2009年11月16日発行
第5巻第5号（通巻20号）
ISSN 1349-8622

SEC journal

別冊

Software Engineering Center

ソフトウェアシステムの開発力向上に向けて

組込みシステム開発技術リファレンス 特集号

Part 1. ソフトウェア開発力強化のすすめかた

1. ソフトウェアの品質と開発効率とは
2. プロセス／プロダクトの改善に向けて
3. 経営者からスタートするソフトウェア開発力強化
4. 現場の管理者が主導する改善
5. 改善に向けて担当者が考えるべきこと
6. 改善を支える技術者教育

Part 2. ESxRを活用した品質・開発効率の改善

1. ESxRとは ～狙いと全体像・関連
2. 実装品質の改善 ～ESCRとは
3. プロセスの改善 ～ESPRとは
4. プロジェクト管理の改善 ～ESMRとは
5. 品質定量コントロール ～ESQRとは

IPA
独立行政法人 情報処理推進機構
http://www.ipa.go.jp/

SEC journal
Software Engineering Center

別冊 ESxR 特集号
目次

本特集号で紹介しているESxRシリーズ：ESCR、ESPR、ESMR、ESQRは一般書籍として販売されています。
また、SEC WebサイトよりPDFをダウンロードすることができます。ご参照ください。
http://sec.ipa.go.jp/publish/index.html#emb

巻頭言

組込みソフトウェアに期待する

IPA/SEC 所長 **松田 晃一**

日本経済を支える組込み産業

世界のマーケットで高いシェアを占めている自動車・情報家電・工作機械等の日本製品は、その中核を組込みソフトウェアが担っていることは今更言うまでもない。2008年の貿易統計によれば、組込みソフトウェア関連製品の輸出額は45.9兆円であり、輸出総額の56.7%を占めている。更に組込み産業は、2008年における国内総生産の13.5%を占めている。身の回りを見渡すと、小売業のPOS端末※1、通信業の端末機器、金融業におけるATM※2、製造業のロボットなど、ほとんどすべての産業が組込み関連機器なくしては事業が成りたたないばかりか、それらが競争力の源泉となっている。つまり、組込みソフトウェアは我が国の経済活動、国民生活の基盤を支えるソフトウェアとなっている。組込みソフトウェア産業が、いっそう力強く発展することが、すなわち我が国の持続的な発展を支えることになるわけである。

組込みソフトウェアこそが生命線

ところで、日本の「ものづくり」における強い競争力は、ハードウェアの量産技術に負うところが大きかった。つまり、量産設計や量産工程での作業手順の絶え間ない改善、歩留まり・品質の向上、徹底した部品調達コスト削減などが日本の強さであった。しかしこれはハードウェアであったからこそ発揮される強みであって、この部分がソフトウェアに置き換えられれば置き換えられるほど、その強みを発揮できる領域が狭まってくる。そもそもソフトウェアに量産という概念は無いわけだから、今後はソフトウェア設計・開発の分野が競争領域になってくる。組込みソフトウェアをいかに素早く、高い品質で、安く作れるかが組込み製品の競争力を決定付けることになる。

更に、機能の実現をソフトウェアが担うようになると、ハードウェアで実現する場合における回路設計や実装設計、部品の条件などの物理的制約が格段に緩くなる。極端に言えば、アイデアがあればソフトでは何でも実現できる。この結果、ハード実装の勝負から、製品にどんな機能を持たせるかのアイデアの勝負、サービスの勝負がより重要になってくると言える。

ESxRの活用で組込みソフトウェアの開発力強化を

SECは、2004年の設立以来、産学官連携の立場を活かし、業界の協力を得て開発現場のベストプラクティスを収集・整理して、組込み開発リファレンス集として取りまとめてきた。これまでにまとめた4種の組込み開発リファレンス（以下、ESxR※3）について、基本的な考え方と内容の概略を紹介し、できるだけ多くの関係の方々にESxRを知っていただき、活用していただきたいとの思いを込めて、この特集を企画した。そして、それぞれの職場での組込みソフトウェア開発に活用していただくと共に、組込みソフトウェア技術者の育成の一助になることが私たちの願いである。

SECでは、これからも動きの速い技術動向の変化に遅れることなく、必要なエンジニアリング技術を幅広く提供し、日本の組込みソフトウェア産業の競争力強化と活性化に貢献していきたいと考えている。

皆さんのこれまでのご支援に感謝すると共に、これからも変わらないご理解とご支援をお願いする。

※1 POS端末：Point Of Sale端末。店舗で販売情報を記録・管理するための端末。
※2 ATM：Automated Teller Machine。銀行などに設置されている現金自動預け払い機のこと。
※3 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。

はじめに

SECが発足して約5年が経過した。この5年間、部会あるいは企業ヒアリングなど様々な形で多くの方々にご協力をいただき、その成果としてソフトウェア開発技術リファレンス ESxR[※1] シリーズ（以下、ESxR）を整備してきた。

ESxRは、最初に刊行した「ESCR[※2]：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］」以来、昨年刊行した「ESQR[※3]：組込みソフトウェア開発向け品質作り込みガイド」までで4冊がそろい、ようやく多くの皆さんに参照・使用していただける形となってきた。

その一方で、先行してお使いいただいている方々からは、実際の利用にあたっての導入ガイドのようなものが欲しいといった声も多く寄せられていた。今回、こうした声を受けて、ESxR導入のために「SEC journal」特集号を発刊することとなった。

実はESxRについては、これまでSEC主催セミナーなどを通じて多くの方々にその技術的な内容をお話してきたが、セミナーの人数枠がそもそも少ないこともあり、希望される方全員をお招きすることができず、加えて、セミナーでは主に技術的なポイントについてお話しすることが多く、導入の考え方や、基本的なコンセプトなどの背景を説明する時間が取れていないという現状がある。

本特集号では、こうした点を踏まえ、ESxRを実際に現場に導入したいと考えている方々や、ESxRとはそもそもどんなものなのかを知りたいと思っていらっしゃる方々に、できるだけわかりやすく説明をすることを狙っている。本特集号のPart 2をぜひ熟読していただければと思う。またそれに先立ち、Part 1ではESxRも含める形で、ソフトウェア開発力強化とはそもそもどのようなものなのか、そして開発力強化に向けて経営者・管理者・担当者はそれぞれどう考え、どう振る舞っていけばよいのかといったことを整理している。Part 1では、日ごろESxRの策定・普及といった業務に携わっているSEC研究員から、ESxR利用者の皆さんへのメッセージという側面も併せ持っている。

もうひとつ、本特集号を読む上での留意事項がある。ESxRの頭の「E」は「Embedded」の略である。当初、ESxRについては、組込みソフトウェア開発者を念頭に整備してきた。しかし、多くの方々がお気付きのように、組込みソフトウェアといってもソフトウェアであることに変わりはない。その意味では、ESxRも一部を読み替えることにより、組込み以外のソフトウェアの開発の参考になる部分も少なくないと考えている。組込みソフトウェアの関係者以外の方々にも、ぜひ本特集号を手に取っていただき、目を通されることを期待している。

本特集号は、ある意味でオムニバス形式となっており、どこから読んでいただいてもその内容が理解いただけるように編集してある。ぜひ、皆様の気に入ったところから、あるいは気になるところから目を通していただき、そこでのヒントをもとに実際の開発現場で、ESxRを利用したソフトウェア開発力強化を推進していただければ幸いである。

2009年 秋
SEC 組込みソフトウェアプロジェクト
研究員一同

※1 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。
※2 ESCR：Embedded System development Coding Reference。2006年5月発行。
※3 ESQR：Embedded System development Quality Reference。2008年12月発行。

Part 1

ソフトウェア開発力強化のすすめ方

品質の高いソフトウェアを効率的に作り上げるためにはどのようにしたらよいのだろうか。Part1では、このためのソフトウェア開発力についてシステム開発に関係する経営者、管理者や実務担当者などそれぞれの立場からの取り組み方について紹介していく。
またソフトウェア開発力強化に向けた人材育成の取り組み方などについても考えてみたい。

Part 1-1

ソフトウェアの品質と開発効率とは

1. ビジネスとしてのソフトウェア開発

メタボ化が進むソフトウェア開発

携帯電話、テレビなどに代表される情報家電機器、あるいは自動車、航空機などの運輸交通機器、そして金融、電力をはじめとする社会インフラシステムなど、様々なところでコンピュータを利用したシステムの利用が広まっている。一般にこれらのシステムは大なり小なりコンピュータを搭載しており、そこで動作するソフトウェアやそれによって制御される様々なハードウェアの連携によって機能・サービスが実現されている。

とりわけ近年、これらのシステムでは実現する機能が多様化しており、システムに搭載されるソフトウェアやそれらが制御するハードウェアの量は極めて膨大なものとなる傾向が続いている。例えば身近な携帯電話などを見てみよう。図1は一般的な携帯電話などの電子機器に搭載されるソフトウェアの規模の変遷を示したものである。最近の携帯電話では約数百万ステップを超えるソフトウェアが搭載されるケースも少なくないと言われている。これは例えば50行程度のソフトウェアのまとまりをひとつの意味を持った部品と考えた場合、携帯電話の中に数十万点の部品が入っているのと同じという計算になる。数十万点という部品点数というと想像が難しいが、最新の航空機などに利用される部品点数が約300万点と言われていることを考えると、この数字がいかに大きな数字であるかがご理解いただけるのではないかと思う。

一方で、こうした多機能な携帯電話が持つ高度な機能を十分に使いこなせているユーザは限られているというぼやきもよく耳にするところである。このように身近なところにある電子機器などの製品を取り上げても、若干、オーバースペックなものは少なくなく、ソフトウェア・システムのメタボリック化と呼んでもよい状況が進行している。しかも、航空機などの場合には開発に通常10年を超える長い年月をかけ、その機能や品質を洗練したものに熟成して開発されていくが、携帯電話などでは長いものでも半年から1年程度といった極めて短い開発期間で開発が行われている点に注目する必要がある。このように現在のソフトウェア・メタボリック・シンドロームは、短い開発期間の中で進んでいる点も特徴のひとつと考えられる。

