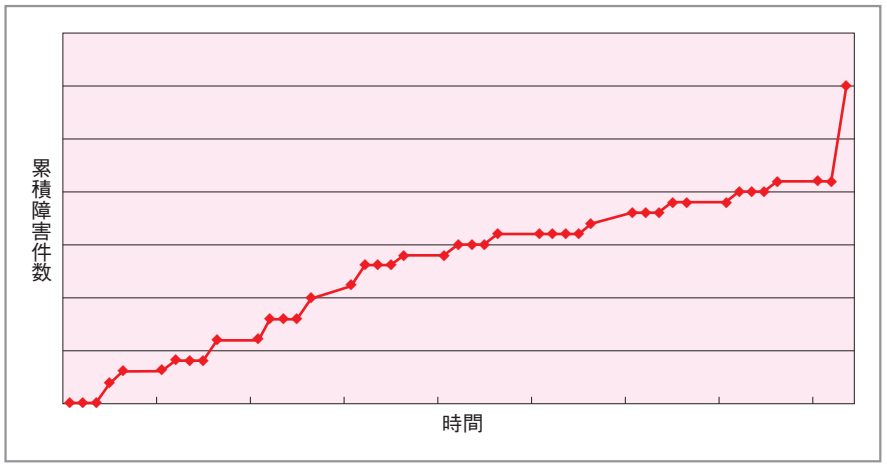

図 11　累積障害件数（急増）

図 11 は、累積障害件数の推移を示したもので、障害追跡システムに登録された障害件数の総数がどのように推移しているかを示しています、図 11 では急に件数が増えています。このとき新しいモジュールの試験が始まったとしたら、そのモジュールの品質が、他のモジュールより悪かったことが考えられます。モジュールを特定して品質を確認するとともに、品質を確保するための対策を実施します。

未解決障害件数の推移で障害解析工数の不足を検出

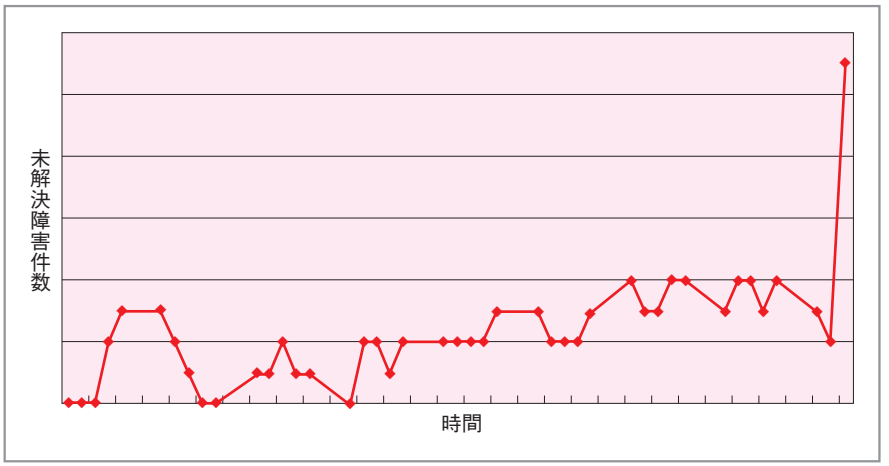

図 12　未解決障害件数（急増）

図 12 は、未解決障害件数の推移を示したものですが、未解決障害件数が急に増えているのがわかります。この場合は、障害を調査するための工数が十分に確保できていない可能性があります。未解決障害の数が少ない場合はそれほど問題にはなりませんが、ある程度数が増えてくると、調査に手を付けていない障害がどんどん増えてしまい、平均障害滞留時間も増加します。

