SEC BOOKS

ESQR Ver.1.1

改訂版

組込みソフトウェア開発向け
品質作り込みガイド

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

はじめに

ESQR Ver 1.1 発行にあたり

本書は、組込みシステムを対象とした品質作り込みを目指して、ソフトウェアの品質を計画・開発段階において定量的にコントロールするためのガイドとして「組込みソフトウェア開発向け 品質作り込みガイド」（英語名＝ ESQR : Embedded System development Quality Reference）として纏めたものです。

ESQRは2008年12月にVer. 1.0をリリースし、組込みソフトウェア開発に係る多くの方々に、ご支持いただき活用していただいてきました。本書は、このESQR Ver. 1.0をベースとして発行時に寄せられたパブリックコメントやその後のヒアリングでの指摘に対して、説明で分かりにくいところや一部誤りを補足・修正し、ESQR Ver. 1.1として発行するものです。

具体的には、

- 開発途中でのフィードバックの掛け方の補足
- 機能安全とのかかわりの明確化
- 参照値の算出方法について説明の詳細化
- プロジェクトプロファイル表の補正値のカスタマイズに関する補足

を中心に改訂しました。ESQR Ver. 1.0にも増して活用いただけると幸いです。皆様からのご意見・ご要望をお待ちしております。

なお、他に要望として、

- 実施率と充当率のような指標の使い分け基準など活用方法の説明
- レビュー指摘率など新しい品質評価指標の追加

を始めとする項目も挙げられていますが、これらはESQRの大幅な拡張に繋がり更なる情報収集と議論が必要になるため、改めて取り組むこととしました。

備考

ESQR Ver. 1.1は、IPA/SEC 組込み系プロジェクト高品質技術部会メンバーの協力によりESQR Ver. 1.0を精査し、見直したものです。

2011年7月

IPA/SEC 組込み系プロジェクト
三原幸博、十山圭介、石井正悟、濱田直樹

謝辞

ESQRの策定に当たっては、SEC研究員を中心として、数多くの企業、そして技術者の方々にヒアリングやインタビュー、あるいはアンケートをさせていただき、それらをもとに原案策定を行いました。また、この原案に対しては、SEC高品質技術部会を中心とする有識者の方々に、企業現場での活用の視点から様々なご意見やコメントをいただきました。改めて、ESQR策定にご協力いただいた皆様に感謝したいと思います。

また、このESQR策定の途中、我々の友人であり、ESQR策定の強力なメンバの1人であったSEC研究員・右近豊さんが亡くなられました。右近研究員には豊富な経験を活かし、ESQR策定のコンセプト作成以来、多くのアイデアや意見、コメントをいただきESQR策定の大きな力として協力いただいたことをここに記しておきます。

目次

第1章 品質作り込みガイドの読み方 1

1.1 品質作り込みガイドの目的と位置づけ 2
1.2 品質指標に基づく組込みシステム開発 5
1.3 本ガイドの想定利用者・利用方法と得られる効果 12
1.4 品質作り込みガイドの構成 16
1.5 本ガイドの利用に関する注意事項 18
1.6 関連規格 20

第2章 プロファイリングを利用した品質目標値の設定 23

2.1 組込みシステムの特性を考えた品質目標設定の考え方 24
2.2 Step-1：システムプロファイリング 28
2.3 Step-2：プロジェクトプロファイリング 34
2.4 Step-3：品質目標値の設定 37
2.5 プロファイリングの事例 45
2.6 システム障害の評価とシステムプロファイリングへの反映 48

第3章 品質指標の定義と参考値 51

3.1 品質指標の定義と意味、利用法 52
3.2 品質指標のカテゴライズ 54
3.3 品質指標 ― 利用上の注意 59
3.4 プロセス品質評価指標 ― 定義と参考値 63
3.5 プロダクト品質評価指標 ― 定義と参考値 79
3.6 基礎指標 ― 定義と参考値 105

第4章 高品質作り込みのためのヒント 135

4.1 開発におけるコミュニケーションと意思決定 136
4.2 ドキュメント 141
4.3 レビュー 147
4.4 テスト 152
4.5 指標を用いた品質作り込み活動 159

付録A 参考文献 168
付録B 高品質作り込みの事例 169
付録C ST-SEISMIC Scale（System Trouble SEISMIC Scale） 172
付録D フィードバックについて 174

あとがき 176

第1章

品質作り込みガイドの読み方

高品質な組込みソフトウェアを実現するためには、その開発過程での品質作り込みが大きなポイントとなります。本ガイド（ESQR：Embedded System development Quality Reference）は組込みシステムを念頭におき品質作り込みを行ううえで欠くことのできない、品質目標値の設定とそれを実現するための具体的な手法を整理したものです。本章ではESQRを初めて手にとる方を対象に、本ガイドの位置づけや利用方法などを紹介します。

1.1 品質作り込みガイドの目的と位置づけ……2
1.2 品質指標に基づく組込みシステム開発 ……5
1.3 本ガイドの想定利用者・利用方法と得られる効果……12
1.4 品質作り込みガイドの構成 ……16
1.5 本ガイドの利用に関する注意事項 ……18
1.6 関連規格 ……20

1.1 品質作り込みガイドの目的と位置づけ

品質作り込みガイドの目的

近年、組込みソフトウェアは需要の急拡大に伴い、その品質や信頼性・安全性などが重視されるようになってきています。従来より、高品質なソフトウェアを作るための手法として、レビューやテスト作業の充実、あるいはソフトウェア設計構造の最適化などさまざまな手法や考え方が紹介され活用されてきました。一方で、ソフトウェアの品質についても、品質の概念としてISO/IEC9126シリーズなどにみられるようにさまざまな定義やそれを測るための手法が提案されてきました。しかし、残念なことに、こうしたこれまでのソフトウェア品質をめぐる議論や手法の中で、「レビューやテストをどれだけやればよいのか」あるいは「このようなシステムであればどの程度の品質が求められ、そのためにどのような手法を利用すればよいのか」といった具体的かつ明確な指針が与えられてきたとは言いがたい状況が続いています。結果として、実際の開発現場では、ソフトウェア開発をめぐって、明確な品質目標を定めることが難しく、そのためにその目標値に近づけるための品質の定量的なコントロール（品質制御）という考え方が十分に実践されているとはいえません。この結果としてソフトウェア不具合やシステム障害が発生し続けています。

本ガイドは、こうしたソフトウェア開発における品質制御の状況を踏まえて、組込みソフトウェア開発の現場に、よりシステマティックな品質コントロール手法を導入するための手引きとして編纂したものです。その意味で本ガイドは「ソフトウェアの成果物・作業の品質を指標を用いて可視化し、その程度に応じて適切な開発手法の利用に道筋をつける」ことを大きな目的としています。

もう少し具体的に本ガイドの特徴を整理すると、

- 製品ごとにユーザから求められる品質をきちんと分析定義し、
- その結果をもとに品質指標を定義し、
- 各プロセスで得られる成果物と、成果物の確認作業の十分性を測り、
- その結果を品質確保のための活動に結び付けていく

ことを提案しています。その出発点には、「このくらいの品質」ということが抽象的・感覚的にではなく、指標を利用することで具体的な数値として表せるようにするというコンセプトがあります。

品質作り込みガイドの位置づけ

SECでは組込みソフトウェアの開発を円滑に進めるための参考資料としてESxRシリーズ(Embedded System development exemplar Reference)を刊行してきました。図1-1は、ESxRの各ガイドをV字開発モデル上に位置付けしたものです。このうち、組込みソフトウェア向け開発プロセスガイド：ESPR(Embedded System development Process Reference)は組込みソフトウェアの開発プロセスの標準的な考え方を整理したものです。また、組込みソフトウェア向けプロジェクトマネジメントガイド：ESMR(Embedded System development Management Reference)は組込みソフトウェア開発プロジェクトのプロジェクトマネジメントを円滑に進めることを目的に、その第1歩としての開発計画書立案方法を紹介したものです。また組込みソフトウェア開発向けコーディング作法ガイド：ESCR(Embedded System development Coding Reference)は組込みソフトウェアのプログラムの品質を直接的に向上させることを目的としてプログラムのコーディングに関するマナーを作法として整理したものです。

本ガイド(ESQR：Embedded System development Quality Reference)は、上記のESxRシリーズとの整合をとりつつ、ESPRで定めた開発プロセスを念頭に、それを実践する際に品質面での作り込み作業であるレビューやテストなどの十分性を評価したり、そこで作成されるさまざまな中間成果物の品質を評価するための指標やその読み方などを提示していきます。また、ESMRではプロジェクトマネジメントを行うベースラインとしての開発計画書を作成しますが、その際に品質面での計画を立てる必要があります。そこで利用する具体的な品質指標やその目標値の設定方法について、本ガイドで詳細に解説していると理解していただくとよいかと思います。なお、本ガイドの本文中でソフトウェア開発のプロセスとして引用している単体テスト、要求分析などの作業プロセスやドキュメントなどの中間成果物名称は基本的にESPRで定義したものを利用しています。必要に応じてESPRも参照いただければと思います。

また、本文中にはソースコードの品質評価に関する仕様が含まれていますが、コーディングルール逸脱などの考え方については、ESCRに掲載したコーディング作法なども参考にしていただけるとよいかと思います。

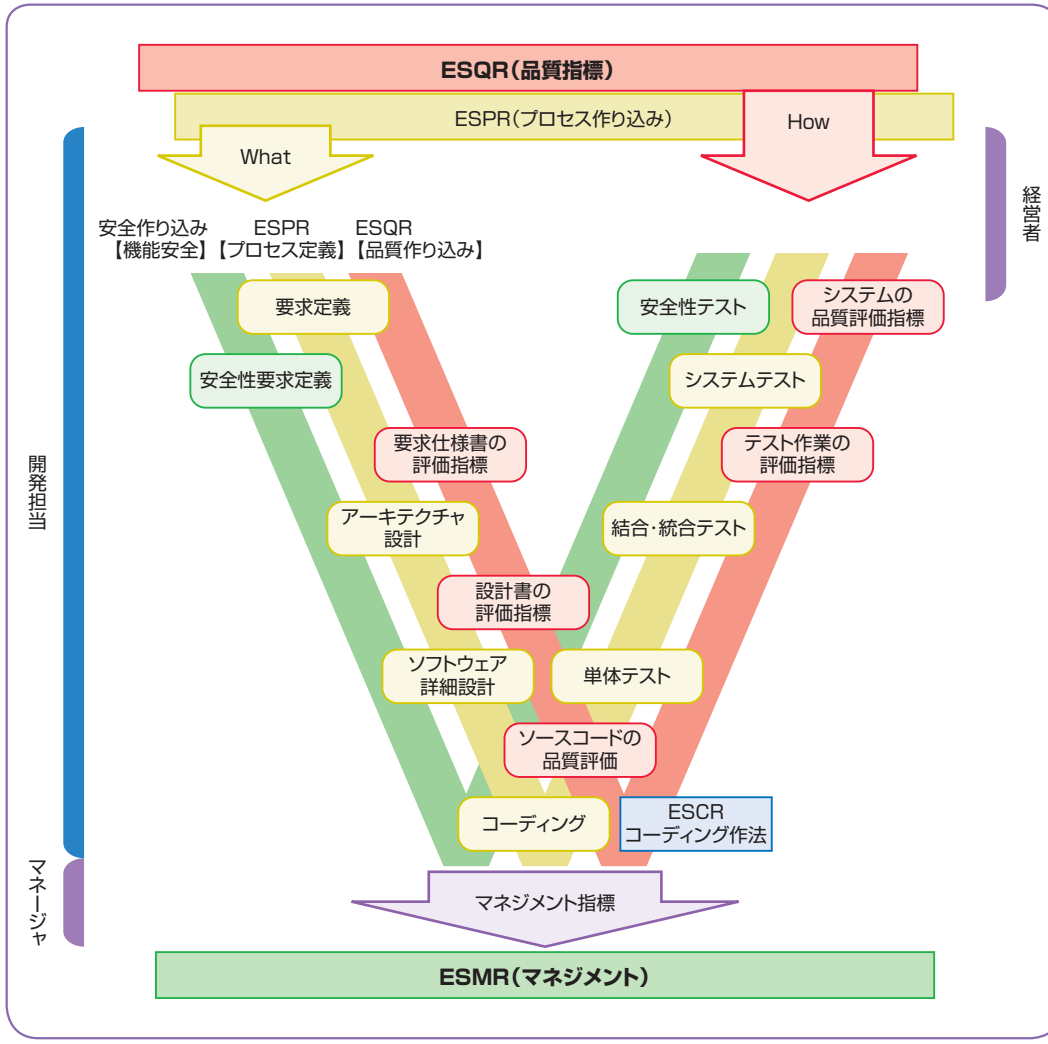


図1-1：ESQRの位置づけ

1.2 品質指標に基づく組込みシステム開発

本ガイドでは「品質指標を用いたソフトウェア品質の定量目標設定とそれに基づく品質コントロール」が中心的な概念になっています。ここではまず、この中心的な概念である品質指標について簡単に説明しておきます。

指標とは

我々の身の回りには実にたくさんの“もの”が存在しています。我々は常日頃から、こうしたさまざまな“もの”について、それらが持つ特質について言及することが少なくありません。たとえば、「りんご」というものを考えてみると、「りんご」がもつ特質—たとえば、重さであり、大きさ、色などさまざまな性質—を感覚的にとらえ、その特質を評価しています。ところが「大きいりんご」を考えた場合、大きい小さいはそれを受けとる人の感覚に左右されてしまいます。たとえば、商品として大きさを基準にりんごの価格を決める場合、こうした人の感覚に頼った大きさの評価はきわめて曖昧で、ある意味、あまり参考にならないかもしれません。このため、一般的には「りんご」の直径を測り、数値として表現することで、個人的な感覚の差異をなくし、より科学的に“もの”のもつ特質を表現することが行われています。そして、りんごの直径がある値より大きいものを大玉、ある値からある値までのものを中玉などといった具合に、直径と基準値を利用してりんごのクラス分けをしたりします。この場合、「りんごの直径」は「りんご」という“もの”の大きさを評価するための指標ということになります。このように、世の中に存在するどのようなものも、それぞれの特質をもっており、それを測り評価するための指標が存在するということになります。

さて、もう一つの例として、たとえば、「口座開設のため銀行の窓口に行った際に、どれくらい待たされるか」といった例があります。この場合の評価の対象は“もの”ではありません。しかし、我々はA銀行はやたら待たされてサービスが悪い、とか、B銀行は比較的待たされずにサービスが良いなどと比較をしたりします。この場合、我々は知らず知らずのうちに、「銀行での口座開設手続き」という形のないサービスあるいは作業の質を比較していることになります。この場合も、直感的にサービスが良い、悪いは人によって感じ方が異なります。このため、サービスの質などを評

価するためには、より客観的な指標、たとえば「平均の待ち時間」といった数値で測れる指標を用いて評価することになります。このように、指標は作業のように対象が無形のものに対しても、その特質を測る手段として利用される場合があります。

このように考えてくると、指標とは以下のようなものであることがわかるかと思います。

> **指標**：有形、無形にかかわらず対象（物、作業、サービスなど）の特質を表現するための科学的な根拠をもったものさし

一般にどのような対象であれ、その対象の特質がどのようなもの・状態であるかを確認するためには、指標を用いて対象の何かを測り、その目標値あるいは経験値と比べてどうか、ということでその対象の特質を判断することが必要となります。

指標の測り方と単位・精度

先に示した「りんごの直径」という指標をもう少し考えてみましょう。「りんごの直径」といっても、図1-2に示すようにりんごのどの部分を測るかで変わってきます。たとえば、りんごの球体の中心を通る面で切ったところで直径を測るのか、もう少し上側の面で切ったところの直径を測るのかで、りんごの直径の値は変わってしまいます。このように測り方によって指標の値が変わってしまっては意味がありません。このように指標はその測り方とセットにして定義を与えておかないと、意味のないものになってしまいます。

また、りんごを測るときに、どのような粗さのものさしを利用するかもまた考えておく必要があります。すなわち、最小目盛が1センチメートルのものさしで直径を測れば、おおよそ10センチメートルとか、7センチメートルといった値が得られます。これに対し、最小目盛が0.1ミリメートルのものさしを用意し、りんごの直径が7.13センチメートルとか7.14センチメートルと騒いでみたところで、0.01センチメートルの差はあまり意味をもちません。この例からわかるように、指標で測る対象の特質に応じた精度を考えて測る必要があるといえます。

さらに、長さの単位を考えると、センチメートル、ミリメートルなどのメートル法のものもありますが、インチといった単位もあります。りんごの直径について考えると、特にメートル法でなければいけないといったことはありませんが、直径を測る人によって、センチメートルだったり、インチだったりと単位が異なってしまうと、りんごの直径を比較したりする場合に混乱が起きてしまいます。

ここまでの例をもとに指標の測り方と単位・精度などを整理すると次のようになり

ます。

指標：指標の計測方法も合わせて定義し、誰が計測しても同じように値を計測できること

指標の精度：測ろうとしている対象の特性を表現し得る適切な精度を意識して測ること

指標の単位：指標値を表現するための単位系は、指標定義と合わせて、明確に一つに決めておくこと

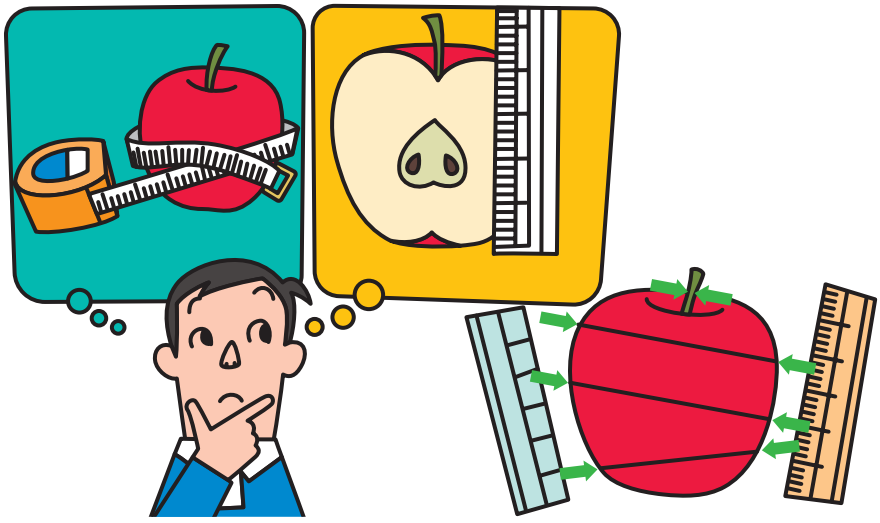


図1-2：指標と計測の意味あい

指標の利用方法

指標は何のために決め、測るのでしょうか？「りんごの直径」を考えてみると、たとえば、「りんごの直径」が基準値よりも大きければ、「より良いりんごとして出荷してもよさそうだ」という判断をすることができるかもしれません。一方、極端に基準値を下回ったりんごは、発育不良という判断をして、その場で収穫をせずにもう少し木にぶら下げておいて大きくなるのを待つ、といった判断をすることもできます。このように、実際に計測された指標は、たとえば収穫や出荷などの判断に使うといった利用法が考えられます。さてここでもう一つ別の指標「りんごの軸（果柄）の

太さ」を考えてみましょう。軸の太さの基準値を2ミリメートルと考え、いくつかのりんごをそれより太いもの、細いものに分けてみましょう。「りんごの軸の太さ」に応じて2つに分けたグループに何か意味はあるでしょうか？　確かに「りんごの軸の太さ」を測ることはできますが、それの太い・細いは商品としてのりんごの価値にはほとんど影響なさそうです。この例のように、指標の対象が有する特質はさまざまであり、それぞれ指標を測ることは可能です。しかし、中には測れるからといって計測しても、ほとんど意味を持たないであろう指標があります。一般的に、「指標を計測する」ということは、測り評価することでその結果を何かに利用することが前提になります。すなわち、指標を決める場合には、まず、「指標を用いて何を言いたいのか、あるいは指標を何に利用したいのか」を明確にしたうえで、それに適した指標を選ぶことがきわめて重要になります。

品質指標とは

本ガイドはより高い品質の組込みシステムを開発することを目的として、対象システムが持つさまざまな特質を品質指標により定量化しコントロールしていく方法を紹介することを目的としています。

すでにソフトウェアも含めて、世の中一般に存在する“もの”や“作業”の特質を表現する手段としての「指標」の概念は説明しましたが、本ガイドでは、さらにそれを推し進めて、ソフトウェアがもつ品質という特質を表現し、その特質をコントロールする手段として、品質指標を導入していきます。

先の例にひいた「りんご」は実際に我々が手にとって観察し計測することができる実体を有しています。これに対し本ガイドが対象とする組込みソフトウェアは我々が直接手にとったり観察したりということがきわめて難しい対象物であるといえます。すなわち、さまざまな製品に搭載される組込みソフトウェアをその製品の中から分離して手にとることはできませんし、開発の現場においても、ソフトウェアはソースコードや設計書といったさまざまな中間成果物の形を借りてのみ、その対象物としてとらえることができるものです。その意味で、ソフトウェアを指標の計測対象と考えた場合には、きわめてとらえづらい対象であるということがいえます。

しかし、その一方で、ちまたに出回っているさまざまな組込みシステムでは、不具合が発生したり、なんとなく使い勝手が悪かったりということもあり、我々はそれを称して「品質が低い」とクレームを訴える場合も少なくありません。この例でわかるように、ソフトウェアは計測の対象としてはきわめてあやふやなものではありますが、他の有形物と同様に「品質」という概念を合わせ持っていると考えることができ

ます。その意味から、本ガイドでは、ソフトウェアを対象として、その品質を計測し表現する手段としての品質指標を明確にしていきます。

また、こうしたソフトウェアに関して、その品質の良し悪しはなぜ発生するかといった、より本質的なところを検討し、コントロールしていくことが重要です。たとえば直径の大きな「品質の良いりんご」を作ろうとした場合、結果としてできたりんごのみを一所懸命、指標で測っても埒があきません。「品質の良いりんご」を手に入れるためには、その育成段階で「どれだけ陽にあたったか」「雨がどれくらい降ったか」あるいは「肥料をどれくらい施したか」などりんごを手に入れるための過程や(農)作業が適切であったかどうかを測り、問題があれば「より多くの肥料を施す」などの工夫をしなければなりません。ソフトウェアの場合、最終品質がきわめてとらえづらいため、こうした考え方はより重要になります。すなわち結果としてのソフトウェアの品質を測るだけではなく、その開発の過程でどれだけ手間隙を掛けたか、あるいは、どれだけ必要な作業をしているかといった点に注目して評価していく必要があります。このように考えてくると、品質指標は下記の2つの概念に整理することができます。

結果を得る過程で実施した作業の適切さを測る指標　⇒　プロセス品質評価指標
結果として得られるソフトウェアの品質を測る指標　⇒　プロダクト品質評価指標

プロセス品質評価指標とプロダクト品質評価指標

上記に述べたように、本ガイドでは組込みシステムの品質という特質を計測し評価するための指標として、プロセス品質評価指標とプロダクト品質評価指標の2種類の指標概念を導入していきます。

①プロセス品質評価指標

ソフトウェアを開発していくにあたって各段階で、「どのような作業を実施していくか」については、「組込みソフトウェア向け 開発プロセスガイド(ESPR)」で示しています。本ガイドでは、その各プロセスでの作業を着実に行っているかどうかを確認するためのものさしについて整理していきます。ソフトウェア開発にかかわる作業はさまざまなものがありますが、本ガイドでは特にソフトウェアの品質を担保する作業に着目し、品質の作り込み、テストの両面から品質を確保、保証する作業が十分に実施されているかどうかを評価するための指標を考えていきます。

一般的にソフトウェアを作るという作業は、下記に示したように「きちんと作り、

確認し、直す」という3つの作業要素の繰り返しと考えることができます。

- **きちんと作る**：要求分析する、構造を設計する、プログラムを実装する
- **確認する**：チェックする（レビューなど）、動作確認をする（テストなど）
- **直す**：デバッグする、設計やドキュメントを修正する

このうち「きちんと作る、直す」という2つの作業要素についてはすでにソフトウェア工学の技術としてさまざまに語られています。本ガイドでは「チェックする、動作確認をする」という2つめの作業要素に焦点を当てています。

②プロダクト品質評価指標

工業製品に用いられるソフトウェアを開発する場合には、開発の途中段階で作成されるさまざまな成果物を適切なタイミングで計測評価し、目標とする品質に合わせこんでいく（コントロールしていく）活動がきわめて重要になります。本ガイドでは、開発作業の結果として得られる成果物を測るための指標としてプロダクト品質評価指標を整備していきます。

先にも述べたように、ソフトウェアという“もの”はある意味、形があるようでないような、とらえどころのないものといえます。結果的にものとしてのソフトウェアの品質を測る場合には、その開発過程で作られたさまざまな中間成果物－仕様書、設計書、ソースコード、実行コード－などを直接的に測って、ものとしての品質－プロダクト品質－を測ることとなります。

品質指標と参考値の考え方

本ガイドの基本的な考え方として、組込みシステムの開発は図1-3に示すように、要求定義から始まり、設計、コーディング、それらのテストを行うプロセスをたどりますが、本ガイドの3章ではこれら各工程の全体を通して、段階的に品質を確保するための指標を提示しています。これらの指標は、その定義や測り方の情報、また計測された値の解釈の指針なども合わせて示しています。本ガイドの3章以降をご覧いただくとわかるように、本ガイドで採用した品質指標の多くは、なるべく簡便に測れ、かつ、誰でも同じ結果が得られるものを提案しています。また、本ガイドでは、これらの品質指標（定義、測り方、解釈）とともに、計測した指標値の妥当性の目安とするために、対象とするシステムの品質レベルなどを考慮した参考値なども合わせて提示しています。この参考値については、ウォーターフォール型の工程をベースに、新規開発の場合を想定しての参考値を提示しています。実際の組込みシステ

ム開発の現場では、このような全くの新規にシステムを開発する場合のみだけでなく、既存のソフトウェア資産を流用あるいは再利用しながらシステムを開発する場合も少なくありません。こうした場合には、本ガイドで提示する指標や参考値、考え方などを参考に、個々の開発の条件などを考慮した読み替えなどをしていただけるとよいかと思います。なお、本ガイドで示した指標の参考値は、SECの活動に賛同いただいた皆様からいただいたデータや議論を経て算定したものであり、その利用は参考値とし、あくまでも目安として、それぞれのプロダクトでカスタマイズして使用していただくことを想定しています。

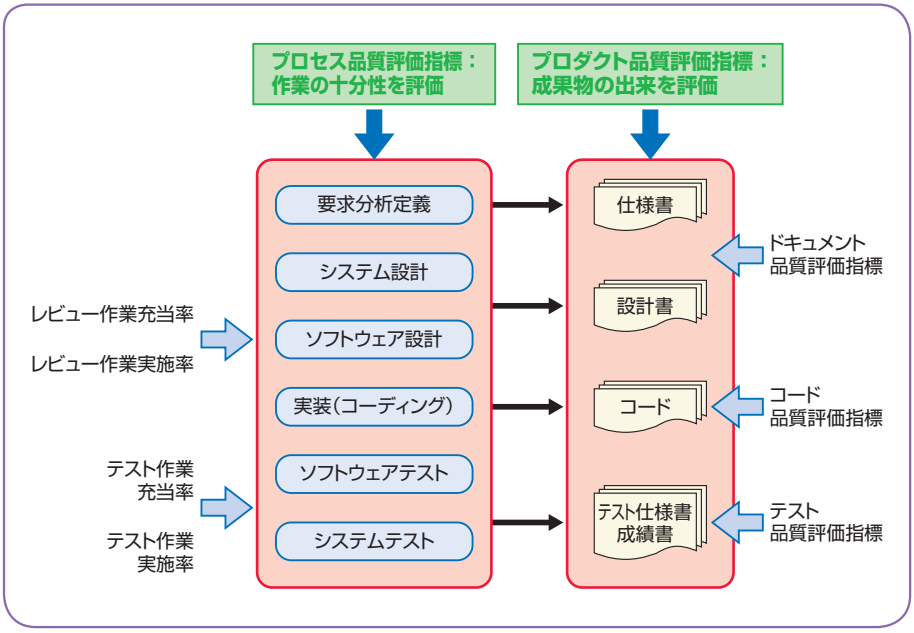


図1-3：プロセス品質評価指標、プロダクト品質評価指標の位置づけ

1.3 本ガイドの想定利用者・利用方法と得られる効果

想定する開発と利用者

想定する開発

本ガイドはあらゆる組込みソフトウェアの開発を対象として利用していただくことを想定しています。実際の開発現場では、

- 機器に搭載されるさまざまな組込みシステムの開発において、全くの新規からソフトウェアを開発する場合
- 既存の設計資産などを流用や再利用してソフトウェアを開発したり、その一部機能を拡張する場合
- 他社からの求めに応じて、あるいは自社のビジネスとして、組込みソフトウェアを実現するための一部の（汎用）機能やミドルウェアなどを開発する場合

などが考えられます。これらのソフトウェア開発は形態やその開発プロセスなどにおいてさまざまなパターンが存在しますが、作られるソフトウェアの品質をその過程において定量的にコントロールしておくという基本的な考え方は同じと考えることができます。このため、本ガイドに記載した考え方や参考指標などをもとに、それぞれの開発条件などを加味し工夫することで適用していただくことができると考えています。また、本ガイドに記載した考え方は基本的に組込みソフトウェアの開発を念頭に整備したものですが、組込みソフトウェア以外のソフトウェア（たとえば企業情報システムを構成するソフトウェアなど）への適用に関しても、多少の読み替えによって適用が可能になると考えています。

想定する利用者

本ガイドは、企業やプロジェクトでソフトウェア開発を行う人および、品質管理を行う人、マネジメントを行う人すべてを対象とした、品質指標を定義し使用するためのガイドです。

本ガイドは、組込みソフトウェアの開発における次の利用者を想定しています。

- 開発プロジェクトや開発組織などをマネジメントし、個々の開発案件において実際の工程管理や品質管理を検討・決定するマネージャやリーダ
- 組込みソフトウェアを開発する組織において、組織や部門の開発プロセスの標準や品質に関する基本的な考え方を整備し、その運用を支援するメンバ
- 組込みソフトウェアを開発する組織において、品質保証など、ソフトウェア開発を間接的に支える支援グループのメンバ

もちろん、このように定量的品質指標を活用してシステムの品質をコントロールする場合には工数をかける等のコストが発生しますので、経営者層の理解・支援やバックアップが必要であることはいうまでもありません。

想定する利用方法

本ガイドは、組込みソフトウェア開発を進める際に品質を担保していくうえで必要な指標や品質向上につながると考えられる作業の進め方などを整理したものです。本ガイドは個々の組織や部門、あるいは個々のプロジェクトに適した品質管理を整備する際に利用することを想定しています。

品質管理技術の整備に関しては下記のような利用シーンが考えられます。

- 製品の特性を分析し、その製品に即した品質目標を定める
- ソフトウェアの工程移行判定の目安を求める
- ソフトウェアの品質管理のための計測等が行われていない場面で、新たに定量的な品質指標を導入し、積極的な品質コントロール（管理）を行う

また、本ガイドに記載した内容の利用手順に関しては図1-4に示すように、自組織、自部門の事業戦略、製品戦略や組織特性などを十分に考慮する形で、システムプロファイリングやプロジェクトプロファイリングを行い、その結果を開発における品質目標に反映させる手順をとります。また、実際に設定した品質指標を計測し、その結果と目標値のギャップ分析に従い、プロジェクトを適切な方向に誘導すべく、必要な手法や技法の導入を進めていきます。このうち本ガイドでは、システムプロファイリングやプロジェクトプロファイリングの方法を2章で、個々の品質指標を3章に紹介しています。また、計測結果をもとにした開発やシステム品質のコントロールに関して、本ガイド4章ではレビューやテストなどの作業に関して進めるうえでのヒントなどをチェックリストの形で整理しています。なお高品質を実現するための設計手法、実装手法などの直接的な手法・技法の紹介については他の書籍などを参考にしていただければと思います。

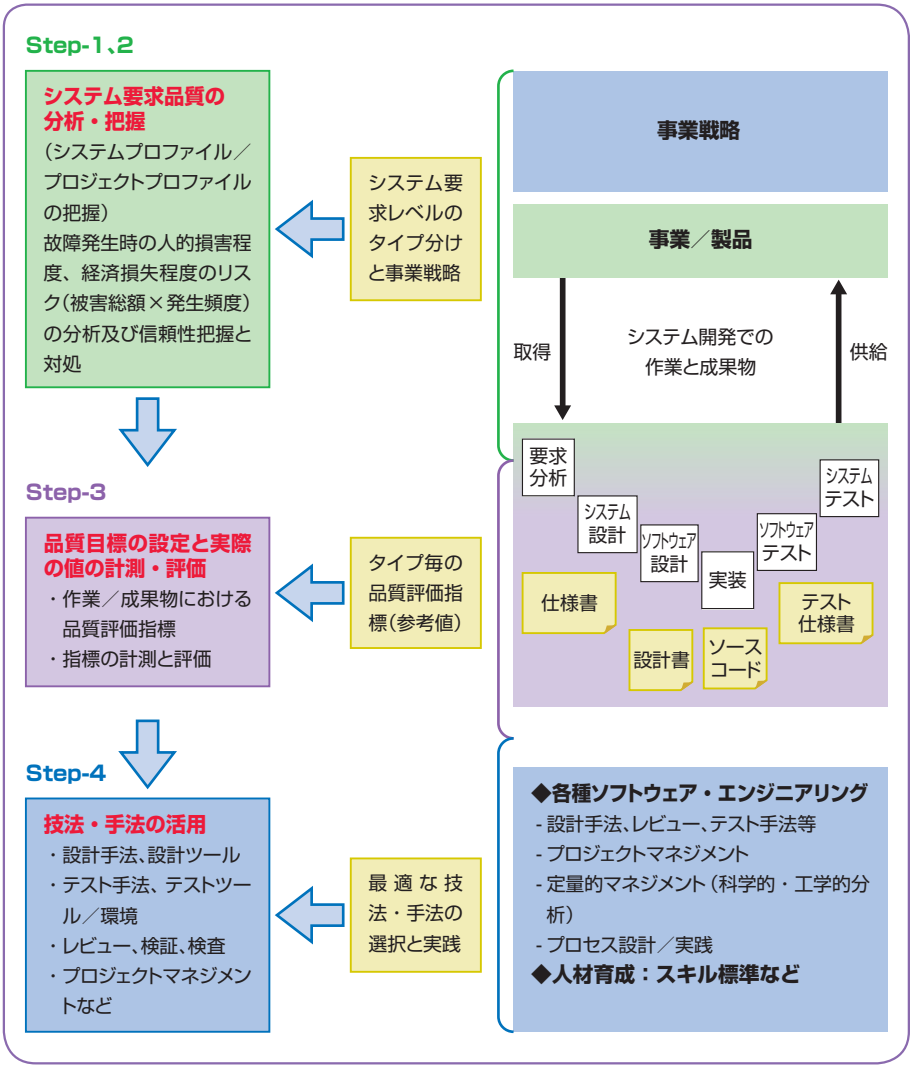


図1-4：本ガイドの利用手順

得られる効果

本ガイドを用いることで図1-4に示したように、以下のステップで品質コントロールを行います。

Step-1：開発をしようとしているシステムの特性を踏まえた要求品質の分析・把握を行い、

Step-2：その開発プロジェクトの特性を分析・把握し、

Step-3：それに対応した品質目標を設定し、実際の開発に即した指標を計測・評価を行い、

Step-4：設定した品質目標値に近づけるためのさまざまな品質向上の手法をとり入れた定量的な品質コントロールを行う

これにより、次のような効果を期待することができます。

開発対象のシステムが有する特性の分析（システムプロファイリング）により得られる効果

- 利用者側から見たシステムの適切な品質レベルを分析考慮することができます。
- 製品を作る際にはとかく、作り手の理由だけで品質のレベルが決まってしまいがちです。利用者側の視点に立って、どのような製品が求められているかということを機能面だけでなく品質面も合わせて考えてみることにより、品質確保のために必要な作業/不要な作業の見直しを行うことが可能となります。

システムを開発するプロジェクトの特性の分析（プロジェクトプロファイリング）により得られる効果

- 対象システムを開発する立場に立って、そのプロジェクトの特性を把握し、品質目標の設定に反映していくことができます。
- プロジェクトの特性が品質面にどのような影響があるかを客観的に考える機会が得られます。

品質レベルに応じた品質目標に基づき製品の品質をコントロールすることによる効果

- システムプロファイリングおよびプロジェクトプロファイリングで得た品質レベルに応じて、品質確保のために必要な作業に対し「十分性」の観点から定量的に評価するための指標を設けます。また、これらの指標に関する目標値（品質目標値）を設定して開発をすすめることで、求める品質を実現するために必要十分な作業を行うことができるようになります。

1.4 品質作り込みガイドの構成

品質作り込みガイドの構成

本ガイドでは、図1-5に示すように組込みシステムの品質を定量的にコントロールしていく作業について、「システムプロファイリング」、「プロジェクトプロファイリング」、「品質指標の定義と手順」という順に整理しています。また、こうした品質定量化の結果をもとに「高品質を作り込むための作業」に必要ないくつかのヒントを紹介しています。

具体的には下記のような構成になっています。

第1章：組込みシステム品質ガイドの読み方
第2章：プロファイリングを利用した品質目標値の設定
第3章：品質指標の定義と参考値
第4章：高品質作り込みのためのヒント

第1章では、本ガイドの位置づけや目的、および本ガイドの中心的な概念である品質指標について概要を説明します。

第2章では、品質指標を利用する前段として、システムを分析定義するシステムプロファイリングの考え方や開発プロジェクトの特性を評価するプロジェクトプロファイリングの考え方などを解説します。システムプロファイリングは開発対象のシステムに関して、ユーザ視点からみた品質や信頼性の観点からシステムにどのようなレベルが求められるかを体系的に分析しシステムをタイプ分けする考え方です。またプロジェクトプロファイリングは開発を進めるプロジェクトに着目し、その特質などを評価し、それらが開発するシステムの品質や信頼性にどの程度の影響を及ぼすかを評価する考え方です。

第3章では、作業の十分性を評価するプロセス品質評価指標と成果物の十分性を評価するプロダクト品質評価指標、さらにそれらの指標のために計測する基礎指標、それぞれの品質指標について考え方と使い方を定義解説します。さらに、2章で定義したシステムのタイプごとの品質指標の参考値を示します。なお、3章に示した指

標群は基本的に、どの組織でも手軽に測れる指標を提示しています。たとえば、規模を示す単位としては本ガイドではソースコードの物理行数：LOC（Lines of Code）を使っています。これはエディタなどを使えば誰でも簡単に測ることができ、かつ、誰が測っても同じ値を得ることができます。

第4章では、コミュニケーション、仕様書・設計書の書き方、レビュー、テストなどそれぞれについて確実に作業をするためのチェックリストとその解説を行います。さらに本ガイドを組織で使いこなしていくためのヒントについても解説します。