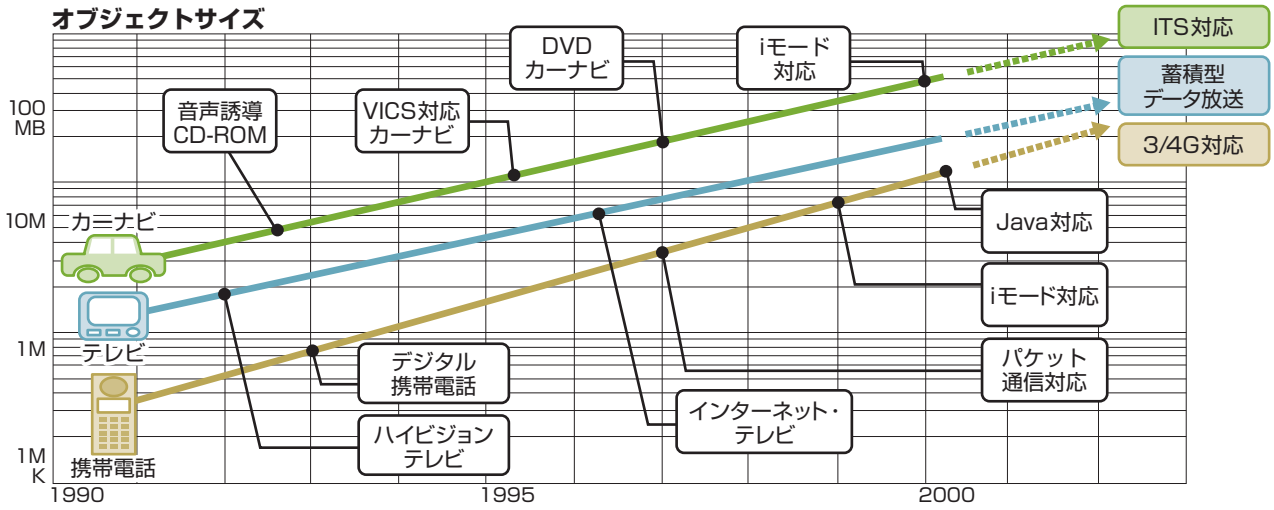


図1 電子情報機器のソフトウェア規模の変遷
出典：「日経エレクトロニクス」2000 9-11（no.778）をベースに追加、修正。

ソフトウェア産業ガラパゴス化の危機

一方で、これらのコンピュータシステムの多くは前述したように情報家電機器、運輸交通機器等の一般消費者が最終ユーザとなる製品に搭載されたり、金融・電力のように一般消費者の社会生活に直結するシステムに多く搭載されている。そして、これらのシステムの多くは、それらを製造・販売する企業や、あるいはそれらを利用してサービスを提供する事業者にとっては、常にビジネスの中心的な存在となっている。これは言い換えると「システムの失敗は即、ビジネスの失敗につながる」ということでもある。もちろん、我が国の多くの企業では、その真摯(しんし)な取り組みによって優秀なシステムを生み出し、より大きな成功を生み出している。しかしながら、2008年後半に発生した世界的な景気後退の波の中で、組込みシステムにかかわる企業群でも、この波を乗り切った企業と、乗り切れなかった企業があるように、大きな明暗が見え始めている。例えば、先に例に出した携帯電話などの分野では、十年程前には国内で十社を越える開発メーカがしのぎを削っていたものが、昨今の景気後退を経て国内メーカは数社に絞り込みが進んでいる。更にこうした携帯電話ビジネスは海外メーカのさらなる攻勢にもさらされ続けており、もはや日の丸携帯は空前のともしびといった感すらある。

さて、それではなぜ日の丸携帯が失速したかということになるが、ここで、あらためてシステム開発をビジネスの視点から見てみよう。一般にビジネスでは投下資本と利潤といった企業の視点と製品やサービスへの期待に代表されるユーザ視点の二つをバランス良くキープすることが求められる。とくに我が国の製造業の現場においては、従来QCD※1のバランスの最適化を強みとして世界を席巻してきた。

しかし、前述の携帯電話の場合、この視点があまりに国内目線になりすぎ視野狭窄(きょうさく)を引き起こしていたとの指摘も少なくなく、ちまたでは我が国の携帯電話は東洋のガラパゴス※2ともささやかれている。

こうした我が国のお家芸であるものづくりビジネスの多く、とくにそれを支えるシステム開発の分野では現在、様々な海外勢力の攻勢の前に守勢に立たされているものが少なくなく、第2、第3のガラパゴス化が目前に迫っているのかも知れない。

2. 組込みシステムと企業情報システム

システムの特徴による区分

我が国の情報システムを取り巻くビジネス環境はこのように待ったなしの状況である。前節では情報家電、運輸交通機器や金融、電力などの社会インフラシステムなどを総じて言及したが、一般的には、これらのシステムは組込みシステム(ET※3システム)、情報処理システム(IT※4システム)といった形で大きく2分して扱われる(図2)。もちろん、この分類はあくまでも便宜的であり、このようにカテゴライズしたところで、それぞれのカテゴリの待ったなしの状況に変わりはない。ここでは、こうした状況をいかにして克服するかということを考える前に、まず、これらの各カテゴリのシステムがどのような特徴を持っているかを整理してみよう。

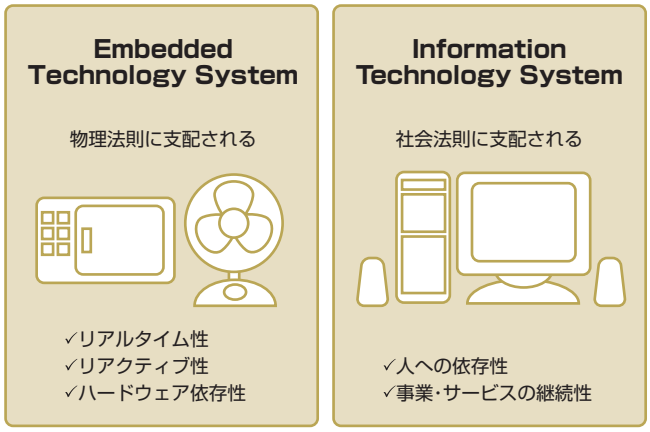


図2 ETシステムとITシステムの違い

ものづくりの伝統を受け継ぐ組込みシステム開発

まず、我が国のものづくりの伝統を受け継ぐと言われている組込みシステムについて考えてみる。このカテゴリは、かつての携帯型ステレオカセットプレーヤに代表されるように軽薄短小が商品価値の支配要因であった時代から、現在は自動車に代表されるような機能複合体としてのものづくりと付加価値創造へと大きく価値観が変遷している。しかしこうしたものづくりに対する価値観が変遷する中でも、組込みシステムの黎明(れいめい)期からの面々と伝わるものづくり産業としてのDNA(遺伝子)は我が国のメーカの中で確実に受け継がれている。では、こうした組込みシステムの最大の特徴はどのようなところであろうか?一言で言うと「システムが動作する環境に支配されながら、それに対応したものづくりが求められる」という点にあるのではないだろうか。

例えば身近なところで電気ポットを例に考えてみよう。ご存じのように最近のポットは中に水を入れると、自動で湯沸かしを行い、保温などもこなすものがほとんどである。この動作の中心は、電熱ヒータと湯温をチェックする温度センサであり、内部の制御ソフトウェアからの指示によって、定期的に水温をセンシングし、必要であればヒータを動作させて加熱するというもので

※1 QCD:Quality, Cost, Delivery。製造業の生産管理における基本要素、品質(Q)、費用(C)、納期(D)。
※2 ガラパゴス:独自の生態系を持つガラパゴス諸島になぞらえ、国内で独自の進化・発展を遂げたため、海外市場に進出できないことを指す。
※3 ET:Embedded Technology
※4 IT:Information Technology

ある。実はこの単純な動作の中に組込みシステムの本質が隠されている。すなわち、ポット内の水は物理法則（正確には熱力学の法則など）に支配されている。つまりヒータによって発生した熱量が、水に伝導され、水は温度上昇を起こし、沸点に達すると沸騰し、温度変化が無くなるというものである。このため極端な場合、海抜が数メートルの東京で動作させる場合と、標高約1,600メートルの米国・コロラド州ボルダーなどの高地で利用する場合では、水の沸点が異なるため、動作が異なってしまう可能性もある。つまり、水の温度をセンシングし、100度になった時点を沸騰したという判断でヒータをオフにするロジックを組むと、高地では水温は100度に達することなく沸騰を始めるため、ヒータがオフにならず、沸騰し続けてしまうという現象が発生してしまうのである。

このように組込みシステムの多くは、システムが動作する環境や制御するデバイス動作に関係する物理法則や自然法則に支配されているという特徴を持っている。

人や社会に支配される情報処理システム開発

一方、情報処理システムの場合はどうであろうか。この分野でも、最近話題のクラウドコンピューティング※5など、計算機の分散処理技術やインターネット技術の進化に伴い、システム形態が大きく変化しようとしている。しかしながら、こうしたシステム形態の変化の中にあっても、これらの情報処理システムの本質は、様々なビジネスデータを多様な形態に加工・処理するデータプロセッシングが中心であり、それによってさらなるビジネス活動の源泉となるデータや情報を生み出すところにあると言える。

例えば、多くのビジネスマンが必ずと言ってよいほど利用する企業内の勤務管理システムなどでは、それぞれの企業内の出退勤処理の業務ルールの上に成り立っており、そこにはそのシステムで勤務時間などの管理や精算を受ける従業員が深く関係している。ここで例えば、ある企業の勤務管理システムがダウンしたとしたどうなるかを考えてみよう。恐らく、システムがダウンした場合に考えられる対応として、まずは個々の従業員に対し、その日の出勤退勤の時刻などを取りあえずメモし、所属するグループの勤務管理担当者などに渡すように指示をされる場合もあるかも知れない。あるいは、個々にメモを残しておき、後日、システムが正常に戻ってから入力をするよう指示されるかも知れない。

このように、情報システムでは、社会のルールに基づいて運用されるものが多く、システムトラブルのリカバリは関係する人間系が介在する場合も少なくないのが特徴とも考えられる。

3. ソフトウェア・システムの品質とは

ソフトウェアの付加価値がビジネスを制する

ここまで述べてきたように組込みシステムの多くは自然法則に支配される一方で、情報処理システムの多くは社会法則に支配されるという点において若干の違いを見て取ることができる。しかしながら、これらのシステムに関与する企業にとっては、どちらのカテゴリのシステムであってもビジネスの源泉そのものであるという事実には変わりがないと言っても差し支えないであろう。そしてビジネスとしてのソフトウェアを考えると、「付加価値の高いものがビジネスの帰趨（きすう）に影響する」という極めて単純かつ明快な法則に支配されることは言うまでもない。

さて、それはもう少し話を進めて、次にソフトウェア製品あるいはソフトウェア・システムの付加価値とは何であるかについて少し考えてみよう。恐らく真っ先に挙がる項目は機能の多様化ではないだろうか。それぞれのシステムの特徴の項でも述べたように、我が国の組込みシステムや情報処理システムの進化の歴史は、ユーザが求める様々な機能を包含し連携することで多機能化を実現し、付加価値の向上を目指してきた歴史とも言える。しかしながら、こうした高機能化による付加価値創造の方式は、先に指摘した携帯電話の事例を見るまでもなく、既にシステム機能の飽和という形で限界に近づきつつあるのかも知れない。さて、こうした高機能追求型による付加価値創造方式に限界が見えてきた今、我が国のソフトウェア・ビジネスはこれに続