調査を担当する技術者の役割分担を見直すか、障害調査のための要員を投入するなどの対策を実施する必要があります。

平均障害滞留時間の推移で解決が難しい障害を検出

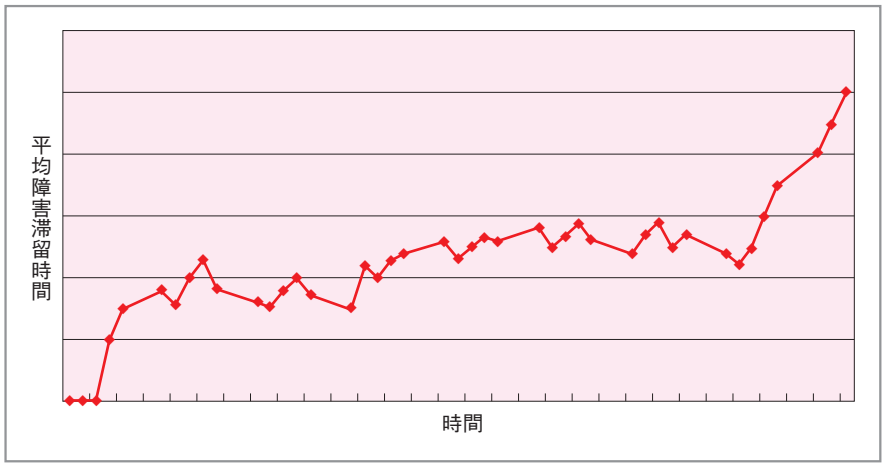

図 13　平均滞留時間（急増）

図 13 は、平均障害滞留時間の推移を示したものですが、急に増加しています。その原因として、解決までに時間がかかっている障害があることが推測されます。

未解決のままになっている障害がある場合には、解決できない理由を明確にし、対策を検討する会議を開催するなど、早期解決のための施策を実施する必要があります。特に担当者が 1 人で抱え込まないよう、調査を分担したり、有識者からアドバイスを受けるなど、解決に向けての体制を作ることが重要です。

変更時期と障害件数から修正が難しい障害を検出

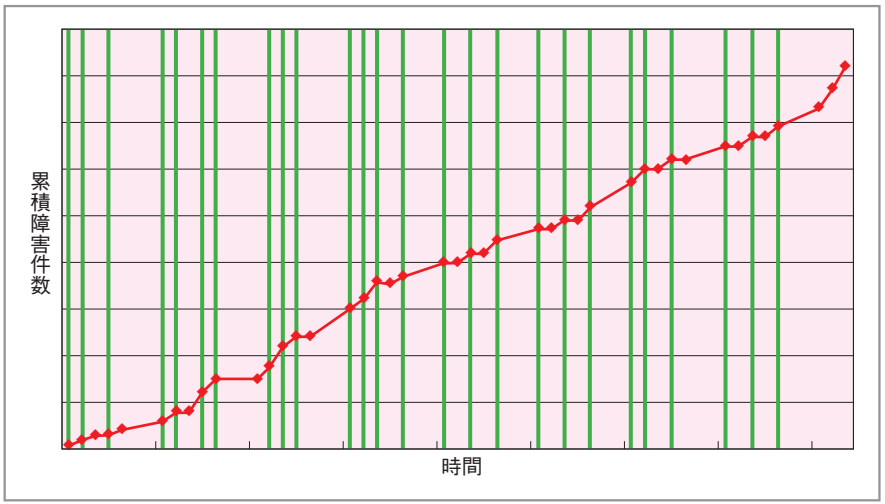

図 14　更新と障害件数（更新なし）

図 14 は構成管理システムへの変更時期（縦の線）と障害件数の累積（折れ線グラフ）を示すグラフですが、障害が増えているのにソースコードの変更が登録されていません。

障害を解決するためのソースコードの修正方法が見出せていない可能性があります。あるいは影響範囲が大きく、調査に時間がかかっていることも考えられます。いずれにしても原因を明確にして、対策を検討する必要があります。