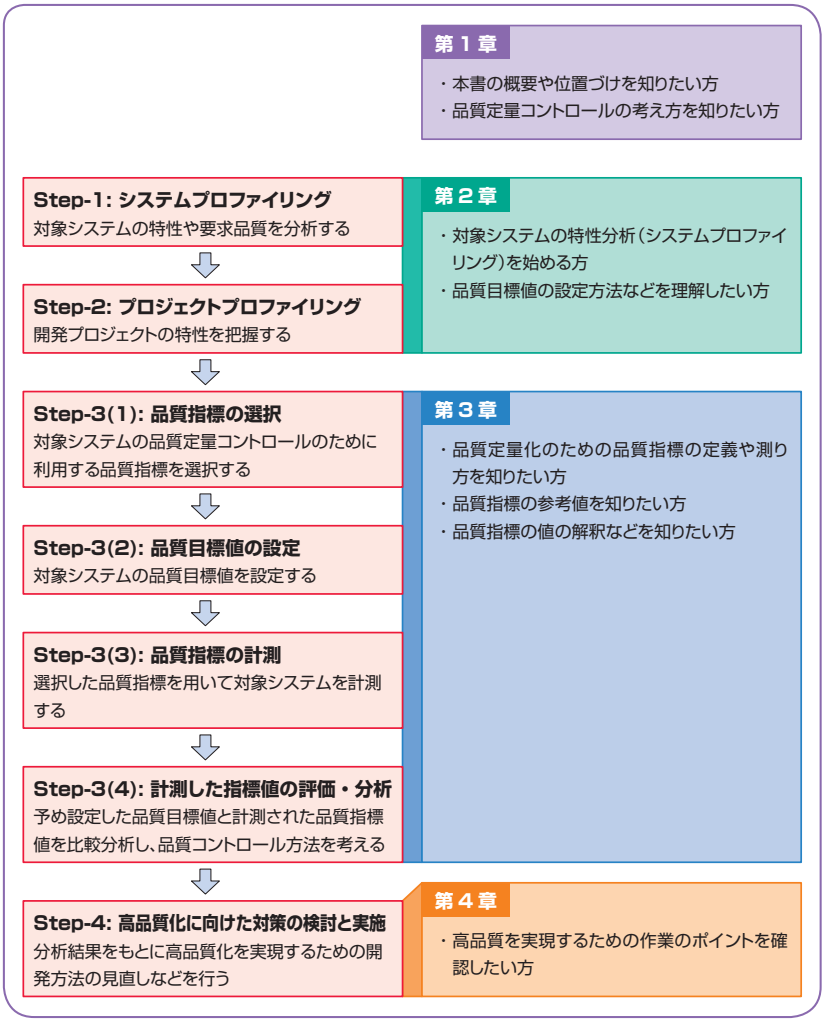


図1-5：本ガイドの構成

1.5 本ガイドの利用に関する注意事項

参考情報としての扱い

本ガイドは、組込みシステムの品質向上を目的として、開発過程での品質指標を用いた品質コントロールの概念に基づき、システムプロファイリングの考え方や品質指標定義や参考値、品質向上に関係すると思われる手法のポイントなどを整理したものです。本ガイドで紹介しているこれらの事項については、高品質な組込みシステムを構築するための一つの考え方であり、あくまでも参考情報として理解していただく必要があります。そのうえで是非、次に示すような点に留意し活用をお願いします。なお、本ガイドに記載された内容についての準拠性について、いかなる認証や認定といった枠組みを想定するものではありません。

①システムプロファイリングの考え方について

本ガイドで提示しているシステムプロファイリングの考え方は、システムに求められる品質や信頼性に関して、ユーザ視点を中心にしてみた場合の一つの考え方として紹介したものです。対象とするシステムによっては、ここで提示した考え方に必ずしもフィットしないものもあると考えられますので、利用の際には十分に理解・吟味した利用を推奨します。

②品質指標の定義およびその参考値はあくまで参考情報であること

本ガイドの3章に提示した品質指標は、ソフトウェアの品質を可視化するための一つの考え方です。世の中には本ガイドに提示したもの以外にもさまざまな品質指標や尺度、品質メトリクスが提案されていますので、それらも参照のうえで、自身の開発に適した指標などを選定することが望ましいといえます。また、同じく3章に提示した品質指標の参考値についても、あくまでも一事例として理解してください。これらの値も対象とする開発やシステムの条件によって、さまざまな値をとる場合があるため、個々の事情や条件を十分に考慮した利用が望まれます。

なお、本ガイド（ESQR Ver1.1）で提示したこれらの値は、Ver1.0検討にあたりSEC

にご協力いただいた皆様からのデータなどを吟味して決め、さらにVer.1.0読者の皆さまからいただいた貴重なフィードバックを反映したものですが、これらの値は適宜、見直しを進めESQRの次のバージョンでさらに精度の高い値へと調整をしていく予定です。

③数値に振り回されないこと

本ガイドではさまざまな指標や数値の概念が提供されています。我が国のソフトウェア開発の世界では、これらの数値にきわめて敏感でかつ安易に便乗してしまう傾向があります。本ガイドの冒頭で述べたように、ソフトウェアの品質を数値化することは、あくまでも高品質なソフトウェアを開発するための手段であって、目的ではありません。SECでさまざまな企業の方々とお話しする中で、よく「色々なデータを集めてはいるが、何が言えるかわからない」あるいは「その結果をどう使ったらよいかわからない」といった方がいらっしゃいます。こうした組織のほとんどは「数値を集めること」が目的化していると考えられます。本ガイドを読了後に、品質定量化のための指標の検討を始められる方は、何のためにその活動を始めるのかを、是非、最初に徹底的に検討してみてください。

SECへのフィードバックのお願い

SECはソフトウェアの開発力強化を目的とした公的な機関です。本ガイドのようなものを整理し発行することで我が国のソフトウェア開発力強化につなげたいと考えています。本ガイドで提示した指標類やその参考値などは、フィールドでの実際の値やフィードバックがきわめて重要になります。本ガイドを策定の段階で多くの企業の方々にさまざまなデータの提示をお願いしてきましたが、より質の高いリファレンスとするためには、より多くの方々からのフィードバックを必要としています。是非、本ガイドで提示した指標類に関する実データやその他の指標データなども含めて、巻末に添付したフィードバックシートなどを利用し、SECにフィードバックをお願いできればと考えています。

1.6 関連規格

国際規格

① ISO/IEC9126（JIS X0129）

この規格は製品としてのソフトウェアが有する品質面の特性を規定したものです。概念的には、ソフトウェア製品は信頼性、機能性他、計6つの側面を持つという概念に基づいて、それぞれの品質副特性や、それらを代弁するための代替特性（外部特性、内部特性）などを規定しています。ソフトウェアの分野では、さまざまなシーンで参照される規格であり、ソフトウェア品質に関する理解としては、そのもっとも根底に位置する規格と考えることができます。

本ガイドとの関係では、ソフトウェア製品の品質という面で、3章に提示したプロダクト品質評価指標の概念の一部がこの国際規格に関係しています。特に、コードの制御文記述率などは信頼性（複雑性）に、またコメント行記述率などは保守性に関係するといった間柄にあります。本ガイドの策定にあたっても、この国際規格を念頭にした議論がベースとなっています。

② IEC 61508（JIS C0508）

この規格は電気電子系の計算機システム（プログラマブル系）に関する機能安全の考え方を整理した国際規格です。この国際規格では、プラントなどのシステムを念頭に、その保安・安全系の機能的な充実度合いをSIL（Safety Integrity Level）という尺度で評価し、そのレベルに対応する形で、開発手法などの推奨をする形をとっています。本ガイドはこの国際規格におけるSILの概念を念頭におきながら、システムに求められる品質や安全性をより簡易的な方法で決定するシステムプロファイリングの考え方を整理しています。本ガイド本文中には示していませんが、本ガイドのシステムプロファイリングの4つのタイプと、この規格におけるSILのレベルは非常に関連が深いものとなっています。

③ ISO/IEC 15939（JIS X0141）

この規格はソフトウェアの計測法や計測プロセスに関しての概念を整理した規格です。大きくはソフトウェア計測に関する流れや枠組みに関して、計測対象物の計測方法の概念と基本測定量やそこから導き出される導出測定量の概念、またそれらを判断基準と比較し意思決定の根拠とする指標といった概念に整理しています。またこれらに関して、計測・評価という行為に関係するさまざまな用語なども定義したものです。

本ガイドではこの規格を参照しつつ、より実践の場でのなじみやすさという視点から、対象を計測し評価する尺度をその計測方法も含めて“指標”という用語に統一しています。

第2章

プロファイリングを利用した品質目標値の設定

組込みソフトウェアの品質目標や品質作り込みを考えた場合、それぞれ対象とする組込みシステムの特徴を考慮した目標値の設定が必要になります。本章では、組込みシステムの品質目標値設定の際に必要となるプロファイリングの考え方と、それに基づく組込みソフトウェアの品質指標設定の方法について紹介します。

2.1 組込みシステムの特性を考えた品質目標設定の考え方 ……24
2.2 Step - 1：システムプロファイリング ……28
2.3 Step - 2：プロジェクトプロファイリング ……34
2.4 Step - 3：品質目標値の設定 ……37
2.5 プロファイリングの事例 ……45
2.6 システム障害の評価とシステムプロファイリングへの反映 ……48

2.1 組込みシステムの特性を考えた品質目標設定の考え方

品質目標設定の基本的な考え方

我々の身のまわりで利用される組込みシステムはさまざまなものがあります。たとえば携帯電話やテレビなどの情報家電、冷蔵庫、電子レンジなどの白物家電、自動車やエレベータなどの輸送機器、銀行のATMや駅の改札機などの社会インフラ機器などがあります。また、発電所などで利用されている複雑な制御装置なども組込みシステムの範疇に入ります。組込みシステムの品質を考える場合、これらの多様な組込みシステムをすべて「組込みシステム」という名称で一括りにして考えるのは少々粗っぽいといえます。たとえば、携帯電話や情報家電ではシステムに障害が発生した場合、利用者が多大な迷惑をこうむることは確かですが、システムの利用者が怪我をするなどといった人的な損害に直結するケースはきわめて稀で、あまり考えられません。一方、自動車やエレベータ、航空機などに搭載される制御システムで障害が発生した際には、場合によっては人的な損害を含めたきわめて深刻な事故を誘発する可能性があります。このように、本来、組込みシステムが製品として持つべき品質、言い換えれば製品として期待される品質は、その製品の特性によって観点や程度に大きな違いがあると考えられます。従って、これらの組込みシステムを開発し、その開発過程で品質の作り込みを実践する場合にも、対象となるシステムの特性を考慮して品質目標を設定し、品質作り込みを進めていく必要があります。

プロファイリングを用いた品質目標値設定の流れ

プロファイリングを利用して、対象システムの特性を考慮した品質目標値を設定するためには、下記の3つのステップを順に実施していくことになります。

Step-1：システムプロファイリングの実施

システム利用・運用時に発生する可能性のあるシステム障害のうち、組込むソフトウェアが関与する場面を想定し、その経済面および人的面の被害額や影響額をもとに対象システムを4つのシステムタイプに分類します。

Step-2：プロジェクトプロファイリングの実施

システムの特性や、システムに組込むソフトウェアの実装面、開発プロジェクトの特性などを考慮して開発プロジェクトとしての特性を評価します。この評価結果は、Step-1で得た4つのシステムタイプに対する補正係数として、次のStep-3の品質目標値設定の際の参考とします。プロジェクトプロファイリングでは全部で10個のファクターをチェックし、その全体の状況から上記の補正係数への変換を行いますが、ここで掲載した10個のファクターはあくまでも一例であるため、必要に応じて見直しや追加などを行うことをお勧めします。

Step-3：品質目標値の設定

本ガイドの3章には組込みソフトウェアの品質作り込みを行ううえで考慮すべき品質目標値を設定するための指標（メトリクス）とその参考値を掲載します。参考値はStep-1の4つのシステムタイプごとに決められていますが、同時に補正ベース値も掲載しています。上述したStep-1、Step-2のシステムプロファイリングおよびプロジェクトプロファイリングの結果をもとに、本ガイド3章に記載された品質指標の情報を見ながら、個々の開発案件に関して採用する品質指標とその目標値の設定を行います。なお3章で紹介する指標群の参考値は、SECの活動を通してヒアリングやアンケートなどによって収集されたデータ群を基礎データとして分析し算出した値であり、あくまで参考値として掲載したものですので、実際に利用する際には、これらの指標の吟味や見直し、追加などを適宜行うことをお勧めします。

品質目標設定の対象と範囲

品質目標の設定は上述したような3つのステップで実施していきますが、ここで問題となるのは、プロファイリングの対象と範囲です。

システムの部分に対する考え方

一般に組込みシステムはさまざまな“部分”から構成されており、結果的に、そこで利用される組込みソフトウェアも複数の部分に分割することができます。そして、これらの部分は当然のことながら、その特質や求められる品質が大きく異なる場合も出てきます。たとえば、自動車というシステムで利用されるソフトウェアを考えてみると、その中でもブレーキ制御を司っているソフトウェアとエアコンなどを司っているソフトウェアでは、当然、求められる品質や信頼性、あるいは性能が異なっていると考えることができます。プロファイリングを行う場合には、こうした異なる特質をも

つ“部分”はそれぞれ別個にプロファイリングをして考えていきます。これによって、個々の部分の特質を反映した品質目標につなげることができるようになります。対象とするソフトウェアをどのような部分に分割して考えていくかについては、それぞれのソフトウェアの機能や構造などによっても異なりますので十分にそれらを検討したうえで、適切な粒度でプロファイリングを行う必要があります。一方で、こうしたシステムの構成部分を注視してプロファイリングの厳密性のみを追及すると、システムの総体としての特性が把握できなくなるといった場合もありますので、システムの部分をみながらも常にシステム全体を意識しながらプロファイリングを行うことが理想的です。

ソフトウェアの流用や再利用に関する考え方

最近の組込みソフトウェア開発は、過去の設計資産を流用したり、再利用する形での開発が中心となっています。このような場合、プロファイリングや後述する品質目標値の設定対象をどのように考えるかが重要になります。特に品質目標値の中でもテスト実施率などのように対象ソフトウェアの全体ボリュームや全体工数で値をノーマライズする場合に、それらをどのように考えるかはさまざまな方法があります。本ガイドでは図2-1に示すように、サブシステムソフトウェアをファイルに分割した際、プログラムやその設計書に1行でも手を加える可能性がある、あるいは実際に手を加えたサブシステムやファイルについては、手を加えた分量によることなくプロファイリングや品質目標値の設定対象として考えていきます。従って、図のように、A、B、Cの3つの要素からなるソフトウェアに対し、B、Cの一部を修正したり追加したりして利用し、また、Dの要素を全く新規に開発し、Aの要素は全く手を加えずに利用した場合には、プロファイリングは図のB、C、Dのサブシステムを対象とし、また品質目標の設定の際のベースとなるソフトウェア全行数はB、C、D部のソフトウェアの全行数と考えて計算していきます❶。また、製品のシリーズ化などに伴う派生バージョンの開発などについても、基本的には上記の考え方を踏襲します。すなわち、派生バージョン開発などの場合、派生に伴い機能追加や機能変更が発生した対象部分（これらのボリュームは問わない）を基本的なプロファイリングや品質目標値の設定対象と考えていきます。

❶：なお、この場合でもAのサブシステムがシステム全体の中で中核的な役割を持ち、テスト作業の対象となるなどの場合には、適宜、ソースコード全行数の計算に含めることも検討してください。

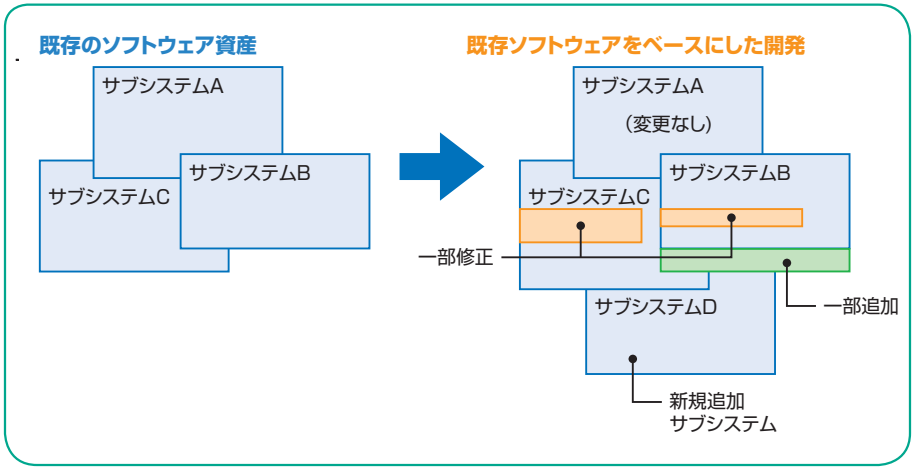


図2-1：ソフトウェアの流用や再利用に関する考え方

2.2 Step-1：システムプロファイリング

システムプロファイリングの役割りと位置づけ

我々の身のまわりにある組込みシステムはさまざまなユーザがさまざまな用途で利用するという点において、限りなく裾野が広がりつつあります。こうした広がりを持った組込みシステムに関して、それらの特性を考慮した品質作り込み活動を行う場合、その第一歩として対象となる組込みシステムの特性を評価する作業が必要となります。本ガイドでは組込みシステムの特性を評価する手法として、システムプロファイリング（SCP：System Characteristics Profiling）を採用します。SCPは近年活発に議論されている電子電気系の機能安全の議論やシステムディペンダビリティの議論をベースに、SECで策定したオリジナルなシステム特性評価方式です。

システムプロファイリングは「対象とする組込みシステムの利用シーンを想定し、システムの利用者の視点に立ったシステムのタイプ分けを行う点」を最大の特徴としています。すなわち「システムが完成し実際の利活用のフェーズに入った際に、何らかのシステム障害が発生した場合に、システムの利用者にどれくらいの迷惑やどれくらいの損害を与えてしまうか」という視点に立ち、システムのタイプ分けを行う考え方です。たとえば、携帯電話の制御システムに不具合があった場合には一般ユーザがどれくらいの被害をこうむるか、あるいは、発電所の制御システムに不具合があり仮に原子炉が暴走したらどれだけの被害が発生するか、といった被害想定を行い、それをもとにシステムのタイプ分けを行います。

このように本ガイドにおけるシステムプロファイリング（SCP）は社会的に求められるシステム品質の定義とタイプ分けを行うためシステム利用・運用時に発生する可能性のあるシステム障害を想定し、その経済面および人的面の被害額や影響額をもとに対象システム（あるいはその構成の一部）を次の4つのタイプに分類します。

システムのタイプ分け

システムのタイプ：SCP（System Characteristics Profiling）

Type-1：Normal：通常レベルの品質や信頼性が求められるシステム

Type-2：Normal Quality Required：通常以上に高い品質や信頼性が求められるシステム

Type-3：Critical：高い品質や信頼性が求められるシステム

Type-4：Highly Critical：きわめて高い品質や信頼性が求められるシステム

本ガイドでは上記のように対象システムを4つのタイプに分け、このタイプに応じてプロダクトで達成すべき品質やその開発プロセスに求められる品質活動の目標値を決めていきます。前述の2つの事例（携帯電話の制御システムと発電所制御システム）では、後者の方が大きな被害が発生するであろうことは想像に難くありません。SCPを適用することによって、たとえば、携帯電話のシステムでは高品質レベル（Type-2：Normal Quality）、発電所関係のシステムでは超高品質・高信頼レベル（Type-4：Highly Critical）が求められるシステムとして求められる品質レベルとタイプ分けすることができます。

システムプロファイルの評価方法

システムプロファイリングでは、対象システムを上記の4つのタイプに分類するために、下図に示すように、次の2つの判断を順に行っていきます。

① 人的損失の判定　　②経済損失の算定

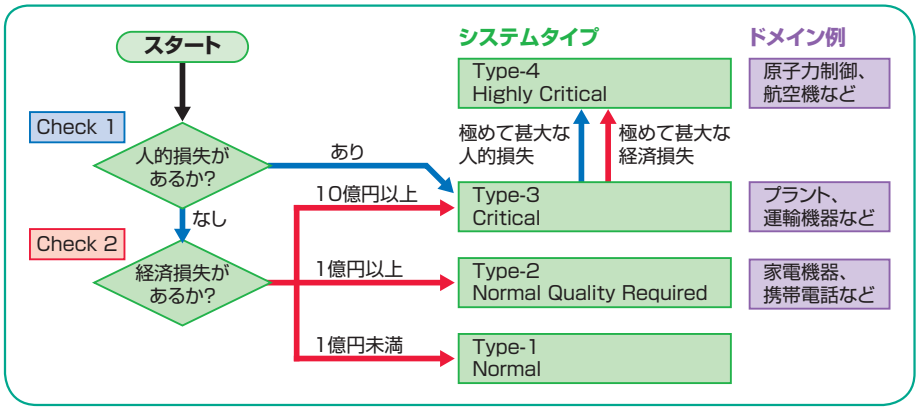


図2-2：SCPチェック手順

①人的損失の判定

対象システムが障害を発生した場合に、その結果発生する事故によって、そのシステムの直接的なユーザや間接的なユーザに怪我や人命損失などの致命的な影響を及ぼすかどうかを分析します。人的損失判定のポイントは下記の通りです。

［人的損失判定のポイント］

- システム障害による事故発生で人的な損失（含む重大傷害）が発生する可能性がある場合、安全性を確保する手段のひとつとして高い品質や信頼性が求められるシステム（Type-3以上）と考えます。
- システム障害による事故発生により相当数の人的損失が発生する可能性があると予想される場合には、Type-4（Highly Critical）と考えます。

人的損失	タイプ	略号	意味
人的損失は発生しない	Type-2以下	NQ以下	Normal Quality Required 以下
人的損失（重大傷害を含む）が発生する可能性がある	Type-3	C	Critical
相当数の人的損失（重大障害を含む）が発生する可能性がある	Type-4	HC	Highly Critical

［判定のヒント］

- **直接・間接の被害認定**

  たとえば原子力発電所が臨界事故を起こしたチェルノブイリ原発のような事故を想定すると、原発で働いていた技術者の人命損失とともに、その周辺住民への深刻な健康被害も想定されます。このような場合には、直接的なユーザに対する被害とともに間接的に発生する周辺住民への被害も人的損失に含めて考えます。

- **1製品による複数の事故の可能性**

  たとえば自動車のエンジン制御ソフトウェアなどの障害によって交通事故が発生する可能性がある場合、1事故ではその自動車の搭乗者の人数が直接的な被害人数となります。しかし、自動車などの場合には、同じソフトウェアが搭載された自動車が生産台数分市場に流通し、事故を引き起こす可能性があります。従って、この場合の人的損害規模は生産台数累積の人的損害数なども考慮して人的損害を評価します。

- **事故発生の確率**

  上記の発電所の事故などの発生確率はきわめて低く、人的被害が発生するよう

な事故が起きてしまうことは稀です。一方で、自動車のエンジン制御ソフトウェアの不具合などは（発電所の不具合に比べると）かなりの頻度で発生する可能性もあります。厳密にはこれらのシステム障害の発生確率や人的被害の発生確率などを考慮して、求めたほうがより正確な被害程度を求めることができます。SCPではシステムのタイプ分けを簡易に行うことを目的とするため、これらの障害発生確率や人的被害発生確率は原則として考慮しませんが、組織・プロジェクトに応じて、事故発生確率なども加味した被害想定を行っても構いません。また、冒頭のチェルノブイリ原発の事故のようにシステム障害が発生した後に、長期にわたりシステム関係者（周辺住民なども含め）に健康被害が生ずる場合などについては、それらの期間を考慮した人的被害を想定して評価していきます。

②経済損失の算定

次に、対象システムが障害を発生した場合に、そのシステムの障害によってユーザがこうむる経済的な損失を分析します。組込みシステム単体の場合、極端な経済的な損失は想像しにくい場合がありますが、近年、組込みシステムを内包した機器がより大きな系（システム、ITシステムなど）の中で利用されることにより、その一部である組込みシステムが故障することによって、系全体の機能が停止するといった場合も想定されます。このような場合は、加味した経済的な損失評価がきわめて重要となってきます。経済損失判定のポイントは下記の通りです。

［経済損失算定のポイント］

- ①でType-4と判断した場合には経済損失の算定は不要です。
- 事故発生で経済損失が発生する可能性がある場合、経済勘案システムと考え1事故あたりに引き起こされる直接ユーザ、間接ユーザの経済的な損失を算定します。
- 1事故あたりで引き起こされる経済的な損失額に応じて以下の4つにタイプ分けします。なお、ここでのタイプ区分値は目安であり、それぞれの製品分野あるいは企業などの事情を考慮した見直しなどを適宜実施していただいても構いません。

経済損失	タイプ	略号	意味
1億円未満の損失	Type-1	N	Normal
10億円未満の損失	Type-2	NQ	Normal Quality Required
10億円以上の損失	Type-3	C	Critical
Type-3のシステムのうち、巨額の損失が想定されるもの	Type-4	HC	Highly Critical

［算定のヒント］

- **直接・間接の被害算定**

  たとえば航空機のチェックインシステムの一部が故障したことにより、航空機の旅客チェックインが不可能となった場合を考えると、航空会社のサービスが滞ることによるビジネス上の損害（経済損）を第一に考えることができます（直接的な被害額）。一方で、個々の旅客も航空機の発着がキャンセルになるなどして、個々人の仕事などに支障が出ることが考えられます（間接的な被害額）。経済損の判定はこれらの直接的被害額と間接的被害額を合計して考えます。

  また、工場の制御システムなどの場合、システムトラブルによってその工場の製品製造が停止し、最終ユーザに製品が提供できなくなるなどの間接的な被害も考えられますが、このような場合にも経済損失として考慮していきます。

- **時間経過に伴う被害の算定**

  システム障害が発生した場合、そのシステムを修理し再稼動させるまでには時間がかかります。ユーザ側の経済損はその間、継続的に発生すると考えられます。このため、経済損の認定では、システム障害によりシステムが提供するサービスが利用できない間に発生する経済的な被害を合算して計算します。

- **メーカ側経済損の除外**

  一方、組込みシステムで不具合が入り込むとシステムの回収（リコール）や修理などメーカ側にも経済的な損害が発生します。しかし、システムプロファイルの第1ステップでは、こうしたメーカ側の経済損は計算から除外し、次のステップであるプロジェクトプロファイリングの評価に委ねます。

実システムの障害経験や事例からのフィードバック

システムプロファイリングでは上記のようにシステム障害が発生した際の人的被害、経済被害をあくまでも想定のうえで計算していきます。しかし現実には、開発したシステムの不具合によってシステム障害が発生した場合に、どのような影響がどの程度に及ぶかを想定するのはかなり難しい場合があります。こうした場合には、本ガイド2.6節に示すように、類似のシステムや過去の類似製品において実際に発生した不具合や事故を振り返って分析し、その被害想定などの参考にしていくのが有効と考えられます。

Column

システムプロファイリングとSILとの関連

SILはIEC 61508においてシステムの安全性能を表す尺度で、SIL1からSIL4まで4レベルが定められ、SIL4を最高レベルとして各SILに対応して目標機能が動作しない/故障するといった確率で規定されています。たとえば、動作要求があった際に動作しない確率の平均が、SIL4では10^{-5}以上10^{-4}未満であることが必要とされるといった具合です。一方、本ガイドで提示したシステムプロファイリングでは、障害が発生した場合の、人的、経済的影響の大きさによって対象システムを4つにタイプ分けします。

2011年に発行される予定の自動車の電子制御系に関する安全規格ISO 26262においては、SILがASIL（automotive SIL）とされ、確率による定義ではなくハザード解析とリスクアセスメントによって見積もってレベル分けするものと定義され、システムプロファイリングの考えに通じるものもあります。

2.3 Step-2：プロジェクトプロファイリング

プロジェクトプロファイリングの意味と位置づけ

本ガイドではStep-1に示したように対象とするシステムの品質レベルをタイプ分けし、そのタイプに対応した品質目標を設定するという考え方を採用しています。たとえば、「Type-1のシステムの場合にはレビュー時間は開発全体の工数の10%を実施する」といったような品質活動の目標値を設定することになります。ところが実際のシステム開発の現場を見ると、たとえば、通常レベルの品質が求められるType-1のシステムだからという理由で安易に考えた結果、開発にまったく不慣れな新人の寄せ集めのチームで開発にあたってしまうという場合もあるかもしれません。こうした場合、Type-1のシステムとして求められるレビュー時間数だけレビューをすれば十分かどうかはかなり怪しくなってしまいます。場合によってはこうした開発メンバのスキルや経験を考慮して、多少レビューの時間を多めにしたほうが安全かもしれません。このため本ガイドではシステムプロファイリングによって求められたシステムのタイプをベースに、その対象システムを開発するプロジェクトの特性も加味して、最終的に品質目標を設定していきます。本ガイドではStep-2として、対象システムの開発にあたるプロジェクトに視点を当て、プロジェクトの特性を評価することを目的にプロジェクトプロファイリング（PCP：Project Characteristic Profiling）の考え方を採用しています。詳細は以降に記載しますが、システム実装時の特性や開発プロジェクトの事情に関する10個のファクターをチェックします。ここで得られたプロジェクトプロファイリングの結果は、本ガイドのStep-3で行う品質目標値設定の際に、システムタイプの補正係数として利用していきます。

プロジェクトプロファイリングファクター

実際に開発する場合のプロジェクトの特性やシステムやソフトウェア実装面の特性などに関して表2-1に示す10個のプロファイリングファクター (以下、ファクターと略す) を補正要因として勘案し、プロジェクトの特性のプロファイルを評価していきます。具体的には、これら10個のファクターに関して表2-1に示したプロジェクトプロファイリングファクターをチェックしていきます。

なお、プロジェクトプロファイリングに使用するこれらの10個のファクターはあくまでも一例として提示したものです。評価を行う対象となる組織やプロジェクトなどの事情に応じて、これらのファクターは適宜見直しすることをお勧めします。

表2-1：プロジェクトプロファイリングファクター

①ソフトウェアの規模	規模が大きい場合、ソフトウェアの不具合が入りやすくなる傾向があるため、より慎重な開発が求められます。
②ソフトウェアの複雑さ	ソフトウェアが複雑な場合にも不具合が入り込みやすくなる傾向があるため、より慎重な開発が求められます。
③システム制約条件の厳しさ	システムの制約条件がきついと、設計・実装で考慮すべき事項が増加し、また、さまざまなケースでのテストなども必要になります。
④仕様の明確度合い	システムの仕様が曖昧なまま開発を続けたプロジェクト、あるいは仕様変更が頻発するプロジェクトでは結果として、後戻りや不具合が多くなる傾向が強いため、開発段階での慎重なチェックが必要になります。
⑤再利用するソフトウェアの品質レベル	再利用による開発を行う場合、再利用もとのソフトウェア品質が低いとその影響を受けやすくなるため、レビューやテストなどもより慎重に行う必要があります。
⑥開発プロセスの整備度合い	開発プロセスが整備され運用できていないと、開発者間の足並みがそろわず、結果的に作業の抜けや誤りが発生し、不具合が発生しやすくなります。
⑦開発組織の分業化・階層化の度合い	開発組織が巨大で分業化していたり階層構造が深いと、情報伝達面などでの問題が発生しやすくなります。
⑧開発メンバのスキル	開発メンバのスキルにばらつきがあったり、低スキルのメンバが多数を占めている場合には、レビューなどを通して作業やその結果の確認をより慎重に行う必要があります。
⑨プロジェクトマネージャの経験とスキル	プロジェクトマネージャ（PM）の経験やスキルが低い場合には必要なタイミングでのレビュー＆チェックが甘くなる可能性があるため、予め、十分なレビュー時間などを確保しておく必要があります。
⑩システム障害時のメーカ側損失額	システム障害が発生した場合にリコールなどメーカ側の損失が巨額になる場合には、開発段階で徹底した品質チェックや作り込み活動を行い、そのリスク低減を図る必要があります。

プロジェクトプロファイルチェック表

表2-2を用いて①～⑩のファクターについてチェックし、プロジェクトプロファイルを評価します。たとえば、当該プロジェクトの開発対象について「①ソフトウェア規模」が「極めて小さければ」該当する欄をチェックします。①から⑩までのすべてのファクターのチェックが終了したら、それぞれ-1の欄にチェックがついた数に-1をかけた数を小計欄（左）に、+1の欄にチェックがついた数に1をかけた数を小計欄（右）にそれぞれ記入します。最後に、小計欄の数値を合計し、その値を合計ポイント数欄に記入します。この合計ポイント欄の数値は、本ガイドのStep-3において品質目標値を求める際のシステムタイプに対する補正係数となります。

表2-2：プロジェクトプロファイル表

ファクター			マイナス補正（－1）		基本		プラス補正（＋1）
①	ソフトウェア規模	□	極めて小さい	□	普通	□	極めて大きい
②	ソフトウェアの複雑さ	□	極めて単純	□	普通	□	極めて複雑
③	システム制約条件の厳しさ	□	制約ゆるい	□	普通	□	制約厳しい
④	仕様の明確度合い	□	極めて明確	□	普通	□	明確になっていない
⑤	再利用するソフトウェアの品質レベル	□	極めて高品質	□	普通	□	極めて品質低い
⑥	開発プロセスの整備度合い	□	整備できている	□	普通	□	整備できていない
⑦	開発組織の分業化・階層化の度合い	□	開発組織が単純	□	普通	□	開発組織が複雑
⑧	開発メンバのスキル	□	メンバスキル高い	□	普通	□	メンバスキル低い
⑨	プロジェクトマネージャの経験とスキル	□	ＰＭスキル高い	□	普通	□	ＰＭスキル低い
⑩	システム障害時のメーカ側損失額	□	極めて小さい	□	普通	□	極めて大きい
小計							
合計ポイント数							

（例）

以下、補正係数を求めるために上記チェック表を用いて計算した例を示します。

この例では、-1 欄の小計が -3 ポイント、+1 欄の小計が +1 ポイントであるため、合計して求められる補正係数は -2 となります。

表2-3：プロジェクトプロファイル表（記入例）

ファクター			-1		0		+1
①	ソフトウェア規模	■	極めて小さい	□	普通	□	極めて大きい
②	ソフトウェアの複雑さ	□	極めて単純	□	普通	□	極めて複雑
③	システム制約条件の厳しさ	■	制約ゆるい	□	普通	□	制約厳しい
④	仕様の明確度合い	□	極めて明確	■	普通	□	明確になっていない
⑤	再利用するソフトウェアの品質レベル	□	極めて高品質	■	普通	□	極めて品質低い
⑥	開発プロセスの整備度合い	□	整備できている	■	普通	□	整備できていない
⑦	開発組織の分業化・階層化の度合い	□	開発組織が単純	□	普通	■	開発組織が複雑
⑧	開発メンバのスキル	□	メンバスキル高い	■	普通	□	メンバスキル低い
⑨	プロジェクトマネージャの経験とスキル	□	ＰＭスキル高い	■	普通	□	ＰＭスキル低い
⑩	システム障害時のメーカ側損失額	■	極めて小さい	□	普通	□	極めて大きい
小計		-3		0		1	
合計ポイント数							-2

なお、ここでプロジェクトプロファイル表の補正値は、すべてのファクターに対してマイナス補正を「-1」プラス補正を「+1」としていますが、これらは組織や企業内での経験に基づいて開発対象のシステムやプロジェクトに応じ、それぞれに最適な値を設定し、カスタマイズしていただくこともできます。

2.4 Step-3：品質目標値の設定

品質目標値とは

開発するソフトウェアの品質目標値は計測する対象（システム・開発）を明確にし、Step-1で得られた対象システムに関するシステムタイプ、およびStep-2で得られたプロジェクトプロファイルに基づくタイプ補正係数を利用し、本ガイドの3章に記載された個々の品質指標の参考値テーブルなどを参考にして設定していきます。

本ガイドでは組込みソフトウェア開発途中での品質の定量的なコントロールを実現するための指標群を3章で定めています。これらの指標は基本的に、開発の途中段階で、たとえば「設計書のレビュー作業をどの程度すべきか」あるいは「ソースコードの複雑さはどの程度に抑えるべきか」といった開発段階での品質の目標を定める目安として利用していきます。このための指標の概念として、本ガイド1章に記載したように、プロセス品質評価指標、プロダクト品質評価指標の2種類の指標を用意します。

- 各中間成果物に関するレビューなど品質をチェックする作業の十分性を評価するプロセス品質評価指標
- 開発途中段階での中間成果物の品質を評価するためのプロダクト品質評価指標

実際の開発においては、これらのプロセス品質評価指標やプロダクト品質評価指標の目標値はプロジェクトとしての品質目標値と考えることができます。品質目標値に関しては、すでに述べたように、開発しているシステムや組込みソフトウェアに求められる品質レベルや信頼性レベルなどに応じて設定する必要があります。これらを考慮しない画一的な品質目標値設定を行ってしまうと、システムやソフトウェアとして求められている品質レベルとのギャップが生じてしまい、過剰品質や過小品質となってしまう恐れがあります。本ガイドでは、前述のSCP（System Characteristics Profiling）およびPCP（Project Characteristic Profiling）の結果を参考に、個々のプロセス品質評価指標やプロダクト品質評価指標の目標値設定を行います。

採用するプロセス/プロダクトの品質指標の検討と決定

品質目標値を設定する前段階として、まず、どのような指標を利用して品質目標値を設定していくかを考える必要があります。このため、図2-3に示すように本ガイドの3章に記載した品質指標を参考に、対象プロジェクトでどの指標を採用するかを検討し決定します。採用する指標の検討に際しては、プロジェクトの開発プロセスや作成予定の中間成果物などを踏まえ、どのようなタイミングでどの指標を計測するかを考えて決めていきます。3章に記載した指標群はあくまでも参考例として提示したものですので、開発対象のシステムやプロジェクトの特性や事情を考慮して適切な指標を選択、採用していくことをお勧めします。また、これらは必要に応じて追加しても構いません。また、この際に、本ガイドの4章の品質定量化に関するヒント(4.5節)にも述べているように、指標計測のコストやその指標を計測する目的、指標の利用方法なども十分に吟味しておく必要があります。