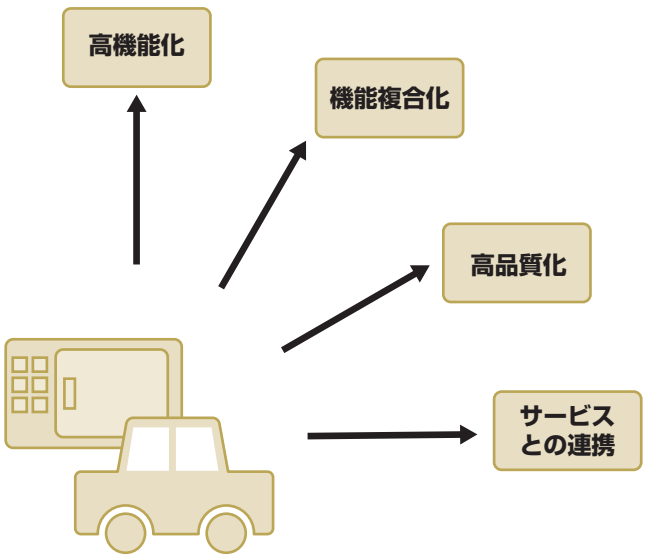


図3 システム付加価値の創造

※5 クラウドコンピューティング：インターネット上に用意された様々なシステムリソースを、ユーザは提供元を意識することなく、サービスとして使用するコンピュータの利用形態。

く新たな付加価値創造の方向性を考え、打ち出していくことが求められる時期にさしかかっていると言ってもよいかも知れない（図3）。

もちろんこうした新たな方向性を模索する際にも当然のことながら、様々な制約条件を加味しながら考えていくことが求められるのは言うまでもない。少なくとも、今我々が考えようとしているソフトウェア・ビジネスの将来ということに関して、その位置付けとしてのものづくり産業、サービス創出産業である点は、恐らく変わらぬ目的となり続けるし、また産業資源としては我が国が優位に立つ勤勉かつ優秀な人材をいかに有効に利用していくかを、その出発点において考えておく必要があろう。

さて、上記のような制約や国内特性を考慮して、我が国のシステム・ビジネスの国際的な勝ち残りに向けた戦略を考えていくと、幾つかのシナリオが考えられるが、ここではシステムが提供するサービスと品質の視点からもう少し掘り下げてみたい。

ものづくりの立場から捉える付加価値創造

システム価値の源泉をどこに求めるかはそれぞれの企業のビジネスや立ち位置によっても大きく異なってくるのは言うまでもない。近年、情報処理分野でもサービスサイエンスといった考え方が議論されるようになってきており、一部では、単純なものづくりの立ち位置からサービス産業としての立ち位置にシフトし、システムを捉えていく動きも見られるようになってきている。

確かに様々なITシステムや組込み機器は最終的に機能実現によってユーザにあるサービスを提供するという側面を持っており、サービスの視点からシステム作りを捉えていくという方向性はある意味、現在の情報処理ビジネスの方向性を暗示していると考えることもできる。しかしその一方で、この数年、世間を騒がせた有名飲食店の食品偽装などの問題に見るように、サービスは一流であっても、そこで提供する料理（＝製品）に誤りがあった場合、顧客は一気に離れビジネスとして立ちゆかなくなってしまうことは言うまでもない。つまり、サービス業といえども、その根底には提供する商品の品質が十分であることが前提となっていると考えてもよさそうである。このように見てくると、ソフトウェア・システムを取り巻くビジネスの基本も、実はその製品の品質の良否に大きく依存していることが理解できるのではないだろうか。

ソフトウェア・システムの品質の多様性

さてそれでは、このソフトウェア・ビジネスの成否を決めかねないソフトウェア品質とはどのようなものであろうか? SECという立場から、これまでにも組込みシステムや情報システムに関連する多くの方々にお会いし、議論させていただく機会が多々あったが、ソフトウェアの品質は、それぞれの方の立ち位置などによって千変万化、多種多様な捉え方があるのに毎回驚かされている。

例えば、ある組込みシステム開発を手がけている企業では中堅の技術者の方が、ソフトウェアの再利用性を重視され、SECが整備したコーディング作法などを積極的に活用されソースコード品質向上を重視して活動されている。また、別の大手車載機器のソフトウェア開発責任者の方は、同じくSECから発行している開発プロセスガイドを参考に、自社のソフトウェア設計作業などを中心に開発プロセスの見直しや開発の見える化を推進されている。その一方で同じソフトウェアの世界でも、昨年、経済産業省とSECの共同事務局で検討を進めた「重要インフラ情報システム高信頼化」の議論の中では、電力や交通など我々の社会生活に欠くことのできない重要インフラシステムについて、これらのサービスを提供する事業者の視点から見た事業継続性や、これらのシステムの運用保守などの重要性が議論されている。このように、ソフトウェアの品質といった場合、それぞれの企業の立ち位置やビジネス形態によって、その議論の方向性は多様なベクトルを持つことがおわかりいただけるのではないだろうか。

ソフトウェア・システムの品質の切り口

このように考えてくると、ソフトウェアの品質は非常に多くの切り口があり、それぞれの切り口に対応する形で、ソフトウェアの品質問題を捉えていかなければならないということになってしまう。

しかし、こうしたソフトウェアの品質の多様な切り口ではあるが、もう少し前述の例を参考に整理してみると、実はサービス構成要素としてのソフトウェア品質と、純粋に計算機システムの構成要素としてのソフトウェア品質という大きく二つの側面に分けることができるのではないだろうか。すなわち、前出の様々な例の中で、例えば、事業継続性の視点やシステム運用の視点から見たソフトウェア品質は、どちらかというとサービス構成要素としてソフトウェアを見た場合の期待品質であり、先に例に引いた企業の勤務管理システムのように、この側面の品質はシステムとそれを取り巻く運用環境・利用条件などと相互補完されるという特性を持っていると考えてもよさそうである。こうした側面から品質を捉えていく場合にはソフトウェアをサービスという視点から見つめ直し、考えていく必要があるかもしれない。

さて、一方で、システムを直接的に実現していくソフトウェアについて、そのソースコードの再利用性や設計の見える化などは、純粋な計算機システムそのものの品質に直結する要素で

あると考えることができる。そしてこれは計算機システムそのものの品質／信頼性向上を目指す考え方に直接つながり、ソフトウェア開発の問題を直接的に扱うソフトウェア工学の範疇（はんちゅう）に入ると考えることができる。もちろん、ソースコードの再利用性といった取り組みの先には、製品をシリーズ化して開発量を抑制し開発の生産性を向上させるというコスト面を意識した戦略も間接的に織り込まれていると考えるのが適切であろう。

4. 品質の高度化による付加価値創造

最近のシステム障害から見る品質への取り組み

さて、このように便宜的にソフトウェア品質に関する視点を二つに区分してみたのだが、既に多くの皆様がお気付きのように、こうした区分もまた、あくまでも簡易的な分類であり、それぞれが密接に関係するものであることは言うまでもない。その一方で、先に指摘した食品偽装の問題でも明らかなように、品質という側面については、まず、商品としての製品自身の品質が保証されていることが大前提となり、その上で、それを活用したサービスあるいはビジネスの品質の議論が成立していくという特徴がある。

それでは具体例をもとに品質の問題をもう少し詳しく見てみよう。表1は最近に発生したシステム障害の例から幾つかを取り上げたものである。

例えば、3番目の事例は、関東地方のモノレールのブレーキ故障によって車両が暴走した事故であるが、運輸安全委員会によると、電子機器から発生した不規則なノイズの影響で制御上の誤動作が発生したとの原因が報告されている。多くの組込みシステムではいわゆる専用のハードウェアデバイスと専用の制御ソフトウェアが相互連携し機能実現を図っているが、そうした中で、物理的なノイズなどの影響でソフトウェアが誤動作するという事象は電子機器の分野では「よくある不具合」のひとつである。こうした不具合は純粋に計算機システムとしての不具合そのものであり、問題はこうした典型的な不具合をなぜ事前にレビューやテストで検出できなかったかということになる。

一方、運輸業界などで最近多く発生している予約発券業務や列車運行管理などのシステムでの不具合の中には、1番目や2番目の事例に見るように、システムのデータセット・ミスやソースコードの修正ミスなども多く報告されている。こうした不具合もソフトウェアを作るという作業から見ると起こるべくして起こる典型的な不具合のひとつということができる。ここでも問題は、修正したコードのチェックや確認の中でなぜこうした不具合を見逃してしまったかということになる。

このように見てくると、先に分類した第2の品質、すなわち計算機システムとして求められる品質の確保が、やや手薄な状況になっていると言えるのかも知れない。こうした計算機システムとしての品質のほとんどは、その開発の過程で作り込まれるものであるが、近年のシステムのメタボ化の中で、品質作り込みという我が国のものづくりの原点が揺らいでいることの現れであるのかも知れない。このような傾向は実は我が国のシステム開発に限った傾向ではなく、システムのメタボ化を背景にしたシステム品質の不安はある意味、世界の様々なところで発生している。これらの傾向を見ると、現在の情報処理分野のビジネスの中では、「いかに良質なシステムを、いかに効率的に提供していくか」という問題を解決していくことが、極めて重要であることがあらためてご理解いただけるのではないだろうか。ここまで整理すると、

表1 最近のシステム障害の事例

2007 May	航空会社 国内線予約システム故障	空港端末とホストコピュータをつなぐルータの管理プログラムの設定ミスによるメモリ故障が発端 ⇒130便が欠航。影響は4万人以上、キャンセルなどによる損失4.5億円
2008 Dec	鉄道会社 運行管理システム障害	鉄道運行管理システムのデータ変更に伴う入力ミスなどによりシステムがダウン ⇒新幹線の100本以上が運休し約13万人が影響
2009 Feb	モノレールトラブル	モノレールのブレーキ制御ソフトウェアが電子波ノイズ影響を受け誤動作 ⇒営業運転中に駅をオーバーラン

①システムやソフトウェアの品質面を確実にすることは情報処理サービス分野や電子機器分野でのビジネスを展開する上での必須要件であり、
②昨今のシステム事情や我が国の特性を考慮すると、システム品質はこの分野で引き続き中心的なプレーヤとして勝ち残っていく上で、極めて有力な武器になる。

ということができるのではないだろうか。

システム・ソフトウェアの品質向上戦略

さてそれではビジネス展開上、これだけ重要なシステムやソフトウェアの品質であるが、具体的にどのようにして向上させていけばよいのだろうか。

詳細は本特集号の以降の記事なども参照にしていただくとして、まず、ソフトウェアの品質を向上させるためには、様々な方々が連携していくことが重要である。つまり、品質向上は品質保証担当だけの仕事ではないし、設計者だけの仕事でもなく、品質は組織的にデザインされた活動の中から生み出されるものである。例えば、設計の品質を向上させようとする場合、直接的には設計書をより明確に定義したり、その設計書をレビューしたりする、あるいは、場合によっては設計記述ツールなどを導入するなど様々なアプローチが考えられる。しかし、こうした様々なアプローチを設計担当者が実践に移す場合には、そのための予算をあらかじめ組んでおくことも必要であり、一方で、設計の様式を従来と変えるのであれば、そのあとの実装担当者やテスト担当者と情報の伝え方を事前に調整しておかなければならない。また、設計の良否をレビューなどによって確認する場合には、レビュー関係者や品質保証担当者などとも話をしておく必要がある。もちろん、レビューなどに顧客が立ち会う場合には、顧客の了解などを取り付けておくことが求められる。このように、システムやソフトウェアの品質向上には組織内外の様々な関係者が連携して取り組むことが極めて重要である。