もし、ソースコードを修正する必要がない障害（たとえば説明書の誤りなど）が多数検出されているのであれば問題ありません。

変更時期と障害件数で仕様変更を検出

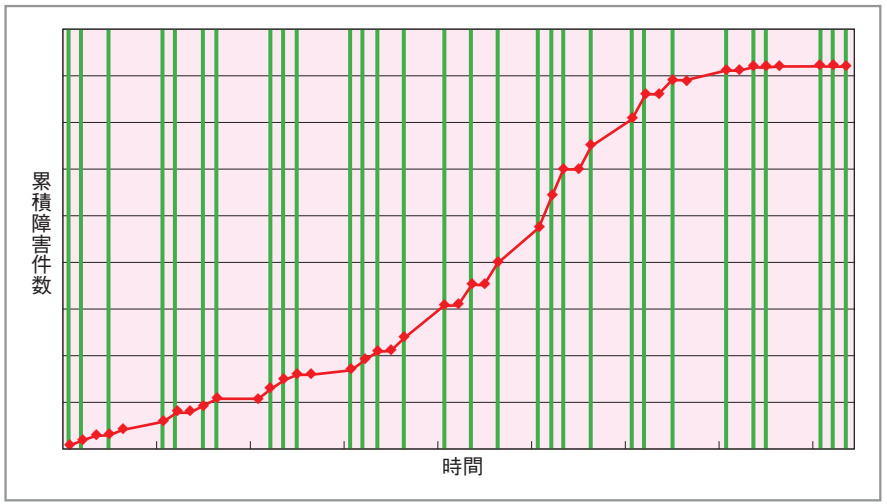

図 15　更新と障害件数（更新のみ）

図 15 は構成管理システムへの変更時期（縦の線）と障害件数の累積（折れ線グラフ）を示すグラフですが、障害件数は増加していないのに、ソースコードが変更されています。これは仕様の変更が行われたため、変更された仕様に合わせてソースコードを修正している可能性があります。この場合、仕様変更の手続きは正しく行われ、変更内容が管理されているかが重要になります。

未解決だった障害への対応のためにソースコードを変更している場合は問題ありません。未解決障害件数の推移と合わせて分析することが重要です。

メール投稿数と更新時期の関連で更新ミスを検出

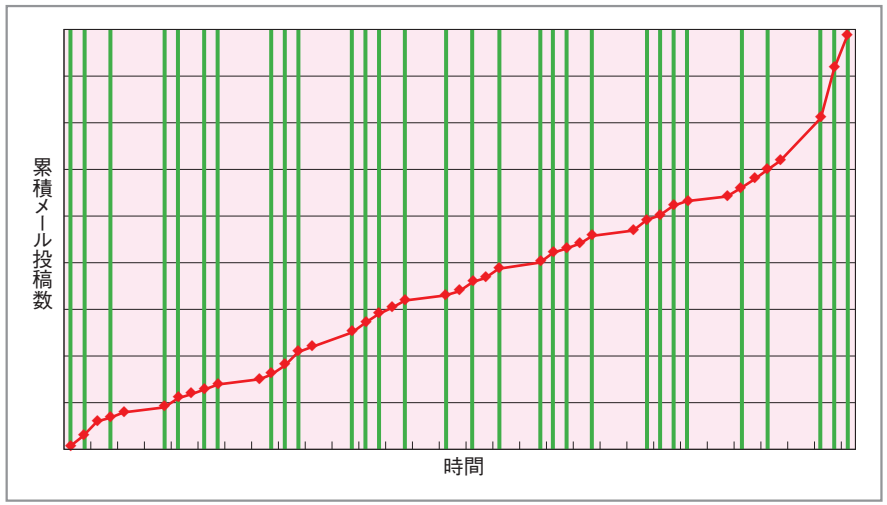

図 16　メール投稿数と更新時期の関連（メール急増）

図 16 は、メールの投稿数の推移（折れ線グラフ）と構成管理システムへの変更時期（縦の線）の関連をグラフに表示させたもので、メール投稿数が急激に増加しているのがわかります。メールが増加している時期にソースコードの変更も行われているので、ソース修正に問題があって、問題点を指摘するメールが多数配信されている可能性があります。この場合には、問題となっている修正を特定し、修正内容を早急に見直す必要があります。