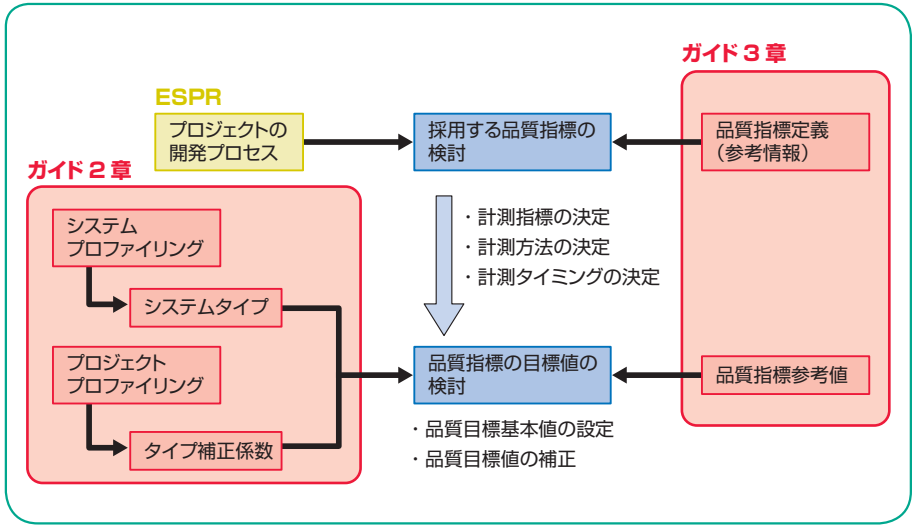


図2-3：品質目標値設定の流れ

品質目標値の設定方法

本ガイドの３章に記載した指標の参考値とStep-1で求めたシステムタイプ、Step-2のプロジェクトプロファイリングの結果から得られたタイプ補正係数を利用して、個々の指標の目標値（品質目標値）を決めていきます。品質目標値の設定に関しては、開発プロジェクトのリーダ、マネージャや実際の開発担当者の考えなども確認しながら設定していく必要があります。

①品質目標値の基本値の設定

プロジェクトで採用する指標を決定した後に、選択した指標の目標値を設定する必要があります。

本ガイドの３章には、個々のプロセス品質評価指標、プロダクト品質評価指標に関する定義とともに、それらの参考値が示されています。たとえば、代表的なプロセス品質評価指標である設計レビュー作業充当率（ID：PR11、RDRE）を見ると、下記のような表が掲載されています。この表から、設計レビュー作業は、対象システムのタイプがNormalの場合には充当率2.00%、Normal Quality Requiredの場合には6.00%、Criticalの場合には10.00%、Highly Criticalの場合には14.00%の値を基本値としてそれぞれ選択します。

表2-4：設計レビュー作業充当率の例

ＩＤ	PR11
名称	設計レビュー作業充当率
略称	RDRE
名称（英語表記）	Ratio of the Design Review Effort

参考値	N	NQ	C	HC	補正ベース値
	2.00	6.00	10.00	14.00	4.00
参考値の範囲	0.00 ～ 6.00	2.00 ～ 10.00	6.00 ～ 14.00	10.00 ～ 18.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 設計のレビューにどれだけ工数をかけているかを設計に費やした工数（設計プロセスの工数）とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、設計作業そのものも工数は多くなります。従って多ければ多いほど良いというものではなく、適切な値であることが求められることになります。				
計算方法	設計レビュー工数/設計作成工数 RDRE = REDE/PEDE				

②品質目標値の補正

次に、①で求めた品質目標値の基本値の補正を行います。ある対象システムについて、前述のシステムプロファイリングを行い、Step-1：SCP（System Characteristic Profiling）結果がNタイプに分類され、かつ、Step-2：PCP（Project Characteristic Profiling）の結果が+4ポイントと評価されたとします。この場合、対象システムに関する設計レビュー作業充当率は、

Nタイプの仕様書レビュー充当率：2.00%
設計レビュー充当率の補正ベース値：4.00
補正係数：+4

より、

対象システムの設計レビュー作業充当率
＝　当該タイプの設計レビュー作業充当率　＋　補正係数/10×補正ベース値
＝　2.00＋（4/10×4.00）
＝　3.60%

と求めることができます。

補正係数の基本的な考え方

上記の計算方法でもわかるように、あるシステムのタイプがStep-1：SCP（System Characteristic Profiling）の結果NやNQなどに決まった場合、Step-2：PCP（Project Characteristic Profiling）による補正効果は最大限でもプラスマイナス10（100%）であり、System Characteristic Profilingのタイプを一つ上げたり、下げたりする範囲内で補正していきます。もう少し平たく言うと、Step-1のタイプ分けであるシステムがType-3：Criticalとなった場合、プロジェクトプロファイリングによる開発の条件を最大限考慮したとしても、Criticalのシステムが2段階下のNormalレベルと同じ品質目標値となることは過小品質になってしまう可能性が高く、好ましいとは言えません。すなわち、Criticalのシステムの場合、開発の都合を考慮し、1段階下のNormal Quality Requiredから1段階上のHighly Criticalの範囲内で、目標値を設定することが望ましいと考えられます。図2-4に、この考え方を示します。

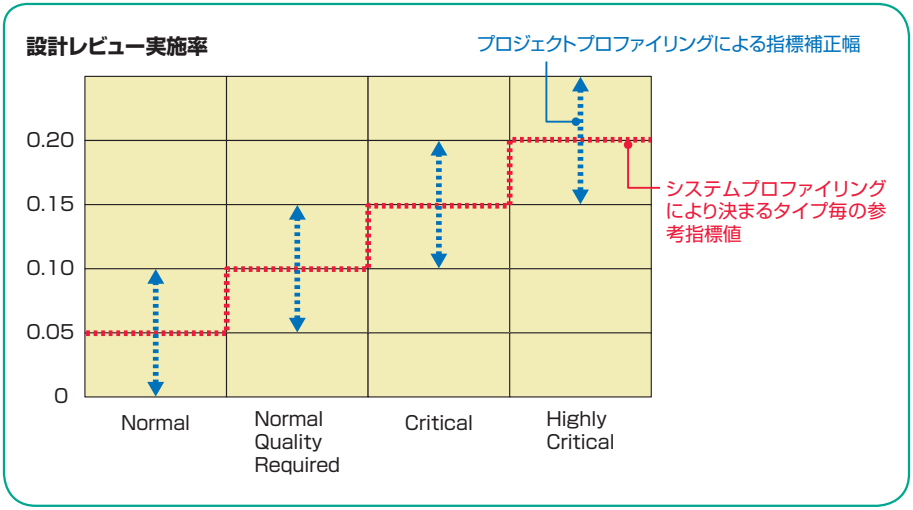


図2-4：補正係数の考え方

実際の品質目標値の設定では、表2-5に示すような品質目標値設定表を利用して指標の選択と目標値設定を行っていきます。

表2-5：品質目標値設定表

対象システムまたはプロジェクト名	

システム／プロダクトプロファイリング結果				
System Characteristic Profiling	Normal	Normal Quality	Critical	Highly Critical
Project Characteristic Profiling			point	

プロセス品質評価指標			当該タイプの参考値	補正ベース値	設定目標値
作業充当率					
	仕様レビュー			4.00	
	設計レビュー			4.00	
	コードレビュー			1.50	
	テストレビュー			1.50	
	テスト			5.00	
	レビュー			4.00	
作業実施率					
	仕様レビュー			2.40	
	設計レビュー			2.40	
	コードレビュー			1.20	
	テストレビュー			2.00	
	テスト			17.00	
	レビュー			8.00	
プロダクト品質評価指標			**当該タイプの参考値**	**補正ベース値**	**設定目標値**
ドキュメント					
	ドキュメントボリューム				
		要求仕様書		4.00	
		設計書		10.00	
		テスト仕様書		10.00	
	ドキュメントバランス				
		要求仕様書		-	
		設計書		-	
		テスト仕様書		-	
コード					
	コードボリューム				
		ファイル行数	2 KLOC	この値以下であること	
		関数の行数	160 LOC	この値以下であること	

表2-5：品質目標値設定表（続き）

	コード特性				
		制御文記述率		5.00	
		コメント行記述率		5.00	
		コーディングルール逸脱率		100	
テスト					
	テスト十分性				
		テスト密度		25.00	
		不具合収束率		0.01	
	動作完全性				
		不具合除去率		3.00	

ドキュメントバランスメトリクス				
	項目No	内容	参考値	設定目標値
要求仕様書	R1.	全体の記述量	100	
	R2.	対象ユーザとその使い方に関する記述	5	
	R3.	動作環境条件に関する記述量	10	
	R4.	主な機能に関する記述量	40	
	R5.	安全に関する記述、並びに非機能に関する記述量	30	
	R6.	システム全体構成に関する記述量	10	
	R7.	例外処理に関する記述量	5	
設計書	D1.	全体の記述量	R1×3	
	D2.	システム全体構成に関する記述量	5	
	D3.	機能ブロックの構成に関する記述量	5	
	D4.	機能ブロックの詳細に関する記述量	50	
	D5.	インタフェース・データに関する記述量	20	
	D6.	例外処理に関する記述量	20	
テスト仕様書	T1.	全体の記述量	R1×3	
	T2.	テスト環境に関する記述	5	
	T3.	テストの手順・条件に関する記述	10	
	T4.	正常系に関する記述	35	
	T5.	異常系・例外処理に関する記述	45	
	T6.	テスト完了基準に関する記述	5	

指標参考値の組織内での継続的な見直しについて

本ガイドでは、品質目標値の設定の目安とするために3章に各指標の定義とその参考値を掲載しています。このうち指標の参考値は本ガイド策定にあたりSECの活動に賛同いただいた企業へのヒアリングやアンケートなどをもとに収集したデータをベースにSECで算出したものです。これらの収集データは、さまざまな分野・さまざまな規模の組織やプロジェクトのデータが含まれているため、あくまでも検討の際の参考値として考える必要があります。

一般的には、こうした指標の基準値などの値は個々の組織においてその特性などを反映した値として蓄積され活用されていく形のほうが、より精度の高いものとなります。この点から、本ガイドで示した参考値を出発点として、各組織や企業内での経験を踏まえて、それぞれの最適な値を整備していくことが望ましいと考えられます。もちろん個々の組織における指標の標準値の整備に関しては、比較的、中長期の指標トレンドなどを分析し、かつ実際の開発プロジェクトへの適用結果などを見ながら、最適標準値を整備していく必要があります。また、その過程において、必要に応じて、これらの数値の統計処理などを行い、データの層別やはずれ値の処理なども行うことで、精度の高い企業・組織内標準値を整備することが可能となります。

2.5 プロファイリングの事例

ここでは簡易な事例を用いてシステムプロファイリング、プロジェクトプロファイリングのサンプルを紹介していきます。なお、付録Bに本ガイドによる高品質を作り込むための作業手順 Step-1 から Step-4 のサンプル事例を紹介します。

想定事例：移動型小型通信機器

開発の条件

主たる機能： 手のひら大の大きさで、パーソナルユースでのメールやWebなどのサービスを受けるための移動型小型通信端末です。主な機能はこれらの通信機能とともに、メール、Webの小型液晶上での表示（一部動画表示あり）やスケジュール管理、電話帳管理などの機能を有します。

主たるユーザ： ユーザは主として若手の社会人を中心として、出荷台数は約10000台を見込んでいます。

ソフトウェア： 機器を構成するソフトウェアの規模は約300万LOC、既存の携帯電話などのソフトウェアを再利用して開発しますが、携帯電話に比べると機能・構造ともにシンプルになる見通しです。また同時にソフトウェアの仕様も比較的単純でわかりやすくなっています。

開発プロジェクト： 開発プロジェクトは既存の携帯電話開発プロジェクトメンバによる混成・急造チームであり、ハード部隊などとの関係は必ずしも整理されていません。また、このため、開発プロセスなどについてもプロジェクトとしては未成熟な状況です。ただし、プロジェクトリーダについては熟練者が担当しています。

製品の位置づけ：開発するメーカとしては、出荷台数がそれほど多くなく、かつ、製品価格も比較的安価なため、万が一、製品トラブルが発生した場合でもリコール費用などが大きくならないと考えています。

プロファイリング

Step-1：システムプロファイリング

このシステムの場合、システム障害による人的損害が発生する確率はきわめて低いと考えられます。またシステム障害により引き起こされる経済損失については、以下のように求めることができます。

- **NU**：Number of Users：　対象ユーザ数
- **RD**：Ratio of Damaged users：　対象ユーザのうちシステム障害によって影響を受けるユーザの率
- **DI**：Damage of Impact：　システム障害が発生しユーザが1日間機器の使用ができないと考えた場合のユーザ1人あたりの損害額
- **ND**：Non Service Days：　システム障害によりユーザがサービスを受けられなくなる日数
- **ED**：Economic Damage：　システム障害によって引き起こされる最終的な経済的損失

以上の要素の間の関係は次式で表すことができます。

ED ＝ ［ND ×（NU × RD）］× DI

本事例の場合、以下のように考えると

NU ＝ 10000 人（ユーザ数）
RD ＝ 1.0（システム障害によりすべてのユーザが不利益をこうむると想定）
ND ＝ 2.0 日（システム障害はメーカ提供のパッチソフトインストールなどを使用し、2日程度で修正可能）
DI ＝ 5000 円（ユーザ1日あたり当該機器を利用することで、5000円程度のサービスを享受していると考える）
ED ＝ ［2.0 ×（10000 × 1）］× 5000
＝ 100,000,000 円

と見積もることができます。この結果から、経済損による被害想定をベースに本システムは、Type-2（Normal Quality Required）にタイプ分けすることができました。

なお、この例で利用したユーザ数や利用者の被害想定の方法などはあくまでも一つの例として提示したものです。

Step-2：プロジェクトプロファイリング

本項冒頭に示した開発の条件などから下表に示すように、タイプ補正係数は 0 となることがわかりますので、この事例では、Step-1 で求めたタイプ（TYPE-2）の参考値をそのまま利用すればよいということになります。

表 2-6：プロジェクトプロファイリング（想定事例の記入例）

			マイナス補正（－1）		基本		プラス補正（＋1）
①	ソフトウェア規模	□	極めて小さい	■	普通	□	極めて大きい
②	ソフトウェアの複雑さ	□	極めて単純	■	普通	□	極めて複雑
③	システム制約条件の厳しさ	□	制約ゆるい	■	普通	□	制約厳しい
④	仕様の明確度合い	□	極めて明確	■	普通	□	明確になっていない
⑤	再利用するソフトウェアの品質レベル	□	極めて高品質	■	普通	□	極めて品質低い
⑥	開発プロセスの整備度合い	□	整備できている	□	普通	■	整備できていない
⑦	開発組織の分業化・階層化の度合い	□	開発組織が単純	□	普通	■	開発組織が複雑
⑧	開発メンバのスキル	□	メンバスキル高い	■	普通	□	メンバスキル低い
⑨	プロジェクトマネージャの経験とスキル	■	ＰＭスキル高い	□	普通	□	ＰＭスキル低い
⑩	システム障害時のメーカ側損失額	■	極めて小さい	□	普通	□	極めて大きい
小計		-2				2	
合計ポイント数							0

2.6 システム障害の評価とシステムプロファイリングへの反映

システム障害の振り返りに基づくシステムプロファイリングへの反映

本章で紹介したシステムプロファイリングからプロジェクトプロファイリング、そしてそれに基づいた一連の品質目標値の設定の流れの中で、最初のステップにあたるシステムプロファイリングでは、対象システムを開発する前段階で、そのシステムが障害を起こした場合を想定して、システムの障害による影響を分析します。しかし、このシステム障害による影響分析は、対象とするシステムの仕様や利用シーンを丹念に分析したとしても、時としてきわめて想像が難しく、その評価が分かれてしまう場合があります。

システムに関する機能安全について定めたIEC61508では、こうしたシステムの障害に関して、実際にシステム障害が発生した場合の振り返りとその経験の次への反映、そしてシステムを開発する際の想定障害を考慮した対策立案などの重要性が述べられています。本章の2.2節～2.4節に記載した内容は、主に開発前段階での品質目標値の設定について述べていますが、実際に障害が発生した際の影響の評価やそこから次に向けた品質コントロールへの振り返りもきわめて重要な要素となります。

ここでは、システム障害に関する影響を評価する物差し（システム障害影響評価スケール）によってシステム障害が発生した場合に、システムがユーザや社会に与える影響を評価する考え方を整理します。

システム障害影響評価スケールの考え方

システムプロファイリングのための参考情報として、実際に類似のシステムなどがシステム障害や不具合を発生した場合に、ユーザの健康面、経済面でどのような被害が発生したか、またその結果、システムの周辺環境や社会にどのような影響を及ぼしたかという事実を正しく認識する必要があります。当然のことながら、実際のシステム障害では障害の程度によって、これらの被害や影響の度合いは異なってきます。システム障害影響評価スケールでは図2-5に示すように、実際にシステム障害が発生した際の評価によるシステム障害の階級をもとに、次のシステム開発や類似のシステム開発を行う際に、システムプロファイリングの①人的被害、②経済的被害

の想定額算出の参考に利用してシステムのタイプ分けに反映させることを目的としています。システム障害影響評価のための物指の事例を付録Cに示しますので、参考にしていただければと思います。

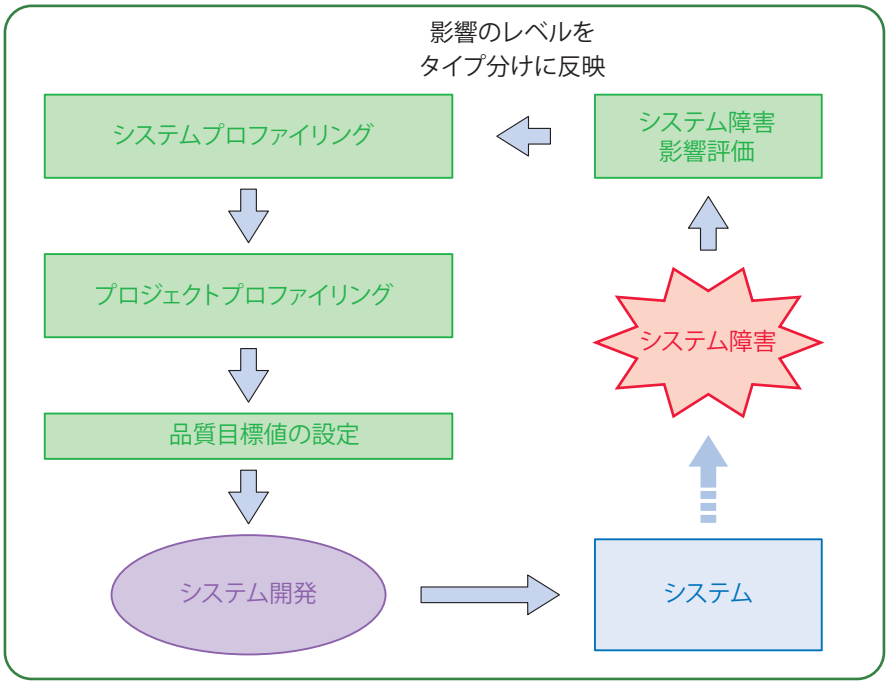


図2-5：システム影響障害評価の利用

第3章

品質指標の定義と参考値

組込みソフトウェアの品質を可視化するためには品質指標が必要となりますが、品質指標の定義などが曖昧だと、測る人や組織によって値がぶれてしまいます。また、計測した値が妥当な範囲にあるかどうかを判断する場合には、計測値の評価基準を決めておかなければなりません。そのため、本章では品質指標の定義とそれらの参考値、利用方法などを紹介します。

3.1 品質指標の定義と意味、利用法 ……52
3.2 品質指標のカテゴライズ ……54
3.3 品質指標 ― 利用上の注意 ……59
3.4 プロセス品質評価指標 ― 定義と参考値 ……63
3.5 プロダクト品質評価指標 ― 定義と参考値 ……79
3.6 基礎指標 ― 定義と参考値 ……105

3.1 品質指標の定義と意味、利用法

1章で紹介した通り、より品質の高いソフトウェアを開発するためには、実際の開発作業の質やその結果得られた成果物を品質という観点から確認する作業を確実に行う必要があります。このためには、これらの作業や成果物の品質を数値として客観的にとらえ、必要に応じてフィードバックループをまわすといった品質指標に基づいた品質の定量コントロールを実施することがポイントとなります。この品質定量コントロールの主役となるのが品質指標です。ここでは、本ガイドにおけるソフトウェア品質指標の考え方などを紹介していきます。一般的にこうした品質指標を考えるうえでは、特に以下のような点を考慮する必要があります。

- 指標の定義と意味、利用法
- 指標の計測方法と計測タイミング

計測する指標の定義を明確にしておく

たとえば、ソフトウェアを計測する代表的なものさし（品質指標）の一つとして、LOC（Lines of Code）と呼ばれるソースコードの行数があります。同じ処理内容を書いたソースコードであっても、C言語で書いた場合と、C++言語で書いた場合、あるいはアセンブリ言語で書いた場合ではLOCは異なった値を示します。また、C言語だけに絞った場合でも、コメント行を計測するか、何も書いていない行も計測するかなどによって、その値は大きく異なってきます。このように品質指標を利用してソフトウェアやその開発作業を定量的に計測し可視化する場合には、個々の指標の定義を明確にしておく必要があります。

計測したデータを開発でどのように利用するかを考えておく

品質指標も含めて開発の途中でとる数値は実際の開発に役立てて初めて意味のあるものといえます。よく耳にするのは、「さまざまな数値データを集めたものの、その利用方法がわからない」といったケースです。こうなってしまうとせっかくの努力が何の意味もなさなくなってしまうばかりか、数値をとることのアレルギーのみが残り、必要なデータさえも確認できなくなってしまう怖れがあります。こうした状況に

陥らないためにも、ソフトウェアや開発作業を品質指標を用いて可視化するためには、計測した数値データをどのように開発にフィードバックしていくかも含めて事前検討しておくことが重要です。

後の節で紹介するプロセス品質評価指標やプロダクト品質評価指標では、個々の指標でソフトウェアや開発作業のどのような側面を可視化しようとしているか、また、その数値が参考値未満の場合、どういうことが想定できるのかなどを参考情報として記載しています。たとえば、不具合発見率が低い場合、テストの内容が不十分であるか、品質が良いために不具合が出ないかのいずれかが考えられます。そこで、テストの内容を確認するとともにコードレビュー作業実施率やコーディングルール逸脱率などの数値を参考にして、どちらの可能性が高いかを検討します。レビューやテストを十分に実施しておりコーディングルール逸脱率も少ない場合、品質には問題がない可能性が高いと考えられます。

一方、不具合発見率が目標値よりも高ければ、不具合を出し切って品質が確保できたか、品質が低くさらなる品質確保の作業が必要であるかのいずれかが考えられます。この場合も先ほどと同様にコードレビュー作業実施率やコーディングルール逸脱率の数値を参考にして、どちらの可能性が高いかを検討します。

なお、レビューなどにおける品質の定量的な予測と評価について、SEC BOOKS[1]などを参考にされてもよいでしょう。

本ガイドでは、品質指標として簡単に計測でき、かつ、開発途中での品質面のフィードバックを行うための指標として、できるだけ厳選したものを掲載しています。これらを参考に、自組織やプロジェクトでどのような指標を採用してソフトウェア開発を可視化し、どのように開発にフィードバックしていくかなどを検討することで、計測するデータを絞り込み、かつ有効利用していただければと思います。

指標の計測方法や計測タイミングを考えておく

製品開発の最中にデータを集めることは開発側に少なくない負担をもたらします。このため指標を用いて数値データを取得する際にはどのような方法で計測するか、あるいはどのようなタイミングで計測するかなども十分に検討しておく必要があります。計測方法に関しては計測する指標のデータとしての精度とも関係しますが、不必要に高い精度のデータをとろうとして、計測に手間がかかったり、開発者に負荷がかかる方法は好ましいやり方とはいえません。指標の計測では、如何に小さい手間で意味のあるデータを収集するかを常に考えることが重要です。

---

[1]：SEC BOOKS「定量的品質予測のススメ」 オーム社 2008

3.2 品質指標のカテゴライズ

通常、ソフトウェアの開発では、要求分析、設計、実装やテストなどさまざまな作業を順次進めていくことで、最終的なソフトウェアが作成されていきます。ここで最終的なソフトウェアの品質をコントロールするためには、

①開発の過程で作成される各種の中間成果物の出来栄えを逐次チェックする
②開発過程で実施される品質向上に直結するであろう作業の実施状況を逐次チェックする
③最終的に開発されたソフトウェアの出来栄えをチェックする

といった多方面からの品質チェックが必要となります。こうした品質チェックや品質コントロールをより客観的に行うために、本ガイドでは品質評価指標（Evaluation Metrics）と品質評価指標を得るために計測すべき数値データとして基礎指標（Basic Metrics）という２つの種類の指標を利用します。品質評価指標は、プロセス品質評価指標とプロダクト品質評価指標から構成され、図3-1の意味づけになります。

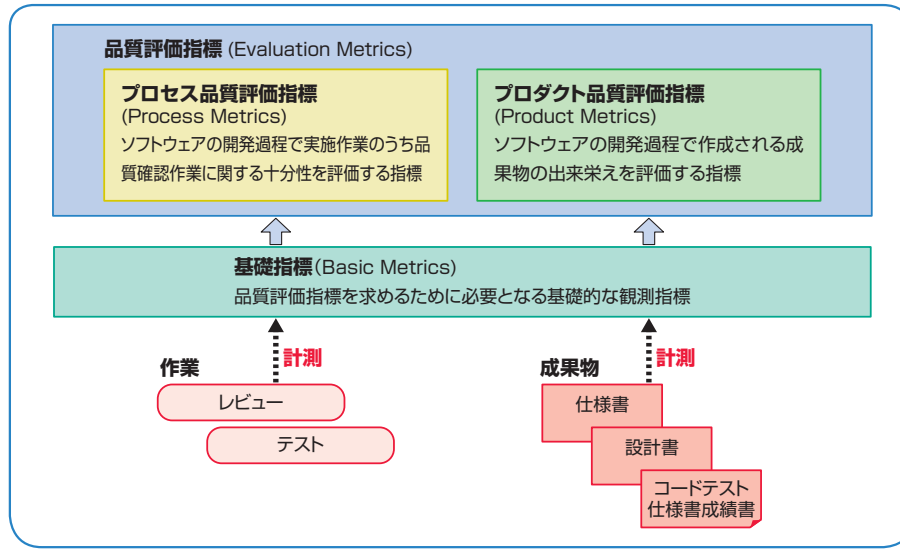


図3-1：品質指標の種類と意味づけ

Column

ソフトウェア工学での品質指標の体系的整理への取り組み

ソフトウェア工学の世界では、ソフトウェアを測る(ソフトウェアメトリクス)という課題に関して研究が続けられており、ソフトウェアに関するデータの収集が行われてきました。代表的なところでは、ISBSGという団体で世界各国からの3000を超えるITプロジェクトのソフトウェア開発データを収集したものなどがあります。

http://www.isbsg.org/japanese/index.htm

我が国ではSECが約1000件のエンタープライズプロジェクトの開発データを収集して整理分析した「ソフトウェア開発 データ白書(2006、2007、2008)」などがあります。これらはソフトウェアの生産性を中心に、さまざまな指標を用いて詳細にプロジェクトの輪郭を浮き彫りにしようとしたものです。

一方、組込みソフトウェアの分野を眺めてみると、こうしたデータ収集の試みはまだほとんど報告されておらず、そもそも開発のどのような側面をどのような指標を用いて計測していくかといった議論が足りているとは思えません。本ガイドをきっかけとして、組込みソフトウェアの分野でも、ISBSGのようなデータ収集が進み、数値でソフトウェア開発をとらえることができるようになっていくと良いのではないかと思います。

品質指標一覧

品質指標（Evaluation Metrics）			
ID	略称	名称	計測方法または計算式
プロセス品質評価指標：作業充当率			
PR10	RSRE	仕様レビュー作業充当率	仕様レビュー工数/仕様作成工数
PR11	RDRE	設計レビュー作業充当率	設計レビュー工数/設計作成工数
PR12	RCRE	コードレビュー作業充当率	コードレビュー工数/コード作成工数
PR13	RTRE	テストレビュー作業充当率	テストレビュー工数/テスト準備・確認工数
PR14	RTWE	テスト作業充当率	テスト工数/開発全工数
PR15	RORE	レビュー作業充当率	全レビュー工数/開発全工数
プロセス品質評価指標：作業実施率			
PR20	ERSR	仕様レビュー作業実施率	仕様レビュー工数/ソースコード全行数
PR21	ERDR	設計レビュー作業実施率	設計レビュー工数/ソースコード全行数
PR22	ERCR	コードレビュー作業実施率	コードレビュー工数/ソースコード全行数
PR23	ERTR	テストレビュー作業実施率	テストレビュー工数/ソースコード全行数
PR24	ERTW	テスト作業実施率	テスト工数/ソースコード全行数
PR25	EROR	レビュー作業実施率	全レビュー工数/ソースコード全行数
プロダクト品質評価指標：ドキュメント品質評価指標			
ドキュメントボリューム品質評価指標			
PD10	RSDV	要求仕様書ボリューム率	要求仕様書ボリューム/ソースコード全行数
PD11	RDDV	設計書ボリューム率	設計書ボリューム/ソースコード全行数
PD12	RTDV	テスト仕様書ボリューム率	テスト仕様書ボリューム/ソースコード全行数
ドキュメントバランス品質評価指標			
PD20	BSDD	要求仕様書バランス	要求仕様書内の各パートのページ数/要求仕様書ページ数の総和
PD21	BDDD	設計書バランス	設計書内の各パートのページ数/設計書全体ページ数の総和
PD22	BTDD	テスト仕様書バランス	テスト仕様書内の各パートのページ数/テスト仕様書ページ数の総和
プロダクト品質評価指標：コード品質評価指標			
コードボリューム品質評価指標			
PD30	FLOC	ファイル行数	基礎指標のファイル行数と同じ
PD31	MLOC	関数の行数	基礎指標の関数の行数と同じ
コード特性品質評価指標			
PD32	ROCS	制御文記述率	制御文数/ソースコード全行数
PD33	ROCL	コメント行記述率	コメント行数/ソースコード全行数
PD34	RDCR	コーディングルール逸脱率	コーディングルール逸脱数／ソースコード全行数
プロダクト品質評価指標：テスト品質評価指標			
テスト十分性品質評価指標			
PD40	DOTI	テスト密度	テスト項目数/ソースコード全行数
PD41	ROFC	不具合収束率	最終10%のテスト期間の不具合発見率/当初90%のテスト期間の不具合発見率
動作完全性品質評価指標			
PD42	ROFE	不具合修正率	修正済み不具合数/検出不具合数

基礎指標（Basic Metrics）			
ID	**略称**	**名称**	**計測方法または計算式**
ソースコードボリューム基礎指標			
B10	TLOC	ソースコード全行数	ファイル毎の行数の総和
B11	FLOC	ファイル行数	各ソースファイルに記述されたソースコードの規模
B12	MLOC	関数の行数	ソースコードを構成する関数一つあたりのソースコードの規模
B13	NOCS	制御文数	ファイル毎の制御文数
B14	CLOC	コメント行数	ファイル毎のコメント行数
B15	NDCR	コーディングルール逸脱数	コーディングルールを逸脱した延べ数
ドキュメントボリューム基礎指標			
B20	VOSD	要求仕様書ボリューム	仕様に関するドキュメントのページ数の総和
B21	VODD	設計書ボリューム	設計に関するドキュメントのページ数の総和
B22	VOTD	テスト仕様書ボリューム	テストに関するドキュメントのページ数の総和
工数ボリューム基礎指標			
B30	RETO	全レビュー工数	レビュー工数の総和
B31	RESP	仕様レビュー工数	仕様レビューにかかった工数の総和
B32	REDE	設計レビュー工数	設計レビューにかかった工数の総和
B33	RECO	コードレビュー工数	コードレビューにかかった工数の総和
B34	RETP	テストレビュー工数	テストに関するレビューにかかった工数の総和
B35	PETO	開発全工数	開発にかかった工数の総和
B36	PESP	仕様作成工数	仕様作成にかかった工数の総和
B37	PEDE	設計作成工数	設計にかかった工数の総和
B38	PECO	コード作成工数	コード作成にかかった工数の総和
B39	PETP	テスト準備・確認工数	テストの準備・確認作業にかかった工数の総和
B3A	PETE	テスト工数	テストにかかった工数の総和
テストボリューム基礎指標			
B40	NOTI	テスト項目数	テスト項目数の総和（ただし、実施していること）
B41	NOET	テスト実施項目数	実施したテスト項目数の総和（重複を含む）
B42	NODF	検出不具合数	単体テスト後に検出した不具合の総和
B43	NOEF	修正済み不具合数	検出した不具合の内、修正した不具合の総和
B44	ROFD	不具合発見率	検出不具合数/テスト実施項目数

品質評価指標のノーマライズと基礎指標

本ガイドでは前述したようにプロセス品質評価指標、プロダクト品質評価指標の2つのタイプの品質評価指標を利用したソフトウェアの可視化を目指しています。ここで問題となるのは、ソフトウェアの規模や開発プロジェクト規模（工数）です。たとえば、プロセス品質評価指標として「設計レビュー工数」などが考えられますが、その値が1人月であったとしても、全体の開発規模が1000人月のプロジェクトの場合と全体が100人月のプロジェクトでは、レビューの十分性という点でまったく異なる評価をせざるを得なくなります。また、プロダクト品質評価指標として「設計書のページ数」を考えてみます。設計書のページ数は開発しようとしているソフトウェアの規模が大きければ多くなることが考えられます。この場合、単純に「設計書のページ数」の値だけをみても、妥当なものであるかどうかは判断がつきません。このように、プロセス品質評価指標やプロダクト品質評価指標の一部では、計測対象部分のソフトウェア規模やその部分の開発に費やした全体工数などとの対比において評価する必要があります。

このため、本ガイドではプロセス品質評価指標やプロダクト品質評価指標を算出する際には、基礎指標として「設計レビュー工数」や「設計書のページ数」などの直接的な値も計測しますが、同時にそれらのノーマライズに必要となる値として、たとえばソースコードの全行数や開発の全工数なども計測します。なお、これらのノーマライズに必要な指標は、後述する各品質指標の計算式の分母として現れます。

3.3 品質指標 ― 利用上の注意

本ガイドでは、本章後半に示すようにプロセス品質評価指標、プロダクト品質評価指標、基礎指標のそれぞれについて、その定義や計測方法、利用方法や参考値などを紹介しています。これらの指標は原則として、システムの要求定義からテストまでを一つの組織内で開発することを想定して紹介しています。しかしながら、実際の開発では、開発の中の一部を切り出したり、既存のソフトウェアを流用したりと、さまざまな開発バリエーションが考えられます。以下に、開発スタイルのバリエーションを考慮した場合の注意点について紹介しておきます。

開発の部分委託など一部を切り出して行う開発の場合

実際の組込み開発のプロジェクトでは、初期の要求定義から最終段階のテストまでを一つのプロジェクトや組織で対応する場合もあれば、これらの作業の一部を外部に委託するプロジェクト形態、あるいは逆に開発の一部を請け負うというようなプロジェクト形態、また、機能を分割しいくつかの部門や複数の企業にまたがって開発するというようなさまざまなプロジェクト形態があります。本ガイドでは3.2節で述べたように、いくつかの品質指標を計測し利用することでプロジェクト全体として開発するソフトウェアの品質を可視化することを目標としています。しかしながら、これらの品質指標は仕様定義からテストまで開発のすべての作業、すべての機能を一つのプロジェクト内で行う場合には比較的確実に計測することができますが、開発作業の一部を切り出したような形のプロジェクトや逆に一部の作業を外部委託しているようなプロジェクト、機能を分散開発するプロジェクト、それらを統合するようなプロジェクトでは、本ガイドに記載した品質指標すべてを計測することは難しい場合があります。このため、このようなプロジェクトでは、本ガイドの利用に際して参考として掲載した品質指標のすべてを利用するのではなく、それぞれのプロジェクトの作業範囲やプロジェクト形態などを考慮して、計測可能な品質指標に絞って利用するなどの工夫をしていただくと良いと思います。すなわち、品質指標を用いて品質コントロールを実施する組織、プロジェクトや部門などでは、それぞれの状況に合わせて運用方法を検討し採用することをお勧めします。

部分開発における基礎指標の算出

上記のように、設計および実装だけを請け負って開発するプロジェクトの場合や、逆に要求仕様の策定と最終的なテストのみを行うようなプロジェクトの場合、評価する品質指標はそれぞれのプロジェクトで実施する作業（レビューなど）やその作業で作成する成果物（設計書、仕様書など）に関係する品質指標のみを評価する形になります。この場合、品質指標を算出するためのベースとする基礎指標（ソースコード全行数、開発工数等）についての考え方は以下を参考にしていただければと思います。

- **ソースコード全行数**

  ソースコード全行数については、作業の一部を請け負う場合などでも開発対象となるソフトウェア全体の行数を参考値として利用します。業務請負などでソースコード全体の行数が把握できていない場合には、発注元に確認するなどしてみてください。また、発注側も有効な品質指標を得るために発注先と協力しあうようにしてください。

- **開発工数**

  開発工数については、プロジェクトで実施するプロセスごとに計測します。ただし、テスト作業充当率およびレビュー作業充当率は開発全工数に対する値を計測しますので、たとえば、テスト作業のみを請け負っている業務などでは、開発全工数を計測できないことが考えられます。この場合、プロジェクトで実施する作業全体の工数を開発全工数として読み替えることも可能ですが、結果としてテスト作業充当率が100%になってしまうことが考えられます。こうしたプロジェクトでは後述する本ガイドの品質指標の参考値は参考とするのに適しません。この場合、テスト作業に関する評価は、テスト作業充当率ではなく、テスト作業実施率を用いるようにします。

品質指標の計測タイミング

本ガイドで示す品質指標に関しては、それぞれのプロジェクトでの作業状況などを考慮して適切なタイミングで計測する必要があります。図3-2に、一般的な計測のタイミングを示します。図中、SWP1-2などはESPRでのソフトウェアエンジニアリングプロセスのタスクを示します。

本ガイドでは品質の定量コントロールのために3.4節以降に示す品質指標を利用します。このうち、実際に計測をする指標である基礎指標群は予め見積もりを行い、開発を進めていくうちに徐々に精度を上げて行くようにします。計測するタイミングとしては、下記の通りとなります。

- 要求仕様書、設計書、テスト仕様書などのドキュメントボリュームは、各成果物のドラフトが完成した時点、または第1版が完成した時点で計測
- コードボリュームはソースコードのコーディングが終了した時点で計測
- 工数については、それぞれ該当する作業の着手から終了までをモニタして所要工数を計測
- テストボリューム関連の基礎指標は主にテストフェーズで計測

計測タイミングによる開発全工数、ソースコード全行数の考え方

実際に品質定量コントロールに利用するプロセス品質評価指標、プロダクト品質評価指標は、基礎指標値を加工して求めます。たとえば、仕様レビュー作業実施率を求める場合、仕様レビューが終了した時点で仕様レビューに要した工数を求め、これをソースコード全行数で割ることが必要になります。ところが通常、仕様レビューを終えた段階では実装は始まっていませんので、当然、この時点でのソースコード全行数は測れないため、仕様レビュー作業実施率を計算で求めることができなくなってしまいます。本ガイドで計算によって求める、作業充当率、作業実施率などは、計算の過程で、開発工数やソースコード全行数を用いますが、要求定義、設計、実装など開発の途中段階ではこれらの実数値は求めることができません。このため、開発途中段階で作業充当率、作業実施率を求める場合には、分母となる開発工数やソースコード全行数は推定値（見積もり値）を利用することになります。