また、一方で、こうした品質向上戦略は、ある日突然実行に移すというわけにはいかない。システム開発に関する様々な技術が氾濫（はんらん）する昨今、それらを導入し、品質向上を目指すという選択肢は数限りなくある。しかしながら、例えば設計支援ツールひとつ取っても、自部門のこれまでの設計の進め方との整合性を確認したり、ツールの利用について習熟したりと、それなりの準備期間が必要になる。その意味では、ソフトウェアの品質向上戦略の実行については、単年度ではなく複数年度を見通す中で、どのように実行に移していくかを検討しておくことが必須であると言える。もちろん、こうした品質向上活動の計画については、一方で、実際の製品開発や製品展開についての中長期計画とリンクさせておくことが求められる。

ESxRシリーズの導入による品質向上

さて本特集号では、こうしたことを踏まえて、SECが発足当初より検討・整備を進めてきた組込みシステム開発向けの開発技術リファレンスESxR※6シリーズの全体を紹介していく。ESxRはSECが組込みシステム開発の開発力強化—とりわけ品質と生産性向上を狙って整備した開発技術リファレンスであり、現在、「ESPR※7：組込みソフトウェア向け開発プロセスガイド」「ESMR※8：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」「ESCR※9：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]」「ESQR※10：組込みソフトウェア開発向け品質作り込みガイド」の4種類の参照ドキュメントを公開しており、本特集号ではこの「ESxRをいかに開発の現場に導入していくか」を主題としている。

これまでSECではESxRシリーズの普及啓蒙に向けて、主催セミナーを延べ50回、一般の展博やセミナーなどでも延べ100回近くの説明の機会を持ってきた。これらのセミナーでは主にESxRの技術的な説明を中心としてきたが、セミナーで寄せられる質問の中には、これらをどのように導入していくか、あるいは、導入を前提にした品質向上戦略の立て方や実行方法などについてのアドバイスを求められることも少なくなかった。このため本特集号では、Part-1として、先に指摘したように、システム開発にかかわる技術者、管理者、そして経営者にどのように捉えていただきたいかを紹介すると共に、Part-2では実際に利用していただく場合を想定し導入を意識したそれぞれの手法解説を試みている。皆さんの組織における品質向上戦略実行の参考にしていただければと考えている。

（IPA/SEC　平山　雅之）

※6 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。
※7 ESPR：Embedded System development Process Reference
※8 ESCR：Embedded System development Coding Reference
※9 ESMR：Embedded System development Management Reference
※10 ESQR：Embedded System development Quality Reference

Part 1-2

プロセス／プロダクトの改善に向けて

1. プロセス／プロダクトとは

ソフトウェアプロセスとは

皆様はプロセスプログラミングという言葉を聞いたことがあるだろうか。「ソフトウェアの開発過程（プロセス）もまたソフトウェアである」という考え方で、1987年に開催されたソフトウェア国際会議（ICSE※1）で米国・マサチューセッツ大学のL.Osterweilによって提唱された。このプロセスプログラミング以降、ソフトウェア・エンジニアリングの中で、ソフトウェアプロセスの話題は極めて重要な技術課題として議論されるようになった。さて、それではこのソフトウェアプロセスとはどのようなものなのだろうか。

SECでいろいろな企業の皆様とお話をしていると、このソフトウェアプロセスが話題に上ることも少なくないが、なかなかどうして、ある企業では工程を指していたり、別の企業ではCMM※2を指していたりと、多様な解釈をされているようである。実はプロセスとは体系的に整理された作業（群）と考えるのが最も適切な考え方である（図1）。

例えば、家を建てることを考えてみよう。家を建てるには、地面を確保し、そこを整地し、土台を造る、柱を立てるなど、様々な作業（プロセス）が必要になる。こうした作業をひとつひとつ丁寧に実施していかないと「家」という製品（プロダクト）はでき上がらないのである。ここで重要なのは、「柱を立てる」という作業（群）の中には、柱を所定の長さに切断したり、水平・垂直を確認したりなど、更に細かな作業が含まれている、すなわち作業が階層的に整理されている点に注目する必要がある。また、もうひとつの重要なことは「地面を整地する前に、土台を造ることはできない」という点、すなわち作業には基本的な順序関係が存在するという点である。このように「ものづくり」において必要な作業の順序関係を含め階層的な体系で整理したものがプロセスと呼ばれるものである。更にもうひとつ、プロセスは順序関係を持つが、時間概念は持たないという点にも注意が必要である。すなわち、家を建てる場合、何月何日までに土台を造り、いつまでに棟上げをするかなどの時間概念は、プロセスをもとにした作業工程の設計という概念で捉えられることを理解しておく必要がある。

ソフトウェアプロセスはなぜ大切か

ここまでで恐らくおわかりいただけたかと思うが、ソフトウェアプロセスとは、ソフトウェアを作る際に必要となる作業を階層的

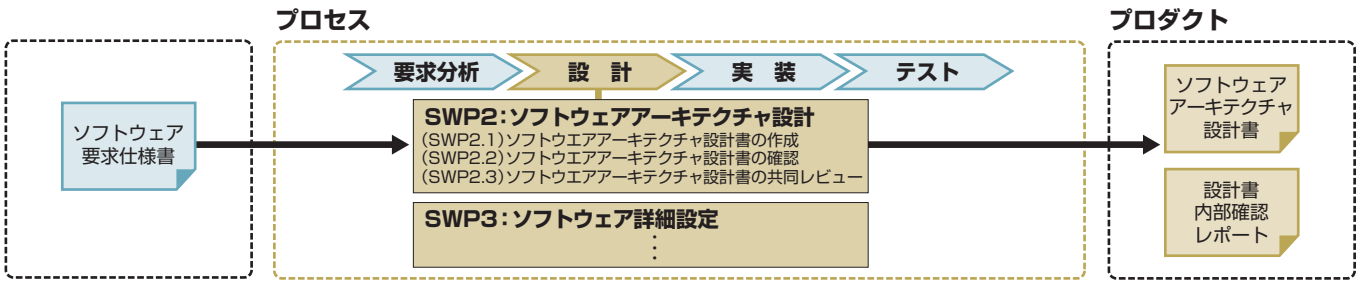


図1 プロセス／プロダクトの関係

※1 ICSE：the International Conference on Software Engineering
※2 CMM：Capability Maturity Model。能力成熟度モデルのこと。その改訂版として開発されたのがCMMI（能力成熟度モデル統合）である。

に整理したものであり、例えば、設計という作業を整理した設計プロセス、テストという作業を整理したテストプロセスといったものが存在することになる。

そして一般的にソフトウェアを作ろうとした場合、やはり、必要な作業（プロセス）をきちんと実施して作る方が、恐らくまっとうなソフトウェアができるであろうことは、家の場合と同じである。つまり、家を作る際に土台を造る作業を手抜きしたりすれば、やはり欠陥住宅ができ上がってしまうのと同じように、ソフトウェアの場合もどこかの作業に漏れがあれば欠陥ソフトウェアができてしまうのである。このように適切なプロダクトを作り上げるためには、適切なプロセスの実行が必須であるという考え方の中で、現在のソフトウェア開発ではソフトウェアプロセスを重視する流れが続いている。

プロセス改善とは

こうしたソフトウェア開発プロセス重視の考え方を背景として、昨今、ソフトウェア開発の現場ではソフトウェア開発プロセス改善の考え方が広まっている。CMMI※3やSPICE※4などがその代表的なもののひとつである。これらに共通しているのは、ある組織を見た場合、そこの組織でソフトウェア開発に関して、本来、やらなければならない作業（プロセス）がどの程度実施されているかを評価（アセスメント）し、問題があれば改善していくという考え方である。「プロセス」と「カイゼン」、5段階の組織成熟度という組織のレベル付けという日本人好みの仕組みで多くの企業が関心を示しているようである。

CMMIを提案している米国・カーネギーメロン大学のサイトには「CMMIを導入した結果、組織の生産性が大幅に向上した」といった事例が紹介されている。もちろん我々がお邪魔しお話を伺った企業の中にも、CMMIなどを利用してソフトウェアプロセス改善を試みてきたところも少なくなく、それなりに組織風土が改善されたところもあるようだ。しかし、その一方では、アセッサ（評価を行う専門家）に依頼してプロセス改善を試みてみたものの、どうもその結果は思わしくないというところも少なくない。その理由のひとつは、道具としてのプロセス改善をどのように開発組織のソフトウェア開発力に結び付けていくかは、それぞれの組織の取り組みに依存する部分が多いという点にあるのかも知れない。こうしたことも踏まえて、今回の特集号では、既存のソフトウェアプロセス改善の話題は少し横に置き、より本質的な問題としてのソフトウェア開発力強化をいかにして進めていくかについて考えてみたい。Part1の本稿以降の記事は、こうした点を踏まえ、ソフトウェア開発の中で経営者、管理者、担当者がどのような姿勢でソフトウェア開発力強化に取り組むべきかを整理してみた。

プロダクトとは

さて一方で、こうした様々な作業（プロセス）を実施すると、その数だけいろいろな成果物が出来上がることになる。例えば、設計というプロセスを実施すれば、当然のことながら、その作業によって設計書というドキュメントが生み出されることになる。これらは開発の過程で生み出されるいわゆる中間成果物であるが、これらを総称してプロダクトと呼ぶことが多い。

当然のことながら、これらのプロダクトに誤りがあったり、抜けがあったりすると、それを入力とした次の作業でも誤りが発生してしまうという誤りの連鎖が発生し、最終的なソフトウェアそのものの不具合に直結してしまう。このため中間段階での成果物の誤りはあってはならず、同時に中間成果物の品質そのものも厳しく問われなければならないのである。

プロダクト自身のカラクリを考える

プロダクトと言えば、3年ほど前にSECの仕事として重要インフラ情報システムに関する信頼性ガイドラインを経済産業省の方と共に執筆させていただいた。このガイドライン策定の中で、話題となったのは、いかにしてプロダクトとしてのシステム信頼性を向上させるかという議論だった。つまり、どんなに開発プロセスを強調し、「きちんと設計すること、異常処理を考えた設計をすること」とお願いしたとしても、それだけでは十分ではないのではないかという点だった。

例えば航空機制御などの非常に高い信頼性が求められる分野では、信頼性を高めるために系の二重化などの設計定石がふんだんに取り入れられている。つまり、具体的にシステムの信頼性を上げたいと思えば、例えば二重化などのカラクリをシステムやソフトウェアの構造面に作りこんでいく必要がある。ソフトウェア開発とよく対比される建築の世界では、建物の構造などについて、例えば、高い耐震性を持たせるのであれば、どのような材料をどのように組み合わせ、どのような構造にすればよいかといった技術情報がかなり整理されている。こうした分野に比べると、残念ながらソフトウェアについてはプロダクトとしての中身や構造に関する議論がやや低調であるという印象を受ける。ソフトウェア開発力という意味では、こうしたプロダクト自身の信頼性、品質を高めるための仕組みやメカニズムも時には検討しておく必要があるのかも知れない。