パレート図で主要障害原因を判別

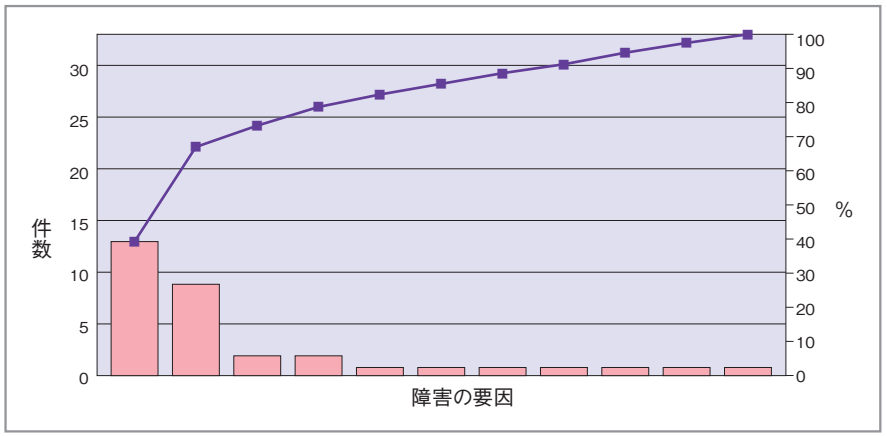

図 17　パレート図（障害原因）

図 17 は、障害原因をパレート図で表示させたもので、最初の 2 つが主要な原因であることがわかります。この 2 つに対して対策を講じることで、品質を向上させることができる可能性があります。

障害原因だけではなく、さまざまなデータに対してパレート図による分析を行うことで、それぞれの項目に対して主要な要素を見出すことができます。品質向上やプロセス改善に向けて取り組むべき対策を考える場合、主要な要素に対する対策を検討することで効率的に改善を目指すことができます。

クロス分析で障害摘出不足を検出

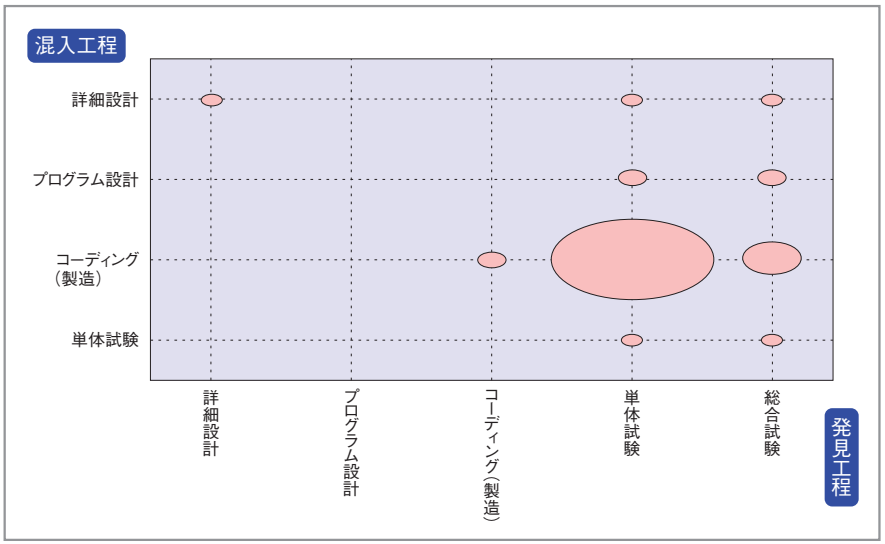

図 18　クロス分析（混入工程－発見工程）

図 18 は、障害追跡システムに登録された障害について、混入工程（障害が入り込んだ工程）と発見工程（障害が発見された工程）の 2 つの観点から分類したもので、コーディング工程で混入し、単体試験で発見された障害が最も多いことを示しています。