これらの推定値については、基本的に、過去の類似開発の値などを参考に、当該システムでの仕様の変更量などを考慮して、概数値を求めて利用する必要があります。

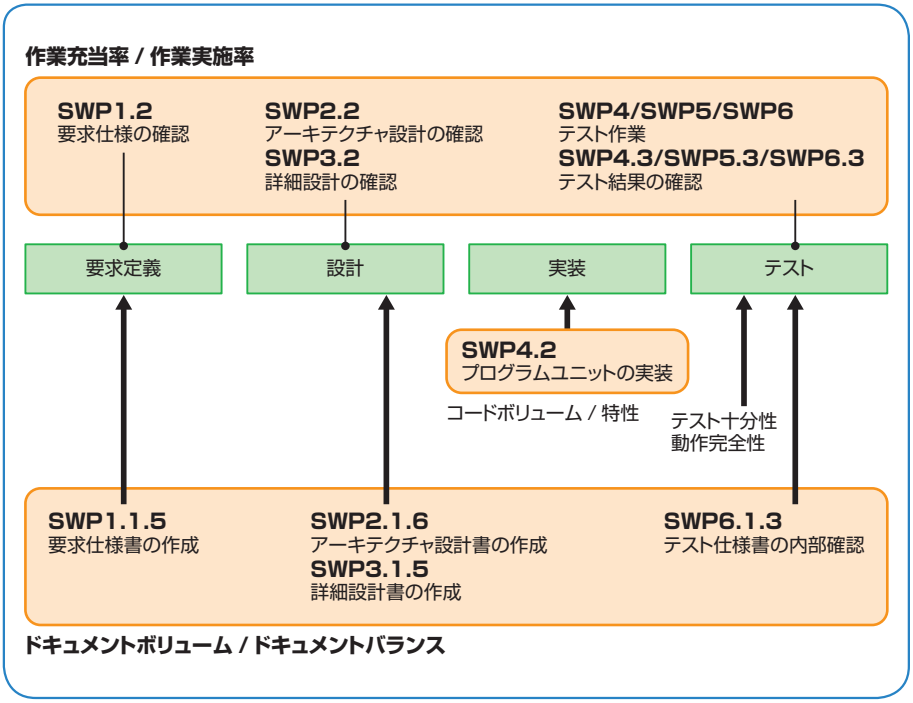


図3-2：指標の計測タイミング

3.4 プロセス品質評価指標 ― 定義と参考値

プロセス品質評価指標とは

プロセス品質評価指標とはソフトウェア開発の過程で実施される作業を計測し評価するための品質指標です。高品質なソフトウェアを作るためには開発の過程で、品質面のチェックや確認などの作業が適切に行われなければなりません。プロセス品質評価指標では、これらの作業の十分性を評価します。

プロセス品質評価指標の評価対象

プロセス品質評価指標の評価対象は、開発の中で実施される作業そのものです。たとえば、仕様検討作業の適切さや、設計書などのレビュー作業の十分性などが対象となります。本ガイドではソフトウェア開発で実施される作業の中でも、特に品質作り込みに直接関係するレビューやテストなど、品質確認に必要な作業の適切さや十分性などを評価します。

プロセス品質評価指標の評価の観点

プロセス品質評価指標の評価対象である“作業”はある意味で大変にとらえづらいものです。一般的には、「作業が適切に実施されているか」「作業が十分に行われているか」「作業の抜けが無いか」などが評価の観点になります。このうち、作業の適切さや十分性については、「作業充当率：ソフトウェア開発に関わる作業全体の中で、その作業にどの程度工数（量）を割いているかをみることで、作業の相対的な十分性を評価」「作業実施率：その作業の作業対象物のボリュームに応じて適切な作業工数（量）を充てているかをみることで、作業のきめの細かさを評価」といった２つの視点から、作業の量的な十分性と作業の質的な十分性を評価していきます。

作業充当率

作業充当率とは、ソフトウェア開発の過程で作成される中間成果物や最終成果物のレビューに費やした工数およびテスト工数をプロセスに要した作業工数との相対で評価することで、作業工数の十分性を示します。仕様、設計、コード作成、テス

ト作成（準備）および結果の作成についてそれぞれのレビューに費やした作業の工数をそれぞれのプロセスごとの作業全体工数の相対値で、またテストについてはテスト作業に費やした工数を開発作業全体工数との相対値で評価します。

たとえば、「設計レビュー作業充当率」は設計レビューに要した作業工数を実際の設計作業プロセス全体に要した工数で割ったものとなります。一般的に設計作業に多くの工数を割いている場合には、それだけ設計が難しかったり、設計量が多かったりといった状況が考えられますので、それに応じて、設計レビューを充実することが望まれます。作業充当率の考え方は、そうした作業の量としての十分性を評価するための指標と位置づけられます。もちろん品質を担保するためのレビューやテストの工数は開発工数と相対的に適切な工数をかけるのが望ましいといえます。すなわち、作業充当率は、単に値が大きければ良いといったものではなく、対象とするシステムの要求品質などとの関係において、その目標値を設定する必要があります。

作業実施率

作業実施率は、ソフトウェア開発の過程で作成される中間成果物や最終成果物のレビューを行う工数およびテスト工数をソフトウェアの規模との相対で評価することで、作業の質的な十分性を示す指標です。仕様、設計、ソースコード作成、テスト作成（準備および結果の作成）についてそれぞれのレビューに費やした作業の工数を、またテストについてはテスト作業に費やした工数を全体ソフトウェア規模との相対値で評価します。ソフトウェア規模としてはソースコード全行数（ID：B10、TLOC）を代替指標として使用します。たとえば、設計レビュー作業実施率の場合、設計レビューに要した工数を開発しているシステムのソースコード全行数で割った値となります。この値は、単位行数あたりどの程度の工数をかけて設計のレビューを実施したかを表し、ソフトウェアのボリュームに対して適切な設計レビューが実施されたかどうかの判断指標となります。もちろん品質を担保するためのレビューやテストの工数は一般的には、多くの工数をかけるのが良いのであり、作業実施率の値が大きいことは質が高いことを示しますが、これについても対象とするシステムの要求品質や開発コストなどと合わせて、その目標値を考えていく必要があります。

プロセス品質評価指標の評価対象範囲

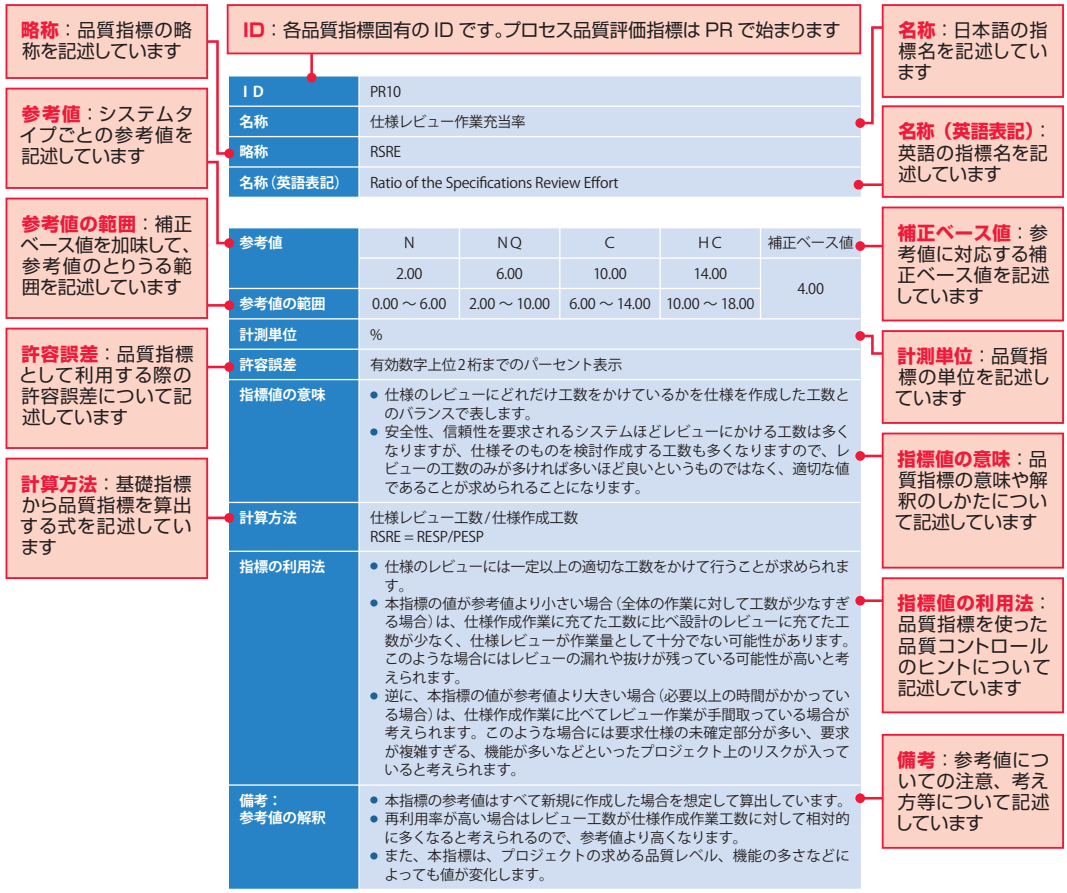
ソフトウェアを作るには、要求仕様検討（仕様作成）、設計、実装（コーディング）、テスト等、大まかに4つのプロセスを経ます。プロセスの詳細については、「組込みソフトウェア向け開発プロセスガイド」（ESPR）を参考にしてください。プロセス品質評価指標としては、作業充当率、作業実施率の2つの考え方がありますが、これらの4プロセスを対象として、

仕様レビュー作業充当率/実施率
設計レビュー作業充当率/実施率
コードレビュー作業充当率/実施率
テストレビュー作業充当率/実施率
テスト作業充当率/実施率
レビュー作業充当率/実施率

といった指標を用いて、レビューやテストなどの作業がきちんと行われたかどうかを確認していくことで各工程の成果物の品質を確保していきます。

プロセス品質評価指標　定義表の読み方

次に、プロセス品質評価指標について1つずつ、解説します。各表は次のような構成になっています。



ID	PR10					
名称	仕様レビュー作業充当率					
略称	RSRE					
名称（英語表記）	Ratio of the Specifications Review Effort					
参考値	N	NQ	C	HC	補正ベース値	
	2.00	6.00	10.00	14.00	4.00	
参考値の範囲	0.00 ～ 6.00	2.00 ～ 10.00	6.00 ～ 14.00	10.00 ～ 18.00		
計測単位	%					
許容誤差	有効数字上位2桁までのパーセント表示					
指標値の意味	● 仕様のレビューにどれだけ工数をかけているかを仕様を作成した工数とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、仕様そのものを検討作成する工数も多くなりますので、レビューの工数のみが多ければ多いほど良いというものではなく、適切な値であることが求められることになります。					
計算方法	仕様レビュー工数/仕様作成工数 RSRE = RESP/PESP					
指標の利用法	● 仕様のレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標の値が参考値より小さい場合（全体の作業に対して工数が少なすぎる場合）は、仕様作成作業に充てた工数に比べ設計のレビューに充てた工数が少なく、仕様レビューが作業量として十分でない可能性があります。このような場合にはレビューの漏れや抜けが残っている可能性が高いと考えられます。 ● 逆に、本指標の値が参考値より大きい場合（必要以上の時間がかかっている場合）は、仕様作成作業に比べてレビュー作業が手間取っている場合が考えられます。このような場合には要求仕様の未確定部分が多い、要求が複雑すぎる、機能が多いなどといったプロジェクト上のリスクが入っていると考えられます。					
備考：参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● 再利用率が高い場合はレビュー工数が仕様作成作業工数に対して相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標は、プロジェクトの求める品質レベル、機能の多さなどによっても値が変化します。					

図3-3：プロセス品質評価指標 定義表の読み方

プロセス品質評価指標

ＩＤ	PR10
名称	仕様レビュー作業充当率
略称	RSRE
名称（英語表記）	Ratio of the Specifications Review Effort

参考値	N	NQ	C	HC	補正ベース値
	2.00	6.00	10.00	14.00	4.00
参考値の範囲	0.00 ～ 6.00	2.00 ～ 10.00	6.00 ～ 14.00	10.00 ～ 18.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 仕様のレビューにどれだけ工数をかけているかを仕様を作成した工数とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、仕様そのものを検討作成する工数も多くなりますので、レビューの工数のみが多ければ多いほど良いというものではなく、適切な値であることが求められることになります。				
計算方法	仕様レビュー工数/仕様作成工数 RSRE = RESP/PESP				
指標の利用法	● 仕様のレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標の値が参考値より小さい場合（全体の作業に対して工数が少なすぎる場合）は、仕様作成作業に充てた工数に比べ設計のレビューに充てた工数が少なく、仕様レビューが作業量として十分でない可能性があります。このような場合にはレビューの漏れや抜けが残っている可能性が高いと考えられます。 ● 逆に、本指標の値が参考値より大きい場合（必要以上の時間がかかっている場合）は、仕様作成作業に比べてレビュー作業が手間取っている場合が考えられます。このような場合には要求仕様の未確定部分が多い、要求が複雑すぎる、機能が多いなどといったプロジェクト上のリスクが入っていると考えられます。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● 再利用率が高い場合はレビュー工数が仕様作成作業工数に対して相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標は、プロジェクトの求める品質レベル、機能の多さなどによっても値が変化します。				

ＩＤ	PR11
名称	設計レビュー作業充当率
略称	RDRE
名称（英語表記）	Ratio of the Design Review Effort

参考値	N	NQ	C	HC	補正ベース値
	2.00	6.00	10.00	14.00	4.00
参考値の範囲	0.00 ～ 6.00	2.00 ～ 10.00	6.00 ～ 14.00	10.00 ～ 18.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 設計のレビューにどれだけ工数をかけているかを設計に費やした工数（設計プロセスの工数）とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、設計作業そのものも工数は多くなります。従って多ければ多いほど良いというものではなく、適切な値であることが求められることになります。				
計算方法	設計レビュー工数/設計作成工数 RDRE = REDE/PEDE				
指標の利用法	● 設計のレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標の値が参考値より小さい場合は、設計プロセスに充てた工数に対する設計レビューの工数の割合が低く、結果的に設計レビューの漏れや抜けが発生する可能性が高いと考えることができます。 ● 逆に、本指標の値が参考値より大きい場合には、設計作業に充てた工数に比べ設計のレビューに工数がかかっていることを意味します。これは仕様が曖昧であるとか、構造が複雑すぎる、機能が多い、といったプロジェクト上のリスクによって設計レビューが手間取っている可能性を示唆する場合もあります。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー工数が設計作業に対して相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標はプロジェクトの求める品質レベル、構造の複雑さ、異常系の処理などにより、値が変化します。				

ID	PR12
名称	コードレビュー作業充当率
略称	RCRE
名称（英語表記）	Ratio of the Code Review Effort

参考値	N	NQ	C	HC	補正ベース値
	2.00	3.50	5.00	6.50	1.50
参考値の範囲	0.50 ～ 3.50	2.00 ～ 5.00	3.50 ～ 6.50	5.00 ～ 8.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● ソースコードのレビューにどれだけ工数をかけているかをソースコード作成工数（実装プロセスの工数）とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、ソースコードを作成する工数も相対的に多くなると考えられますので、多ければ多いほど良いというものではなく、適切な値であることが求められます。				
計算方法	コードレビュー工数/コード作成工数 RCRE = RECO/PECO				
指標の利用法	● ソースコードのレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標の値が参考値より小さい場合には、実装に要した工数に比べソースコードレビューに充てた工数が十分でなく、ソースコード上に不具合が残りやすいと考えることができます。 ● 逆に、本指標の値が参考値より大きい場合には、実装に要した工数に対し、ソースコードレビューに充てた工数が多すぎるということで、ソースコードの理解性、保守性等が良くないなど、プロジェクト上のリスクが入っていることが考えられます。 ● 特にソースコードレビューは、とかくレビュー箇所が偏りがちになる傾向があるので、本指標による数値だけを鵜呑みにするのではなく、モジュールごと分布を見るなどして重要な部分をレビューできているか、また、レビュー箇所に漏れはないかなど、カバレッジの概念を利用して確認すると良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの利用率が高い場合はソースコード作成工数に対し、レビュー工数が相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標はプロジェクトの求める品質レベル、構造の複雑さ、異常系の処理などにより、値が変化します。				

ID	PR13
名称	テストレビュー作業充当率
略称	RTRE
名称（英語表記）	Ratio of the Test Review Effort

参考値	N	NQ	C	HC	補正ベース値
	2.00	3.50	5.00	6.50	1.50
参考値の範囲	0.50 ～ 3.50	2.00 ～ 5.00	3.50 ～ 6.50	5.00 ～ 8.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● テスト作業のレビュー（テスト仕様・項目・結果の作成等のレビュー）にどれだけ工数をかけているかをテスト準備・確認作業の工数とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、これらのシステムではテストレビュー対象物の作成そのものも相対的に多くなりますので、多ければ多いほど良いというものではなく、適切な値であることが求められます。				
計算方法	テストレビュー工数/テスト準備・確認工数 RTRE = RETP/PETP				
指標の利用法	● テストの作業のレビューには一定以上の適切な工数をかけることが求められます。 ● また、テストを開始する前に行うレビュー：テスト仕様書、テストスクリプトのレビュー等と、テスト実行後に行うレビュー：テスト結果、テスト報告書のレビュー等の2つの計測ポイントがあります。それぞれ、以下の観点で指標を利用することが考えられます。 ● テストを開始する前に行うレビューの利用ポイント：テスト開始前のレビューを十分に行うことによって、不具合の検出が不十分になることを事前に防ぐことを狙います。本指標値が低い場合にはテストそのものの質の確認が不十分であり、結果的に不具合の検出が不十分となる可能性があると考えられます。逆に本指標値が高い場合にはレビュー方法が良くない、テスト仕様書のレビューが仕様レビューになってしまっている、テスト仕様書そのものの質が良くなくレビューに手間取っているなどの可能性があると考えられます。 ● テスト実行後に行うレビューの利用ポイント：テスト十分性の検証の指標となります。本指標値が低い場合には、テスト結果の判断が十分でなく、不具合を見逃している可能性があると考えられます。逆に本指標値が高い場合にはプロダクトの品質が低く不具合が数多く検出されて、その結果判断に時間を要している、などのプロジェクト上のリスクが入っていることが考えられます。 ● いずれの場合もレビューの値が適正でない理由を明確にして対策を行いましょう。				
備考：参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー工数が相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標はプロジェクトの求める品質レベル、複雑さ、データのバリエーションの多さなどにより、値が変化します。				

ＩＤ	PR14
名称	テスト作業充当率
略称	RTWE
名称（英語表記）	Ratio of the Test Work Effort

参考値	N	NQ	C	HC	補正ベース値
	30.00	35.00	40.00	45.00	5.00
参考値の範囲	25.00 ～ 35.00	30.00 ～ 40.00	35.00 ～ 45.00	40.00 ～ 50.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 成果物である実行コードそのものをテストする作業（テスト準備・実施から確認までの作業すべて）にどれだけ工数をかけているかを開発全工数とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどテスト作業にかける工数は多くなりますが、全体の工数に対する比率なので、多ければ多いほど良いというものではなく、適切な値であることが求められます。				
計算方法	テスト工数/開発全工数 RTWE = PETE/PETO				
指標の利用法	● テストには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合は、テストの効率が良いということも考えられますが、テストが不十分で不具合の検出が不十分であるとも考えられます。 ● 逆に、本指標値が参考値より大きい場合は、テスト実行前の作業（テスト仕様書の作成等）が不十分でテスト設計に工数がかかっている、プロダクトの品質が良くなく、何度もテストを繰り返すなどしていることが考えられますので、テスト対象のプロダクトの品質が良くない、製品の理解性テスト容易性等が低いなどのプロジェクト上のリスクが入っていることが考えられます。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● 設計やコードの再利用率が高い場合はテストに関する工数が全開発工数に対して相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標はプロジェクトの求める品質レベル、複雑さ、データのバリエーションの多さなどにより、値が変化します。				

ＩＤ	PR15
名称	レビュー作業充当率
略称	RORE
名称（英語表記）	Ratio Of the Review Effort

参考値	N	NQ	C	HC	補正ベース値
	4.00	8.00	12.00	16.00	4.00
参考値の範囲	0.00 ～ 8.00	4.00 ～ 12.00	8.00 ～ 16.00	12.00 ～ 20.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 開発に関わるすべてのレビューにどれだけ工数をかけているかを開発全工数とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなりますが、全体の工数に対する比率なので、多ければ多いほど良いというものではなく、適切な値であることが求められます。				
計算方法	全レビュー工数/開発全工数 RORE = RETO/PETO				
指標の利用法	● レビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、開発に費やした工数全体に比べレビューに充てた工数が少なく、レビューによる不具合の検出が不十分となっている可能性が考えられます。 ● 逆に、本指標値が参考値より大きい場合には、レビューの進め方に問題がある、プロダクトの品質が良くない、製品の理解性、レビュー容易性等が良くないことなどのプロジェクト上のリスクが入っていることが考えられます。また、適切なレビュー工数であっても、レビューそのものの質が良くないこともよくあります。レビュー記録等でレビューそのものの質を評価することも検討しましょう。 ● レビューの進め方のポイントについては、4章にもヒントを掲載していますので参考にすると良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー工数が相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標はプロジェクトの求める品質レベル、複雑さ、データのバリエーションの多さなどにより、値が変化します。				

ID	PR20
名称	仕様レビュー作業実施率
略称	ERSR
名称（英語表記）	Execution Ratio of the Specifications Review

参考値	N	NQ	C	HC	補正ベース値
	7.20	9.60	12.00	14.40	2.40
参考値の範囲	4.80 ～ 9.60	7.20 ～ 12.00	9.60 ～ 14.40	12.00 ～ 16.80	
計測単位	人時／ KLOC				
許容誤差	有効数字上位3桁まで				
指標値の意味	● 仕様のレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	仕様レビュー工数/ソースコード全行数 ERSR = RESP/TLOC				
指標の利用法	● 仕様のレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合（規模あたりのレビュー工数が少なすぎる場合）は、仕様の確認が不十分であり、レビューの漏れや抜けが残っている可能性が高いと考えられます。このため、レビューの内容を再確認する必要があります。 ● 逆に、本指標値が参考値より大きい場合（必要以上の工数がかかっている場合）は、要求仕様の未確定部分が多いために仕様レビューが仕様検討の場になってしまっているとか、要求が複雑すぎる、機能が多いといったプロジェクト上のリスクが入っていることが考えられますので、レビュー結果が仕様にきちんと反映されているかどうかを確認する必要があります。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー対象が規模に対して少なくなると考えられるので、参考値より少なく見積もることができます。 ● また、本指標は、プロジェクトの求める品質レベル、機能の多さなどによっても値が変化します。				

ID	PR21
名称	設計レビュー作業実施率
略称	ERDR
名称（英語表記）	Execution Ratio of the Design Review

参考値	N	NQ	C	HC	補正ベース値
	7.20	9.60	12.00	14.40	2.40
参考値の範囲	4.80 ～ 9.60	7.20 ～ 12.00	9.60 ～ 14.40	12.00 ～ 16.80	
計測単位	人時／KLOC				
許容誤差	有効数字上位3桁まで				
指標値の意味	● 設計のレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	設計レビュー工数/ソースコード全行数 ERDR = REDE/TLOC				
指標の利用法	● 設計のレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、設計のレビューが甘く結果として設計の漏れや抜けが残りやすくなります。 ● 逆に、本指標値が参考値より大きい場合には、仕様が曖昧なために設計レビューがしづらくなっている、設計構造が複雑すぎる、機能が多い、といったプロジェクト上のリスクが入っていることが考えられます。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー対象が規模に対して少なくなると考えられるので、参考値より少なく見積もることができます。 ● また、本指標はプロジェクトの求める品質レベル、構造の複雑さ、異常系の処理などにより、値が変化します。				

ＩＤ	PR22
名称	コードレビュー作業実施率
略称	ERCR
名称（英語表記）	Execution Ratio of the Code Review

参考値	N	NQ	C	HC	補正ベース値
	3.60	4.80	6.00	7.20	1.20
参考値の範囲	2.40 ～ 4.80	3.60 ～ 6.00	4.80 ～ 7.20	6.00 ～ 8.40	
計測単位	人時／ KLOC				
許容誤差	有効数字上位3桁まで				
指標値の意味	● ソースコードのレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	コードレビュー工数/ソースコード全行数 ERCR = RECO/TLOC				
指標の利用法	● ソースコードのレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、対象ソースコードの必要な部分に十分目が行き届かずソースコード上の不具合が残りやすくなります。 ● 逆に、本指標値が参考値より大きい場合には、ソースコードの理解性、保守性等が良くない、レビューの効率が良くないなどのプロジェクト上のリスクが入っていることが考えられます。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー対象が規模に対して少なくなると考えられるので、参考値より少なく見積もることができます。 ● また、本指標はプロジェクトの求める品質レベル、構造の複雑さ、異常系の処理などにより、値が変化します。				

ＩＤ	PR23
名称	テストレビュー作業実施率
略称	ERTR
名称（英語表記）	Execution Ratio of the Test Review

参考値	N	NQ	C	HC	補正ベース値
	6.00	8.00	10.00	12.00	2.00
参考値の範囲	4.00 ～ 8,00	6.00 ～ 10.00	8.00 ～ 12.00	10.00 ～ 14.00	
計測単位	人時／ KLOC				
許容誤差	有効数字上位３桁まで				
指標値の意味	● テストにかかわる作業のレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	テストレビュー工数/ソースコード全行数 ERTR = RETP/TLOC				
指標の利用法	● 本指標値が参考値より小さい場合はテスト作業に対するレビューが不十分であり、テスト自身の不具合が残っているなどの可能性があります。 ● 逆に本指標値が参考値より大きい場合にはテスト仕様書や成績書などの出来が悪くレビューに手間取っている可能性があります。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー対象が規模に対して少なくなると考えられるので、参考値より少なく見積もることができます。 ● また、本指標はプロジェクトの求める品質レベル、複雑さ、データのバリエーションの多さなどにより、値が変化します。				

ＩＤ	PR24
名称	テスト作業実施率
略称	ERTW
名称（英語表記）	Execution Ratio of the Test Work

参考値	N	NQ	C	HC	補正ベース値
	34.00	51.00	68.00	85.00	17.00
参考値の範囲	17.0 ～ 51.0	34.0 ～ 68.0	51.0 ～ 85.0	68.0 ～ 102.0	
計測単位	人時／ KLOC				
許容誤差	有効数字上位３桁まで				
指標値の意味	● 一定の規模あたり、テスト作業（テスト準備・実施から確認までの作業すべて）にどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどテストにかける工数は多くなります。				
計算方法	テスト工数/ソースコード全行数 ERTW = PETE/TLOC				
指標の利用法	● テストには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、開発対象規模に対し、テスト作業の割合が小さく、結果として不具合の検出が不十分であると考えられます。 ● 逆に、本指標値が参考値より大きい場合には、必要以上の時間がかかっており、プロダクトの品質が良くない、製品の理解性、テスト容易性等が良くないことなどのプロジェクト上のリスクが入っていることが考えられます。				
備考： 参考値の解釈	● 1人月あたりトータルの生産性を1KLOC/人月と考えた場合（1人月=155人時換算）、NQではテストフェーズに全工数の約1/3を充当すると考え、テスト作業実施率は約50人時（155×0.3）を基準としています。 ● テスト作業は再利用があったとしてもすべてのソースコードに対してテストを行うことを原則としているので、本指標は、再利用率による影響はありません。 ● 本指標は簡易的にソースコード規模＝ソースコード全行数でノーマライズしていますが、テストはオブジェクトコードでの相対量を計測するとより精度の高い評価を行うことができます。計測可能な場合は、ソースコード全行数からコメント全行数および空白全行数を除いた値で評価すると良いでしょう。				

ＩＤ	PR25
名称	レビュー作業実施率
略称	EROR
名称（英語表記）	Execution Ratio Of the Review

参考値	N	NQ	C	HC	補正ベース値
	24.00	32.00	40.00	48.00	8.00
参考値の範囲	16.0 ～ 32.0	24.0 ～ 40.0	32.0 ～ 48.0	40.0 ～ 56.0	
計測単位	人時／ KLOC				
許容誤差	有効数字上位３桁まで				
指標値の意味	● 開発に関わるすべてのレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	全レビュー工数/ソースコード全行数 EROR = RETO/TLOC				
指標の利用法	● レビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、ソフトウェアの規模に対してレビューの工数が足りておらず、結果としてレビューによる不具合の検出が不十分な可能性があると考えられます。 ● 逆に、本指標値が参考値より大きい場合には、プロダクトの品質が良くない、製品の理解性、レビュー容易性等が良くないことなどのプロジェクト上のリスクが入っていることが考えられます。また、レビューそのものの進め方に問題がある場合も考えられます。また、適切なレビュー工数であっても、レビューそのものの質が良くないこともあります。レビュー記録等でレビューそのものの質を評価することも検討しましょう。 ● レビューの進め方のポイントについては、４章にもヒントを掲載していますので参考にすると良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● NQのソフトウェアの場合、1KLOCのソフトウェアの仕様からコードのレビューに約３人日、テストのレビューに１人日を費やすことを念頭に参考値を提示しています。 ● ソフトウェアの再利用率が高い場合はレビュー工数が相対的に多くなると考えられるので、参考値より高くなります。 ● また、本指標はプロジェクトの求める品質レベル、複雑さ、データのバリエーションの多さなどにより、値が変化します。				

3.5 プロダクト品質評価指標 ― 定義と参考値

プロダクト品質評価指標

ソフトウェア開発の過程では、設計書やソースコード、テスト仕様書などさまざまな成果物（ドキュメント）が作成されます。プロダクト品質評価指標はこうした開発の過程で作成される各種の中間成果物や最終段階で作成される成果物の出来栄えを評価するための品質指標です。

プロダクト品質評価指標の評価対象

多くの組込みソフトウェアの開発では、上述したように、開発の中間成果物として作成される仕様書や設計書などのドキュメントやソースコード、あるいはテスト仕様書、テストデータなどが主たる計測評価の対象となります。また、最終段階の成果物もほぼこれらと同じものが最終版として用意される場合がほとんどですので、これらの最終段階での成果物もまた対象ということになります。また、これ以外にも、最終成果物としてのソフトウェア製品やシステム製品としてのテスト作業、テスト結果等もプロダクト品質評価指標の評価対象と考えていきます。

プロダクト品質評価指標の評価の観点と種類

プロダクト品質評価指標の評価対象である、成果物自身は目に見えるものですが、その品質を評価しようとすると、途端にとらえにくいものになります。本ガイドではプロダクトの持つ特質の中から、比較的容易に測ることのできるもの ― たとえばページ数、行数などのボリューム、不具合数 ― などを計測し、特に品質に結びつく要素が反映されているのかどうかといった観点で評価します。

また、プロダクト品質評価指標は、評価する対象物によって、ドキュメント、ソースコード、テストの3つに分けて考えることができます。ISO/IEC9126では内部品質測定対象としてドキュメントおよびソースコード、外部品質測定対象としてテスト、と分類し、計測観点を示していますので、これらも参考にしていただければと思います。

プロダクト品質評価指標の評価対象範囲

近年の組込みソフトウェアはさまざまな機能の複合体となっています。たとえば、自動車というソフトウェアを考えると、自動車にはエンジンやブレーキ、エアコンなどさまざまな機能が搭載され、それぞれの機能を司るコンピュータユニット（ECU）毎にソフトウェアが実装され、それらが専用ネットワークなどの通信機能を通じて連係動作しています。このため、同じ自動車のなかでも個々の機能や部分によって、ソフトウェアに求められる特質は異なります。従って、プロダクト品質評価指標の評価では、システムを構成する要素を見きわめ、求められる品質特性が同一と考えられる範囲を対象に評価する必要があります。

プロダクト品質評価指標の概要

①ドキュメント品質評価指標

ドキュメント品質評価指標とは、ソフトウェア開発の過程で作成される中間成果物や最終成果物のうち、いわゆる仕様書、設計書などの文書（ドキュメント）の品質を評価するための指標です。組込みソフトウェアの開発の中で作成されるドキュメントには要求仕様書や設計書、テスト仕様書などがあります。

ドキュメント品質評価指標は、

(i)純粋なドキュメントの規模あたりのボリュームを評価するドキュメントボリューム品質評価指標

(ii)ドキュメントの記述内容面のバランスを評価するドキュメントバランス品質評価指標

の２種類の指標を利用します。

なお、このドキュメント品質評価指標の対象となる要求仕様書、設計書、テスト仕様書に記載されるべき内容や位置づけについては「組込みソフトウェア向け開発プロセスガイド（ESPR）」を参照してください。

ドキュメントボリューム品質評価指標

ドキュメントが品質という視点から見て適切なものかどうかを測る第１の指標として、そのボリューム（ページ数）が妥当なものであるかどうかという指標を考えることができます。ただし、これらのドキュメントのページ数は、当然のことながら開発しているソフトウェアの規模によって変わりますし、ドキュメントのフォーマットなどによっても変わってきます。このため、単純にドキュメントのページ数を数えるだけではまったく意味をもちません。

本ガイドのドキュメントボリューム品質評価指標では、まず、ページ数の計測ルールとして、どのようなフォーマットで作成したとしても、そこに記載されている文字情報や図表の情報について、1ページあたり2000字（40字×50行）換算でおおよそのページ数に関する基礎指標として求め、さらに全体のページ数をソフトウェア規模で割ることで、単位規模（1 KLOC）あたりどの程度のドキュメントを整備しているかを評価していきます。

なお、本ガイドではドキュメントボリュームについて単位規模でのノーマライズを行いますが、当然、システムの特徴―たとえばGUIの占める部分が多い、制御ロジックの占める部分が多いなど―によってベースとなるシステムのコードサイズは大きく変わります。このため、ドキュメントボリュームを評価する際には、個々のシステムの作りなど特徴を十分に考慮した評価が必要となります。

ドキュメントバランス品質評価指標

また一方で、ドキュメントのボリュームとしては問題なくとも、ドキュメントとして盛り込むべきことが抜けていたりすると、開発の大きな支障になってしまいます。このため本ガイドでは要求仕様書、設計書、テスト仕様書に標準的に盛り込むべき項目と各ドキュメント全体を100とした場合に、各項目がどの程度の記述量が含まれているかのバランスを表す、ドキュメントバランス品質指標を用意しています。たとえば、要求仕様書の中で、機能要求に関する記述は全体の何パーセントか、あるいは非機能要求に関する記述は全体の何パーセントを占めているかなどを測ることで、要求仕様書の記載内容にまで踏み込んで、ドキュメントの内容面の評価をすることを狙っています。なお、ここでベースとする各ドキュメントの記載項目などはESPRの定義を参考にしています。

②コード品質評価指標

コード品質評価指標は、組込みソフトウェアの実装時の最終成果物であるソースコードを直接観察し、品質を評価する品質指標です。

ソースコードのボリュームを行数を用いて測る評価指標としてはソースプログラムを構成する行のうち、コメント等を除いた行数を計測する論理行数と、それらを含めて単純に行数を計測する物理行数とがあります。本ガイドではより簡便な計測方法を採用としたいため、ソースコード品質評価指標としては、後者を採用しています。ただし、ソースコードの行数は当然のことながら、開発者の記述の仕方により、2～3倍の差が表れます。ソースコードの書き具合を均一化するために、たとえば、コーディング作法ガイドを利用してプロジェクトのコーディングルールを作成し、誰

が書いても同様のコメント率になるようにプロジェクトをコントロールすることで、ソースコード品質評価指標の有効性を発揮することができるようになります。

また、ソースコードがプロダクト内で一つの言語のみで記述されることはあまりありませんが、本ガイドでは基本的に、組込みソフトウェアで一般に使用されているC言語に換算して参考値を考えています。アセンブリ言語やC++等、他の言語を併用している場合には、一定の係数を乗ずるなどして、組織の経験に基づいて使用してください。

コード品質評価指標は、

(i)ファイルあるいは関数単位のボリュームを評価するコードボリューム品質評価指標

(ii)ソースコードの記述内容面のバランスを評価するコード特性品質評価指標

の2種類の指標を利用します。

Column

ソフトウェア規模を測る

ソフトウェア規模を測る方法としては、ファンクションポイント数(FP)、行数(LOC)などさまざまな方法があります。このうちファンクションポイント(FP)はアプリケーションの機能に着目し、入出力やアプリケーションのインタフェースに基づいてプログラムの規模を計測する方法です。ファンクションポイントはプログラム言語を特定せずに使えること、実装前に規模が決定できることなどいくつかの利点はありますが、一方、複数の計測方法が提唱されており、それらはいずれも測定ルールが複雑であり、測定する人、測定の仕方によって大きな違いが発生します。そのためツールを用意したり、訓練を受けた専任者を用意したりする必要があります。一方、行数(LOC)は視覚的に明らかであり、機械的に計測可能で、たとえばエディタの機能などを使うことにより誰が測っても同じ値を得ることが可能となります。

コードボリューム品質評価指標

ソースコードが品質という視点から見て適切なものかどうかを測る指標の一つとして、そのボリュームが妥当なものであるかどうかという指標を考えることができます。一定単位あたりのソースコード行数がある規模以上の場合、保守性、理解性などに影響を与えてしまうと考えられます。

コードボリューム品質評価指標は

(i)**ファイル行数**：各ソースファイルに記述されたソースコードの規模（ボリューム）
(ii)**関数の行数**：ソースコードを構成する関数一つあたりのソースコードの規模（ボリューム）

の２種類の指標を利用します。

また、ファイル行数の総行数（総和）はソースコード全行数としてプログラム自身の規模を表すための代替指標として用います。

コード特性品質評価指標

また一方で、ソースコードがボリュームとしては一定の値に収まっていたとしても、記述の仕方に偏りがある場合はレビュー時の可読性、保守性などの品質が損なわれ、ソースコードの品質に大きく影響します。このため、本ガイドではソースコードの特質に着目したコード特性品質評価指標を用意しています。たとえば、コメント行はそのソースコードがなすべき機能について、読者（レビュー者、保守者）に補足情報を与える貴重な情報源になります。組織で定めたコーディングルールに従って記述されるべき補足情報が不足していないか、あるいは冗長な情報がないかを評価する手段としてコメント行記述率を評価することにより、これら情報の過不足をある程度判断することができるようになり、レビューを行う際に内容確認を再度行うなどの利用方法を考えることができます。

コード特性品質評価指標は

(i)制御文記述率
(ii)コメント行記述率
(iii)コーディングルール逸脱率

の３種類の指標を利用します。

③テスト品質評価指標

テスト品質評価指標は実装時の最終成果物であるオブジェクトコードを実際に動かすテストの十分性を評価するための指標です。

テストはいろいろな目的がありますが、大きく分けて、

(i)システムに要求される仕様（機能、性能等）と異なる動作を行わないか（不具合の検出）

(ii)システムに要求される仕様（機能、性能等）を満たしているか（動作の確認）

の、2つの特性を持っています。この2つの特性は相反するようですがどちらも大切で、テスト設計の際には目的、場面に応じてどちらを主にするかを判断して作業を行います。本ガイドではこのような視点からテストの十分性を確認することでソフトウェアの品質を間接的に評価します。