※3 CMMI：Capability Maturity Model Integration。能力成熟度モデル統合とも呼ばれ、開発プロセスを5段階の成熟度で定義している。
※4 SPICE：Software Process Improvement and Capability dEtermination。ISO/IEC 15504の愛称。ソフトウェア開発を中心とした工程評価のフレームワークである。

2.ソフトウェア開発力強化を考える

ソフトウェア開発力とは

ここまでソフトウェアのプロセスやプロダクトについて考えてみたが、次にこれらとソフトウェア開発力強化の関係についてもう少し考えてみよう。ソフトウェア開発力とは、より高い品質のソフトウェアを効率的に生み出す力であり、ある意味、昨今の情報機器、情報処理ビジネスの根幹を支える力である。既に述べたようにソフトウェアを開発する上では、ソフトウェアの開発プロセスとその結果生み出されるプロダクトの両面を考えていくことが求められるが、とくにソフトウェア開発力を伸ばし強化するためには、この両方が車の両輪として、リンクし回っていくことが求められるのは言うまでもない(図2)。

図2 ソフトウェア開発力とは

例えば、発電所の制御ソフトウェアなどのように非常に高い信頼性が求められるソフトウェアなどでは、開発プロセスとしては「システムに求められる信頼性を考慮してシステムの内部構造を決める」という作業が求められる。

実際の設計者はこの作業の中で、例えば「そのシステムに入力される情報を2系統の処理ルーチンに振り分けてダブルチェックする」などの具体的なメカニズムを考え導入することになる。実際にはこの作業の結果をもとに処理を二重化した内部構造を持つソフトウェアの設計が決まり、その結果が設計ドキュメントに記載され、そうしたソフトウェアが実装されることになる。 ここまでくると、ソフトウェアプロセスが実行された結果、プロダクトとしての設計書やソフトウェアが生み出されることになるが、この場合、プロダクトの確認をすることで正しいプロセスが実行されたかどうかを確認することも可能である。このように、プロセスとプロダクトはある意味、独立ではなく、それぞれが密接に関係することで、ソフトウェア開発力は実現されていくと考えても差し支えない。

進化する周辺技術

次にシステムを構成する要素という視点から、ハードウェアに目を移してみよう。ほとんどのソフトウェアはハードウェアとインテグレートされてひとつのシステムを構成しており、ソフトウェア開発力を考える上では周辺のハードウェア技術も十分に考慮しておかなければならない。

例えば最近の組込み分野をのぞいてみると、高度なメディア処理などを行う分野では、ひとつのユニットに複数のプロセッサコアを搭載したマルチコアが脚光を浴びている。このように近年、システムを構成する要素技術は非常に速いスピードで進化しており、ソフトウェア開発力という点からはこれらも考慮しておく必要がある。例えばマルチコアなどの場合、どのプロセッサコアでどのような仕事をさせるかといった、システム機能の分割やそれに伴う機能連携、あるいは、必要なデータの保持方法などが極めて重要になると言われている。これらを考えるとハードウェアの方式としてマルチコアという方式を採用した場合、そこに搭載するソフトウェアの設計は従来のシングルコアの時代の設計の考え方、進め方とは若干異なった考え方やアプローチが必要になるのは想像に難くない。

このようにシステムの要素技術の進化は、一方で対応するソフトウェア開発技術にも大きな影響を与えると考えておいた方がよさそうである。このためソフトウェア開発力強化を達成しようとするのであれば、現在のシステム要素技術や自社の製品をベースにするだけでなく、この先、数年単位での技術の進化も視野に入れながら、検討を進めておいた方がよいということになる。

ソフトウェア開発力と技術者スキル

ソフトウェア開発力を考える上でもうひとつ重要な要素はいわゆる技術者のスキルの問題である。ソフトウェア開発はその開発に携わる技術者のスキルによるところが極めて大きいと言われている。すなわち、ソフトウェア開発プロセスを実施するためには、様々な開発技術やツールなどを使いこなさなければならないが、そのためにはこれらの「道具」を使いこなせるだけの力量が技術者に備わっていなければならないのは言うまでもない。

例えば、ソフトウェア開発の中で欠くことのできない作業のひとつである設計という作業を考えてみよう。ここでは具体例として「単純に1,000個の数字があったとして、それを大きい順に

並び替える」という処理をプログラムとして作成する場合を考えてみる。これは極めて初歩的な問題であるソーティング処理という問題であるが、ソーティング処理のアルゴリズムには単なる大小比較を繰り返して並び替えをしていく単純ソートという方式から、並び替えの速度を重視したクイックソートという方式まで様々な方式が考えられる。ここで設計技術者の「力量」としては、これらの様々な方式の中から、決められた制約条件（例えば処理時間やコードサイズ）なども考慮して最適な方式を採用し、ソフトウェア機能として実装できるという能力が、求められることになる。

こうした技術者の力量は、前述のソフトウェア開発プロセスを実践に移し、期待されるプロダクトを生み出すために欠くことのできない要素である。そして、ソフトウェア開発の個々の作業・プロセスごとに必要となる技術者のスキルやその力量は異なってくると考えられ、開発に入る段階でそうした必要なスキルや力量を持った技術者をチームとしていかに編成するかが、ソフトウェア開発力を最大限に活かすための必要条件となる。

3. ソフトウェア開発力強化とESxR

ESxRの位置付け

ここまでソフトウェアのプロセス／プロダクトを中心にソフトウェア開発について考えてきたが、最後にソフトウェア開発力強化のための道具としてのESxR※5の導入方法などについて考えてみたい。SECでは、実際のソフトウェア開発力強化は、2段階で進められるべきものと考えている。すなわち、まず第1段では開発力の基礎となる開発の基本的な体力を向上させ、そのあと、第2段としてより先進的な技術導入によってより高度な開発力を身に付けていくという考え方である（図3）。この考え方は、SEC発足時に多くの企業の方々と意見交換する中で、昨今話題になっている新しい技術導入を図ろうとしても、それ以前の開発の手順やプロセスといったところが十分に機能していない中で、なかなか効果的な技術導入が難しいという意見を反映した考え方である。つまり、第1段ではまず、こうした障害を取り除くために開発の基礎となる体力部分を強化しようという考え方に立っている。これに対し昨今話題の技術の中には、こうした基礎力を前提としてさらなる開発力強化を目指すための技術やその指針となる考え方が含まれているものも多い。

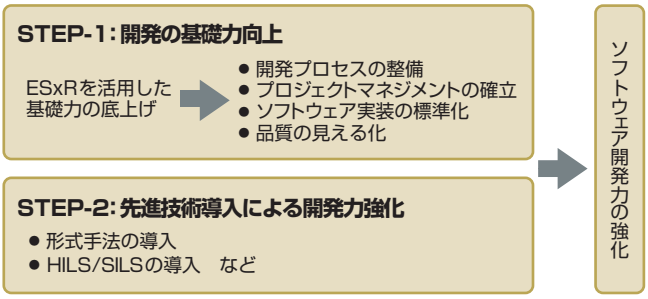


図3　ESxRの位置付けの図

今回の特集で紹介するESxRシリーズは、この意味ではまず第1段の開発の基礎体力を強化するという点に狙いを絞って整備してある。このため、既にこうした基礎的な開発体力が備わっている組織ではESxRシリーズは若干物足りない、あるいは当たり前のことを紹介しているという印象を持たれるかも知れない。しかしそうした組織でも現状を振り返り、さらなる開発力強化を目指す際の参考として利用いただくこともできると考えている。また逆にこうした基礎体力をまだ整備できていないという組織にとっては、比較的、参考となる情報が多く含まれているのではないかと考えている。

ESxRの導入について

ESxRシリーズはソフトウェア開発力強化に向けたあくまでも参考情報（Reference）として整備したものであり、ESxRに記載されたすべてを遵守（じゅんしゅ）しなければいけないわけではない。

ESxRシリーズはそれぞれの読者の立場や状況に応じて、必要な情報を選んで参考にしていただければよいと考えている。ESxRに記載されている表面的な部分をなぞるだけではなく、ぜひ、本特集 Part1 に紹介したソフトウェア開発力強化への取り組みという視点からも読み込んでいただきたい。ESxRシリーズの編さんにあたっては担当研究員のみだけでなく、多くの産業界の皆様にご協力いただく形の委員会で非常に濃密な議論を経て策定したものである。結果として我が国のソフトウェア開発におけるベストプラクティスがふんだんに盛り込まれていると同時にソフトウェア開発力強化への思いも盛り込まれている。こうした産業界の有効な知見を国として集め整理し活用していくことも、また、我が国全体としての開発力強化につながっていくと考えている。ぜひ、多くの皆様にESxRシリーズを参考にしていただければ幸いである。

（IPA/SEC　平山　雅之）

※5 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。

Part 1-3

経営者からスタートするソフトウェア開発力強化

1. 開発力強化のスタートポイント

見えないソフトウェア開発は経営上の不安要因

SECという立場上、システムの開発に携わっていらっしゃる経営者の方とお話しする機会も少なくない。開発技術面、開発管理面や技術経営面など様々なお話をさせていただく中で、皆さんよく話題にされる事項のひとつは、「ソフトウェア開発はなかなか見えない」そして「その見えないソフトウェア開発にどれだけのコストを投下したらよいかの判断が難しい」というものだ。

確かにソフトウェアは物理的な形状を持っておらず、従って体積や重量といった特性で捉えることが大変に難しい。畑で育てるトマトやリンゴなどであれば、その生育状況は目視で確認でき、適切なタイミングで生育のための対策を実施し、その効果を確実に把握できる。しかし、物理的な形状や特性を持たないソフトウェアの場合、その品質などに対してどれだけのコストを投下し、どのような対策を実行すれば、どの程度の品質の製品が実現できるかという評価が極めて難しいのは言うまでもない。かくしてソフトウェア開発に携わる多くの経営者が、目に見えないソフトウェアやその開発を相手に、どこまで、どのようにすればよいかを迷うという図式ができ上がってしまい、経営上の一大不安要因になっている場合が少なくないようである。実際、経済産業省とSECが毎年実施している組込み産業に関する実態調査のデータの中から経営者に対して行った調査結果の一部を示した図1を見てもわかるように、多くの企業経営者が「設計品質の向上」を課題認識の1番目に挙げており、経営者の悩みはなかなか解決しそうもない。

ソフトウェア開発は本当に見えないか

経営者のソフトウェアに対する不安要因のひとつが開発やその品質が見えないことであると指摘したが、それでは本当にソフトウェアの開発や品質は見えないのだろうか。

通常、ソフトウェア・システムを開発するためには、現場の担当技術者、プロジェクトマネージャ、ライン管理者、そして経営者と組織内の様々な役割の方々がチームとなって取り組んでいる。現在の複雑なソフトウェアから構成されるソフトウェア・システムの多くは、こうしたチームの中で開発されており、恐らく、このチーム内では設計書やソースコード、そして開発の状況などをまとめた進ちょく状況などが、何らかの形で共有されなければ実際問題として開発はままならない状況にある。これは言い換えると、多少の齟齬(そご)はあるにせよ、開発の現場では、様々な情報が何らかの形で担当者や現場のマネージャなどの間で共有されてチームとして機能していることを意味している。