一般に、障害を発見する工程が混入した工程から遅くなるほど修正に要するコスト（工数）が増える傾向があります。

コーディングで混入した障害は、コーディング工程で検出するのが最も効率が良いのですが、図 18 の場合、コーディング工程での検出がほとんどありません。生産性の向上を考えると、コーディング工程での障害検出件数を増やすことが必要です。コーディングビューの方法や工程の完了基準を見直すなどの対策が必要です。

ソースコードの規模推移からわかる開発経緯

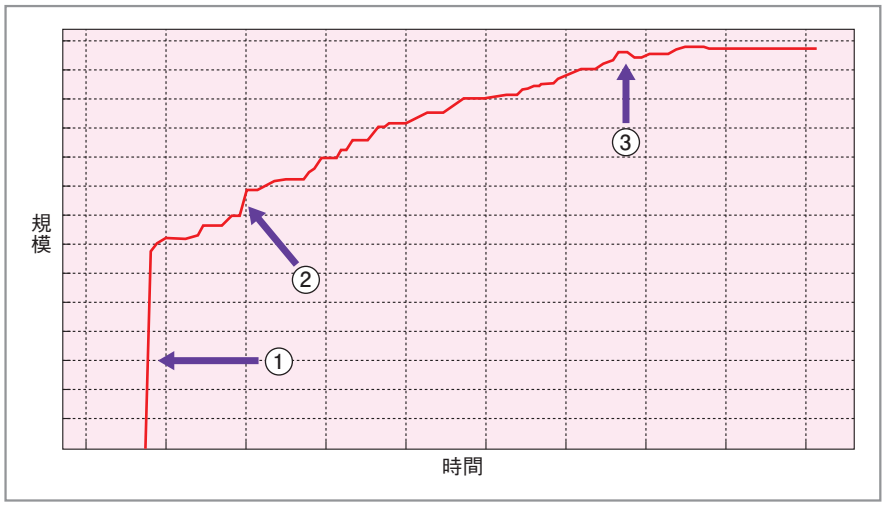

図 19　ソースコードの規模推移

図 19 は、ソースコードの規模推移を示したグラフで、開発作業は終了しています。開発作業が終了した時点で、グラフの変化と開発過程でどのようなことが起きていたかを関連付けることが重要です。

①では、ソースコード規模が大きく増えていますが、これはコーディング工程の最初にプログラム設計書に基づいて、主要な部分を一気にコーディングして構成管理システムに登録したためです。

その後、レビューを行っている途中で比較的大きな規模の増加が②で発生しています。これは、ある機能について、別の開発プロジェクトで開発されたコンポーネントの一部を流用することになっていて、その流用部分を登録したためです。

その後も、ソースコードレビューをしながらコーディング障害を修正していましたが、あるモジュールで他のモジュールと類似の処理を行っていることが判明し、ソースコードを共通化しました。そのため、③で全体のソースコード規模が若干少なくなっています。

②や③のような比較的小さな変化においても、何らかの理由があります。プロジェクト進行中は、小さな変化を見逃さずに、その変化の理由をさまざまな情報を総合して把握していくことが重要です。

このようなグラフの変化とそのとき起きていた事象の関連付けを蓄積していくことが重要です。新たなソフトウェア開発において、同じようなグラフの変化が現れた場合に、状況を推測する手がかりになります。