テスト品質評価指標は、

(i)テストとして十分な量が行われているかを評価するテスト十分性
(ii)テスト時に検出した不具合の修正具合を評価する動作完全性

の2種類の指標を利用します。

テスト十分性品質評価指標

テストの十分性という観点から見て適切なものかどうかを測る指標の一つとして、テストの結果としてのデータが十分であるかどうかという指標を考えることができます。たとえば、テストを実施した項目の数が規模に対して十分であるかどうかを測ることで、ソースコードに対し必要十分なテストがされているかというボリューム面での評価をすることを狙っています。もちろん、テスト項目数はテストフェーズ、開発しているソフトウェアの規模や複雑性、I/Oの多少によっても変わりますし、また、1項目とは何を指すかということもプロジェクト、場合によっては担当者によって異なります。このため、むやみにテスト項目数を数えてもまったく意味がありません。

本ガイドでは一般的な小項目、あるいはスクリプトレベルでの計数を参考値の目安としています。プロジェクトに応じ、基準を決めて測るのが良いでしょう。

テスト十分性品質評価指標は、

(i)テスト密度
(ii)不具合収束率

の2種類の指標を利用します。

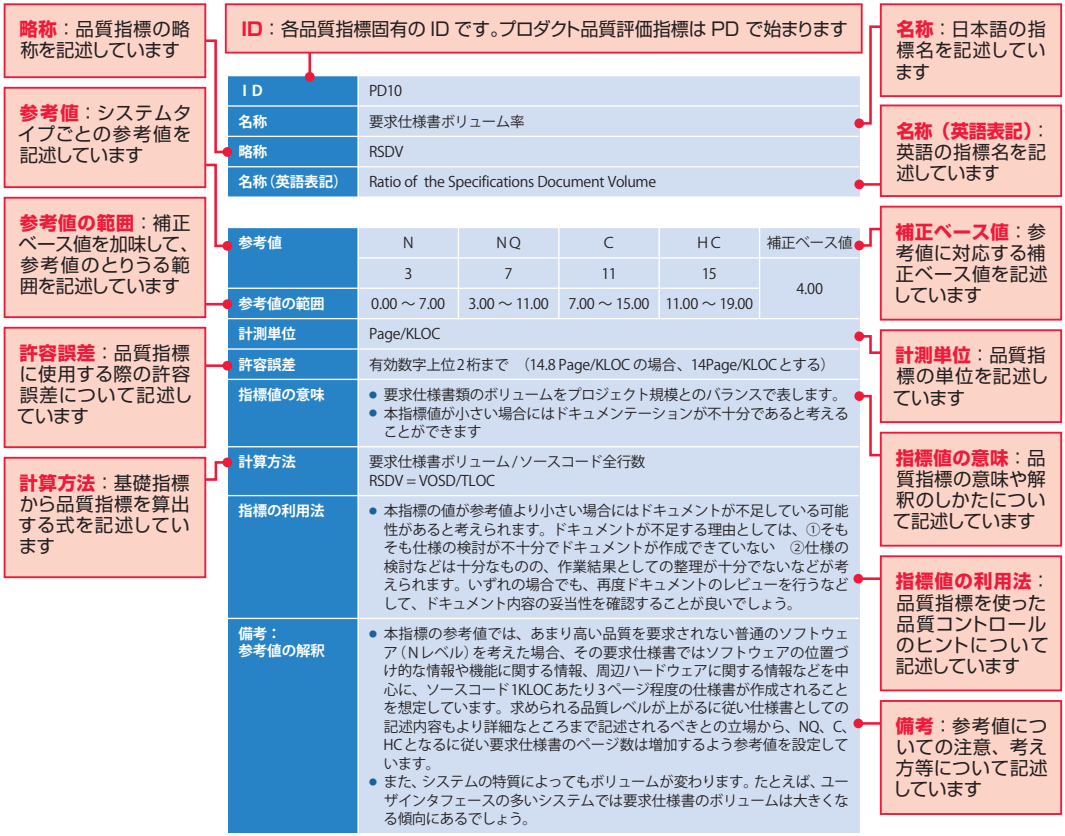
動作完全性品質評価指標

一方、テストによって不具合が検出されたとしても、そのすべてを修正するかどうかはプロジェクトの判断によります。動作完全性品質評価指標はテスト作業後のソフトウェアの品質がどの程度安定しているかを評価するための指標です。テストで検出された不具合に対してどの程度修正されているかを評価することで、出荷後のソフトウェアのリスクを評価します。このため動作完全性品質評価指標は、不具合

修正率 の1種類の指標を利用します。

プロダクト品質評価指標　定義表の読み方

次に、プロダクト品質評価指標について1つずつ、解説します。各表は次のような構成になっています。



ＩＤ	PD10				
名称	要求仕様書ボリューム率				
略称	RSDV				
名称（英語表記）	Ratio of the Specifications Document Volume				
参考値	N	NQ	C	HC	補正ベース値
	3	7	11	15	4.00
参考値の範囲	0.00 ～ 7.00	3.00 ～ 11.00	7.00 ～ 15.00	11.00 ～ 19.00	
計測単位	Page/KLOC				
許容誤差	有効数字上位2桁まで　（14.8 Page/KLOC の場合、14Page/KLOCとする）				
指標値の意味	● 要求仕様書類のボリュームをプロジェクト規模とのバランスで表します。 ● 本指標値が小さい場合にはドキュメンテーションが不十分であると考えることができます				
計算方法	要求仕様書ボリューム/ソースコード全行数 RSDV＝VOSD/TLOC				
指標の利用法	● 本指標の値が参考値より小さい場合にはドキュメントが不足している可能性があると考えられます。ドキュメントが不足する理由としては、①そもそも仕様の検討が不十分でドキュメントが作成できていない　②仕様の検討などは十分なものの、作業結果としての整理が十分でないなどが考えられます。いずれの場合でも、再度ドキュメントのレビューを行うなどして、ドキュメント内容の妥当性を確認することが良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値では、あまり高い品質を要求されない普通のソフトウェア（Nレベル）を考えた場合、その要求仕様書ではソフトウェアの位置づけ的な情報や機能に関する情報、周辺ハードウェアに関する情報などを中心に、ソースコード1KLOCあたり3ページ程度の仕様書が作成されることを想定しています。求められる品質レベルが上がるに従い仕様書としての記述内容もより詳細なところまで記述されるべきとの立場から、NQ、C、HCとなるに従い要求仕様書のページ数は増加するよう参考値を設定しています。 ● また、システムの特質によってもボリュームが変わります。たとえば、ユーザインタフェースの多いシステムでは要求仕様書のボリュームは大きくなる傾向にあるでしょう。				

図3-4：プロダクト品質評価指標 定義表の読み方

プロダクト品質評価指標

ＩＤ	PD10
名称	要求仕様書ボリューム率
略称	RSDV
名称（英語表記）	Ratio of the Specifications Document Volume

参考値	N	NQ	C	HC	補正ベース値
	3	7	11	15	4.00
参考値の範囲	0.00 ～ 7.00	3.00 ～ 11.00	7.00 ～ 15.00	11.00 ～ 19.00	
計測単位	Page/KLOC				
許容誤差	有効数字上位2桁まで （14.8 Page/KLOC の場合、14Page/KLOCとする）				
指標値の意味	● 要求仕様書類のボリュームをプロジェクト規模とのバランスで表します。 ● 本指標値が小さい場合にはドキュメンテーションが不十分であると考えることができます				
計算方法	要求仕様書ボリューム/ソースコード全行数 RSDV = VOSD/TLOC				
指標の利用法	● 本指標の値が参考値より小さい場合にはドキュメントが不足している可能性があると考えられます。ドキュメントが不足する理由としては、①そもそも仕様の検討が不十分でドキュメントが作成できていない ②仕様の検討などは十分なものの、作業結果としての整理が十分でないなどが考えられます。いずれの場合でも、再度ドキュメントのレビューを行うなどして、ドキュメント内容の妥当性を確認することが良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値では、あまり高い品質を要求されない普通のソフトウェア（Nレベル）を考えた場合、その要求仕様書ではソフトウェアの位置づけ的な情報や機能に関する情報、周辺ハードウェアに関する情報などを中心に、ソースコード1KLOCあたり3ページ程度の仕様書が作成されることを想定しています。求められる品質レベルが上がるに従い仕様書としての記述内容もより詳細なところまで記述されるべきとの立場から、NQ、C、HCとなるに従い要求仕様書のページ数は増加するよう参考値を設定しています。 ● また、システムの特質によってもボリュームが変わります。たとえば、ユーザインタフェースの多いシステムでは要求仕様書のボリュームは大きくなる傾向にあるでしょう。				

ＩＤ	PD11
名称	設計書ボリューム率
略称	RDDV
名称（英語表記）	Ratio of the Design Document Volume

参考値	N	NQ	C	HC	補正ベース値
	9	19	29	39	10.00
参考値の範囲	0.00 ～ 19.00	9.00 ～ 29.00	19.00 ～ 39.00	29.00 ～ 49.00	
計測単位	Page/KLOC				
許容誤差	有効数字上位2桁まで （14.8 Page/KLOC の場合、14Page/KLOCとする）				
指標値の意味	● 設計書類のボリュームをプロジェクト規模とのバランスで表します。 ● 本指標値が小さい場合にはドキュメンテーションが不十分であると考えることができます。				
計算方法	設計書ボリューム/ソースコード全行数 RDDV = VODD/TLOC				
指標の利用法	● 本指標値が参考値より小さい場合には、設計ドキュメントが不足している可能性があります。設計ドキュメントが不足する理由としては、①そもそも設計の検討が不十分でドキュメントが作成できていない ②設計の検討などは十分なものの、作業結果としての整理が十分でないなどが考えられます。いずれの場合でも、再度ドキュメントのレビューを行うなどして、ドキュメント内容の妥当性を確認することが良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値では、あまり高い品質を要求されない普通のソフトウェア（Type-1：N）を考えた場合、要求仕様書の1ページに対して、その内容をきちんと設計に反映させた場合、設計書のボリュームは要求仕様書の3倍程度が適切との立場で参考値を設定しています。 ● また、システムの特質によってもボリュームが変わります。たとえば、ユーザインタフェースの多いシステムでは設計書のボリュームもまた大きくなる傾向にあるでしょう。				

ＩＤ	PD12
名称	テスト仕様書ボリューム率
略称	RTDV
名称（英語表記）	Ratio of the Test Document Volume

参考値	N	NQ	C	HC	補正ベース値
	9	19	29	39	10.00
参考値の範囲	0.00 ～ 19.00	9.00 ～ 29.00	19.00 ～ 39.00	29.00 ～ 49.00	
計測単位	Page/KLOC				
許容誤差	有効数字上位2桁まで （14.8 Page/KLOC の場合、14Page/KLOCとする）				
指標値の意味	● テストに関する仕様書類のボリュームをプロジェクト規模とのバランスで表します。 ● 本指標値が小さい場合にはドキュメンテーションが不十分であると考えることができます。				
計算方法	テスト仕様書ボリューム/ソースコード全行数 RTDV = VOTD/TLOC				
指標の利用法	● 本指標値が参考値より小さい場合には、テストドキュメントが不足している可能性があります。テストドキュメントが不足する理由としては、①そもそもテスト項目の検討が不十分でドキュメントが作成できていない ②テストの検討などは十分なものの、作業結果としての整理が十分でないなどが考えられます。いずれの場合でも、再度ドキュメントのレビューを行うなどして、ドキュメント内容の妥当性を確認することが良いでしょう。				
備考： 参考値の解釈	● 本指標の参考値では、あまり高い品質を要求されない普通のソフトウェア（Type-1：N）を考えた場合、そのソフトウェアをテストする場合には例外動作などさまざまなバリエーションをテストすることを想定し、要求仕様書の3倍程度のボリュームが適切との立場で参考値を設定しています。 ● また、システムの特質によってもボリュームが変わります。たとえば、ユーザインタフェースの多いシステムではテスト仕様書のボリュームもまた大きくなる傾向にあるでしょう。				

Column

再利用率の指標値への影響

最近のソフトウェア開発では、全くの新規から作るケースは減少し、既存のソフトウェア資産を再利用する、ベースソフトウェアを元に派生開発を行うなどのケースが増加しています。こうしたソフトウェアの再利用や流用などによる開発では、再利用や流用した部分とそれらを含めたソフトウェア全体の両方を考慮しながら、品質をコントロールしていく必要が生じます。再利用などの度合いがソフトウェア開発にどの程度影響を及ぼすかについてはプロジェクトやソフトウェアごとに異なるため、再利用を行ったプロジェクトのレビュー作業実施率などの参考値は一律には決めづらいといった側面があります。このため本ガイドでは考え方のベースとして新規開発を念頭に置いた場合の参考値を提示してあり、再利用率の影響などについてはこの参考値をそれぞれのプロジェクトの状況などに合わせて読み替えたり、見直していただくことを推奨しています。

ＩＤ	PD20
名称	要求仕様書バランス
略称	BSDD
名称（英語表記）	Balance of the Specifications Document Description

参考値/参考値の範囲	N	NQ	C	HC	補正ベース値
	次ページの表参照				なし
計測単位	各ドキュメントの記載項目の%で表示				
許容誤差	5%刻みのパーセント表示				
指標値の意味	● 要求仕様書に記載すべき各項目がどの程度の割合で記載されているかを要求仕様書の全体記述量とのバランスで表します。ドキュメントの内容面での十分性を評価することができます。				
計算方法	● 要求仕様書内の各パートのページ数/要求仕様書ページ数の総和 ● 要求仕様書に記載されている各内容について、要求仕様書記述量全体の何パーセントを占めているかを計測します。 ● 記述ボリュームの計測に関しては、たとえば、「R1 対象ユーザとその使い方に関する記述」が何ページあるかを確認し、そのページ数を全体ページ数で割ることで、この項目に関する記述量のパーセントを求めます。なお、要求仕様書にどのような項目を盛り込むかについては、ESPRのSYP1.1 システム要求仕様書の作成 および SWP1.1 ソフトウェア要求仕様書の作成の内容をベースとしています。				
指標の利用法	この指標値について、ある記述項目に関しての値が次ページの表の参考値に示された記述バランスから乖離して小さい場合、その項目に関する記述や検討が不十分と判断し、ドキュメントの見直しや再検討などの対象とします。				
備考：参考値の解釈	● 本指標の計測・評価では、要求仕様書に記載すべき事項と、それぞれのドキュメントの全体記述量を100とした場合のおおよその目安となる個々の記載項目の記述量を示しています。たとえば、要求仕様書では「R2：対象ユーザとその利用方法に関する記述」は全体の5%程度、「R3：動作環境条件に関する記述」は全体の10%程度といった参考値を提示しています。これらの値などを参考にして、評価するドキュメント（たとえば要求仕様書）にどのような項目がどの程度記載されているかを計測し、ドキュメント内容の適切さを評価します。 ● これら各項の比率は対象となるシステムの特質により値が変わることに注意してください。参考値は、標準的なシステムでの比率を表しています。たとえば、安全性が求められるシステムではR5の比率が、また、例外処理の多いシステムではR7の比率が相対的に高くなります。システムプロファイルの際に重視すべき項目は何かということを確認し、目標値を決める際に反映すると良いでしょう。				

ドキュメント	項目No	内容	参考 %
要求仕様書	R1.	全体の記述量	100
	R2.	対象ユーザとその使い方に関する記述	5
	R3.	動作環境条件に関する記述量	10
	R4.	主な機能に関する記述量	40
	R5.	安全に関する記述、並びに非機能に関する記述量	30
	R6.	システム全体構成に関する記述量	10
	R7.	例外処理に関する記述量	5

Column

機能要求 vs 非機能要求

ソフトウェアを開発するためには、まずどのようなものを作るかを明らかにする必要があります。そのためにはいわゆる要求分析・定義という作業を行います。要求分析・定義では、ソフトウェアの機能として実現すべきことを表す機能要求と性能や使い勝手、安全性など、機能以外の面を表す非機能要求とを明確にします。

開発をしようとしているソフトウェアがどのような機能要求、非機能要求を持つか、そしてその割合がどの程度かは対象とするシステムによって大きく異なってきます。たとえば、不特定多数のユーザが利用する機器などで用いられる組込みソフトウェアの場合、ソフトウェアとして提供する機能もさることながら、その使い勝手や信頼性などの非機能面などを十分に検討しておくことが求められます。このようなソフトウェアでは非機能要件として記載すべき事項が増加しますので、要求仕様書の記述内容バランスを調整する必要があります。

ID	PD21
名称	設計書バランス
略称	BDDD
名称（英語表記）	Balance of the Design Document Description

参考値/参考値の範囲	N	NQ	C	HC	補正ベース値
	次ページの表参照				なし
計測単位	各ドキュメントの記載項目の%で表示				
許容誤差	5%刻みのパーセント表示				
指標値の意味	● 設計書に記載すべき各項目がどの程度の割合で記載されているかを設計書の全体記述量とのバランスで表します。設計ドキュメントの内容面での十分性を評価することができます。				
計算方法	● 設計書内の各パートのページ数/設計書全体ページ数の総和 ● 設計書に記載されている各内容について、設計書記述量の何パーセントを占めているかを計測します。記述ボリュームの計測に関しては、たとえば、「R2：システム全体構成に関する記述」が何ページくらいあるかを確認し、そのページ数を全体ページ数で割り、この項目に関する記述量のパーセントを求めます。なお、設計書にどのような項目を盛り込むかについては、ESPRのSYP2.1 システムアーキテクチャ仕様書の作成 およびSWP2.1 ソフトウェアアーキテクチャ設計書の作成の内容をベースとしています。				
指標の利用法	● この指標値について、ある記述項目に関しての値が次ページの表の参考値に示された記述バランスから乖離して小さい場合、その項目に関する記述や検討が不十分と判断し、ドキュメントの見直しや再検討の対象とします。				
備考：参考値の解釈	● 本指標の計測・評価では、設計書に記載すべき事項と、それぞれのドキュメントの全体記述量を100とした場合のおおよその目安となる個々の記載項目の記述量を示しています。たとえば、設計仕様書では「D2：システム全体構成に関する記述対象」は全体の5%程度、「D3：機能ブロックの構成に関する記述」は全体の5%程度といった参考値を提示しています。これらの値などを参考にして、評価するドキュメント（たとえば設計書）にどのような項目がどの程度記載されているかを計測し、ドキュメント内容の適切さを評価します。 ● これら各項の比率は対象となるシステムの特質により値が変わることに注意してください。参考値は、標準的なシステムでの比率を表しています。要求仕様書バランスの際にバランスの目標値を変更した場合は本設計書バランスの目標値も相対的に変更するよう、注意してください。				

ドキュメント	項目No	内容	参考 %
設計書	D1.	全体の記述量	100
	D2.	システム全体構成に関する記述量	5
	D3.	機能ブロックの構成に関する記述量	5
	D4.	機能ブロックの詳細に関する記述量	50
	D5.	インタフェース・データに関する記述量	20
	D6.	例外処理に関する記述量	20

なお、D1：全体の記述量は、R1：全体の記述量（要求仕様書）の3倍を目安とします。

Column

設計内容の表現方法

ソフトウェアの設計とはそのソフトウェアに求められている要求事項（機能要求、非機能要求）を如何にして実現するかを考え、ソフトウェアの静的な構造や動的な仕組みを検討し決めていく作業です。ソフトウェアの静的構造、動的構造は場合によっては日本語などの自然言語で説明するよりも、それらを端的に表現できる図や表などを用いて表す場合が少なくありません。近年、ソフトウェア設計の世界では設計モデリングという考え方が浸透してきていますが、これもソフトウェアの設計を決められた図表表記によって抽象的に整理するための考え方の一つです。このように図表を用いたソフトウェア設計の表現は、直感的あるいは論理的に設計構造を表現できるという利点がある一方で、細かい部分や設計根拠などの情報が表現しきれないなどの課題もあります。ソフトウェアの設計内容を正確に伝えるためにどのような表現系を用いるかも含めて設計書に記載すべき内容などは予め吟味しておくことが求められます。

ID	PD22
名称	テスト仕様書バランス
略称	BTDD
名称（英語表記）	Balance of the Test Document Description

参考値/参考値の範囲	N	NQ	C	HC	補正ベース値
	次ページの表参照				なし
計測単位	各ドキュメントの記載項目の%で表示				
許容誤差	5%刻みのパーセント表示				
指標値の意味	● テストに関する仕様書に記載すべき各項目がどの程度の割合で記載されているかをテストに関する仕様書の全体記述量とのバランスで表します。ドキュメントの内容面での十分性を評価することができます。				
計算方法	● テスト仕様書内の各パートのページ数/テスト仕様書ページ数の総和 ● テスト仕様書に記載されている各内容について、テスト仕様書記述量の何パーセントを占めているかを計測します。 ● 記述ボリュームの計測に関しては、たとえば、「T2：テスト環境に関する記述」が何ページくらいあるかを確認し、そのページ数を全体ページ数で割り、この項目に関する記述量のパーセントを求めます。なお、テスト仕様書にどのような項目を盛り込むかについては、ESPRのSYP4.1 システムテストの準備、SWP5.1 ソフトウェア結合テストの準備およびSWP6.1 ソフトウェア総合テストの準備の内容をベースとしています。				
指標の利用法	● ある記述項目に関しての値が次ページの表の参考値に示された記述バランスから乖離して小さい場合、その項目に関する記述や検討が不十分と判断し、ドキュメントの見直しや再検討の対象とします。				
計測のヒント	● 個別の記述項目の記述量については、単純に1.5ページ位などのようにページ換算で求める、あるいは行換算で求めるなどしておおよその記述量を把握できればよいものとします。場合によっては、記述エリアの面積や距離などの物理的な値によって記述されたパーセントを求める方式でも構いません。				
備考：参考値の解釈	● 本指標の計測・評価では、テスト仕様書に記載すべき事項と、それぞれのドキュメントの全体記述量を100とした場合のおおよその目安となる個々の記載項目の記述量を示しています。たとえば、テスト仕様書では「T2：テスト環境に関する記述」は全体の5%程度、「T3：テストの手順・条件に関する記述」は全体の10%程度といった参考値を提示しています。これらの値などを参考にして、評価するドキュメント（たとえばテスト仕様書）にどのような項目がどの程度記載されているかを計測し、ドキュメント内容の適切さを評価します。 ● これら各項の比率は対象となるシステムの特質により値が変わることに注意してください。参考値は、標準的なシステムでの比率を表しています。要求仕様書バランスの際にバランスの目標値を変更した場合は本テスト仕様書バランスの目標値も相対的に変更するよう、注意してください				

ドキュメント	項目No	内容	参考値
テスト仕様書	T1.	全体の記述量	R1*3
	T2.	テスト環境に関する記述	5
	T3.	テストの手順・条件に関する記述	10
	T4.	正常系に関する記述	35
	T5.	異常系・例外処理に関する記述	45
	T6.	テスト完了基準に関する記述	5

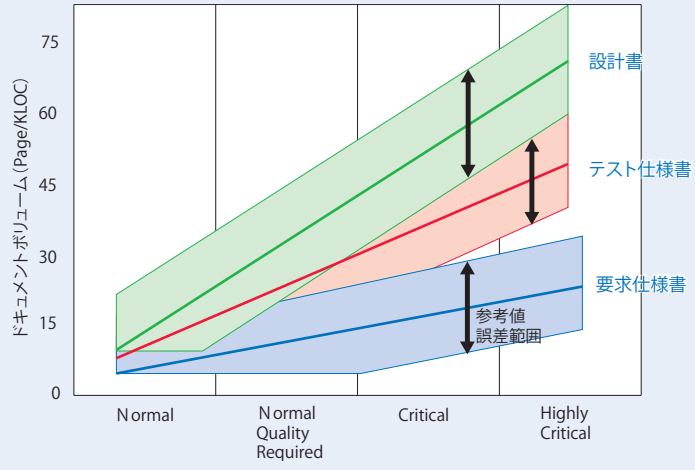
なお、T1：全体の記述量は、R1：全体の記述量（要求仕様書）の3倍を目安とします。

Column

数値指標

一般的に数値指標は、さまざまな理由から計測誤差を含む場合が少なくありません。本ガイドで示した参考値の解釈についても、下図のように参考値±15%程度の誤差範囲を考慮して適宜修正して利用することが望ましいと考えられます。

また、一部の指標についてはソフトウェア規模が大きくなった場合には純粋に規模に応じて単調増加しないものもあります。



ＩＤ	PD30
名称	ファイル行数
略称	FLOC
名称（英語表記）	File Lines Of Code

参考値/参考値の範囲	N	NQ	C	HC	この値以下であること
	2.00	2.00	2.00	2.00	
計測単位	KLOC				
許容誤差	有効数字上位2桁まで　（1231LOCの場合、1.2KLOCとする）				
指標値の意味	● 品質計測・評価の対象となる部分のソフトウェアの規模を表す指標です。 ● この値が大きい場合には、対象ソフトウェアの規模が大きいことを意味します。 ● 1ファイルあたりの行数を計測し、一定量以上になっているファイルがあるかどうかを確認します。1ファイルあたりの行数が多すぎる場合、可読性、保守性の妨げとなることが考えられます。また、このメトリクスはシステムプロファイルのタイプに影響を受けません。				
計算方法	ファイル行数：基礎指標のファイル行数をそのまま使用します。 FLOC＝FLOC				
指標の利用法	● 本指標値は値の大きいファイルをとらえて評価の対象とします。 ● 参考値よりも1ファイル内の行数が大きく超えている場合には、保守性の低下が考えられますので、ソースコードレビューなどを早めに実施する必要があります。 ● また、本指標の総計（全ファイル行数）はソフトウェアの規模をあらわす代替指標として使用します。開発者あるいはプロジェクトマネージャやリーダが感覚的にとらえているソフトウェア規模（全行数）とこの計測値が大きく乖離している場合には、開発者の仕様の取り違えや内部処理の誤りなどが含まれていると考え、ソースコードレビューなどを早めに実施する必要があります。 ● 本指標値は、C言語換算になっていることに注意してください。複数の言語を使用している場合の換算方法などについては基礎指標のファイル行数（ID=B11、FLOC）を参照してください。				
備考：参考値の解釈	● 人が論理を見通せる範囲には限界があります。小説や技術文書など、約1章分であれば把握できる範囲でしょう。本指標の参考値としては、本の1～2章分：80行×25ページ（2000行）として換算しています。				

ID	PD31
名称	関数の行数
略称	MLOC
名称（英語表記）	Module Lines Of Code

参考値/参考値の範囲	N	NQ	C	HC	この値以下であること
	160.0	160.0	160.0	160.0	
計測単位	LOC				
許容誤差	有効数字上位2桁まで （231LOCの場合、230LOCとする）				
指標値の意味	● 処理単位の基本である関数の規模を表す指標です。 ● 1関数あたりの行数を計測し、一定量以上になっている関数があるかどうかを確認します。1関数あたりの行数が多すぎる場合、可読性、保守性の妨げとなることが考えられます。また、このメトリクスはシステムプロファイルのタイプに影響を受けません。				
計算方法	関数行数：基礎指標の関数行数をそのまま使用します。 MLOC = MLOC				
指標の利用法	● 本指標は値の大きい関数をとらえて評価の対象とします。 ● 一定行数以上の関数を抜き出します。すべての関数の値を計測するのが困難な場合、値の大きそうな関数を選んで計測するのでも良いでしょう。 ● 参考値よりも1関数内の行数が大きく超えている場合には、保守性の低下が考えられます。関数を分割する、ソースコードレビューなどを早めに実施するなどの対策を検討する必要があります。				
備考： 参考値の解釈	● 人が論理を見通せる範囲には限界があります。レビューなどで一度に見渡すのはプリントアウトのイメージで見開き2ページが適当でしょう。本指標の参考値としては、80行×2ページとして換算しています。				

ＩＤ	PD32
名称	制御文記述率
略称	ROCS
名称（英語表記）	Ratio Of Control Statement

参考値	N	NQ	C	HC	補正ベース値
	35	30	25	20	-5.00
参考値の範囲	30.00 ～ 40.00	25.00 ～ 35.00	20.00 ～ 30.00	15.00 ～ 25.00	
計測単位	%				
許容誤差	有効数字上位2桁まで				
指標値の意味	● 計測対象ファイルに含まれる制御文数をソースコード全行数からのバランスで表します。制御文記述率はソースコードの複雑度計測が繁雑であることから代替指標として提案しています。 ● 複雑度が高いということはソースコードによる記述が複雑であることを示し、保守性、理解性等の低下につながり、ひいては品質低下を招いてしまいます。すなわち、この数値が高いということは分岐等が多いことから、設計もしくはコード自身の複雑度が高く、信頼性、保守性に問題があるかもしれないということを示します。				
計算方法	制御文数 / ソースコード全行数 ROCS = NOCS/TLOC				
指標の利用法	● 制御文数が多い場合には、ソースコードが複雑になっており、保守性、可読性等が落ち、バグの温床になることも考えられます。あまり多い場合は、設計を見直す、ソースコードレビューで信頼性面に問題がないか確認する、テストでパスを網羅するなどの工夫をしましょう。 ● 対象となるシステムの特性により、ソフトウェアに要求される複雑さもさまざまです。参考値を参照し、組織毎、モジュール毎に妥当な値を設定するのが良いでしょう。				
備考： 参考値の解釈	● ツールを使ってサイクロマティック複雑度などを計測する方法もあるので、それを使っても構いません。その場合は組織ごとに決められた指標を用いるのが良いでしょう。 ● 本ガイドでは、なるべく簡便に、費用をかけずに計測できる代替指標として制御文記述率を提案しています。これはソースコード上の分岐を表すキーワードを拾い出すだけで計測できるので、手軽な代替指標と考えることができます。 ● 本指標の目安はNタイプのソフトウェアの場合,100行あたり３～４個の制御文が使用されると考えて設定しています。				

ＩＤ	PD33
名称	コメント行記述率
略称	ROCL
名称（英語表記）	Ratio Of Comment Line

参考値	N	NQ	C	HC	補正ベース値
	20.00	25.00	30.00	35.00	5.00
参考値の範囲	15.00 ～ 25.00	20.00 ～ 30.00	25.00 ～ 35.00	30.00 ～ 40.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 計測対象ファイルに含まれるコメント行数を、ソースコード全行数からのバランスで表します。 ● コーディングルールなど一定のルールに従って記述されたコメントはソースコードを理解保守する上で最低限必要な情報をもたらします。ヘッダコメント、関数、変数などの説明が適切なボリュームで含まれることで、ソースの可読性を上げることができます。 ● この数値が高すぎるということは、不必要なコメントが入っていたり、既に不要なコメントやソースコードが削除されずに残っている可能性が高く、ソースコードの可読性を下げている可能性があります。				
計算方法	コメント行数 / ソースコード全行数 ROCL = CLOC/TLOC				
指標の利用法	● この指標を有効に扱うにはまず、コメントを記述するルール（内容、箇所、更新ルール）が規定されていることが大切です。コーディングルールに則った記述であれば、相当の分量がソースコード内に記述されていることでコードの可読性を保障することが可能となります。 ● ただし、コメントは決まった項目以外は自然言語で書きますので、記述する人によってボリュームはかなり前後します。2～3割までの余裕をもって運用するにしましょう。値が極端に指標と異なるものはソースコードレビューを早めに行うなどして、状況を確認する必要があるでしょう。 ● また、記述する人の傾向を見ることによって、コメント行記述率が低い場合は指導の補助に使うなどといった利用法も考えられます。				
備考： 参考値の解釈	● 本指標はコーディングルールなどに則ってコメントが挿入されることを前提としています。 ● ファイル毎に大きなばらつきはないか、あまりに大きい値がないか、逆に小さすぎる値はないか、という観点で評価すると良いでしょう。 ● あまりに大きい値の場合は、使用しないコードをそのままコメントアウト化しているなど、無駄な、かつ危険なコメントが入っているというケースが考えられます。 ● また、小さい値の場合は必要なヘッダファイルが入っていないなどのケースが考えられます。コーディングルールに則って記述されているか、確認しましょう。				

ＩＤ	PD34
名称	コーディングルール逸脱率
略称	RDCR
名称（英語表記）	Ratio of Deviation of Coding Rules

参考値	N	NQ	C	HC	補正ベース値
	310	210	110	10	-100.00
参考値の範囲	210.00 ～ 410.00	110.00 ～ 310.00	10.00 ～ 210.00	0.00 ～ 110.00	
計測単位	箇所/KLOC				
許容誤差	有効数字上位2桁まで				
指標値の意味	● 組織あるいはプロジェクトで決めたコーディングルールから逸脱している記述がどの程度あるかをソースコード全行数とのバランスで示します。				
計算方法	コーディングルール逸脱数/ソースコード全行数 RDCR = NDCR/TLOC				
指標の利用法	● 逸脱率が高い場合にはレビュー等での確認時間を多くするなどして、品質の確保に努める必要があります。一部のモジュールで逸脱率が突出している場合にはそのモジュールの特殊性などがあるか、また、記述した人に依存している部分があるかなども確認すると良いでしょう。				
備考： 参考値の解釈	● 逸脱に対する手続きがとられないことが多い、逸脱を免れるために不適当にソースを変更する箇所が多いなどの傾向が見られる場合は、コーディングルールそのものの見直しも検討する必要があるでしょう。				

ID	PD40
名称	テスト密度
略称	DOTI
名称（英語表記）	Density Of Test Items

参考値	N	NQ	C	HC	補正ベース値
	25.00	50.00	75.00	100.00	25.00
参考値の範囲	0.00 ～ 50.00	25.00 ～ 75.00	50.00 ～ 100.00	75.00 ～ 125.00	
計測単位	項目 /KLOC				
許容誤差	有効数字上位2桁まで				
指標値の意味	● ソース規模あたりのテスト実施項目数を示します。この値は動的テストの十分性、ソースに対するテスト網羅性の目安となります。				
計測方法	テスト項目数 / ソースコード全行数 DOTI = NOTI/TLOC				
指標の利用法	● テスト密度は対象の規模が大きくなれば、組み合わせの数などが増えるので増加する傾向にあります。ただし、やみくもに増やせばよいというわけではなく、効果のある組み合わせを考えてテストを設計することが必要です。また、対象システムの複雑さ、入出力の数などにより、テスト密度はそれ以上に変化しますので、補正係数などを考慮して十分バランスをとるようにすることが必要です。 ● 本指標が参考値より小さい場合は、対象システムの規模に対して適切な数のテストが実施されていないことを意味し、テストによる不具合の検出が不十分になる可能性があると考えることができます。 ● 逆に本指標が参考値より大きい場合は、余計なテストを実施しており、テストの効率が良くないなどの可能性があります。				
備考： 参考値の解釈	● 参考値は、関数1つの行数を120行とした場合、NQで関数1つに対し、6項目のテストを用意するといった目安で算出しています。 ● 本指標値をさらに有効活用するには、ソフトウェア全体のテスト密度を測るだけでなく、モジュールごと、機能ごとに難易度、複雑度が高い部分、入出力の多い部分などを考慮してテスト密度が適当であるかなどを見ると良いでしょう。				

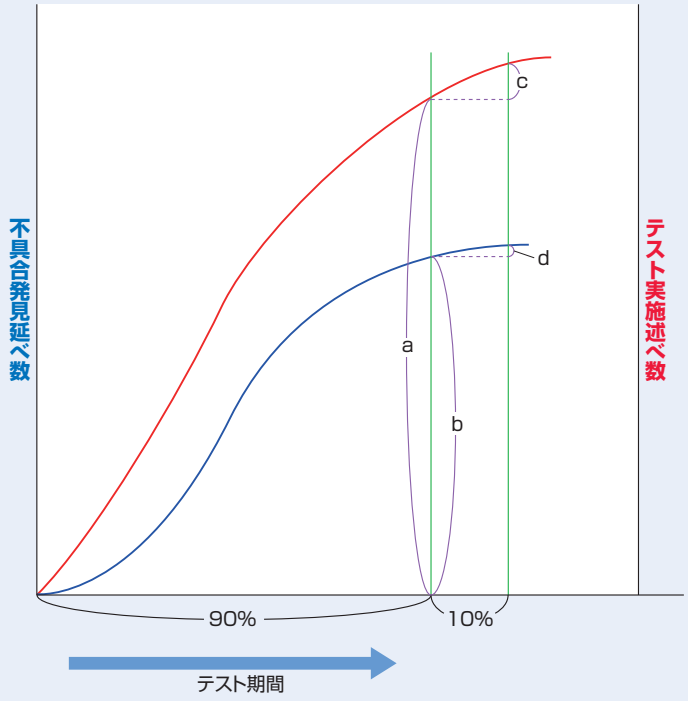
ＩＤ	PD41
名称	不具合収束率
略称	ROFC
名称（英語表記）	Ratio Of Fault detection in Comparison

参考値	N	NQ	C	HC	補正ベース値
	0.05	0.04	0.03	0.02	0.01
参考値の範囲	0.04 ～ 0.06	0.03 ～ 0.05	0.02 ～ 0.04	0.01 ～ 0.03	
計測単位	不具合発見率の比				
許容誤差	有効数字上位2桁まで				
指標値の意味	● テストの終了条件にはいろいろありますが、発見する不具合数が収束していくこと＝発現する不具合が一定の確率以下になったと見込めることでテスト終了するというのも一つの方法です。この指標では、テスト終盤とそれ以前のテスト期間との不具合発見率を比較することで、テスト期間の最終段階での対象ソフトウェアの安定度を測ります。 ● テスト期間最終段階で、テスト実施項目数は増加しているのに対し、発見する不具合数は0に近づいている（増加しない）という状態が確認できれば、ソフトウェアを運用した際に顕在してしまう不具合が収束しつつあるということになります。				
計算方法	最終10%のテスト期間の不具合発見率 / 最終10%手前までのテスト期間の不具合発見率				
指標の利用法	参考値に示された記述バランスから乖離して値が高い場合、テストの十分性が足りないと判断し、テスト項目の見直し、実施の状況などを再検討します。				
備考： 参考値の解釈	● 本指標は、①発見した不具合は修正する。②テストはリグレッションテストを繰り返しながら項目数は増加していくということを前提としています。従って、テスト実施項目数は開発が進むに従って増加し、発見不具合数は減少していくということになります。テストを中断していた期間が長期に渡る場合にはその期間を省いて算出するなど、期間と項目数の関係については組織の事情に応じて勘案する必要があるでしょう。				

Column

不具合収束率

ソフトウェアの世界では不具合の収束を予測する考え方については、いわゆるバグ曲線と呼ばれるソフトウェア信頼度成長モデルなどの考え方が多く用いられています。そのモデルとしてはたとえば、需要傾向、経済成長などの予測に用いられるロジスティック曲線、ゴンペルツ曲線などがありますが、これらはモデルが多岐にわたり、ユーザが選択を誤ると評価結果に信頼がおけなくなってしまう、また、確率論に基づく統計的処理となるため、適用が極めて難しいなどの側面があります。本ガイドではより簡単に不具合収束を判断する方法として、テスト期間のある時期（期間の90%までと期間の最後10%）における不具合検出率を比較する方法で、テスト最終段階で不具合が検出されつくしているかどうかを評価する方式を採用しています。