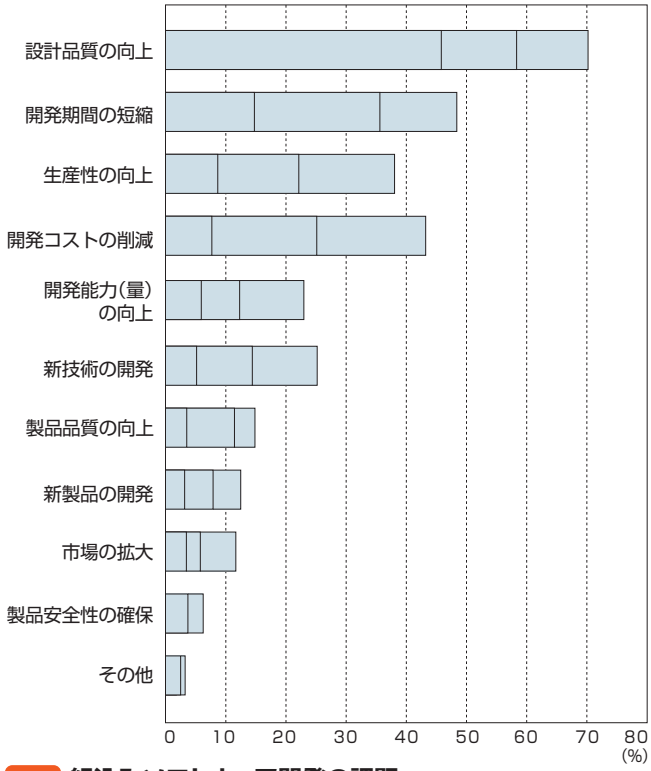


図1 組込みソフトウェア開発の課題
(出典「2009年版組込みソフトウェア産業実態調査報告書」経営者及び事業責任者向け調査)

さて、それではなぜ、こうした現場の情報が経営者に伝わらず、結果として経営者からソフトウェアは見えにくくなってしまうのだろうか。この答えは、ある意味、非常に簡単である。すなわち、実際の開発の現場内にある様々な情報を、開発サイドが見えるように、そして経営サイドが見るように努力しているか否かということに尽きるのかも知れない。この記事をお読みいただいている皆さんは、最近、実際にソフトウェアを開発している開発現場に足を踏み入れ、開発の状況や現場が抱える課題点などについて、担当者や現場のマネージャたちと虚心担懐（きょしんたんかい）に意見交換する時間をどれくらい取っただろうか。ソフトウェア開発に関する状況を整理した報告書や図表の形でまとまっていなくとも、現場の技術者やマネージャと意見交換をすれば、おのずと自組織の抱える課題やその解決の方向性が見いだせるはずである。

しかし、実際の多くの企業では、こうした機会が十分に確保されず、経営者と現場サイドの間に大きな情報伝達や情報咀嚼（そしゃく）の溝がある場合が少なくないようである。もちろん、現場の開発担当者やマネージャが、それぞれの状況をわかるように経営者に伝える努力も必要であり、その一方では、経営者が現場を理解する努力も求められると言える。この双方の努力がかみ合ったとき、初めて、ソフトウェア開発は見えるようになり、その改善に向けたスタートを切ることができるのかも知れない。その意味で、ソフトウェア開発力強化の原点は「コミュニケーション」であることを、経営者の方々にも再認識していただければと思う。

2. 品質ビジョンを立て推進する

経営者の役割は旗を立てること

さて上記のような形でソフトウェア開発力の改善や強化に向けたスタートポイントに立ったところで経営者としては次に何をすべきなのだろうか。ここでひとつ参考になる事例がある。

1990年代後半、不振に陥った我が国を代表する自動車メーカのひとつが、海外から強力な指導者を迎え入れ、見事に復活を遂げた例がある。その当時、その会社を任された指導者は、大胆な車種絞り込みや調達から開発に至るまでのムダの排除、社内コミュニケーションの充実などをリバイバルパッケージとして全社の旗印にして、社内の意識改革を推進していった。この例は我が国を代表する巨大企業の一例にすぎないかも知れないが、ここでこの指導者が採った施策、すなわち、全社員の開発力強化や改革・改善に向けた意思を統一し、方向性を示していくという考え方は、企業規模の大小にかかわらず極めて重要な第一歩であると考えてもよさそうである。

ソフトウェア・システム開発では、現場の技術者、マネージャや品質スタッフ、そして営業スタッフに至るまで非常に多くのメンバが関係する。上記の例からもわかるように、多くのメンバの意識をソフトウェア開発力強化というひとつの方向性に向けていくためには、組織全体を引っ張る経営者自らが、その方向性をより具体的に示す旗、すなわちビジョンを明確に示すことが極めて重要であると言えるのではないだろうか。

品質ビジョンの実行に向けて

それでは、次にこのようにして策定されたソフトウェア開発に関する開発力強化や品質改善のためのビジョンをどのように実行に移していけばよいのだろうか。

企業レベルで立案した開発力強化のためのビジョンとは例えば、「向こう3年間でソフトウェアバグを半減する」といったものや「ソフトウェアの生産性を2倍にする」といったもののように、より具体的な達成目標が明示されていることが重要であるが、その一方で、こうしたビジョンをどのような施策によって達成に導いていくかを初期の段階で十分に検討しておく必要がある。このためには経営者はビジョンを立てるだけではなく、実際の開発現場にそのビジョンをどのように落とし込んでいくかという視点から、主体性を持って、実際の開発担当者やマネージャたちと議論し、方向性を探っていくことが求められる。この局面においてもその成功の鍵を握るのは、経営者と開発現場の間のコミュニケーションであることは間違いない。ビジョン実行に際して、経営サイドや開発サイドの双方が、円滑なコミュニケーションを通じて、それぞれの考えや問題意識、解決の方向性について、率直に語り合うといった努力を惜しまないことが、ある意味で開発力強化に向けたビジョン実行の大きなポイントになると言える。

組織の持続的な成長を目指して

ところで実際にソフトウェア開発力強化を推進する場合、どれくらいの時間がかかるのであろうか。先に例に引いた自動車メーカの場合、ビジョン策定からそのビジョンが実効性を持った施策に展開され、その効果が実際に経営上に現れるようになるまでには数年を要したと報告されている。

こうした例を見るまでもなく、ソフトウェア開発力強化の場合にも、ある程度の時間が必要であることは同様である。その点を考慮すると、経営者が策定した品質ビジョンを組織内に徹底し、それに向けた具体的な施策を展開するためには、単年度ではなく複数年度を視野に入れた推進を織り込んで全体計画を用意していく必要があるのではないだろうか。

しかし、その一方、経営的な視点からすると、明日にでも自社のソフトウェア開発力を改善し向上したいと思うのは当然のことであり、現実的な考え方としては、ソフトウェア開発力強化に向けた時間スパンをいかに実際の企業経営の時間スパンに合わせていくかという点を慎重に検討しておくことが望まれる。

段階的なアプローチを考える

次に、開発力強化に向けて時間軸をどのように合わせていくかを考えてみたい。例えば、ソフトウェア開発にかかわる組織力を強化する考え方のひとつにSPI※1という方法がある。この考え方の代表的なもののひとつが米国・カーネギーメロン大学から提案されたCMMI※2である。このCMMIの中ではソフトウェア開発組織の能力レベルをレベル1からレベル5までの5段階に分類し、各レベルではどのようなことができていなければいけないかを明示している。

このCMMIにおける組織レベルの分け方やその利用方法については様々な意見があるが、ソフトウェア開発力強化を考える場合、それに向けた施策を段階的に捉え実施していくという方法は大変に参考になり、有効なアプローチのひとつであることは間違いない。例えば「自社のソフトウェア不具合を50%削減する」という品質ビジョンを実践することを考えると、テストを強化して下流フェーズでの不具合検出率を上げ不具合を水際で除去する考え方もあれば、開発過程での設計などの段階で集中的にレビューを実施して不具合混入の機会を減少させる考え方、あるいは実際の設計段階でできるだけ不具合が入らないように厳密に設計定義と表記を行うなど様々な施策を考えることができる。

しかしながら、実際の手法導入を考えると、それぞれの手法には長所／短所が存在し、当然のことながら、導入に要するコストや期間、求められるスキルレベルなども異なっている。もちろんこれらの施策の効果という側面についても、効果は限定的であるが比較的短期に実行に移せる施策もあれば、比較的実行に移すのに時間を要する施策もある。このため、実際のソフトウェア開発力強化を目指す場合には、それぞれの施策の効果と実効期間などを十分吟味し、優先度を付けて、組織の中長期の成長戦略とリンクさせて考え、実行することが求められる。

開発力強化の道具としてのESxR

SECではソフトウェア開発に携わる企業のソフトウェア開発力強化をあと押しする道具としてESxR※3シリーズを整備している。ESxRシリーズの詳細は本特集号のPart2以降を参考にしていただければと思うが、このESxRシリーズを活用して開発力強化を試みる場合にも、段階的な手法導入という基本原則は同様にあてはまる。ESxRシリーズでは、いわゆるプログラムの