累積・未解決障害件数および平均障害滞留時間からわかる障害対応状況

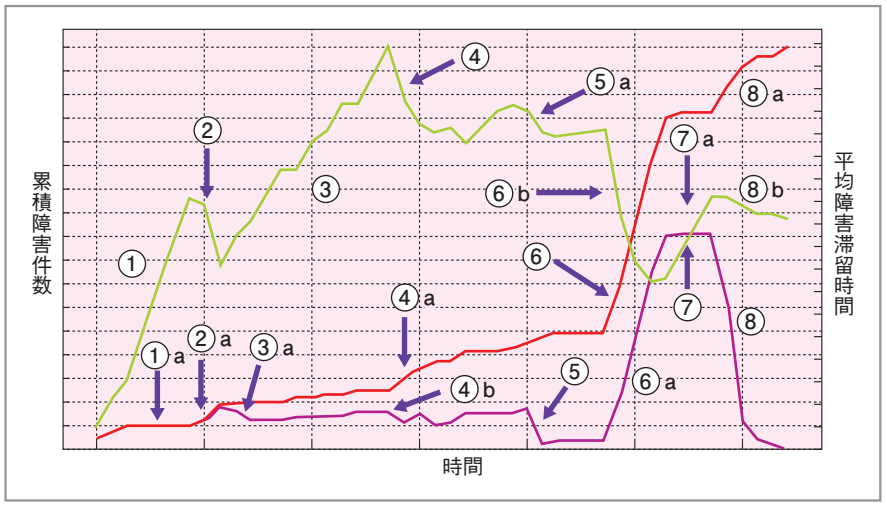

図 20　累積・未解決障害件数および平均障害滞留時間

図 20 は、累積障害件数、未解決障害件数、平均障害滞留時間の推移を示しています。この 3 つの折れ線グラフは互いに関係して変動します。それぞれの関係を理解しておくことは、障害対応状況を把握するうえで重要です。

①で平均障害滞留時間が増加しています。これは、検出された障害の調査に着手していなかったためですが、それは累積障害件数と未解決障害件数の線が重なっている（① a）のでわかります。

②で平均障害滞留時間が減っていますが、これは新たな障害が検出され（② a）、障害件数の母数が増えたためで、障害が解決されたからではありません。これも、累積障害件数と未解決障害件数の線が重なっているのでわかります。

③では平均障害滞留時間が増加しています。障害の一部を解決しています（③ a）が、それほど多くの障害を解決していないのと、どちらかというと新しく検出された障害を解決したため平均障害滞留時間は減少していません。

④では平均障害滞留時間が大幅に減少しています。これは新しく検出された障害があった（④ a）一方で、障害を解決した（④ b）ためです。

⑤では、未解決障害のほとんどを解決しています。しかし、平均障害滞留時間があまり減少していません（⑤ a）。これは古い障害（検出されてから時間が経っている障害）が解決されていないためです。

⑥では累積障害件数が大幅に増加しています。これは、総合試験工程に入って、新たな障害が多数検出されたためで、未解決障害件数が大幅に増加している（⑥ a）ことからもわかります。障害の全体数が急激に増えたため、平均障害滞留時間は大幅に減少しています（⑥ b）。

⑦では、未解決障害件数が横ばい状態になっています。これは総合試験工程での試験の実施が終了し、新たな障害検出はなくなったものの、障害の解決には至っていないためです。そのため、平均障害滞留時間は増加しています（⑦ a）。

⑧では、未解決であった障害を多数解決しています。その一方で、新たな障害も検出されています（⑧ a）。最終的に、平均障害滞留時間は若干減少して終了しています（⑧ b）。

EPM とオープンソースソフトウェア

EPM ツールは、原則としてオープンソースソフトウェアと組み合わせて使用されます。

構成管理システムは、オープンソースの「CVS（Concurrent Version System）」または「Subversion」、メール管理システムでは、「Mailman」「FML」「Majordomo」、障害管理システムでは、「GNATS」や「Bugzilla」「影舞」といった、それぞれオープンソースのソフトウェアと組み合わせて使用します。