不具合収束率＝(最終 10%のテスト期間の不具合発見率)/
(最終 10%手前までのテスト期間の不具合発見率)
＝(d/c)/(b/a)

ＩＤ	PD42
名称	不具合修正率
略称	ROFE
名称（英語表記）	Ratio Of Fault Elimination

参考値	N	NQ	C	HC	補正ベース値
	94.00	97.00	100.00	100.00	3.00
参考値の範囲	91 ～ 97	94 ～ 100	97 ～ 100.00	97 ～ 100.00	
計測単位	%				
許容誤差	有効数字上位2桁までのパーセント表示				
指標値の意味	● 検出した不具合数のうち、どの程度修正したかを示します。 ● 上限を100とし、できるだけ100に近い数値になっていること、すなわち検出した不具合はできるだけ修正されることが望ましいでしょう。				
計算方法	修正済み不具合数 / 検出不具合数 ROFF = NOEF/NODF				
指標の利用法	● 検出不具合数は評価の対象となるもののすべての不具合の数の総計を表す指標であり、検出不具合数が多いということは対象物の品質に課題が潜んでいるかもしれないことを意味します。 ● 開発途中で本指標値を計測することで、不具合の修正収束状況を見ることができます。本指標値が高ければ不具合が収束しつつあることを示します。逆に、低ければ検出不具合数に対して修正が追いついていないということを示すので、プロジェクト上困難な状態にあることが予測できます。 ● 開発終盤で本指標値が低い場合、クレーム発生率が高くなることが予測されますので、リリースの是非について再検討する必要があります。 ● 不具合については重大なもの、軽微なものなどさまざまですので単純な数の評価だけではなく、不具合内容なども考慮した評価が必要になります。				
備考： 参考値の解釈	● 検出した不具合が開発後も残存していることはある一定以上の品質を要求するシステムでは考えられないので、Type-3：Critical以上のシステムでは目標参考値を100としています。 ● 修正しない不具合がある場合、仕様を見直したり制限事項としたりすることで（修正したとみなして）本指標の数値を上げることができますが、不具合の発生頻度、危険率を考慮してどう扱うかを決めるのが大切です。また、その際、ユーザの誤操作や誤解が生じないよう、回避方法を明確にするなどの工夫も必要でしょう。 ● 本指標の同意の指標として、残存未修正バグ数があります。残存未修正バグ数を単純に数値としてとらえているプロジェクトが多いかもしれません。もちろんこうした方法でも構いませんが、組織内に大きさの異なるプロジェクトが複数存在する場合、本指標を用いることでノーマライズして比べることができるようになります。				

3.6 基礎指標 ― 定義と参考値

基礎指標とは

基礎指標とは、前述したプロセス品質評価指標、プロダクト品質評価指標を算出する際に基礎となる指標であり、ソフトウェア開発の各場面で実際に計測する指標です。

基礎指標は

①ソースコードボリューム基礎指標
②ドキュメントボリューム基礎指標
③工数ボリューム基礎指標
④テストボリューム基礎指標

の4種類の指標を利用します

ソースコードボリューム基礎指標

ソースコード全行数

前述した、プロダクト品質評価指標でのコードボリューム品質指標の考え方に沿って、物理行数をファイル毎に計測します。これはエディタや行数カウントコマンド等を使うことで計測できます。また、すべてのファイルの行数の総和であるソースコード全行数をプログラム規模の代替指標として使用します。計測時にファイル毎の一覧を作成しておくことでファイル行数（ID：PD30、FLOC）への適用が楽になります。

制御文数

ソースプログラム中に含まれる制御文の数をファイル毎に計測します。制御文の数はC言語の場合、以下の予約語

If / while / for / case / default / else

の個数を指します。これらをキーワードとしてエディタなどで検索することで、おお

よその値を得ることができます。計測時、ファイル毎の一覧を作成しておき、制御文記述率（ID：PD32、ROCS）で使用します。

コメント行数

ソースプログラム中に含まれるコメント行の数をファイル毎に計測します。ツールを使用して計測してもよいですし、C言語の場合、コメントを示す“/* ”および“//”の数を計測してもおおよその値が得られるでしょう。その場合、コメントが複数行に渡る場合の記述方法について、コーディングルールを作成してコントロールすると良いでしょう。

ドキュメントボリューム基礎指標

ソフトウェア開発を行ううえでの各プロセスである、要求定義、アーキテクチャ設計・詳細設計、結合テスト～システムテストのそれぞれの成果物であるドキュメント類のそれぞれのページ数を前述した、プロダクト品質評価指標でのドキュメントボリューム品質指標の考え方（1ページ2000字換算）に沿って計測します。

また、計測時、ドキュメントのパート毎の一覧を作成しておくことで要求仕様書バランス（ID：PD20、BSDD）、設計書バランス（ID：PD21、BDDD）、テスト仕様書バランス（ID：PD22、BTDD）への適用が楽になります。

工数ボリューム基礎指標

それぞれの作業にかかった工数の総和を計測します。作業記録を元に、おおよその値を計算することができるでしょう。

各レビュー工数

各プロセスのレビューに要した工数の総和を計測します。レビュー記録票などが手がかりとなるでしょう。ただし、個人的な机上レビュー、ピアレビューなどで比較的多く見られるようにレビューの記録を残さない場合がありますが、このようなレビューはレビュー工数に含めないようにします。

各作成工数

各プロセスに要した工数の総和を計算します。作業記録票などが手がかりとなるでしょう。計測対象となる工数は、成果物を生成するあるいはレビューするためのものとします。会議時間は含むが、教育時間は含まないなど、各プロジェクトで計測対象を明確にしておくことで混乱を防ぐことができます。

テスト工数

ソフトウェアあるいはシステムをテストした工数の総和を計算します。テスト設計、テストデータ作成、テストシナリオ作成、テスト期待値作成などの工数もこの工数に含めます。また不具合記録作成、テスト報告書作成などの作業工数もテスト工数に含みます。ただし、単体テストの工数は含みません。

なお、単体テストの位置づけについてはESPR SWP4.1,4.2の考え方に準じ、プログラムユニットレベルのテストと考えます。

テストボリューム基礎指標

テストのプロセスを通じて得られる成果物のボリュームを測ります。

テスト項目数

単体テストを除く、結合テスト、総合テスト、システムテストなど、テストの項目数の総和です。項目数はさまざまな粒度の表現がありますが、対象プロジェクト内では粒度を均一にできるよう、意思統一を行う必要があります。なお、テスト項目として計測するのは、テスト項目として有効なもの、すなわち、一度以上テストに使用しているものとします。作っただけで使用していないテスト項目は計測対象としません。

テスト実施項目数

テストを実施した延べの項目数です。テスト項目数と同じく、単体テストは計測しません。上記テスト項目×テスト実施回数で計測することができます。テストを実行している途中で中止してしまったような場合は、中止した箇所より後のテスト項目は計測しない、ということになりますので注意してください。

検出不具合数

単体テストフェーズ終了以降～計測時点までに検出した不具合数を計測します。不具合を検出するのはテスト部隊だけでなく、プログラマ、社内関係者、その他ステークホルダすべてになります。検出した不具合は一件一葉の不具合票または不具合データベースに入れて管理します。不具合票の枚数を計測して検出不具合数とする場合などは結果として不具合でなかったもの（仕様の不理解、操作ミス、マニュアル不備等）と区別できるようにしておくことが必要ですが、本指標での検出不具合数は、厳密に細かい数を出すのではなく、不具合報告のうち、真に不具合なものは全体として何割あるか、というようなことをサンプリングで算出し、その割合を全体不

具合報告に乗ずる、というような簡易な方法を用いても構いません。

修正済み不具合数

上述した、不具合として検出されたもののうち、修正された不具合の数を計測します。不具合が修正されたかどうかは不具合票あるいは不具合データベース等を用いて管理されるべきですが、修正結果が反映されにくいプロジェクトもあるので注意が必要です。検出不具合数から残存（未修正）不具合数を減算することで、求めることもできます。

不具合発見率

この基礎指標は測定するのではなく、検出不具合数/テスト実施項目数で計算して求めます。テスト実施した規模あたり、どのくらいの不具合を検出できたかという値を示します。この値は、期間で区切って計算し、開発初期～開発後半と、開発終盤との比を見ることでテスト十分性品質評価指標の不具合収束率（ID：PD41、ROFC）を求めます。

Column

検出不具合のランク

一般的にテストで検出される不具合は、重大な不具合から軽微なものまでいろいろあります（これらをバグランクなどと呼んで区別する企業もあります）。不具合発見率を算出する際に必要となる不具合数の計測について、本ガイドでは不具合ランクに関してあまり触れていません。

実際の運用に際しては、不具合ランクを考慮するなどして、不具合のカウントルールを決めておく必要があります。

基礎指標　定義表の読み方

次に、基礎指標について1つずつ、解説します。各表は次のような構成になっています。

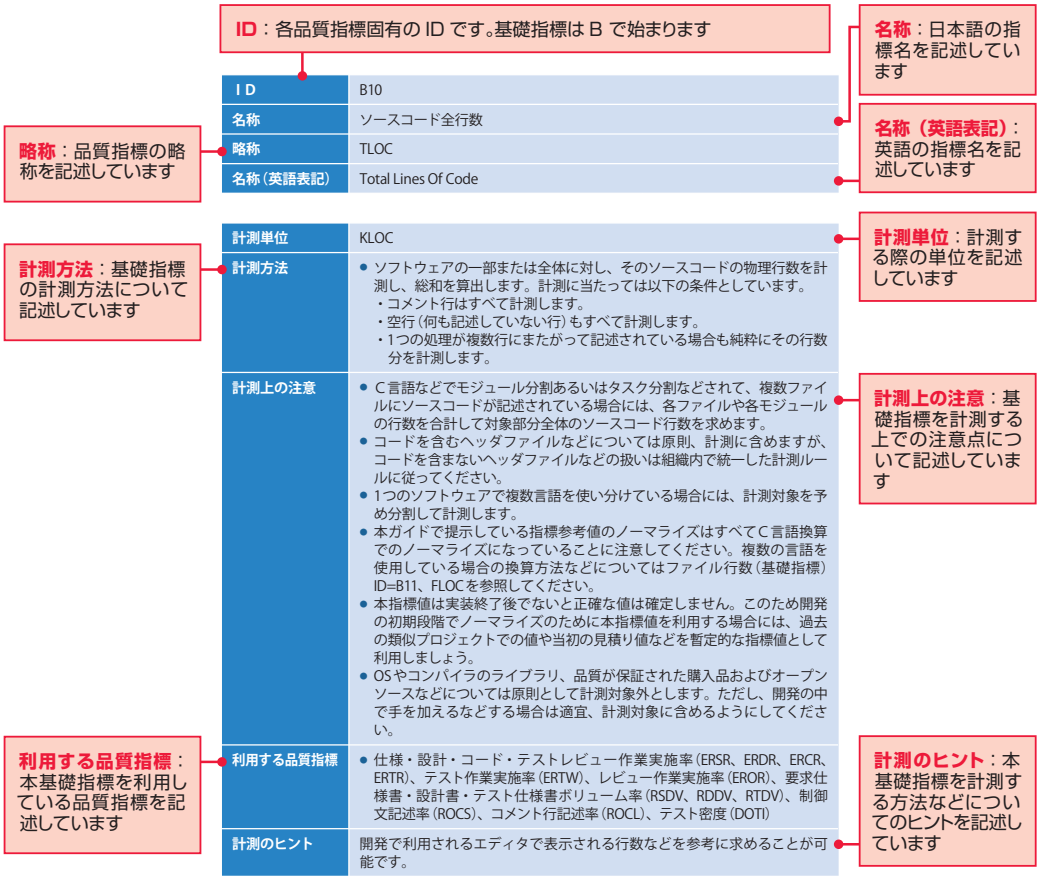


図3-5：基礎指標 定義表の読み方

基礎指標

ID	B10
名称	ソースコード全行数
略称	TLOC
名称（英語表記）	Total Lines Of Code

計測単位	KLOC
計測方法	● ソフトウェアの一部または全体に対し、そのソースコードの物理行数を計測し、総和を算出します。計測に当たっては以下の条件としています。 ・コメント行はすべて計測します。 ・空行（何も記述していない行）もすべて計測します。 ・1つの処理が複数行にまたがって記述されている場合も純粋にその行数分を計測します。
計測上の注意	● C言語などでモジュール分割あるいはタスク分割などされて、複数ファイルにソースコードが記述されている場合には、各ファイルや各モジュールの行数を合計して対象部分全体のソースコード行数を求めます。 ● コードを含むヘッダファイルなどについては原則、計測に含めますが、コードを含まないヘッダファイルなどの扱いは組織内で統一した計測ルールに従ってください。 ● 1つのソフトウェアで複数言語を使い分けている場合には、計測対象を予め分割して計測します。 ● 本ガイドで提示している指標参考値のノーマライズはすべてC言語換算でのノーマライズになっていることに注意してください。複数の言語を使用している場合の換算方法などについてはファイル行数（基礎指標）ID=B11、FLOCを参照してください。 ● 本指標値は実装終了後でないと正確な値は確定しません。このため開発の初期段階でノーマライズのために本指標値を利用する場合には、過去の類似プロジェクトでの値や当初の見積り値などを暫定的な指標値として利用しましょう。 ● OSやコンパイラのライブラリ、品質が保証された購入品およびオープンソースなどについては原則として計測対象外とします。ただし、開発の中で手を加えるなどする場合は適宜、計測対象に含めるようにしてください。
利用する品質指標	● 仕様・設計・コード・テストレビュー作業実施率（ERSR、ERDR、ERCR、ERTR）、テスト作業実施率（ERTW）、レビュー作業実施率（EROR）、要求仕様書・設計書・テスト仕様書ボリューム率（RSDV、RDDV、RTDV）、制御文記述率（ROCS）、コメント行記述率（ROCL）、テスト密度（DOTI）
計測のヒント	開発で利用されるエディタで表示される行数などを参考に求めることが可能です。

ID	B11
名称	ファイル行数
略称	FLOC
名称（英語表記）	File Lines Of Code

計測単位	KLOC
計測方法	● ソフトウェアの一部または全体に対し、それらを構成するファイル毎に行数を計測します。 ● 品質計測・評価の対象となる部分のソフトウェアの規模を表す指標です。この値が大きい場合には対象ソフトウェアの規模が大きいことを意味します。
計測上の注意	● ファイル内の物理行数をファイル毎に計測します。コメント行、空行などもそのまま計測します（行数の計測ルールはソースコード全行数（ID：B10、TLOC）に準じます）。 ● 多言語を使っている場合の換算は、たとえばCapers Jones[1]は下記のように経験則としてFP（ファンクションポイント）から比率データを例示しています。このように、過去のプログラムの実績で換算式を作成すると良いでしょう。 1FP ＝ 320ステートメント：アセンブリ言語 1FP ＝ 128ステートメント：C言語 1FP ＝ 53ステートメント：C++
利用する品質指標	ファイル行数（ＦＬＯＣ）
計測のヒント	開発で利用されるエディタで表示される行数などを参考に求めることが可能です。

1：「ソフトウェア見積りのすべて」Capers Jones 著　構造計画研究所　2001

ＩＤ	B12
名称	関数の行数
略称	MLOC
名称（英語表記）	Module Lines Of Code

計測単位	LOC
計測方法	● 関数毎の行数を計測します。1行の考え方はソースコード全行数（ID：B10、TLOC）の計測に準じます。 ● 一つのソフトウェアで複数言語を使い分けている場合には、言語に即した関数相当（サブルーチン、メソッド等）の計測を行ってください。換算の方法についてはファイル行数（ID：B11、FLOC）を参照してください。
計測上の注意	● 関数内の物理行数を関数毎に計測します。なお、関数内とは関数宣言部直後の{から}で囲まれる関数処理部本体を指します。 ● コメント行、空行等もそのまま計測します。
利用する品質指標	関数の行数（MLOC）
計測のヒント	すべての関数の値を計測するのが困難な場合、値の大きそうな関数を選んで計測するのでも良いでしょう。

ID	B13
名称	制御文数
略称	NOCS
名称（英語表記）	Number Of Control Statement

計測単位	LOC
計測方法	● ソフトウェアの一部または全体に対し、そのソースコードに含まれる制御文数を計測します。制御文数は、C言語の場合、次の予約語の個数を指します。 If / while / for / case / default / else
計測上の注意	● C言語以外の場合は、計測する制御文の種類をプロジェクトで決めて運用してください。 ● ソースコードのコメント部分を除いた部分を計測対象としますが、煩雑になる場合（本指標の計測のヒント欄を参照してください）は、おおよその数値で良いでしょう。
利用する品質指標	制御文記述率（ROCS）
計測のヒント	● エディタ、簡単な計測ツールやスクリプトを用いてソースコード中の制御文の数を計測することで制御文数の大まかな値を得ることができます。ただし、コメントを英文で記述している場合など、条件文（forなど）がコメントに頻繁に表れることがあります。そのような場合は、予めコメントを除外してから計測すると良いでしょう。計測の都合上、コメント部分を除外しきれないなどの場合には、ソースコードを目視確認し、コメント部分内に記載されている条件文等のおおよその割合を考慮して、総数から除外しておくなどすると良いでしょう。

ＩＤ	B14
名称	コメント行数
略称	CLOC
名称（英語表記）	Comment Lines Of Code

計測単位	K LOC
計測方法	● ソフトウェアの一部または全体に対し、そのソースコード中に含まれるコメント行数を計測します。 ● C言語の場合、/*から*/で囲まれた部分の行数、および、//で始まる行の数を数えるということで、おおよその値を求めることができます。また、コメントを書くときのスタイルをコーディングルールで定めておくと計測がより正確または楽になります。
計測上の注意	● 実行行の文末のコメントについては計測しません。 ● 組織またはプロジェクトでコメント行数の計測ルールを統一しておくと、ぶれなく効率よく計測することが可能になります。 ● コメント行数を計測できるツールがあればそれを利用しても構いませんが、ツールによっては計測のルールが異なる場合がありますので、利用するツールのルールを確認しましょう。ルールが変えられるのであれば、本指標で提案する方法に変えても構いませんし、プロジェクトで独自にコメントの計測ルールを決めてしまっても構いません。
利用する品質指標	コメント行記述率（ROCL）

ID	B15
名称	コーディングルール逸脱数
略称	NDCR
名称（英語表記）	Number of Deviation of Coding Rules

計測単位	箇所
計測方法	● コーディングルールを逸脱した延べ数を計測します。
計測上の注意	● 同じルールの逸脱でもソースコードの場所が異なる場合には重複して計測します。たとえば、同じルールの逸脱を2箇所で行った場合には、2とします。 ● また、組織内またはプロジェクト内コーディングルールに基づき、逸脱の手続きをとったものは、コーティング逸脱数の総数から除いて計測します。 ● すなわち、コーディングルール逸脱数として計測するものは以下の2つになります。 ・逸脱の手続きがあるルールで逸脱の手続きをとっていないもの ・逸脱を許さないルールのもの
利用する品質指標	コーディングルール逸脱率（RDCR）
計測のヒント	コーディングルール毎に逸脱数をまとめておくとコーディングルールそのものの見直しに役に立ちます。

Column

コーディングルール

一般にコーディングルールは組織などでユニークなルールを決め、開発者が遵守することが求められます。組織内の運用ルールによりますが、ルールとして定めたもののうち、一部についてはシステムとして求められている機能により注目すべき品質特性が異なるために定めたルール通りに記述すると目的が達成できないことがあります。その場合は、ルール適用除外の手順を定めて運用します（ESCR Part.1 3.2節参照）。ルール違反が検出されたもののうちルール適用除外の手順に従って処理されるものについては、コーディングルール逸脱数の計測に含めないものとします。

ID	B20
名称	要求仕様書ボリューム
略称	VOSD
名称（英語表記）	Volume Of the Specifications Document

計測単位	ページ
計測方法	● プロジェクトで作成するドキュメントのうち、要求仕様を記述するすべての文書の総ページ数を計測します。 ● ソフトウェア開発におけるドキュメントはWord他のツールや場合によってはExcel他の表や図などさまざまな形式で作成されます。また、同じWordなどで作成されても1ページあたりの行数や文字数はさまざまです。このため、この指標の計測にあたっては、2000字相当（横40字×縦50行）を1ページとみなし、他のツールなどでドキュメントを作成した場合でも、このページ換算でおおよそのページ数を算出してください。
計測上の注意	● 文書に図や写真、ダイヤグラムなどが含まれる場合には、それらをテキストで表現した場合に何ページ分くらいかを考えページ数に加算します。 ● 既存ソフトウェアの再利用などに伴い、ドキュメントの一部を再利用している場合は、それらのドキュメントも含めて全体のページ数を計測します。
利用する品質指標	要求仕様書ボリューム率（RSDV）
計測のヒント	● ドキュメントのページ数については、その記述方法によってかなりの差異が生じます。このため、本指標ではその値に極端な厳密さは要求せず、おおよその値でドキュメントの十分性を評価することを主眼としています。
補足	（ドキュメントバランス記述量計測の場合） ● 個別の記述項目の記述量については、単純に1.5ページ位などのようにページ換算で求める、あるいは行換算で求めるなどしておおよその記述量を把握できればよいものとします。場合によっては、記述エリアの面積や記述の物理的な幅（縦・横の長さ）などの物理的な値によって記述されたパーセントを求める方式でも構わないでしょう。

ＩＤ	B21
名称	設計書ボリューム
略称	VODD
名称（英語表記）	Volume Of the Design Document

計測単位	ページ
計測方法	● プロジェクトで作成するドキュメントのうち、設計に関する記述の文書の総ページ数を計測します。 ● ソフトウェア開発におけるドキュメントはWord他のツールや場合によってはExcel他の表や図などさまざまな形式で作成されます。また、同じWordなどで作成されても1ページあたりの行数や文字数はさまざまです。このため、本指標の計測にあたっては、2000字相当（横40字×縦50行）を1ページとみなし、他のツールなどでドキュメントを作成した場合でも、このページ換算でおおよそのページ数を算出してください。
計測上の注意	● 文書に図や写真、ダイヤグラムなどが含まれる場合には、それらをテキストで表現した場合に何ページ分くらいかを考え、ページ数に加算します。 ● 既存ソフトウェアの再利用などに伴い、ドキュメントの一部を再利用している場合は、それらのドキュメントも含めて全体のページ数を計測します。
利用する品質指標	設計書ボリューム率（RDDV）
計測のヒント	● ドキュメントのページ数については、その記述方法によってかなりの差異が生じます。このため、本指標ではその値に極端な厳密さは要求せず、おおよその値でドキュメントの十分性を評価することを主眼とします
補足	（ドキュメントバランス記述量計測の場合） ● 個別の記述項目の記述量については、単純に1.5ページ位などのようにページ換算で求める、あるいは行換算で求めるなどしておおよその記述量を把握できればよいものとします。場合によっては、記述エリアの面積や記述の物理的な幅（縦・横の長さ）などの物理的な値によって記述されたパーセントを求める方式でも構わないでしょう。

ＩＤ	B22
名称	テスト仕様書ボリューム
略称	VOTD
名称（英語表記）	Volume Of the Test Document

計測単位	ページ
計測方法	● プロジェクトで作成するドキュメントのうちテストに関する仕様書類の総ページ数を計測します。 ● ソフトウェア開発におけるドキュメントはWord他のツールや場合によってはExcel他の表や図などさまざまな形式で作成されます。また、同じWordなどで作成されても1ページあたりの行数や文字数はさまざまです。このため、この指標の計測にあたっては、2000字相当（横40字×縦50行）を1ページとみなし、他のツールなどでドキュメントを作成した場合でも、このページ換算でおおよそのページ数を算出してください。
計測上の注意	● 文書に図や写真、ダイヤグラムなどが含まれる場合には、それらをテキストで表現した場合に何ページ分くらいかを考え、ページ数に加算します。 ● 既存ソフトウェアの再利用などに伴い、ドキュメントの一部を再利用している場合は、それらのドキュメントも含めて全体のページ数を計測します。
利用する品質指標	テスト仕様書ボリューム率（RTDV）
計測のヒント	● ドキュメントのページ数については、その記述方法によってかなりの差異が生じます。このため、本指標ではその値に極端な厳密さは要求せず、おおよその値でドキュメントの十分性を評価することを主眼とします。
補足	（ドキュメントバランス記述量計測の場合） ● 個別の記述項目の記述量については、単純に1.5ページ位などのようにページ換算で求める、あるいは行換算で求めるなどしておおよその記述量を把握できればよいものとします。場合によっては、記述エリアの面積や記述の物理的な幅（縦・横の長さ）などの物理的な値によって記述されたパーセントを求める方式でも構わないでしょう。

ID	B30
名称	全レビュー工数
略称	RETO
名称（英語表記）	Review Effort in TOtal

計測単位	人時
計測方法	● 仕様・設計・コード・テストすべてのレビューにかけた工数の総和を計測します。 ● 計測にあたっては、以下のように行います。 ・仕様作成、設計、実装、テスト準備、テスト実施などに関する工数は含みません。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	レビュー作業実施率（EROR）
計測のヒント	レビュー記録票などをもとに、毎回の時間×実施回数×参加者数で算出します（Σ 時間 x 参加者数）。

ID	B31
名称	仕様レビュー工数
略称	RESP
名称（英語表記）	Review Effort for SPecification

計測単位	人時
計測方法	● 仕様検討のアウトプットである、仕様書のレビューに要した工数を計測します。 ● レビュー対象は、ESPR（開発プロセスガイド）のSYP1（システム要求定義）およびSWP1（ソフトウェア要求定義）でのアウトプット、すなわち、システム要求仕様書、ソフトウェア要求仕様書、そのほかプロジェクトに応じた仕様書類です。 ● 計測にあたっては、以下のように行います。 ・仕様作成に関する工数は含みません。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	仕様レビュー作業充当率（RSRE）、仕様レビュー作業実施率（ERSR）
計測のヒント	レビュー記録票などをもとに、毎回の時間×実施回数×参加者数で算出します（Σ 時間 x 参加者数）。

ID	B32
名称	設計レビュー工数
略称	REDE
名称（英語表記）	Review Effort for DEsign

計測単位	人時
計測方法	● 設計のアウトプットである、設計書のレビューに要した工数を計測します。 ● レビュー対象は、ESPR（開発プロセスガイド）のSYP2（システムアーキテクチャ設計）およびSWP2・3（ソフトウェアアーキテクチャ設計、ソフトウェア詳細設計）でのアウトプット、すなわち：システム・アーキテクチャ設計書、システム動作設計書、システムインタフェース設計書、システム・アーキテクチャ設計書、ソフトウェア構成設計書、ソフトウェア動作設計書、ソフトウェア・インタフェース設計書、ソフトウェアアーキテクチャ設計書、ソフトウェア詳細設計書、プログラムユニット機能/構成設計書等、そのほかプロジェクトに応じた設計書類となります。計測にあたっては、以下のように行います。 ・設計に関する工数は含みません。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	設計レビュー作業充当率（RDRE）、設計レビュー作業実施率（ERDR）
計測のヒント	レビュー記録票などをもとに、毎回の時間×実施回数×参加者数で算出します（Σ 時間 x 参加者数）。

ＩＤ	B33
名称	コードレビュー工数
略称	RECO
名称（英語表記）	Review Effort for COde

計測単位	人時
計測方法	● ソースコードのレビューに要した工数を計測します。 ● レビュー対象は、ESPR（開発プロセスガイド）のSWP4（実装および単体テスト）でのアウトプット、すなわちプログラムユニット、ソースコードです。 ● 計測にあたっては、以下のように行います。 ・実装および単体テストに関する工数は含みません。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	コードレビュー作業充当率（RCRE）、コードレビュー作業実施率（ERCR）
計測のヒント	レビュー記録票などをもとに、毎回の時間×実施回数×参加者数で算出します（Σ 時間 x 参加者数）。

ID	B34
名称	テストレビュー工数
略称	RETP
名称（英語表記）	Review Effort for Test Preparation

計測単位	人時
計測方法	● テスト関連のアウトプットのレビューに要した工数を計測します。レビュー対象は、ESPR（開発プロセスガイド）のSYP3（システム結合テスト）、SYP4（システムテスト）、SWP5（ソフトウェア結合テスト）およびSWP6（ソフトウェア総合テスト）すなわちそれぞれのテスト仕様書、テスト環境、テストデータ、テスト結果、テスト報告書などです。 ● 計測にあたっては、以下のように行います。 ・テスト実行、環境整備、文書作成等に関する工数は含みません。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	テストレビュー作業充当率（RTRE）、テストレビュー作業実施率（ERTR）
計測のヒント	レビュー記録票などをもとに、毎回の時間×実施回数×参加者数で算出します（Σ 時間 x 参加者数）。

ID	B35
名称	開発全工数
略称	PETO
名称（英語表記）	Process Effort in TOtal

計測単位	人時
計測方法	● プロセス品質評価指標、プロダクト品質評価指標が対象とするソフトウェアの一部または全体の開発に要した全工数を計測します。 ● 計測にあたっては、以下のように行います。 ・要求、設計、実装、テストなどの開発に関する直接作業および品質管理他の開発に関する間接作業などすべての工数を合計して求めます。 ・工数の算出では、作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。 ● 本指標は基本的に開発終了後でないと正確な値は確定できません。このため、この指標を開発途中で利用する場合には、過去の類似プロジェクトなどの値を参考に想定される値あるいは見積りの値を暫定値として利用します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。 ● 要員の教育など、直接的な成果物のない作業は含めません。
利用する品質指標	レビュー作業充当率（RORE）、テスト作業充当率（RTWE）
計測のヒント	基本的には各工程での投入（予定）人数と工程の見込み期間をもとに全体工数を算出する程度でも構いません。

Column 指標の解釈に関するヒント

開発者あるいはプロジェクトマネージャやリーダが感覚的にとらえているソフトウェア開発の全工数と開発全工数の実測値とが大きく乖離している場合には、開発に際して開発者が不要な作業をしたり、作業に手間取ったりしている可能性が考えられます。このため管理者は必要に応じて作業内容の確認やレビューなどを行う必要があります。

ID	B36
名称	仕様作成工数
略称	PESP
名称（英語表記）	Process Effort for SPecification

計測単位	人時
計測方法	● 仕様を作成検討するための作業に要した工数を計測します。 ● 対象は、ESPR（開発プロセスガイド）のSYP1（システム要求定義）およびSWP1（ソフトウェア要求定義）でのアウトプット、すなわち、システム要求仕様書、ソフトウェア要求仕様書、そのほかプロジェクトに応じた仕様書類になります。 ● また仕様のレビューに要した工数も含めるものとします。 ● 計測にあたっては、以下のように行います。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	仕様レビュー作業充当率（RSRE）
計測のヒント	

ID	B37
名称	設計作成工数
略称	PEDE
名称（英語表記）	Process Effort for DEsign

計測単位	人時
計測方法	● 設計を行うための作業に要した工数を計測します。 ● 対象は、ESPR（開発プロセスガイド）のSYP2（システムアーキテクチャ設計）およびSWP2・3（ソフトウェアアーキテクチャ設計、ソフトウェア詳細設計）でのアウトプット、すなわち：システム・アーキテクチャ設計書、システム動作設計書、システムインタフェース設計書、システム・アーキテクチャ設計書、ソフトウェア構成設計書、ソフトウェア動作設計書、ソフトウェア・インタフェース設計書、ソフトウェアアーキテクチャ設計書、ソフトウェア詳細設計書、プログラムユニット機能/構成設計書等、そのほかプロジェクトに応じた設計書類となります。 ● 設計のレビューに要した工数も含めます。 ● 計測にあたっては、以下のように行います。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	設計レビュー作業充当率（RDRE）
計測のヒント	

ID	B38
名称	コード作成工数
略称	PECO
名称（英語表記）	Process Effort for COde

計測単位	人時
計測方法	● ソースコードを作成するための作業に要した工数を計測します。 ● 対象は、ESPR（開発プロセスガイド）のSWP4（実装および単体テスト）でのアウトプット、すなわちプログラムユニット、ソースコードです。 ● ソースコードレビューに要した工数も含めます。 ● 計測にあたっては、以下のように行います。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。
利用する品質指標	コードレビュー作業充当率（RCRE）
計測のヒント	

ID	B39
名称	テスト準備・確認工数
略称	PETP
名称（英語表記）	Process Effort for Test Preparation

計測単位	人時
計測方法	● テストにかかわる作業のうち、テスト仕様書作成、テスト環境整備、テスト結果の確認、およびテストのレビューなどに要した工数の合計を計測します。なお、テスト実施に要した工数は含みません。 ● 対象は、ESPR（開発プロセスガイド）のSYP3（システム結合テスト）、SYP4（システムテスト）、SWP5（ソフトウェア結合テスト）およびSWP6（ソフトウェア総合テスト）になります。 ● 計測にあたっては、以下のように行います。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。 ● テストにかかわる作業のうち、実行を除くものすべての工数を計測します。 ● SWP4（単体テスト）の項目は含みません。
利用する品質指標	テストレビュー作業充当率（RTRE）
計測のヒント	

ID	B3A
名称	テスト工数
略称	PETE
名称（英語表記）	Process Effort for TEst

計測単位	人時
計測方法	● テスト関連の作業に要したすべての工数を計測します。すなわちテストの準備、テスト環境整備、テスト実行、テスト結果確認テストレビューなどが含まれます。 ● 作業対象は、ESPR（開発プロセスガイド）のSYP3（システム結合テスト）、SYP4（システムテスト）、SWP5（ソフトウェア結合テスト）およびSWP6（ソフトウェア総合テスト）になります。 ● 計測にあたっては、以下のように行います。 ・作業期間および投入人員などをもとに作業工数を人時で求めます。 ・組織での工数計測の単位が「時間」でなく、「日数」である場合は、一日あたりの時間数を乗じて算出します。
計測上の注意	● 作業時間の計測を厳密に行う場合には、工数記録、作業日誌他から算出しますが、本指標の趣旨やデータ精度を考えた場合、極端に厳密な工数測定は必要としません。作業者の自己申告などおおよその概算の値やみなし工数を採用しても構いません。 ● レビュー時にプロジェクト外からレビューアが参加した場合、その工数も含めて計上します。 ● テストにかかわる工数すべてを計測します。（テスト準備・確認工数＋テスト実行工数） ● 工数の算出では、作業期間および投入人員などをもとに作業工数を人時で求めます・SWP4（単体テスト）は含みません。
利用する品質指標	テスト作業充当率（RTWE）、テスト作業実施率（ERTW）
計測のヒント	（1回のテスト工数×テスト要員数×テスト回数）でおおよその値を計測できます。

ID	B40
名称	テスト項目数
略称	NOTI
名称（英語表記）	Number Of Test Items

計測単位	項目
指標値の意味	● テスト項目数はテストの規模（テストを実施した数）を表す指標です。 ● 対象ソフトウェアの規模が大きい、複雑である場合にはこの値も大きくなります。
計測方法	● テストを実施した項目数を計測します。 ● 対象テストは、ESPR（開発プロセスガイド）のSYP3（システム結合テスト）、SYP4（システムテスト）、SWP5（ソフトウェア結合テスト）およびSWP6（ソフトウェア総合テスト）です。 ● テスト項目の単位は対象製品、組織により、また、テストの種類によっても小項目単位であったり、スクリプト数であったりとさまざまです。また、同じ組織でも、テストの目的により、記述方法が異なる場合も多いでしょう。このため、この指標の計測にあたっては、プロジェクト内で表現を決め、ばらつきが少なくなるよう、調整する必要があります。 ● 本ガイドではテスト項目の計測方法は、ESPRの2.3 ドキュメント・テンプレート例の、P.192 ～ 195テスト仕様書 / テスト報告書に記述されているテスト項目詳細の項をベースとしています。すなわち、テスト内容、テストデータ、期待する出力、環境条件、テスト結果などを一行で表していますが、この一行を一項目としています。これを参考に、項目の値を計測する、あるいは組織の計数の仕方を考慮して換算しても構いません。
計測上の注意	● 計測にあたっては、以下を注意します。 ・SWP4（単体テスト）の項目は含みません。 ・既存ソフトウェアの再利用などに伴い、テスト項目も再利用している場合、再利用したテスト項目も計測の対象とします。 ・一度も実施していないテスト項目は計測の対象としません。
利用する品質指標	テスト密度（DOTI）
計測のヒント	● テスト仕様書内のテスト項目から厳密に測定しても構いませんが、項目を記述したページ数×1ページあたりの行数などからおおよその値を換算することができます。 ● テスト項目の数え方は、テスト仕様書のフォーマットや記述の粒度によって異なる場合があります。このため、テスト項目の数え方の一つの目安として、テスト仕様書に記述されたテスト項目について、100字（40字×2.5行）を一単位として、この分量を内容に関わらず1項目と計測しても構いません。

ID	B41
名称	テスト実施項目数
略称	NOET
名称（英語表記）	Number Of Executed Test Items

計測単位	項目
計測方法	● 実施したテストの項目の延べ実行数を計測します。 ● 対象となるテストは、ESPR（開発プロセスガイド）のSYP3（システム結合テスト）、SYP4（システムテスト）、SWP5（ソフトウェア結合テスト）およびSWP6（ソフトウェア総合テスト）になります。 ● テスト項目の単位は対象製品、組織により、また、テストの種類によっても小項目単位であったり、スクリプト数であったりとさまざまです。また、同じ組織でも、テストの目的により、記述方法が異なる場合も多いでしょう。このため、この指標の計測にあたっては、プロジェクト内で表現を決め、ばらつきが少なくなるよう、調整する必要があります。
計測上の注意	● 計測に当たっては、以下を注意します。 ・SWP4（単体テスト）の項目は含みません。 ・延べ実施項目数を計測します。たとえば、同じテストセットを複数回実施した場合は（テスト項目数×テスト実施回数）項目として計測します。 ・テストを途中で停止した場合、それ以降の（実施しなかった）テスト項目は計測しません。
利用する品質指標	不具合収束率（ROFC）、不具合発見率（ROFD）
計測のヒント	● テスト報告書に記載された実施テスト項目を積み上げることで計測します。 ● テスト成績書の実施結果などの項目に関して、テスト項目数の場合と同様に100文字を一単位として実施したテスト項目数の概算を出す方式でも構いません。

ＩＤ	B42
名称	検出不具合数
略称	NODF
名称（英語表記）	Number Of Detected Fault

計測単位	件
計測方法	● 結合テスト以降のレビュー、テスト、そのほかあらゆる場面で検出した不具合数を計測します。 ● ただし、SWP4（単体テスト）で検出した不具合については含みません。
計測上の注意	● 不具合票等を数えることで計測が可能です。 ● ただし、同じ原因による不具合、誤報告などは除く必要があります。不具合が一つ一つ管理され、計測できることが望ましいのですが、そうでない場合、一定の乗率を総件数に乗ずるなどし、おおよその値を算出すれば良いでしょう。 ● 不具合は重大なもの、軽微なものなどさまざまですので単純な数の評価だけではなく、不具合内容なども考慮した評価が必要になります。特に重大な不具合については、重み付けの計数を乗ずるなどして調整すると良いでしょう。 ● また、テスト時に不具合としたもので、最終的に仕様または制限として開示するものは不具合の総数から除いて計測します。
利用する品質指標	不具合収束率（ROFC）、不具合修正率（ROFE）、不具合発見率（ROFD）
計測のヒント	● 不具合票のページ数、一覧にしている場合は総数などで計測できます。 ● また、不具合データベースなどを使用することで計測は容易になります。