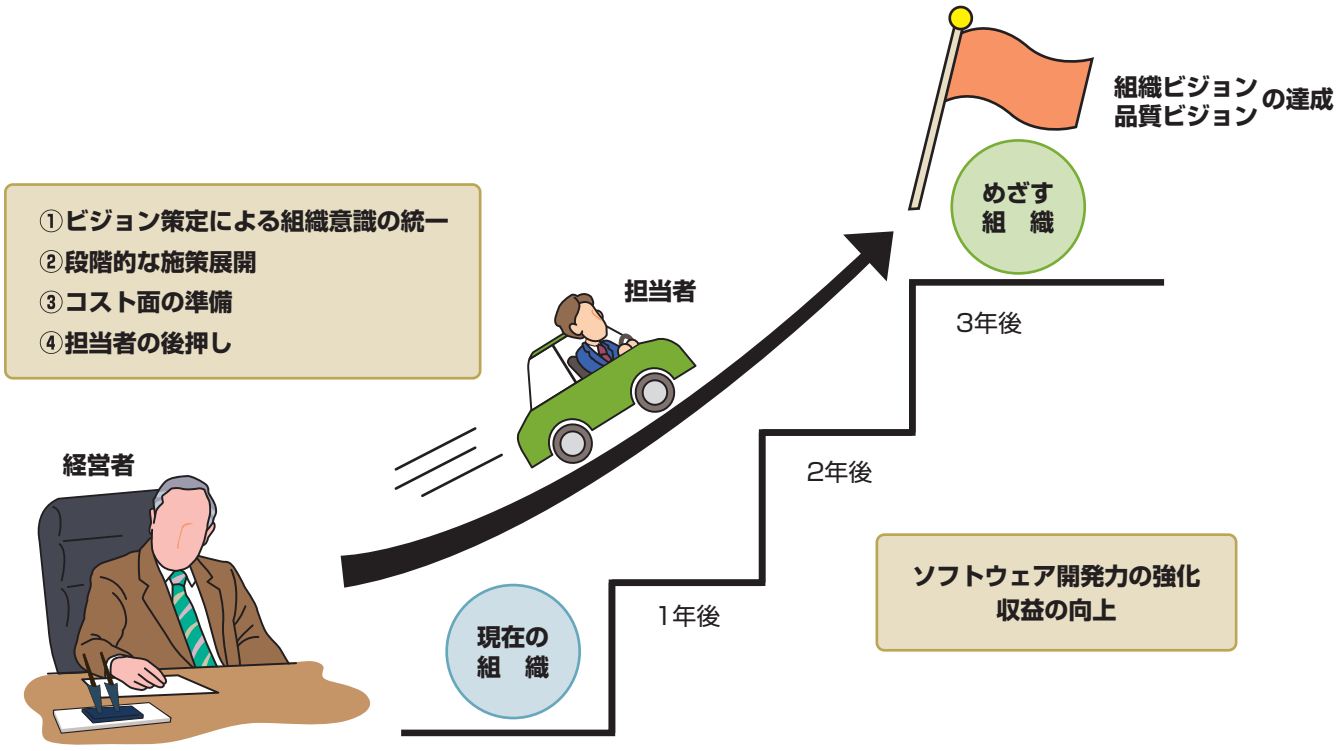


図2 ソフトウェア開発力強化に向けた経営者の役割

※1 SPI：Software Process Improvement。ソフトウェアプロセス改善のこと。
※2 CMMI：Capability Maturity Model Integration。能力成熟度モデル統合とも呼ばれ、開発プロセスを5段階の成熟度で定義している。
※3 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。
※4 ESCR：Embedded System development Coding Reference。2006年5月発行。

直接的な品質の向上を実現する「ESCR※4：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]」、開発のベースとなる開発プロセスの整備を狙った「ESPR※5：組込みソフトウェア向け開発プロセスガイド」、組織的な開発のベースラインとなるプロジェクト計画書策定をあと押しする「ESMR※6：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」や開発過程での品質の定量コントロールを実現する「ESQR※7：組込みソフトウェア開発向け品質作り込みガイド」の4冊が準備されている。

しかしこれらを品質ビジョンの実現のための道具として捉えた場合、品質定量化やプロジェクト計画書は、当然のことながら組織としての開発プロセスが整備されていることが基本要件となるため、組織として導入する場合にはおのずと導入の順番は決まってしまう。一方、プログラムの品質という点に関しては、こうした流れとは独立して現場の技術者の意識次第で個別に導入しコーディングルールなどを運用することも可能であり、比較的、短期間で即効性のある施策展開の道具として利用することも可能となっている。

3. 開発力強化のあと押し

経営者による開発力強化のあと押しは必須

ここまでソフトウェアの開発力強化にどのように取り組んでいくべきかについて紹介してきた。しかし開発力強化というテーマを推進する上では、実際には様々なハードルが待ち受けているのもまた事実であり、経営者による開発力強化のあと押しは必須である(図2)。

コスト面のハードルを越える

実際にSECの立場で様々な企業での開発力強化への取り組みをヒアリングする中でよく指摘される事項のひとつは、コスト面の悩みである。実際にソフトウェア開発力を強化するためには品質ビジョンを展開し実行に移す段階で、様々な手法やツール導入が必要となる局面が少なくない。こうした場合、多かれ少なかれそのための費用が発生することになる。経営層から見るとソフトウェア開発はその多くを技術者の能力に依存すると思われがちであるが、実は開発力強化のためには計画的な設備投資が必要であることを理解しておかなければならない。昨今の企業経営の中では、ムダを排除したスリムな経営を追求する方向にあるが、時には多少の余裕をもってソフトウェア開発力強化に向けた先行投資として、そのためのコストを準備しておくことが望ましいと言えるかも知れない。

現場をエンカレッジする

また、品質ビジョンの実現に向けたもうひとつの大きなハードルはソフトウェア開発技術者の持つ特性そのものなのかも知れない。我が国のソフトウェア開発の現場を見ると、大小様々な課題が見え隠れしているものの、最終的には製品としてのシステムが開発され市場にリリースできているという現実がある。これを現場の技術者の目から見ると、製品リリースという基本的なビジネスゴールは達成できているということになり、現在の自分たちの開発のやり方でも何とかなるという論理が展開されるケースが少なくない。こうした場合、ソフトウェア開発力強化というビジョンの中でこれまでの自分たちの開発の進め方や手法を変えることについて強い抵抗が発生してしまうことになる。その点において現場の開発技術者は非常に保守的な側面を持っている場合が多いようだ。この現場サイドの保守的な側面を打破し、新たな品質ビジョンや開発力強化に向けての行動を促すためには、実は経営層による強力なサポートとあと押しが必要である。

少し古い例になるが、20年ほど前のメーカの製造現場では自分たちの作業改善を目的としたQC※8サークル活動が盛んに行われていた。当時、組織だった企業では、このQCサークルの中でとくに優秀なものを会社として表彰したり、その活動成果を評価したりするといった形で、全社を挙げて盛り上げあと押ししていた。実際、こうした企業内での評価の仕組みとかみ合う形で、各QCサークルはそれぞれの作業改善を目標に、非常に活発なカイゼン活動が展開されていた。こうした事例を振り返ると、経営層も含めて全社一丸となってどのように開発力強化に向けた活動を盛り上げていくかという点もあらためて考える必要があるのかも知れない。

4. おわりに

さてここまで、思い付くままにソフトウェア開発力を向上させるというテーマについて、どのように考え、何をすべきかを紹介してきた。もちろん各企業で取り扱うソフトウェア・システムは様々であり、組織的な規模や特性は大きく異なっている。それらを十分に考慮した上で、それぞれの企業に最適な開発力強化を推進していただきたい。また、そのための手法、技術などの道具立ては昨今、様々なものが提案されており、これらをどのように活用して自社の品質ビジョンを実現し開発力強化を進めていくかについても、ご検討いただけると良いと考えている。

(IPA/SEC　平山　雅之)

※5 ESPR:Embedded System development Process Reference。2006年10月初版発行、2007年11月改訂。
※6 ESMR:Embedded System development Management Reference。2006年11月発行。
※7 ESQR:Embedded System development Quality Reference。2008年12月発行。
※8 QC:Quality Control。品質管理のこと。

Part 1-4

現場の開発管理者が主導する改善

1. ソフトウェア開発における開発管理者の役割

開発管理者の役割

日夜続くソフトウェア開発の中で、開発管理者やプロジェクトマネージャの役割はどのようなものだろうか。ソフトウェアの企画から設計、実装、そしてテストに至る一連の作業が目まぐるしく続く中で、獅子奮迅の働きをされている開発管理者の方が多いのではないかと思う。

開発管理者は企業の経営層と開発の実務担当者の間に立ち、その経営方針を咀嚼(そしゃく)して実務レベルの作業に展開し、ビジネスとしてのソフトウェア開発プロジェクトを成功裏に導くための主役である。しかし多くの開発管理者はこの「開発プロジェクトを成功裏に導く」という重要なミッションを持つ一方で、結果的に目の前のプロジェクトに対する個別対応を重視する形になりがちのようである。ここでは目の前のプロジェクトから少し離れて、組織としてのソフトウェア開発力を強化するという視点から、開発管理者の果たす役割を考えてみよう。

ソフトウェア開発力とは

近年のソフトウェア・システムは実現する機能やサービスが多様化し、機能規模的にも非常に大きなものとなってきている。そして、これらのシステムにおけるソフトウェアの位置付けも相対的に非常に重要なものとなってきており、多くのシステムでその価値を決定する主要な要素のひとつとなっている。このため、システム内で用いられるソフトウェアをいかに効率的に作るか、そしていかに高品質なソフトウェアを提供するかがビジネス上の重要な要素となっている。ソフトウェア開発力とは、高品質なソフトウェアを効率的に開発するための組織としての能力であると定義することができる。

さてここでソフトウェア開発力をもう少し具体的に考えるために、実際のソフトウェア作りを考えてみよう。実際のソフトウェア作りに求められる要素としては、

- ソフトウェア開発に関与する個々の技術者、開発管理者の持つ技術やスキル
- 個々の組織やプロジェクトが持つ、それぞれの開発作業の仕組み(プロセスや開発管理)
- 上記の人と仕組みを有機的につなげるための組織風土

の三つの要素があり、これらがうまく整合し連携がとれた場合に、その組織におけるソフトウェア開発力が最も力を発揮するのではないだろうか。

少し簡単な例を想像してみる。ある組織に百戦錬磨のスーパープログラマが舞い降りたとしてみよう。しかし、その組織内でコーディング規約が整備されていなかったり、コードレビューが実施されていなかったりしたらどうなるだろうか。その担当者が作ったプログラムは確かにエレガントなものに仕上がるかも知れない。だがチームのほかのメンバが作ったプログラムとのインタフェースが確認できない。あるいは、スーパープログラマとその他の技術者のレベルが違いすぎて会話が成り立たないなどの問題が起こり、プログラムに不整合が出る恐れがある。このように、ソフトウェア開発力とは、設計・実装やテストなどの個別の技術だけではなく、上記の三つの側面なども考慮したトータルな組織力ということになり、それを強化するためには現場を客観的に見ることができる開発管理者が必要ということになるのではないだろうか。以下ではこの三つの側面について、開発管理者がどのように考え、どのように施策を実行をしていくべきかについて考えてみたい。

2. ソフトウェア開発力を強化するための技術導入

ソフトウェア開発のための技術は多種多様

ソフトウェア開発を効率的に進めるためには、様々な手法や技術を導入したり、そうした技術を使いこなしたりすることができるように技術者のスキルを高めていくことが必要になる。まず、技術の導入という点に関して考えてみると、ソフトウェア

開発のための手法や技術は、開発の最上流である要求分析から最終段階のテストに至るまで様々なものが用意されている。例えば、ソフトウェアの設計手法ひとつを取っても、構造化設計、オブジェクト指向設計、データ中心設計等、実に様々な手法があり、更に最近はアスペクト指向※1や設計モデリング、あるいはADL※2に至るまでまさに百花繚乱の観すらある。

どの技術を導入するかを考える

こうした中で、開発管理者に任される大きな役割のひとつは、これらの様々な手法・技術の中から自分たちの組織やプロジェクトに導入する手法・技術を決めることである。SECでは後述するESxR※3シリーズの検討や普及のために、様々な企業の方々とお話をする機会が少なくないが、現場の管理者の方によく質問される事項のひとつは、「ソフトウェアの品質を向上させるための良い手法を教えてほしい」というものである。要は、自社の開発力を強化するという点においていろいろな手法や技術がありすぎて、一体どれを導入したらよいかがわからないということのようだ。さてこのような場合、どのようにしたらよいのだろうか。これらの手法・技術については、いろいろな書籍が発行されている場合もあれば、各種のセミナーなどで技術紹介や事例紹介がされている場合も少なくない。まずは、これらを利用してそれぞれの手法・技術がどのようなものかを調べてみるのがよいかも知れない。もちろんこの場合、それぞれの手法・技術の利用について