また、トランスレータの開発にはオープンソースの「Ruby」を採用しています。さらに、インポータとアナライザは、オープンソースの Web アプリケーションサーバーである「Tomcat」上で Java サーブレットとして開発しています。

Web 表示用の Web サーバーとして採用している「Apache」、EPM が稼動する OS である「Linux」もオープンソースソフトウェアです。

8 プロジェクトマネジメントを支援

EPM ツールの分析結果をプロジェクトマネージャやプロジェクトリーダが確認することによって、プロジェクトの異変や問題点を把握できます。ソースコードの規模推移や累積・未解決障害件数および平均障害滞留時間などの分析結果は、即座に対応が必要になります。EPM ツールを有効に利用するには、プロジェクトマネージャがいつでも分析結果を確認できるようにしておくことが重要になります。

一方、障害に関する混入工程と発見工程とのクロス分析などは、分析対象となるデータがある程度の件数になってから確認することになります。多少の間隔をあけて確認しながら、前の工程でのレビュー不足が懸念される場合には、再レビューの実施なども考慮する必要があります。また、工程終

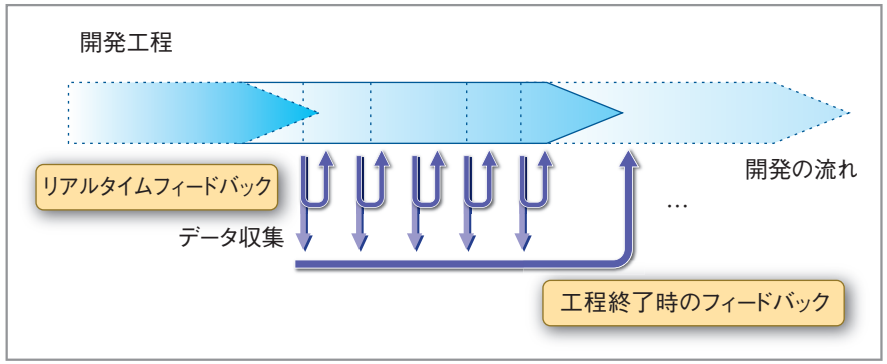

図 21　プロジェクトマネジメントへのフィードバック

了時には、さまざまな分析を行い、プロジェクトの状況を把握しておくことが重要です。

EPM ツールは、ソースコードの規模や障害の検出状況をリアルタイムに分析できます。進捗を把握するために、これらの情報を担当者にヒアリングする必要がなくなり、プロジェクトマネージャの負荷を減らすことができます。

また、複数のプロジェクトに対して EPM ツールを導入することによって、担当するプロジェクトマネージャが統一した視点で複数のプロジェクトをマネジメントすることを可能にします。また、品質管理部門や PMO（プロジェクト・マネジメント・オフィス）での活用も有効です。

EPM ツールは、プロジェクトの異変をリアルタイムに把握することができます。その異変を見逃さずに、適切に対応することによって、プロジェクトが失敗に陥らないようマネジメントすることが可能になります。