ID	B43
名称	修正済み不具合数
略称	NOEF
名称（英語表記）	Number Of Eriminated Fault

計測単位	件
計測方法	● 結合テスト以降にレビュー、テスト、そのほかあらゆる場面で検出した不具合数のうち、修正されたものを計測します。
計測上の注意	● 不具合票を用いて計測できます。 ● 不具合総数から残存不具合数を引き算することで求められます。 ● 不具合票では修正済みになっていても、修正の確認ができていないもの、また、修正したのに不具合票には反映されていないものなど、計測するタイミングによっては発生しますので注意が必要です。
利用する品質指標	不具合修正率（ROFE）
計測のヒント	● 修正済み不具合数は開発開始時点では0であり、開発とともに検出不具合数の後追いで増加します。すなわち、検出不具合数≧修正済み不具合数は常に成り立ちます。 ● 不具合票の一覧ページを用意しておくと計測は容易になるでしょう。

ＩＤ	B44
名称	不具合発見率
略称	ROFD
名称（英語表記）	Ratio Of Fault Detection

計測単位	%
計測方法	不具合発見率＝一定期間内の（検出不具合数/テスト実施項目数）
計測上の注意	● 本指標は、直接計測するのではなく、上記の計算式で求めます。 ● ある一定期間に検出した検出不具合数をその期間に行ったテスト実施項目数に対するバランスで示します。
利用する品質指標	不具合収束率（ROFC）
計測のヒント	● 本指標を利用する品質指標：不具合収束率（ROFC）では開発期間の当初～90%までと残り10%に分けて本指標を利用します。従って開発期間と、検出不具合数および、テスト実施項目数を対応付けられることが必要となります。不具合票および、作業日誌他の記録などから対応付けを行うこととなりますので、本指標を利用する場合は、日程を追って記録できるようにしておきましょう。

第4章

高品質作り込みのためのヒント

2章、3章では客観的に測ることのできる定量的な指標を用いて品質をコントロールする考え方を説明しましたが、本書では少し視点を変えて、開発の各局面での組込みソフトウェアの品質を向上するという観点から作業の質を定性的に確認するためのヒントを紹介します。よりよい品質を作り込むためには、適切なタイミングで適切な手だてを用いて開発を進める必要があります。品質を高めるための技術、手法、工夫などはすでに世の中に知見として数多く紹介されています。本章ではそれら手だてが開発に活かせているか、職場が活性化できているか、などについてまとめました。これらのヒントの多くは実際の開発の現場からいただいたもので、皆さんの開発の現場でも、何かしら気づきを得られるものも含まれていると思います。本章ではこれらのヒントを5つの側面にまとめてから整理し、それぞれチェック表の形にしました。どうぞ、皆さんの職場の活性度合いなどをこのチェック表を用いて見直してみてください。
また、参考となる書籍も巻末に紹介しますのであわせてご覧ください。

4.1 開発におけるコミュニケーションと意思決定 …… 136
4.2 ドキュメント …… 141
4.3 レビュー …… 147
4.4 テスト …… 152
4.5 指標を用いた品質作り込み活動 …… 159

4.1 開発におけるコミュニケーションと意思決定

コミュニケーションと意思決定の重要性

ソフトウェア開発プロジェクトの失敗原因を分析していると、「技術上の問題」では無く、「ヒトに関する問題」が浮かび上がることが多くのケースであります。特に「ヒトに関する問題」には、チーム内あるいはステークホルダとのコミュニケーションに関する問題や、意思決定の問題などがあり、これらがプロジェクトの失敗につながることがよくあります。また、近年のソフトウェア開発は規模が拡大し、多人数による開発形態が主流になり、「コミュニケーションの重要性」は増すばかりです。

ソフトウェア開発でのコミュニケーションには大きく分けて、以下の２つのタイプが考えられます。

- 情報を伝達する（一方向）
- 相談する・検討して意思決定をする（双方向）

一人で開発する場合にはコミュニケーションは不要ですが、複数人で開発を進めるときには情報伝達と、それぞれの事情を考えた意思決定が特に重要になってきます。

多くの人手によって行われるソフトウェア開発で、高い生産性や高い品質を達成するには、リーダとメンバ、メンバ間、ステークホルダ間での信用・信頼と確実な情報伝達・意思決定を司るコミュニケーションが必要不可欠となります。

しかし、こうしたコミュニケーションを円滑に行うためには、経験によって養われるテクニックなどもあり、また個人のセンスに依存することがあるのも事実です。円滑なコミュニケーションや速やかでしこりの残らない意思決定を行うためには以下のような点に留意する必要があります。

- コミュニケーションの目的に応じたやり方、しくみを考える
- 前提として、人間が行う活動であるので、人間的な側面を考える

コミュニケーションに関するチェック項目

NO.	チェック項目	チェック
1	円滑なコミュニケーションを行う土壌があるか	
2	コミュニケーションの相手を尊重しているか	
3	会議の事前準備、環境が整っているか	
4	必要なテーマに対して、必要な時間を割いて情報展開、議論しているか	
5	過去の経験や有識者の経験を尊重しているか	
6	議論、話している内容の本質的なところが共有されているか	
7	意思決定の道筋は合意が得られているか	
8	必要な情報が必要なメンバに正しく伝わっているか	
9	メールなどの非同期コミュニケーションに依存しすぎていないか	
10	生産性/品質の向上につながるようなコミュニケーションがとられているか	
11	会議やコミュニケーションの結果が開発に反映されているか	
12	会議が終わった後に無常感が漂っていないか	

各項目に関する解説

Check 1 円滑なコミュニケーションを行う土壌があるか

プロジェクトの目的や目標を共有し、仕事を進めていくには、円滑なコミュニケーションが必要です。円滑なコミュニケーションを成立させるには、個人・チーム・ステークホルダ間で風通し良く、報告・連絡・相談できる土壌（環境や雰囲気）が必要です。こうした土壌が形成されているかどうかについては、人間的な側面からも考えなくてはなりません。このためには　**①挨拶がきちんとできている　②プロジェクトに笑顔がある／全員の顔が明るい　③隣の人が何をしているか知っている　④組織間の風通しが良い**　このような観点からチェックしてみると良いでしょう。

Check 2 コミュニケーションの相手を尊重しているか

円滑なコミュニケーションを実施し、高い品質を維持するためには、コミュニケーションの場におけるメンバの満足度を上げ、相互に十分理解できるようにする工夫も必要です。力関係でメンバを圧倒してしまうことや、まして個人攻撃でメンバのモチベーションを削ぐようなコミュニケーションは論外です。

①相手の立ち位置（背景）を理解してコミュニケーションする、②メンバ全員が

理解できる言葉を使うことを心掛けましょう。

Check 3　会議の事前準備、環境が整っているか

会議で検討されるテーマや目的が周知されていないと事前の準備や検討もできず、趣旨の説明などにも貴重な時間を奪われます。しっかりした事前準備を行うことで、会議の重要性の認識や集中力が高められ、充実した議論につながります。このためには　**①会議は事前に目的が告知されている　②会議のアジェンダは予め周知されている　③必要なアイテム（ホワイトボード・プロジェクタなど）が準備されている**、といった点などを確認してみてはいかがでしょうか。

Check 4　必要なテーマに対して、必要な時間を割いて情報展開、議論しているか

プロジェクトの目的や目標を共有し、そのゴールを達成するには、**①必要なテーマに対して十分な時間を割き、②異なる意見や方法論も交え、十分な議論を行い、③関わるメンバ全員が決定事項に納得し、高いモチベーションを持って仕事を進めていく**ことが重要です。情報展開や議論の重要性は会議だけでなく、日常の報告・指示でも同じであることはいうまでもありません。対立する意見や反対意見の提案、指摘事項に対しては、対案や対策への意見が出されているかなど、十分検討されることが大切です。

Check 5　過去の経験や有識者の経験を尊重しているか

正しい意思決定を行うには、チームや組織で蓄積した過去の知見や経験を活かす事が重要です。このためには　**①会議には必要な人が参加（メンバが適切である）していること**、かつ参加したメンバの　**②過去の経験が活かされるように工夫すること**がポイントになります。

Check 6　議論、話している内容の本質的なところが共有されているか

ソフトウェア開発といっても、そのプロセスは要求、設計から実装、テストまで多岐に渡ります。またこうしたプロセスにはソフトウェア担当者だけではなく、ハードウェア担当者、あるいは経営者や、時には法規担当者など、多くの役割の人が携わることが多々あります。このため開発におけるコミュニケーションの場では、特に　**①ステークホルダを明確にし、開発のそれぞれの場面で異なる役割の人たちの要求を正しく理解し、一貫したソフトウェア設計・実装を行っていく**必要があります。このためには　**②すべての関係者間で議論内容の本質が共有されていること**が必要です。

Check 7 意思決定の道筋は合意が得られているか

意思決定とは、ある問題に際して、いくつかの案の中から、とるべき方向を決めるプロセスです。

ソフトウェア開発のコミュニケーションの場では、さまざまな意思決定がプロジェクトの成否を左右する重要な役割を持ちます。このため意思決定を行う場合には、単にその結果そのものを導くだけではなく、**①意思決定を行ううえでの道筋に合意が形成され**、その **②責任所在が明確になること**により、**③目的・目標の共有がなされること**が重要です。また、リーダにより、トップダウンで意思決定された場合にも、その内容に関して適切な判断根拠の説明などを心がける必要があります。

Check 8 必要な情報が必要なメンバに正しく伝わっているか

高品質なソフトウェアを開発するには、仕様や仕様変更に関する情報などに、不足や不明な部分があってはなりません。コミュニケーションや意思決定の観点から情報の不足や不明点について考えた時、重要なのは **①必要な情報が関係者全員に正しく伝達され、②情報に不足や不明な部分があった場合には確認できる体制が整えられ、③仕様変更、不具合などが発生した時の連絡体制が整備されていること**です。こうした点に配慮して必要な情報が必要なメンバに適切に伝わるよう工夫することが求められます。

Check 9 メールなどの非同期コミュニケーションに依存しすぎていないか

コミュニケーションには、目的に応じたやり方、仕組みがあります。近年は電子メールやスケジュール管理ソフト、電子決済システムなどの便利なツールにより、生産性や管理の精度も向上しています。しかし、こうしたコミュニケーションツールの進歩の一方で人と人が実際に対峙して交わされるコミュニケーションが省略され、人間同士の接触から得られるコミュニケーションによる重大な情報や意思決定を逃すことも少なくありません。円滑なコミュニケーションの達成のためには、**①電子ツールなどを使用した仮想のコミュニケーションと現実のコミュニケーションとを目的に応じて使い分けること**も重要です。

Check 10 生産性／品質の向上につながるようなコミュニケーションがとられているか

ソフトウェア開発での円滑なコミュニケーションの大きな目的は「生産性と品質の向上」です。打合せや会議を実施すること、それ自体が目的ではありません。正しい役割分担を行い、先人の経験や知見を活かし正しい判断を迅速に行って

いくことが大切です。良いコミュニケーションができている組織では　**①新しい仕事が発生したときに自然と担当者が決まっている　②コミュニケーションを通して、後進をきちんと育てている**といった兆候が見られます。

Check 11　会議やコミュニケーションの結果が開発に反映されているか

円滑なコミュニケーションのもと、会議や意思決定がされることは重要ですが、**①会議の決定や意思決定された事項がすべてのメンバに周知され、開発の各場面に反映され**なければ意味がありません。それには**②議事録に必要十分な内容が整理されて書かれ、③議事録、必要な資料は必要なメンバに配布され、④上位層の意思が下位レベルまで伝わること**が大切です。

Check 12　会議が終わった後に無常感が漂っていないか

無常感とは「虚しさ・あきらめ・しらけ」のことです。この感覚が漂うコミュニケーションや会議では高い生産性や品質は保てません。会議中、途中離席や内職する人が多数いたり会議終了後に問題が提起されたりするようなケースは良好なコミュニケーション環境が形成されていない兆しです。参加者全員が目的を共有し、しらけない環境作りが重要です。**①目的を持って会議を開催する　②結論が出るようリーダシップを発揮する　③適切な時間内で終わらせる　④マンネリ化しないような工夫をする**など心掛けると良いでしょう。

4.2 ドキュメント

ドキュメントの目的と役割

ソフトウェアは仕様、設計、実装、テストなど、複数のプロセスを経て、複数の人数で開発されます。複数のプロセスにまたがる仕事、複数の人数で行う仕事では何をどのように行うかということを記したレシピが必要です。

ドキュメントは、記載される情報の入出力（IN/OUT）を明確にし、何をしたら良いか、何をしたかを他人（未来の自分を含みます）に伝える役割があります。つまり、ドキュメントの良し悪しは、「このドキュメントを見て次にやることがすべてわかるか」「このドキュメントを見て何が行われたかすべてわかるか」を目安に確認すれば良いでしょう。一方で、ドキュメントの問題となるのは

- 情報欠落・間違い
- 誤解がおきやすい・古い・引用ミス
- ドキュメントそのものがない
- うそ
- 読みとれない

などが挙げられます。

ドキュメントそのものが存在しない場合は論外としても、せっかく作成されたドキュメントも信頼性を欠いたものでは、まったく意味がありません。これらの問題を防ぐことがよいドキュメント作りと考えることができるでしょう。

さらに、仕様書をはじめとするドキュメント類の記述ルールを定めることにより次の利点を得ることができます。

設計書を例にとると、次のようなことが考えられます。

正確な設計が可能となる

書き方・読み方が定まった記述方法を用いることにより、記述漏れ、曖昧さ、重複などの少ない、より正確な設計を行えるようになります。それに伴い設計結果が適切かどうかをより厳密に確認できるようになります。

設計が見えるようになり、レビュー効率が上がる

ソフトウェアの設計がより把握しやすくなり、レビュー効率がアップします。また、品質管理のベースとしてさまざまな計測を行いやすくなります。

コミュニケーションが円滑になる

正確かつ理解しやすい設計書を記述することにより、開発に関わるさまざまな人の間でのコミュニケーションが円滑に行われるようになります。

標準化が促進され再利用が容易になる

標準記法を用いることにより、ドキュメントの標準化が促進され、ソフトウェア開発上の各成果物の再利用が容易となります。

ドキュメントに関するヒント

NO.	チェック項目	チェック
1	何をいつ作成するかが定義されているか	
2	改版などのルールがあるか	
3	承認の手順は決まっているか	
4	構成・内容は適切か	
5	必要なドキュメントは揃っているか、不要なドキュメントはないか	
6	決められたフォーマットに従いきちんと記載しているか	
7	用語が定義されているか	
8	第三者が読むことを意識して書かれているか	
9	ドキュメントを作成する際のもととなる情報、ドキュメントに記載された情報を利用する次のドキュメントは明確か	
10	誤字脱字は許容範囲内か、省略形はきちんと説明されているか、意味不明な表現はないか	
11	作成者以外の誰かが目を通しているか	
12	ドキュメントとして盛り込むべき必要な内容が含まれ、システムの規模に応じた適切なボリュームとなっているか	

各項目に関する解説

Check 1 何をいつ作成するかが定義されているか

ソフトウェア開発の過程ではさまざまなドキュメントが作成されますが、それらには個々の目的があります。「どのようなドキュメントをいつ作成するか」は、ソフトウェア開発プロセスに対応して定義すべきであり、適切なタイミングで目的に合致した適切なドキュメントをまとめていく必要があります。ドキュメントの作成タイミングやその内容の定義は、質のよいドキュメントを作る上での最も基本的な要件になります。このためには、まず　**①ソフトウェア開発プロセスに対応したドキュメント作成計画を立てる**ところから始めることが大切です。

Check 2 改版などのルールがあるか

昨今のシステム開発では新しいシステムのすべてを新規に開発することはほとんどありません。既存のシステムの機能を再利用して新しいシステムを設計することになります。製品開発では、僅かなバージョンの違いや機能や処理の類似した部品の取り違いによって不具合、不整合が発生することも珍しい話ではありません。既存機能を再利用した開発を行う場合には　**①改版時のルールを定め、②履歴を作成することが必要**です。また、**③改版時には通知ルートが徹底され、④トレーサビリティが確実に行える**ようにすることも忘れてはなりません。

Check 3 承認の手順は決まっているか

承認手順とは、ドキュメントや成果物が確定したことを確認するための手段で、作業標準や安全基準から裏付けされた責任の明確化ともいえます。

ドキュメントの承認は、品質やコスト、安全面からの審査、検証を予め定めた資格者が実施します。この際に、**①品質やコスト、安全面を考えた承認手順が決められていること**が重要です。

Check 4 構成・内容は適切か

構成や内容が適切でないドキュメントは読者に誤解を与えてしまいます。

たとえば、設計書と書かれているドキュメントにコストやプロジェクトの進捗が記載されていても意味がありません。ドキュメントはタイトルと構成、内容が矛盾なく　**①目的や伝えるべき内容が明確であること**が大切です。

②ドキュメントは重要な情報であり、ドキュメントの読み手が、対象物のドキュメントを確認できるための唯一の手段であることを忘れてはなりません。

Check 5 **必要なドキュメントは揃っているか、不要なドキュメントはないか**

製品開発を行ううえで、**①必要なドキュメントが揃っていること**はいうまでもありませんが、逆に不要なドキュメントが存在すると開発を混乱に陥れます。検討途中の内容のドキュメントや、差し戻しや廃棄になったドキュメントなどは正式なドキュメントとは別に管理されるべきであり、**②ドキュメントはステータスが明確でないような状態で管理されるべきではありません。**

Check 6 **決められたフォーマットに従いきちんと記載しているか**

ドキュメントを作成していくうえでは　**①組織やプロジェクトで定義されたフォーマットが存在する**ということが開発効率を高めるだけでなく共通の理解を得る意味でも重要です。ドキュメントのフォーマットやテンプレートとして最低限、タイトル、作者、目次、承認者、履歴、作成日といった基本的な情報が付与されたものを組織やプロジェクトで用意しましょう。**②開発者は記述ルールに従い、記載する**ようになり、ドキュメントの質が安定します。

Check 7 **用語が定義されているか**

日本語で記述された設計書の場合、たとえば、各所に出てくる「本システム」「当該システム」「対象システム」等々といった用語がいったい何を指すのかは、書いた当人以外はわからないことがあります。こうした未定義の用語を用いると、思わぬ誤解がおこり、設計の誤りや品質の低下を誘発してしまいます。受け手により解釈が異なってしまうような曖昧な用語が使用されていないことが重要です。また、記述する側には当たり前の専門用語や略語も読む側の設計者・実装者側からすると理解しがたい用語であることもあり、さらに思わぬ誤解を招く場合もあります。コミュニケーションを円滑にするためには、**①受け手により曖昧になってしまう用語がなく、②専門用語、略語についても、きちんと定義されていること**が望ましいでしょう。

Check 8 **第三者が読むことを意識して書かれているか**

ドキュメントは読み手の「読みたくなる」といったモチベーションが促進されることも重要です。同じことを表現するとしても、読みやすく理解しやすくするために、**①日本語としての表現は適切である　②語尾がはっきりしている　③文章が受身になっていない　④レイアウトが読みやすい**といったポイントをチェックしてみると良いでしょう。

Check 9 **ドキュメントを作成する際のもととなる情報、ドキュメントに記載された情報を利用する次のドキュメントは明確か**

ドキュメントを作成する際には入力と出力を明確にする必要があります。まず、入力としては、必要となる情報、すなわち、**①入力となる情報（ドキュメント等）が明確に定義され、準備されていること**が前提となります。次に、出力としては、作成するドキュメントの　**②次のプロセス（ドキュメント）を明確にし、次のプロセスのためにどのような情報をどのように記載しなければいけないのかを明らかにする**必要があります。このように入力と出力を明確にしておくことでそのドキュメントに書くべき内容が明確になり、漏れや抜け、余計な情報が入ってしまうことなどの防止につながります。

Check 10 **誤字脱字は許容範囲内か、省略形はきちんと説明されているか、意味不明な表現はないか**

誤字脱字が多く、意味不明な表現が多いドキュメントは読み手に不信感を与え、せっかく作成されたドキュメントが有効に活用されません。レビューを行う際に、誤字脱字があまりに多いとその指摘に終始してしまい、肝心の内容に踏み込んで審査することが困難になってしまうようなことがしばしばあります。また、省略形の用語が説明されていないとか、最後まで何を意味するのか分からないような表現があるドキュメントでは読み手に肝心なことが伝わらず、ドキュメントとして意味がありません。かといって、チーム内などの内部資料では、あまり完璧を求めてしまうと時間ばかりかかってしまうこともあります。**①誤字脱字や省略形、表現について最低限の許容範囲を定め守る**ことも必要です。

Check 11 **作成者以外の誰かが目を通しているか**

レビューはソフトウェア開発過程の品質作り込みのための重要な作業として位置づけることができます。開発過程で作成したさまざまな成果物を対象に、開発工程の節目でレビューを実施することにより、成果物の精度を高め、品質を向上させていきます。レビューのやり方には、自己レビューのように成果物の作成者本人により、チェックリストなどを使って確認する方法もありますが、インスペクションやウォークスルーなど、複数の関係者が集合して行う方法もあります。成果物の作成を担当した技術者だけでなく、**①開発に関係するメンバやステークホルダなど、多視点でのレビューにより確認すること**により、課題や問題など多くの気付きを得ることができるでしょう。**②レビューで確認された課題や問題点はその検討担当者と期日を決めてレビュー記録として残し、確実に反映され**

たことを確認する仕組みも大切です。

Check 12 **ドキュメントとして盛り込むべき必要な内容が含まれ、システムの規模に応じた適切なボリュームとなっているか**

ドキュメントは量ではなく、質（内容）が大切であることは言うまでもありません。つまり、そのドキュメントに記述されるべき内容が正しく書かれていることが重要です。また、記述されるべき項目としてはすべてを満たしていたとしても、量的に少な過ぎる場合には、必要な情報が記述されていない可能性が高いと言えるでしょう。本ガイドのドキュメントボリューム品質評価指標でも示しましたように、**①システムの規模に応じた適正な量のドキュメントが作成されている**か否かも、ひとつのチェックポイントになります。

また、量的なことが充たされていても、**②特定の情報に偏りがなく、必要な内容が漏れや抜けなく含まれていること**も確認する必要があります。

4.3 レビュー

レビューの目的と役割

人間はどんなに正しく行動したと思っていたとしても、一定の確率で間違いを起こしてしまうものと言われています。製品開発を行っていくうえではその間違いを、製品を次の工程に移行する前に複数の目で未然にみつけ、防止して行くことが大切です。ソフトウェア開発の世界では間違いや欠陥・不具合の検出のために、テストとともに、レビューが用いられます。レビューの実施は、ソフトウェアの品質の改善だけでなく、テストの実施だけでは見えない、内部構造の確認や、プロジェクトの規約や基準の順守の確認のほか、成果物を他人に説明する、もしくは他人からのフィードバックを得ることによってお互いから学ぶという開発者の教育ツールとなる側面もあります。

レビューでは、テストでは検出困難な不具合や欠陥などをテストよりも効率的に検出できる場合が多くあります。しかしながら、レビューの効果は、その実施方法により大きく左右されます。効果あるレビューとするには、

- 適切なタイミングで
- 適切な参加者のもと
- 適切な手法で

実施する必要があります。

レビューの方式や進め方のテクニック（例）

方式	内容
インスペクション	モデレータと呼ばれる主催者（成果物の作成者以外）が参加者の選定や、個々の役割の規定などを行う、体系的で厳格なレビュー方式。エラーについては、修正や確認の実施まで行う。
ウォークスルー	成果物の作成者がレビュー参加者へ説明を実施し、コメントを求める方式。
ペアレビュー	作成者とレビュー者の2名でペアを組み、成果物の確認を実施する方式。
パスアラウンド	電子メールや成果物のコピーなどを、複数のレビュー者に配布し、確認を求める方式。回覧方式がとられることもある。

レビューに関するヒント（会議を主体とするレビューの場合）

NO.	チェック項目	チェック
1	担当者はよどみなく説明できたか	
2	何をレビューするか、進め方についてわかっているか	
3	レビュー者の参加者意識はあるか	
4	レビュー対象物は完成しているか	
5	レビュー記録票を用意し、きちんとレビュー記録をとっているか	
6	結論が出たか、結果を必要なメンバに通知する仕組みになっているか	
7	指摘された課題が明確か	
8	力関係がレビューに出ていないか	
9	時間を十分にかけ、適切な手法でレビューしたか	
10	指摘が有効に行われているか	
11	レビュー後のフォローアップが出来ているか	
12	レビューの実施タイミングはプロジェクトの全体の進行状況を踏まえて適切な要員に予め声をかけて召集できているか	

各項目に関する解説

Check 1 担当者はよどみなく説明できたか

ソフトウェアのレビューは被レビュー者が、成果物を説明するところからスタートします。その意味では被レビュー者は自身の作成した成果物の内容を十分に把握・整理して簡潔に説明できるように準備しておかなければなりません。成果物の説明に関しては曖昧性などをできる限り排除し、**①一意に解釈できるよう分かりやすく説明すること**はもちろん説明は**②一貫して矛盾がないように説明すること**が重要です。こうした説明ができていないと、レビューの参加者間でさまざまな誤解が生まれ、レビューで本来確認したり議論すべきことの焦点がぼけてしまう可能性があります。

Check 2 何をレビューするか、進め方についてわかっているか

効率良く、効果的なレビューを実施する為にはレビュー参加者に対して　**①レビュー対象や観点を明確にしておくこと**がまず重要です。たとえば、ソースコードレビューは対象とする機能、構成ファイルなど、レビュー対象を明確にしたうえで、信頼性の観点からレビューする、などの観点を明確に伝えておくというこ

とです。次に、**②どのような手法でレビューを実施、進めるか共有すること**も重要です。それにより、被レビュー者は事前にいつまでに何を準備しておけばよいか、レビュー者は自分の役割は何か、事前準備をどのくらい行っておけばよいのか、を確認することができます。

Check 3 レビュー者の参加者意識はあるか

レビューは　**①参加者全員が当事者意識を持って臨むこと**が重要です。参加した人のモチベーションはレビュー効果に大きく影響します。意識が高い人が多いほど、参加者の脳が活性化され、どんどん指摘が出てくるようになります。逆に、たとえば、寝ている人がいたり、ほかの事を行っている参加者がいると、他の参加者のモチベーションを下げ低調なレビューになってしまいます。

Check 4 レビュー対象物は完成しているか

レビューの手法にもよりますが、レビュー者は、**①レビュー対象物を事前にチェックしていること**が大切です。レビュー対象物に白紙やTBD（To Be Determined）[1]が多いと、レビュー者のモチベーションは下がります。核心である部分の検討が不十分になってしまう場合もあります。程度によっては、レビューそのものが意味をなさないケースもあります。このような場合には、レビューの開催が決定していたとしても、**②リーダは、準備不足と判断し、レビューを開催しない**（改めて設定しなおす）という決断をする勇気も時として求められます。

Check 5 レビュー記録票を用意し、きちんとレビュー記録をとっているか

レビューを実施して、指摘事項が出たとしても、「言い放ち」だけでは意味がありません。**①レビュー者の発言は「指摘」なのか、「要望」あるいは「質問」や「コメント」なのか整理し記録していくこと**がその後の対応を行うために重要です。**②組織やプロジェクトで予めレビュー記録票を定め、容易に整理された形で記述できる仕組みを用意しておく**と良いでしょう。また、記録漏れが起きないよう、**③レビュー開始時にレビュー記録者を決めること**も忘れてはなりません

Check 6 結論が出たか、結果を必要なメンバに通知する仕組みになっているか

レビューを進めるにあたっては、常に結論を出すことを念頭に置くべきです。**①指摘項目に対して、対応・期限・担当者を明確にすることが重要です。**また、気をつけなければいけないのは、レビュー者の発言が要望的発言やコメント的な曖昧な発言や指摘であると、担当者はどのように対応したらよいかどうか、迷って

[1] **TBD（To Be Determined）**：「現在未決定だが、将来決定する」という意味

しまうことがあるので、**②対応の仕方**（個々の指摘事項に対する具体的な修正方法ではなく、誰がどのように検討するか）**を議論し結論付けること**が必要です。そして、**③レビューの結果については必要なメンバに通知**されなければなりません。

Check 7 指摘された課題が明確か

レビューにおいては、活発な指摘が行われることは重要ですが、たとえば、「何となく変な気がする」といったように、抽象的な指摘や、問題の本質が説明されない指摘、また指摘の内容が広範囲にわたり絞りきれないような指摘では、担当者が理解できません。抽象的な指摘などはなるべく具体化、絞込みなどを行うことにより、**①担当者が問題や課題の本質を理解でき、②対応策が検討できる指摘を行う**ように心掛けましょう。

Check 8 力関係がレビューに出ていないか

レビューの場に限りませんが、ユーザと開発者、上司と部下のように力関係により仕様や修正方針などが決まってしまうことがあります。しかし、レビューの場では参加者が平等な立場で純粋に技術面からの議論をしなければなりません。しばしば声の大きい人の主張に（無理やり）流されてしまうことなども見かけますが、参加者は真摯な態度で議論に臨むよう心掛けましょう。**①安易な力関係や妥協だけで方針を決めてしまってはいけません。**

Check 9 時間を十分にかけ、適切な手法でレビューしたか

大規模なシステムではすべてのプロセスの成果物すべてのレビューを実施することは時間的に不可能なケースがあります。

このため、設計や仕様の新規作成部分など一部のみをレビュー対象とし、そのほかの再利用部分などはレビューを割愛することがあります。しかし、割愛する際の検討が不十分なために重要な箇所がレビュー対象として除外されてしまうことがあります。このようなことを防ぐために、レビューにおいては、**①レビュー対象の優先順位を定め、②適切かつ十分なレビュー時間を確保すること**が必要です。ただし、レビュー時間はただ長くとれば良いという訳ではありません。効率性を考えれば　**③必要以上に時間をかけすぎないこと**も重要です。また、時間切れで計画上設定していたレビューを省略してしまったり、重要度や内容の判断をせずに安易に回覧レビューに切り替えてしまうことは避けなければなりません。

Check 10 **指摘が有効に行われているか**

レビュー対象のドキュメントが未熟な場合に、仕様や設計の重要な問題の指摘などが置きざりにされて、ドキュメントの形式や誤字や脱字、項番の振りかたや、インデントなどの形式的なことに終始してしまうことがあります。もちろん、これらドキュメントの形式上の問題も軽視してはなりません（ドキュメントの形式について重点的にレビューするのであれば、回覧レビューにより朱書きチェックを行うなどが有効です）が、レビューでは　**①仕様や設計の内容面についての議論や指摘を中心にすること**が重要です。レビュー時にはレビュー者は指摘事項の有効性に常に気をつけ、被レビュー者は指摘されたことをしっかり理解し、修正などを確実に行えるように心掛けましょう。

Check 11 **レビュー後のフォローアップが出来ているか**

レビューは対策や担当者、期限が決定されても、その後の確認や実行するためのフォローが実施されなければ意味がありません。レビュー結果の反省を行い、指摘事項が非常に多い、指摘事項の反映結果が重要であるなどの判断があれば再レビューなどを実施しましょう。また、実際にレビュー後の対策に対してはレビュー記録票などに　**①対策・期限への確認結果が記録されること、②対応に対してリーダや他のメンバによる確認がなされていること**が重要です。

Check 12 **レビューの実施タイミングはプロジェクトの全体の進行状況を踏まえて適切な要員に予め声をかけて召集できているか**

①レビューは予め計画を立てて、必要なタイミングに実施します。レビュータイミングをマイルストーンとし、**②計画通りに実施できるようスケジュール管理していくこと**も、円滑にプロジェクトを進めるうえで大切です。また、レビューでは開発のメンバだけでなく、**③テーマによっては必要なメンバ**（たとえば、安全性の問題であれば法規の専門家といったように）**を召集すること**でレビューの質が高まります。

4.4 テスト

テストの目的と役割

ソフトウェア開発でのテストとは広義にはソフトウェアに潜む不具合をできる限り多く発見することを目標としてソフトウェアを実行し、正しく動作するかどうか確認する作業のことです。

ソフトウェア開発の開発局面においては、さまざまな目的でテストが実施されます。たとえば、回帰テストはプログラムに追加や修正を行った際に、その変更によって予想外の影響が表れていないかどうか確認することを目的としたテストです。ユーザビリティテストはユーザの使い勝手や感覚的なことに対して問題がないか検証・評価を行うことを目的としたテストです。

効果的なテストを実施するためには、何のためにそのテストをするのか、その目的と役割を明らかにしたうえでテスト計画を立案することが必要です。

テストの種類と目的／役割の例

＜フェーズによる観点＞

テストの種類	目的／役割
単体テスト	プログラムの関数やクラスなどソフトウェアを構成する小さな単位での不具合検出を目的とするテスト
機能テスト	実装した機能が、意図通りに動作するか検査するテスト
結合テスト	ソフトウェアの部品を部分的に結合した実行単位が、意図通り動作するか、また、例外処理を受け入れるかについてインタフェースを確認検査するテスト
システムテスト	ハードウェアへの組込みや通信の実施などを行って、製品としての動作確認を行うテスト
退行テスト・回帰テスト	プログラムを変更した際に、その変更によって予想外の影響が表れていないかどうか確認するテスト
マニュアルテスト	製品がマニュアル（説明書）通りに動作できるか確認するテスト

＜目的による観点＞

テストの種類	目的／役割
性能テスト	処理速度や情報処理量の期待値が確保できているか検査するテスト
負荷テスト	I/OやCPU、通信などに負荷を掛けても正しく動作するかテスト
使いやすさテスト	使い手（ユーザ）の視点から使い勝手を確認するテスト
耐久テスト	製品の長時間稼動を行った際でも正しく動作するか確認するテスト
回復テスト	障害発生後の回復性について確認するテスト

＜技法による観点＞

テストの種類	目的／役割
構造テスト	プログラムの内部構造に着目し検査し、論理矛盾が無いか検査するテスト
入出力依存テスト	プログラムの内部構造に着目せず、入出力のみから結果を検査するテスト
パス解析	カバレッジ（記述されたコードの網羅率）を取得しながらプログラムの挙動について解析し検証を行うテスト

上記は、ソフトウェアのテストの種類と目的と役割の一例です。各企業や製品ドメインにより、テストの呼び名や位置付けが異なる事がありますが、テストの担当者はその目的を十分に理解し、目的に沿ったテスト方法に従い、実施することが必要です。

テストに関するチェック項目

NO.	チェック項目	チェック
1	楽しくテストをしているか	
2	テスト担当者が独立しているか	
3	テスト対象が明確になっており、各テストのフェーズで適切なテスト手法がとられているか	
4	不具合の記録が行われ、修正した不具合の確認を行っているか	
5	システムで提供する機能の確認をするのに十分なテスト項目が実施されているか	
6	非機能要件について（性能など）のテストは実施されているか	
7	テスト環境は実際のシステム動作環境と同等もしくはそれに等しいものか	
8	テスト計画が明確になっており、適切な時間をとってテストをしているか	
9	明確な判断基準のもとにテスト結果を確認しているか	
10	検出した不具合に対する原因分析を行っているか	
11	テスト完了基準は明確か	
12	テストですべての品質をおさえようとしていないか、テストに過度な期待を抱いていないか	

各項目に関する解説

Check 1 楽しくテストをしているか

ソフトウェア技術者にとって、プログラミングの楽しさの大部分は、「プログラムが、上手く動くように考える作業」でしょう。ではテストはどうでしょうか。テストはエラーをみつけるつもりでプログラムを実行する作業と、製品やソフトウェアの品質が十分に達成できているかどうかを確認する作業という2つの面を持っています。いずれも、テストの戦略を持ち、テスト設計から実行まで計画を立てて進めていくべきものですが、まだまだ場当たり的にテストが行われている現場も少なくありません。特に後者に重きを置いているテスト作業の場合、往々にして機能確認のための単調な作業が強調され、時として義務や責任のみが重視され、〝テストは楽しくない〟、と思ってしまいがちです。「楽しくないテスト」は作業効率や作業の質という面から必ずしも好ましいとは言えません。戦略からきちんとブレークダウンしていけば開発と同じようにテストのモチベーションも保てるようになるでしょう。特に「楽しいテスト」にするためには、**①テスト担当者が高いモチベーションで作業できるよう、テストの効果が見えるようにする**、優れたテストをした担当者に高い評価を与えるなど、**②テストが楽しく行える仕組み作り**が必要です。

Check 2 テスト担当者が独立しているか

一般に、テスト担当者が専任でおり、開発者と分けている組織と、開発者がテストまでの一連の作業を行う組織があります。前者は開発者とテスト担当者が分離されているので、製品検査という点から客観的な評価を実施することができます。また、標準や規則に遵守しているかといった視点からもチェックされやすいという点からも優れています。後者は仕様作成から設計、コーディング、テストまで、一連の理解が出来ていることから、時間的な融通がきき、また生産性が高いとも言われています。本ガイドでは、どちらが良いといった教条主義にはこだわりませんが、大切なのは　**①テスト実施者がテストの目的／役割を理解しテストを実施し、②実装とテストが混同しないようにする**ということです。テスト担当者が独立していない組織では、テストの時間を計画的に確保すること、テストを行う際には開発者とは立場や気分を切り替えてしっかりテストすることが望まれます。

Check 3 **テスト対象が明確になっており、各テストフェーズで適切なテスト手法がとられているか**

テストはこの節の冒頭の表に示したようにフェーズ、観点、手法などに関してさまざまなものが存在します。実際のテスト作業はこれらを組み合わせた形で実施することになります。このため実際の開発現場におけるテストは多くのバリエーションが存在します。こうした多様なテストのバリエーションの中から、対象とするシステムやソフトウェアに関して、より有効なテストとするためには、まず、テストフェーズごとに　**①テスト対象・範囲を明確に定義し、テスト設計やテスト結果判定を行うために必要な資料（システム仕様書、インタフェース仕様書など）を集めます**。そのうえでテストの目的を明確にし、適切な手法を選択しテスト設計を行う必要があります。つまり、**②それぞれのフェーズにおいて、それぞれ目的に応じた種類のテストが実施されること**が大切です。また、それぞれの局面のテストでは、**③テスト対象・範囲を明確に定義する**必要があります。

Check 4 **不具合の記録が行われ、修正した不具合の確認を行っているか**

テストを実施してもその結果の記録がされていなければ意味がないのは言うまでもありません。また、検出された不具合の現象については、不具合検出後の修正や再テストなどに備えて、テストの再現手順も含め　**①不具合の記録を詳細に残すこと**も必要です。その不具合についての判断、修正の記録なども残す必要があります。多くのプロジェクトでは既存のプログラムソースからの流用を行う場合や派生バージョンを持つ場合が少なくありません。このため、その製品（バージョン）固有の情報だけでなく、関係するシステムでの影響範囲を確認し、**②他バージョンへ影響がある場合、方針・対策が記録・管理される**必要があります。