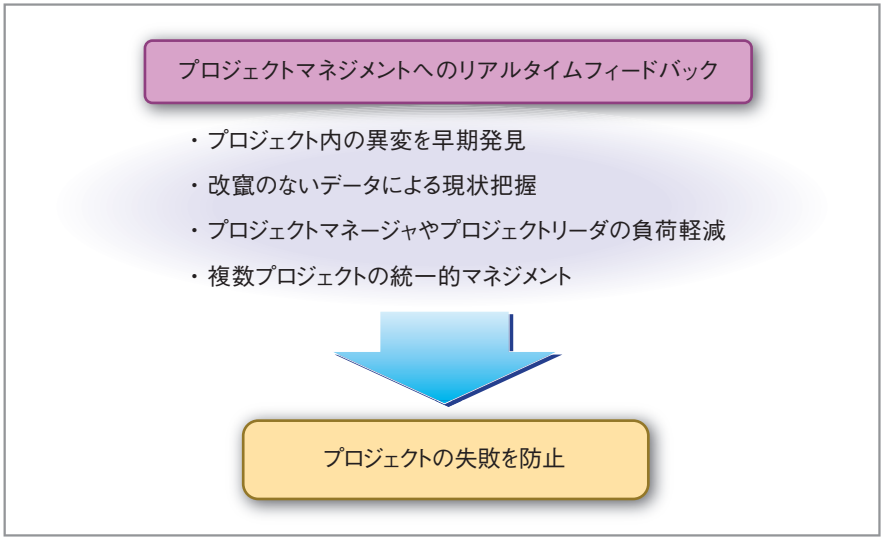

図 22　リアルタイムフィードバックの効果

9 プロセス改善に向けて

何かを改善しようとしたときには、まず己を知る必要があります。現状がどうなっているかを正確に把握することが出発点です。現状がわかってから、どこを改善するか、どう変えればよいかの議論ができます。

EPM ツールは、開発者に負担をかけずに、現状を定量的に把握します。EPM ツールで収集されたデータは、プロジェクトの経緯そのものです。EPM ツールによって収集されたデータを蓄積しておき、改善後のデータと比較することで改善の影響や効果がわかります。

同じデータを長年蓄積することで、経年変化を見ることができ、長期的な改善効果がわかります。

EPM ツールによる分析結果であるグラフのパターンの比較や収集されたデータの推移を比較して、プロセスの類似性を見出すことで、進行中のプロセスの今後の推移を推測することにも活用できます。

10 EASE プロジェクト

2003 年から開始された文部科学省のリーディングプロジェクト「e-Society：基盤ソフトウェアの総合開発」の中の「データ収集に基づくソフトウェア開発支援システム」をテーマにしたプロジェクトで、奈良先端科学技術大学院大学と大阪大学が中心となって活動しています。

EASE とは、Empirical Approach to Software Engineering（ソフトウェア工学へのエンピリカルアプローチ）の略で、ソフトウェア開発において定量的なデータに基づく科学的手法を適用することで、生産性や品質の向上を目指そうというものです。定期的にエンピリカルソフトウェア工学研究会を開催するなど、エンピリカルソフトウェアエンジニアリングの普及と実践を行っています。

EASE プロジェクトについて詳しくは以下を参照してください。

http://www.empirical.jp/

11
IPA の取組み

2004 年 10 月に設立された SEC における先進ソフトウェア開発プロジェクトで、エンピリカルソフトウェアエンジニアリングの実践が行われました。この詳細は『ソフトウェアエンジニアリングの実践』（2007 年、翔泳社）で紹介されています。先進ソフトウェア開発プロジェクトでは、EPM のほかにも、EASE プロジェクトで開発された協調フィルタリング分析ツール「Magi」や相関ルール分析ツール「NEEDLE」、大阪大学で開発された「CCFinder」（後に開発者である現独立行政法人 産業技術総合研究所の神谷年洋氏によってバージョンアップ版である「CCFinderX」が開発された）などの適用が行われました。

先進ソフトウェア開発プロジェクトにおいて EPM が有効に機能したことから、IPA は 2006 年から「ソフトウェア開発技法普及ツール開発事業」を展開し、「ソフトウェア開発プロジェクト可視化ツールのパッケージ化（EPM ツール）」を開始して、EPM の機能強化と普及活動を展開しています。

エンピリカルソフトウェアエンジニアリングの勧(すす)め

2007年10月11日　初版第1刷発行　PDF版

編　著　者　**独立行政法人 情報処理推進機構**
　　ソフトウェア・エンジニアリング・センター
発　行　人　佐々木幹夫
発　行　所　株式会社翔泳社（http://www.shoeisha.co.jp/）
印刷・製本　凸版印刷株式会社

ISBN978-4-7981-1406-4　　Printed in Japan