Check 5 **システムで提供する機能の確認をするのに十分なテスト項目が実施されているか**

利用者の要求を満足するためにソフトウェアが実現しなければならない機能についての要件を機能要件と呼びます。昨今のシステム機能の拡大に伴いソフトウェアとしての機能要件は増加の一途をたどっています。ソフトウェアとして実現する機能は、システムの動作コンテキスト（動作環境、動作条件など）によってさまざまであり、場合によっては複数の機能の重複や並行動作まで考えなければいけないといったケースも出てきます。この点から、システムやソフトウェアで実現する機能に関して、その　**①動作バリエーションなども考慮した十分なテスト項目を用意しテストすること**が求められます。各テストでは、目的に応

じたテスト項目が作成・実施されることが重要であり、きちんと確認作業も行うことが必要です。ソフトウェア開発の各成果物（ドキュメント類、ソースコードなど）と同じく、**②テスト項目は第三者に漏れ・妥当性についてレビューされる**のが理想です。また、特にユーザインタフェースの多いシステムなどでは、**③ユーザがシステムを使う局面を考慮して、適切な操作、シナリオに基づき、テスト項目が網羅できるようにする**など、開発対象に応じたテスト項目を作成実施することが必要です。

Check 6　非機能要件について（性能など）のテストは実施されているか

前項の機能要件に対して、機能要件以外のすべての要素（性能、容量、情報セキュリティ、拡張性など）は非機能要件と呼びます。非機能要件は、多くの組込みシステムでは機能要件と同様にきわめて重要であり、時として、そのボリュームや重要性は機能要件をも上回る場合も少なくありません。開発者は機能要件のみならず、この非機能要件についても考慮しテストをする必要があります。非機能要件のテストとは、たとえば、**①性能に関する検討・テスト、②容量に関する検討・テスト、③情報セキュリティに関する検討・テスト、④拡張性に関する検討・テスト**などが含まれます。これらは対象とするシステムの特質やシステムプロファイルなどによっても大きく変わってきます。仕様や設計段階で検討された非機能要件に関する事項を確認し、必要な非機能側面からのテストを実施することが求められます。

Check 7　テスト環境は実際のシステム動作環境と同等もしくはそれに等しいものか

動作環境、接続環境としての対象ハードウェアが多数存在するシステム、たとえば携帯電話向けアプリケーションのテストなどを実施する際には、膨大な種類の携帯電話で何を対象とするか選定が困難な場合があります。また、組込みソフトウェア開発では対象システムのハードウェア機器の開発とソフトウェアのが開発が並行して行われるため、実際のハードウェアが存在しないうちにソフトウェアのテストを実施しなければならない場合があるなど、悩ましい場面が多々あります。このような場面において重要なのは、動作環境として最適なものを如何に判断するか、仮動作環境としてどのような環境を準備構築したらよいかなど、**①各局面（フェーズ）において適切なテスト環境を用意すること**が求められます。

Check 8　テスト計画が明確になっており、適切な時間をとってテストをしているか

ソフトウェアの開発では、プロジェクト計画とともにテスト計画も立案すること

が有効ですが、テスト計画を独立させて作るのが難しい場合、プロジェクト計画に含めてしまうのも一つの方策です。いずれにせよ、**①プロジェクト計画時には実施するテストの種類、範囲、時期、期間を定めておく**ようにしましょう。また、テストは開発スケジュールに追われ、往々にして必要な時間がなくなってしまいがちです。プロジェクト計画時にテスト計画を明確にすることにより、予め　**②適切な時間を確保しテストを実施すること**が重要です。

Check 9　明確な判断基準のもとにテスト結果を確認しているか

テスト作業には通常、準備、実施、結果確認という３つの作業が含まれます。このうち、テストの準備作業で作成したテスト項目や実施したテスト項目に関しては、テスト対象を明確に意識したうえで、その内容についてきちんと確認作業も行うことが必要です。すなわち、ソフトウェア開発の各成果物（ドキュメント類、ソースコードなど）と同じく、**①テスト項目は漏れ・妥当性については第三者がレビューする**のが理想です。

テストを行うにはテスト対象を明確に意識して、テスト項目、テスト方法を確認してから行うことが重要です。同様に、テスト結果を確認する際は、テスト手順、テスト項目などに記載されている期待されるテスト結果と実際のテスト結果を比較し、判断を行う必要があります。そのためには、テスト手順、テスト項目などを作成する際に、テスト結果への明確な判断基準を記述しておくことが必要です。また、テストを行った際は　**②テスト結果はしっかりと記録し**、特に　**③再現性の低い現象などは、その時の状況など、できるだけ詳細な情報を記録し**、開発者をはじめとする関係者に正確に伝えられるようにしましょう。

Check 10　検出した不具合に対する原因分析を行っているか

テストで　**①検出された不具合については原因分析を行うこと**が重要です。原因分析を行うことで、類似の不具合をさらに検出したり、派生開発製品の不具合を未然に防止したりすることが可能です。さらに、真の原因を次の製品開発にフィードバックすることで次の製品の品質向上に大きく役立てることができるようになります。特に、重大な不具合や影響範囲の広い不具合に対しては、背後にある要因を掘り下げ、水平展開を行うことで　**②組織的に再発防止を検討する**ことが重要です。

Check 11　テスト完了基準は明確か

テストの難しさのひとつに、どこまでテストを行うかがあります。完璧なテスト計画を作るのは難しく、テストを実施する時間には制約があります。限られた時

間で効率よくテストを実施することが必要です。そもそもソフトウェアやシステムのテストは対象を動作させて確認するという点において、システムのライフサイクル（寿命）の一時点を切り出して確認するスナップショットとしての確認であることを理解しておく必要があります。これはすなわち、ソフトウェアテストにはおのずと限界があることを意味しています。それには必要とするシステムに求められる品質・信頼性を十分に把握し定めたうえで、テストでどこまでを確認するかを検討して、テスト計画段階で　**①テストの完了基準を決め、テスト計画をきちんと立てることで　②同じテストを無駄に繰り返したりしないようにする**ようにしましょう。

Check 12　テストですべての品質をおさえようとしていないか、テストに過度な期待を抱いていないか

本節冒頭でテストの種類について紹介しましたが、どのテストも万能ではありません。また、テストはソフトウェア出荷の最後の砦ではありますが、もともと品質が良くないものをいくらテストしたところで品質の向上には限界があります。複数のテストを組み合わせ、テスト計画を作成することはもちろん必要ですが、ソフトウェアの品質向上を実現するためには、テストやレビュー、あるいは仕様検証などさまざまな手法の長所を活かした総合的な戦略をたて、実施していくことが求められます。もちろん、これらの前提として開発・テストを含む要員のスキル調達・向上やプロセス改善などといった、ソフトウェア品質の向上の前提となる、より本質的な活動もあり、それらも視野に入れつつ　**①総合的に組み合わせた品質マネジメント計画を作成・運用すること**が必要です。

4.5 指標を用いた品質作り込み活動

品質指標を組織に定着させる

本ガイドでは、開発する製品を理解し（システムプロファイリング：SCP）、次にプロジェクトの状況を理解し（プロジェクトプロファイリング：PCP）、それらの理解をベースに品質指標を定義設定して品質作り込みを行うことについて述べてきました。品質指標による品質作り込み活動は品質指標を設定・計測するだけでなく、得られた結果をプロジェクトに反映することではじめて効果を発揮します。より効果的に品質作り込み活動を行っていくためには、本ガイドの内容を組織として理解し、概念を組織に定着させることも重要です。また、品質指標はあくまでも客観的、間接的な指標です。さまざまな手法を取捨選択し、作業そのものの質を向上させる努力こそが必要です。

本節では本ガイドのまとめとして、有効な品質指標を設定し、組織に活かすためのヒントを整理してみました。

指標を用いた品質作り込み活動を進めるに当たってのヒント

NO.	チェック項目	チェック
1	定量データの評価のみに目を奪われていないか	
2	システムプロファイリングをきちんと行えているか	
3	プロジェクトプロファイリングをきちんと行えているか	
4	品質指標は有効性を考慮して選択し、かつ無理のない測定方法を選択しているか	
5	過不足無く品質指標の目標値を設定できたか	
6	すでにプロジェクト内で有効な確立した手法がある場合にはそれを有効活用する、読み替えるルールにしているか	
7	測定結果をプロジェクトに反映し、次の開発にフィードバックする仕組みを設けたか	
8	本ガイドを適用することの意義と方法がメンバに理解されているか	
9	メンバが勉強する時間、機会を用意しているか	
10	自分たちのスキルはきちんと把握できているか	
11	システムプロファイリング～品質目標値の設定まで、レビューを行ったか	
12	品質定量化の活動を通して得られた知見を広く社会に還元することを考えているか	

各項目に関する解説

Check 1 定量データの評価のみに目を奪われていないか

本ガイドを通じて、システムプロファイリング→プロジェクトプロファイリング→品質指標の設定というプロセスを経て、その参考値を自分たちのプロジェクトにマッチした値にするという作業を紹介してきました。人は数値が示されると、どうしても数値そのものに目を奪われがちです。しかし、本ガイドで提案したいのは、その数値の表現する意図を理解し、品質の作り込みに役立てていただきたいということです。つまり、数値そのものに一喜一憂せず、その数値の意味するものの本質をとらえて対応することが大事ということになります。

まずは　**①参考値をそのまま使ってはいけない**ということが挙げられます。本ガイドで示した参考値はあくまで目安であり、絶対的な目標値ではないということです。とかく管理する側は数値のみを見てしまいがちですが、本質をとらえて使うようにしてください。

次に　**②数値を達成することのみを目標としてはいけない**ということがあります。本ガイドでは客観的に測ることができるものを品質指標として使用しています。これらは作業についての量の把握はできますが、作業内容の質そのものは把握することができません。たとえば、同じテストを何回も行うことでテスト実施工数の値は上がり、テスト作業の充当率や実施率の値を上げることができます。しかし、同じテストをただ繰り返すだけでは対象ソフトウェアの品質はなんら変わることはありません。また、システム仕様書のレビューをゆっくりやる、多人数でやるなどすれば仕様書レビュー工数の値は上がり、仕様レビューの実施率、充当率を上げることができます。このように数値の操作は簡単ですが、それを行ってしまうと本当の目的であった品質コントロールはできなくなってしまい、せっかく決めた目標値の意味がなくなってしまいます。本来何を行うべきなのかをきちんと認識して品質コントロールを行うようにしましょう。

Check 2 システムプロファイリングをきちんと行えているか

システムプロファイルを定義するにあたって一番重視しているのは、提供された物を　**①使う側の視点に立って品質要求をとらえる**ということです。皆さんの部門では何を基準に品質目標を決めているでしょうか。「なんとなくこれくらい」で決めていないでしょうか。作る側の理屈である、「工数がここまでしかない」とか「リリース時期が目前だ」とか「バグは（とにかく）許さない」とかの意識が働いて決めていないでしょうか。それは経験値としてしばしば当っていることも多

いのですが一方、品質目標が妥当でないために結果として必要な品質が確保されず、出荷後の不具合が頻発してしまうとか、それとは逆に必要以上の品質保証作業を行っているために開発者が必要以上に疲れ切ってしまうとか、なかなかリリースができないといった事態が生じてしまうことがあります。「このくらいでよいだろう」をシステムプロファイリング（SCP）はきちんと裏づけするための手法です。

システムプロファイリングをきちんと行い、自分たちの作るべき製品を客観的にとらえることで、冷静に品質目標を考えることができるようになります。また、部分開発しか行っていない部門でも、自分たちの作っているソフトウェアがシステム全体のどこにどのような影響を与える、ひいては品質にどのような影響がある、ということを考えることができるようになります。**②大きな視点で品質を考えられるようになること**で、さらなる品質の向上が望めるようになります。

また、システムプロファイリングを行うにあたって、**③発注元、ハードウェアメンバなどのステークホルダと議論**をされたでしょうか。議論をしていく中でシステム関係者間共通の認識を持つことができると考えています。システムプロファイリングはソフトウェアのみならず、システムとしての品質を確認しますので、**④ハードウェアの品質目標設定もあわせて行うこと**ができるでしょう。是非、システムプロファイリングを行う過程で製品が世の中から求められる品質を考えてみてください。

Column

システム障害時のメーカ側の損失額

プロジェクトプロファイリングファクターの10番目の「システム障害時のメーカ側の損失額」は、SECの中でもシステムプロファイルの中に入るのではないかとの議論がなんどかありましたし、パブリックコメントでもリコール費用を経済的損失に加えてもよいのではないかというようなご意見をいただきました。しかし、使う側の視点で考えた場合にはこの項目はシステムプロファイリングにはあたらないという結論に達しました。システムプロファイリングでは自分たちの都合でなく、まず、使う側が求める品質を考えるという作業を行っていただければと思います。

Check 3 プロジェクトプロファイリングをきちんと行えているか

プロジェクトプロファイリングでは品質を作り込む作業を行っていく中で障害となる要因、また逆に追い風となる要因を考えます。言い換えれば、**①ウィークポイントと安心材料を品質作り込みという視点でとらえ、表す**ということになります。たとえば、2章でも述べましたように、新人ばかりのプロジェクトであれば、きちんと作れたかということを多くの目で確認するためのレビューを強化することは品質を確保するうえで必要です。また、逆に、ソフトウェア規模がきわめて小さい場合には、レビューやテストにかける工数は相対的な規模よりさらに少なくて済むことも考えられます。つまり、優秀なマネージャであれば暗黙のうちに当然行っているようなことを、プロジェクトプロファイリングは数値化します。プロジェクトの状態を見ながら按分していくことを数値として予め表しておくことで周囲の了解も得やすくなるでしょう。

プロジェクトプロファイリングを効果的に行うには、**②自分たちの状況をきちんととらえているか**、ということが重要になります。プロジェクトプロファイリングを行うために開発者からヒアリングを行い、また、議論を重ねることで、自らのプロジェクトの持つ問題を再認識することができるでしょう。

また、補正係数を算出するにあたってのプロジェクトプロファイリングファクターを何にするかということも検討するとよりプロジェクトに合致した値を導くことができるようになります。本ガイドで示した補正係数のファクターは参考情報です。プロジェクトプロファイリングファクターはそのすべてが全作業に影響するとは限りません。自分たちの作業に合わせて影響の大きいファクターを検討するのが良いでしょう。

Check 4 品質指標は有効性を考慮して選択し、かつ無理のない測定方法を選択しているか

開発プロジェクトを進めていく最中に、データをきちんととり、分析することは、慣れない組織ではなかなか難しいものです。基礎指標のデータを得るために開発側に大きな負担をかけてしまうようではプロジェクトそのものが成り立たなくなってしまいます。今までデータをとっていないプロジェクトでは、まず、品質指標の意味するところを理解したうえで、自分たちに最低限必要な品質指標を定め、そこに必要な基礎指標の計測から始めましょう。**①自分たちができることから始め、少しずつ向上させていくこと**が息切れしない、長期的なプロジェクトの成功につながります。

また、取得したデータを有効に使用するためには、**②データのとり方を予め決め**

ておく（方法、タイミング、報告方法など）ことが大切です。報告にぶれをなくすためには　**③帳票を用意すること**などをしても良いでしょう。また、ツールを使用しての自動収集など、楽にデータをとる手段を設けることも有効です。

Check 5　**過不足無く品質指標の目標値を設定できたか**

品質指標を定義することで取得すべき基礎指標、評価する品質評価指標は決まります。次は品質目標値の設定です。プロジェクトの状況を何も見ずに最初から高い目標を設定しても後が続かなくなってしまいます。**①プロジェクトプロファイリングの結果はきちんと反映できた**でしょうか？自分たちで計測できる範囲や正確さの度合いなども考えながら　**②過不足なく品質指標を設定できている**かを振り返ってみてください。参考値から得た目標値を目指すことは必要ですが、自分たちの経験、過去のデータを元に振れ幅の範囲を決めましょう。ある程度高い目標値を決めたならば、それを実行するための手段を設けることを忘れないようにしてください。

Check 6　**すでにプロジェクト内で有効な確立した手法がある場合にはそれを有効活用する、読み替えるルールにしているか**

本ガイドでは品質コントロールを行うための品質指標の定義とその評価のための基礎指標の計測について述べていますが、すでに同様の目的を持った手法が組織内に適用されている場合には、あえてルールを変える必要はありません。**①すでに取得しているデータがあれば、本ガイドと比較して、有効な方法を選択し、読み替えて**ください。

たとえば、プログラムの規模を表す指標として本ガイドではソースコードの物理行数を採用しています。規模をあらわす指標としてFPを採用している組織では、規模でノーマライズしている指標（作業実施率、ドキュメントボリュームなど）の目標値を組織の実績と照らし合わせて設定してもよいですし、FPからコードに換算して本ガイドの目標値を適用しても良いでしょう。FPからのコードの換算率は組織の経験から算出してもよいですし、複数の文献に参考値が掲載されています（Capers Jonesなど）。また、ソースコードの複雑性を簡易的に測る指標として、本ガイドではソースコードの制御文記述率を設定していますが、すでにツールを用いて複雑度（サイクロマティックなど）を測定し評価しているような場合は新たに制御文の計測をすることなどは不要でしょう。

また、データは取得しているものの、うまく評価に活かせていないというような場合は、本ガイドを参考に見直していただければと思います。

Check 7 **測定結果をプロジェクトに反映し、次の開発にフィードバックする仕組みを設けたか**

品質指標で得られる値は一回の開発の評価のためだけに使うのではなく、次の開発に活かすことでプロダクトの継続的な品質向上、プロジェクトメンバのスキルアップにつなげていくことができます。開発終了後に品質指標の達成度合いを測り、未達の部分、過剰な部分、どちらも目標値と異なってしまった場合の原因、あるいは目標値の妥当性などを分析し、**①その結果を次の開発にフィードバック**しましょう。プロジェクトプロファイリングファクターの評価、品質指標の参考値など、最初はどう扱ったらよいか、わかりにくいかもしれません。その場合は、まずはそのまま運用してみましょう。開発終了後に振り返り、目標を達成できなかった、あるいは達成するのが困難であった品質指標があれば、**②何が原因だったのか、開発終了後に分析を行ってその結果を次の開発時に反映する**と良いでしょう。

システムプロファイリングの分析結果は2.6節に示しましたように、システム障害影響評価スケールを使用して、現実のプロジェクトと常に見比べ、適切な値になるよう（妥協ではないことに注意）、現場にフィードバックしていくことができます。

データを蓄積していくことはプロジェクトの経験の強化になります。それは同時に、組織の体力アップを意味します。ぜひ、数値の上下に一喜一憂することなく、長期的な視点でデータをプロジェクトにフィードバックすることを心掛けるようにしてください。

Check 8 **本ガイドを適用することの意義と方法がメンバに理解されているか**

ソフトウェアの品質は直接は目に見えません。1.2節でりんごの例を示しましたように、本ガイドではプロジェクト内で品質指標という共通のものさしを使うことで少しでも品質が見えるようにすることを狙っています。

つまり、**①ものさしを使うことで作っているものの品質がどの程度であるかを知る、②どのような物を作るべきかを認識すること**ができるようになります。ものさしをきちんと使うためには、そのものさしがどういうものであるかを知ることが必要です。また、ものさしをどのように使うか（物をどのように測るか）、何に使われるかを理解することでさらに有効に使うことができます。

品質指標を理解し、納得して使うことで　**③押し付けではないデータの蓄積**が可能となります。それによって、より大きな効果を得ることができるでしょう。

Column

プロセスと作法

ソフトウェアの開発では、適切なプロセス、適切な作法を採用することで、より高い品質を実現することができます。品質指標の参考値は各プロセスで「適切な作業」を行った場合の目安です。ここで「適切な作業」というのは決められたプロセスに従い、決められた手順できちんとものを作っているということです。プロセスを定義し進めることとそれに伴うドキュメント類の書き方についてはSECで提案している「開発プロセスガイド」ESPRが参考になるでしょう。また、コーディング作法についてはSECで提案している「コーディング作法ガイド」ESCRが参考になるでしょう。その他の技術的な技法を適用することについては、本ガイドの4章の各節および付録Aの参考文献を参考になさってください。

Check 9 メンバが勉強する時間、機会を用意しているか

品質指標により品質コントロールを行うことは品質向上のためにもちろん必要なことですが、要求仕様をきちんと実現する、あるいは不具合を作り込まない、言い換えると品質作り込みを行う活動はより重要です。そのためには、一連の技術向上を常に行うことが重要です。システムやソフトウェアをきちんと作るためには、ソフトウェア工学の技術だけではなく、ドメイン知識、ヒューマンスキル、管理技術、品質管理技術、要素技術などさまざまな知識や技術が必要となります。また、新しいものを作っていくための技術開発ができることも必要です。

そのためには必要な知識や技術を持った人を集めるということも手段の一つですが、なかなかそうは行きません。本を読む、職場で勉強会を開く、セミナに参加する、お客さまに伺うなど、メンバが知識や技術を学ぶ方法はいろいろあります。OJTもその方策の一つでしょう。

付録Aに参考文献を提示しましたように、世の中には良い本がたくさんあります。それらを手に入れ、読んでみてください。また、**①セミナに参加するなど、メンバの内側からのスキル向上、体力づくりを行い、常に技術の向上**を目指しましょう。

Check 10 自分たちのスキルはきちんと把握できているか

品質指標を効果的に使い、自分たちの身の丈に合った品質コントロールを行うためにはまず、自分たちが何をできるかという足元を見極めることが重要になりま

す。プロジェクトプロファイリングおよび、品質目標の設定を行う過程において自分たちがどのような状態であるかをまずは把握できたのではないでしょうか。また、「ものを開発する技術＝プロジェクトを遂行する技術、レビューする技術、テストする技術、プロセスやプロダクトを計測しまとめ上げる技術」などはそれぞれ必要なスキルが異なります。**①それぞれの場面でどのようなスキルが必要なのかを見極め、②自分たちのスキルを把握していく**必要があります。そしてその結果として、そもそもプロジェクトの遂行に無理なことがあるとわかれば、必要な技術を導入するとか、勉強の機会を設けるとかの対策を立てましょう。

Check 11　システムプロファイリング～品質目標値の設定まで、レビューを行ったか

本ガイドでは品質を測るための手段として、レビューのプロセス、成果物のレビューの十分性などを計測することを提案しました。本ガイドを使ううえでももちろん、レビューをすることを忘れてはなりません。**①システムプロファイリング、プロジェクトプロファイリング、品質目標値を決定する、それぞれの場面でレビュー**を行いましょう。レビューを行うプロセスで、また、ステークホルダと会話することで新たな認識が生まれるかもしれません。さらに、**②レビューを行うことでメンバ間のコミュニケーションを図れ、スキル、意識向上につながればなおよい**と考えます。

Check 12　品質定量化の活動を通して得られた知見を広く社会に還元することを考えているか

品質作り込みの手立ては、世の中にいくつもあります。ただ、それらがどのように効果的に行われているか、どの程度行ったらよいかを測るための目標値は職場の中に閉じてしまい、公に語られることは少なかったように思います。SEC でのヒアリングやアンケート活動を通じても品質指標について「他社はどうしているか知りたい」「目標設定をどうしたらよいか良くわからない」という声を多く聞きました。そこで、本ガイドは1つの試みとして、測るべきものとタイミング、その目標値を提案しました。しかし、これらの値は各社へのヒアリングに基づき算出したものの、日進月歩であるソフトウェアの産業、技術の発展につれて変化していくものであるとも認識していますし、また、まだまだ精査の余地があり、フィールドでの値がきわめて重要になるものと認識しています。本ガイドを策定の段階でさまざまな企業の方々にそうしたデータの提示をお願いしてきましたが、残念なことに、これらのデータの提供をしていただけない企業もありました。本ガイ

ドを手にされた方々には、是非、本ガイドで提示した指標類に関する実データやその他の指標データなども含めて、**付録に添付したフィードバックシートなどを利用し、SECへフィードバックをお願い**できればと考えています。フィードバックシートでいただいた内容は、本ガイドの次バージョンとしてとり入れていきたいと考えています。また、SECのみならず、学会などへ発表されることにより、世の中へ貢献していっていただきたいとも思います。近年、学会などに投稿される論文数は激減しています。論文を通じての情報公開、議論が進まない状態です。こうした学会発表などは技術への貢献という側面とともに、自分たちの活動を整理し、より多くの方々からの意見を聞く場でもあり、自部門あるいは皆さん自身にとっても大きなメリットのある活動です。

付録A 参考文献

以下は本書を執筆するうえで参考にした、あるいはソフトウェアの開発から品質保証を行う場面で参考になるであろう書籍のリストです。近年、ソフトウェアエンジニアリングをはじめとして、多の書籍が出版されていますが、それらの中から主に基本的なもの、古くから読み継がれているものをリストアップしています。

タイトル	編著者	出版社	発行年
SEC組込みエンジニア領域発行書籍			
[改訂版]組込みソフトウェア開発向けコーディング作法ガイド[C言語版](ESCR)	SEC	翔泳社	2007
[改訂版]組込みソフトウェア向け開発プロセスガイド(ESPR)	SEC	翔泳社	2007
組込みソフトウェア向けプロジェクトマネジメントガイド[計画書編](ESMR)	SEC	翔泳社	2006
組込みスキル標準ETSS概説書[2008年度版]	SEC	翔泳社	2008
エンジニアリング			
ソフトウェア・エンジニアリング序説	ロジャー・S.プレスマン	ＴＢＳ出版会	1983
ソフトウェアエンジニアリング	イアン・ソマーヴィル	フジ・テクノシステム	1993
実践ソフトウェア工学(全3巻)	Roger S. Pressman	日科技連出版社	2000
ソフトウェアテスト			
ソフトウェア・テストの技法	Glenford J. Myers	近代科学社	1980
ソフトウェアテスト技法	ボーリス・バイザー	日経BP出版センター	1994
品質保証			
ソフトウェア品質保証入門	保田勝通、奈良隆正	日科技連出版社	2008
メトリクス			
ソフトウェア品質のガイドライン	Capers Jones	構造計画研究所	1996
ソフトウェア開発の定量化手法(第2版)	Capers Jones	構造計画研究所	1998
実践的ソフトウェア測定	John McGarry他	構造計画研究所	2004
ソフトウェア品質工学の尺度とモデル	Stephan H. Kan	構造計画研究所	2004
初めて学ぶソフトウェアメトリクス	ローレンス・H・パトナム他	日経BP社	2005
ソフトウエア開発の持つべき文化	カール・E・ウィーガーズ	翔泳社	2005
レビュー			
ソフトウェアインスペクション	Tom Gilb他	構造計画研究所	1999
一般			
ピープルウエア 第2版	トム・デマルコ他	日経BP社	2001

付録B 高品質作り込みの事例

ここでは、無線通信波形計測や分析、信号発生などを行う計測器を対象システムとして、本文記載のStep-1からStep-4で品質を定量的にコントロールして行く作業の簡単な事例を紹介します。対象システムは既存製品を高周波に対応して展開するもので、ユーザはメーカ等の開発者であるとします。

Step-1：システムプロファイリング

- ■ 人的損失はない → Type-3 以下
- ■ 修理に1週間かかる、各ユーザの使用頻度は低い
  （毎日使うものではない。5人に1人程度のユーザが不利益をこうむると想定）
- ■ 損害額は1ユーザあたり1日2,000円と見積る
- ■ 経済損失：被害日数（5日）× ユーザ数（3,000人）× 影響率（0.2）
  × 損害額（2,000円）
  ＝ 6,000,000円 → Type-1：N に決定

Step- 2：システムプロファイリング

- ■ 仕様は明確、他は突出した特徴は無い（下表参照）→ 補正係数：- 0.1

1	ソフトウェア規模	☐ 極めて小さい	☑ 普通	☐ 極めて大きい
2	ソフトウェアの複雑さ	☐ 極めて単純	☑ 普通	☐ 極めて複雑
3	システム制約条件の厳しさ	☐ 制約ゆるい	☑ 普通	☐ 制約厳しい
4	仕様の明確度合い	☑ 極めて明確	☐ 普通	☐ 明確になっていない
5	再利用するソフトウェアの品質レベル	☐ 極めて高品質	☑ 普通	☐ 極めて品質低い
6	開発プロセスの整備度合い	☐ 整備できている	☑ 普通	☐ 整備できていない
7	開発組織の分業化・階層化の度合い	☐ 開発組織が単純	☑ 普通	☐ 開発組織が複雑
8	開発メンバーのスキル	☐ メンバースキル高い	☑ 普通	☐ メンバースキル低い
9	プロジェクトマネージャの経験とスキル	☐ PMスキル高い	☑ 普通	☐ PMスキル低い
10	システム障害時のメーカ側損失額	☐ 極めて小さい	☑ 普通	☐ 極めて大きい
小計		-1		0
合計ポイント数				-1

Step-3：品質評価指標の選定と目標値の設定

■ 品質評価目標の選択

● 計測すべき基礎指標の確認

● レビュー作業充当率、設計書ボリューム率を仮に選択

✓ レビュー作業充当率 ＝ 全レビュー工数 / 開発全工数

✓ 設計書ボリューム率 ＝ 設計書ボリューム / ソースコード全行数

ID	PR15
名称	レビュー作業充当率
略称	RORE
名称（英語表記）	Ratio Of the Review Effort

参考値	N	NQ	C	HC	補正ベース値
	4.00	8.00	12.00	16.00	4.00
参考値の範囲	0.00 ～ 8.00	4.00 ～ 12.00	8.00 ～ 16.00	12.00 ～ 20.00	
計測単位	%				

ID	PD11
名称	設計書ボリューム率
略称	RDDV
名称（英語表記）	Ratio of the Design Document Volume

参考値	N	NQ	C	HC	補正ベース値
	9	19	29	39	10.00
参考値の範囲	0.00 ～ 19.00	9.00 ～ 29.00	19.00 ～ 39.00	29.00 ～ 49.00	

■ 品質目標値の設定

● レビュー作業充当率

✓ 補正値の算出：補正ベース値：4.00 、補正係数：-0.1

∴ 補正値 = 4.00 × (-0.1) = -0.4

✓ 品質目標値の設定：レビュー作業充当率のType1：Nの参考値 = 4.00

∴ 4.00 - 0.4 = 3.6

● 設計書ボリューム率

✓ 補正値の算出：補正ベース値：10.00、補正係数：-0.1

∴ 補正値 = 10.00 × (-0.1) = -1.00

✓ 品質目標値の設定：設計書ボリュームのType1：Nの参考値 = 9.00

∴ 9.00 – 1.00 = 8.00

ここでは、以下の見積り値を仮定して指標値の評価を行います。

- 開発全工数：20人 × 9ヶ月 × 155時間 = 27,900人時
- 開発ソフトウェア：8万Lines

Step-4：実際の開発フェーズでの指標値の測定と評価

■ レビュー作業充当率の評価の例

- 開発全工数の見積り値：27,900人時
- 『結合テスト開始時点』（＝テスト関連のみ見積り値、他は実績値）での全レビュー工数見積り値：900人時
- レビュー作業充当率＝ 全レビュー工数 / 開発全工数
  ＝ 900 / 27,900 = 0.032 = **3.2 (%)**

目標値 3.6(%) に対して、若干少ない

これまでのレビューを再度見直す、テストを厚くするなどの対応を行う

■ 設計書ボリューム率の評価の例

- ソースコード全行数の見積り値：80KLOC
- 『設計書レビュー開始時点』（＝ソースコードはまだ存在しないため、全て見積り値）での設計書ページ数：500ページ
- 設計書ボリューム率＝ 設計書ボリューム / ソースコード全行数
  ＝ 500 / 80 = **6.25**

目標値 8.00 に対して、若干少ない

設計書に抜けもれがある可能性が高いことに気をつけ、レビューを実施する

付録C ST-SEISMIC Scale (System Trouble SEISMIC Scale)

システム障害影響評価スケールとしてSECで検討したST-SEISMIC（System Trouble SEISMIC）を用いた事例を示します。これは地震の震度の概念（SEISMIC Scale）をシステム障害の段階になぞらえたもので、実際のシステム障害による影響がどのレベルに対応するか評価することを目的としています。

ST-SEISMICをシステムプロファイリングでのシステムのタイプ分けに反映させる目安としては、表に示すように8段階に区分したシステム障害の階級をもとにします。

システム障害影響評価スケールは、どの程度の不具合や障害であればどのような対策が必要かといったシステム障害への未然の対応の指針としても利用することが可能です。個々のシステムによって特性は異なりますので、それに応じた対策を考えたり、またシステムプロファイリングへフィードバックして開発段階での品質や信頼性折込みといった視点で利用したりすることができます。

表D-1：ST-SEISMIC Scaleとシステムのタイプとの関係

階級		ユーザ操作への影響	ユーザに対する健康被害の状況	ユーザに対する経済的被害の状況	システム周辺の環境への影響	システムタイプ
0	無	不具合に気付かない				Normal
1	微	一部のユーザがシステムの動作に違和感を覚える				
2	軽	ユーザの多くが不具合に気付く				Normal Quality Required
3	弱	ユーザのほとんどが不具合に気付きクレームを訴える人がいる	一部のユーザに軽微な健康被害の可能性あり	一部のサービスが停止し、ユーザの一部で軽微な経済損が出る可能性がある		
4	中	ユーザの目的が一部達成できなくなる	一部のユーザに健康被害がでる	一部のサービスが停止し、ユーザの一部で経済損が出る可能性がある		Critical
5	強	ユーザの目的が達成できなくなる	一部のユーザに重大な健康被害がでる	一部または全部のサービスが停止し、ユーザの一部で多大な経済損が出る可能性がある	システムの不具合が限定的ながら、社会的不安を引き起こす、又は利便性を損なう	
6	烈		多くのユーザに重大な健康被害がでる	一部または全部のサービスが停止し、多くのユーザで多大な経済損が出る可能性がある	システムの不具合が、社会的不安を引き起こす	Highly Critical
7	激		ほとんどのユーザに重大な健康被害がでる	一部または全部のサービスが停止し、ほとんどのユーザで多大な経済損が出る可能性がある	システムの不具合が、大きな社会的不安を引き起こす	

付録D フィードバックについて

本ガイドで提案した内容に関して、具体的なご意見をお聞かせください。
いただいた内容はESQRの向上に反映してまいります。
ご意見は添付のフィードバックシートの内容を、メールまたはIPA Webサイトのお問合せからお送りください。

宛先

- **電子メール**

  「フィードバックシート」の項目をメールにご記入の上、下記のメールアドレスまでお送りください。

  メールアドレス：ipa-info@ipa.go.jp

- **Webサイト**

  次の手順でお願いします。

  1. 次のWebページへアクセスしてください。

     URL：http://sec.ipa.go.jp/

  2. 「お問合せ」→「お問い合わせ（セキュリティを除く事業全般）」のリンクから、「お問い合わせ：一般」のページに移り、必要事項とご意見の内容をご入力ください。
  3. 入力が終わりましたら「送信する」ボタンを押してください。

ご協力よろしくお願いいたします。

ESQR フィードバックシート

送　信　日	年　　月　　日
送信者情報	氏　名
	所属名
	連絡先　（電話）　　（メール）
フィードバック項目	□　システムプロファイルについて
	□　プロジェクトプロファイルについて
	□　品質指標の定義・参考値について
	□　第4章の各ヒントについて
	□　その他

自由記述欄

※上記の内容に対し、SEC の問い合わせに対応　　□　可　　□　不可

ご意見ありがとうございました。頂きましたご意見は、改訂版に反映させて頂きます。

あとがき

SECで提供する手法はコンセプトシートを作成するところから作業を開始します。

初期の頃の品質作り込みガイド：ESQRのコンセプトシートには以下の数々の文が躍っていました。

- 対象システムをカテゴリ分けし、品質指標を何パターンか提供する
- ESPRのプロセス定義を使用しプロセスメトリクスを定義
- プロダクトメトリクスはアンケート集計を基に定義
- 数値メトリクスを明確に決めて品質折込みをする
- 数値の一人歩きは容認

これらのコンセプトを元に企業ヒアリングやアンケートを実施し、高品質技術部会の方々や他のESxRシリーズのセミナに参加いただいた方からもいくつも貴重なご意見をいただきました。その後、データをまとめ、品質指標の項目を検討するという作業を繰り返し、この品質作り込みガイド：ESQRが出来上がりました。

冒頭にあげたコンセプトのうち、「数値の一人歩きは容認」、すなわち「データを提示することの是非について」は幾度となく大きな議論となりました。SECのような公的な機関から品質目標値に相当するようなデータを提示することは今までにないことであり、大きな決心を必要としましたが、データを示すことで世の中の議論が一歩も二歩も進むであろうということを確信し、参考値の提供という手段を選ぶことになりました。

おりしも世の中は組込みソフトウェア全盛期。ソフトウェアの品質が組込みシステムの品質を左右する時代です。増大する多くのソフトウェアの品質問題が社会問題になる一方、終わりの見えないソフトウェア開発・テストに悩み苦しみ、疲弊していく人たちが多くいることも別の社会問題となっています。品質を確保するための手段、手法はいろいろ提示されていますが、それらをどこまで実行したらよいのか、という目標は企業機密の壁に阻まれ、なかなか共通の認識を作り出すことができずにいます。

ESQRは最初の一歩を踏み出したところです。提示しました品質指標の内容や参考値などに関してはまだまだいろいろなご意見があることと思います。この本を手に取られた方はぜひ、巻末に掲載しましたシートを使ってデータの精査に向けて議論にご参加いただきたいと切望しています。

執筆者

三原 幸博（IPA 技術本部 ソフトウェア・エンジニアリング・センター）

十山 圭介（IPA 技術本部 ソフトウェア・エンジニアリング・センター）

石井 正悟（IPA 技術本部 ソフトウェア・エンジニアリング・センター）

濱田 直樹（IPA 技術本部 ソフトウェア・エンジニアリング・センター）

平山 雅之（日本大学）

吉澤 智美（日本電気株式会社）

山口さゆり

右近 豊

レビュー協力

井沢 澄雄（日本電気株式会社）

伊藤 雅子（富士通株式会社）

上田 直子（富士通株式会社）

大野 克己（トヨタテクニカルデベロップメント株式会社）

古賀 惠子（株式会社日立ソリューションズ）

宍戸 文男（株式会社イーソルエンベックス）

野中 誠（東洋大学）

三橋二彩子（日本電気株式会社）

山崎 太郎（日本ユニシス株式会社）

組込みソフトウェア開発向け
品質作り込みガイド

2008 年　1 2 月 2日　初版第 1 刷発行
2010 年　3 月 10 日　初版第 2 刷発行
2011 年　7 月 25 日　改訂新版第 1 刷発行
2012 年　9 月 10 日　改訂新版第 2 刷発行

編著者　　独立行政法人情報処理推進機構
技術本部 ソフトウェア・エンジニアリング・センター
（http://sec.ipa.go.jp/）

ISBN978-4-905318-13-2
C3055 ¥952E

定価 1,000円
（本体952円＋税５％）

IPA 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

SEC-TN12-001

VEGETABLE OIL INK

R70
70%リサイクル用紙を使用しています。

リサイクル適性Ⓐ
この印刷物は、印刷用の紙へリサイクルできます。