- **利用についてどのような前提条件や技術的制約があるか**
- **手法・技術を利用した場合に結果としてどのような効果が得られるか**
- **それらを使いこなすためにはどのような技術スキルが求められるか**
- **導入のための期間・コストはどれくらい考えられるか**

などをきちんと整理して確認しておくことは言うまでもない。もちろん、これらは部下に指示をして整理させるのもひとつの方法であるが、ときには開発管理者自らも調べ、確認することも重要である。こうした確認を怠ったままで、安易に手法・技術の導入を進めても、期待通りの結果は得られない場合が多いようである。

技術導入のステップ

さてそれでは、具体的に手法や技術を導入する場合、どのようなステップを踏めばよいのだろうか。ここではSECで整備している「ESCR※4：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]」（以下、ESCR）の場合を考えてみよう。このガイドは組織内のコーディング規約を整備し、ソースコードの品質を向上することを狙ったものである。例えば自部門にコーディング規約が整備されていない組織の場合を想定してみる。

【Step 1】

開発上の課題点、解決すべき課題点を明確に把握する

まず、手始めとして、開発管理者が音頭を取って開発の中心となるメンバを集め、自分たちのソースコードの書き方が統一されているか、そして、人によって書き方がまちまちではないかなどをチェックする作業を指示してみてはどうだろうか。恐らくコーディング規約が備わっていない組織では、開発者によってコードの書き方はまちまちであり、コードの品質も人によってバラつきがあることを、発見できるだろう。

【Step 2】

導入する手法について理解し、導入方法を考えてみる

次のアクションとしては、開発管理者と開発のキーパーソンで実際にESCRの勉強会を開催し、自分たちの組織に根付かせるための方策を議論してみよう。恐らく自分たちの組織内でESCRを利用する場合には、これをベースに自組織のコーディング規約を策定する作業、そして、それを開発者に周知徹底させるための教育が必要であることが話題になるのではないかと思う。

【Step 3】

成功体験を用意する

さてその次のステップであるが、こうした新しい手法を導入し成功に導くための秘訣（ひけつ）のひとつは成功体験であり、チームのモチベーションを上げるためにも重要である。ここでは、できれば少人数・短期間のモデルプロジェクトなどを決めて、実際にコーディング規約を用意し、実際に関係者に教育し、開発の中で使わせるという工夫をしてみるとよいかも知れない。

SECでESxR実証トライアルとして、協力いただいている企業の多くは、こうした成功体験を上手に利用しているようである。

【Step 4】

成功体験をもとに組織内普及を加速する

このような成功体験を踏まえ、組織内に展開する場合のメリットや展開した場合の課題点、工夫すべき点などを整理し、それを組織内の全メンバに紹介してみよう。ある組織において、隣のプロジェクトでうまくいった試みは、その例を紹介することで比較的導入のハードルや抵抗感を低く抑えることができるものである。

※1 アスペクト指向プログラミング：オブジェクト指向プログラミングの問題点を補うために考え出されたプログラミング手法。
※2 Architecture Description Language。ソフトウェア・アーキテクチャやシステム・アーキテクチャを記述するための言語。
※3 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。
※4 ESCR：Embedded System development Coding Reference

もちろん、ここで紹介した手法導入のステップはひとつの考え方で、これ以外にも様々な導入ステップは考えられる。しかし、どのような導入ステップを採るにせよ、そのポイントのひとつは、実際にそうした手法を実践するそれぞれの技術者をいかに納得させるかであり、そのためには開発管理者がイニシアチブを取って進めていくことが常に求められる。また、こうした導入ステップの中で、もうひとつのポイントは、技術導入とセットにした技術者スキルの向上であり、新しい手法導入を試みるためには、技術者教育の仕組みも併せて考えておいた方がよいかも知れない。

3. 開発作業の仕組みの構築

開発作業の仕組みとは

さて、開発管理者として気を配らなければならない事項として、開発組織内の開発作業の仕組み構築を挙げることができる。開発作業の仕組みというといまひとつピンとこないかも知れないが、要は、どのような開発作業をどのような順序で進めていくかといういわゆる開発プロセスを明確にすることと、そのプロセスが開発の中で確実に実行できるようにフォローしていく開発管理がその中心になる。

開発作業の仕組みは管理者が責任を持つ

プロセスや開発管理（マネジメント）というと、「既に専門家にお願いしてCMMI※5をやっています」あるいは、「PMO※6を組織してマネジメントを推進しています」という方もいらっしゃるのではないだろうか。これまでの経験から、少しだけ過激なことを言わせていただくと、「こうした活動をやっています」と主張される開発管理者の方のお話の何割かはあまり当てにはならないと思っている。

CMMIは自組織で実践すべき開発作業（プロセス）についてアセッサと呼ばれる専門家が評価し、課題点を指摘し、それを受けて改善活動を進めていくというスタイルが多いようである。確かに第三者が見た方が組織内のプロセスを客観視できるし、指摘もしやすいという側面はあるかも知れない。しかし、自組織の開発作業に目を光らせ、その課題を見抜いて是正していくのは実はほかならぬ開発管理者ではないだろうか。また、PMOはプロジェクトマネジメント技術を身に付けたマネジメントの専門家集団により、開発マネジメント支援を担わせるというものであるが、やはり自組織における開発がどのように進みどのような課題があるか、それをどのように解決していくかを考える中心はあくまでも開発管理者ではないだろうか。もちろん開発管理者が任される業務は多岐にわたるため、その一部を補助するスタッフは必要かも知れない。しかしそういった話とは別に、第三者に開発管理者が本来担うべき業務を委ねてしまうことは本末転倒な印象を受ける。CMMIやPMOが悪いと言っているのではなく、そうした第三者依存のフレームに胸を張る開発管理者はある意味、開発管理者としての責任を放棄してしまっているのではないかとも思うのだが、いかがだろうか。

開発管理の仕組み作り

開発プロセスにせよ、開発管理にせよ、どちらも開発作業の根幹を成す重要な要素である。このためこれらの要素は、ある日、突然に見直したり新しい仕組みに変えたりということは極めて難しい。最近のソフトウェア開発のほとんどはプロジェクト形式で進められるものが多いが、開発プロセス、開発管理の仕組みを変えるひとつのタイミングは、プロジェクトが動き出す前の準備段階がよいかもしれない。

もちろん、個別の技術導入の場合と同様に、こうした開発の仕組みを変える場合にも多少ステップを考えておいた方がよいのは言うまでもない。基本的なステップは技術導入の場合と同じであるが、最初のステップではこれまでの開発の仕組みの課題点を整理することがポイントになる。この場合、やはり本来、理想的な開発の仕組み（プロセス、マネジメント）はどのような姿かを念頭に整理してみるとよく、SECから提案されている「ESPR※7：組込みソフトウェア向け開発プロセスガイド」（以下、ESPR）や「ESMR※8：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」（以下、ESMR）を参考に、自組織の現状とのギャップ分析などを試みるとよいかも知れない。

こうした課題整理をもとに次のステップでは、自身の組織の開発の仕組みのあるべき姿を明確に決める作業が必要となる。例えばESPRでは開発に関係する様々な作業がプロセスとして整理されているが、ここでは、自分たちの組織の現状や製品に求められる品質や特性を考慮してプロセスのテーラリング（定義・実行）を行うことがポイントとなる。もちろんこの過程で、あまり重要でないと考えたプロセスを簡略化するなど様々な工夫をすることがポイントになる。そして、このようにして決めた開発の仕組みは先の手法導入ステップと同じように、モデルプロジェクトなどで実践評価したあとに、組織内に拡大適用していくなどの順次展開のシナリオを用意しておいた方がよいようである。

※5 CMMI：Capability Maturity Model Integration。能力成熟度モデル統合とも呼ばれ、開発プロセスを5段階の成熟度で定義している。
※6 PMO：Project Management Office。組織内において、プロジェクトマネジメントの支援活動を行う組織・部門。
※7 ESPR：Embedded System development Process Reference
※8 ESMR：Embedded System development Management Reference

4. 技術と仕組みを有機的につなぐ組織風土の醸成

組織風土の問題

冒頭に示したスーパープログラマの例のように、どのように素晴らしい技術を持っていても、そして、どのように素晴らしい開発の仕組みを持っていても、やはり組織としてそれを融合させ効果的に機能させる風土が定着していないと、開発は円滑には進まないようである。それでは組織風土はどのようにして醸成されるのだろうか。これまでにも様々な考え方などが議論されているが、恐らく重要な要素のひとつとして「コミュニケーション」を挙げることができる。ソフトウェア開発には様々な人がかかわり進んでいく。こうしたヒトとヒトとのかかわりの中で進むソフトウェア開発では、その間を取り持つコミュニケーションがとりわけ重要であることは間違いない。

コミュニケーションの良否は管理者で決まる

自社の組織内のコミュニケーションがどの程度であるかはぜひ、ESQRの4章に紹介されている表1のコミュニケーション・チェック項目を参考に評価してみていただきたい。恐らく、幾つかの項目で思い当たるものが含まれているのではないだろうか。こうしたコミュニケーションの良し悪しであるが、やはり開発管理者次第というところも多いようである。筆者が知っているある組織の管理者は、会議のたびに何も発言せず、時には開発者の前で居眠りをするという態度を取り続けていた。もちろん開発管理者にとってはあまり関心のないことが議論される会議もないわけではないだろう、しかし、その会議に同席した開発者はそうした上司の態度を見てどう思うだろうか。

その組織で自分たちの開発について管理者も交えた活発な議論が交わされたという話はついぞ聞くことはなかった。このように開発管理者が組織内のコミュニケーションにどれくらい意識を持って望んでいるかで、組織内の風土は大きく変わってくるようである。

5. まとめ

ここまで「技術導入」「仕組みの整備」「組織風土の醸成」という三つの問題について開発管理者がどのようにかかわっていくべきかを考えてみた。ここに述べた考え方は、ひとつの例であるが、ぜひ、多くの開発管理者の方々に、自身の組織の開発力強化という視点から開発管理者としての役割を考えるきっかけにしていただければ幸いである。

（IPA/SEC　大野　克巳）

表1 コミュニケーションチェック項目

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、議論したか	□
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	□

Part 1-5

改善に向けて担当者が考えるべきこと

1. ソフトウェア業界の業務特性

皆さんはこの1カ月、どれくらい技術書を読まれただろうか。あるいはどのくらい周りの同僚と自分たちの仕事の進め方などについて議論されただろうか。この業界に限らず、最近はどこの業界も忙しく、勉強したり業務について議論をして見直したりする時間がなかなかとれないというぼやきをよく耳にする。その一方で、ソフトウェア業界は3K※1と呼ばれているという話もあり、きつい仕事の仲間入りをしている。果たして本当にそうなのだろうか？　9年連続の200本安打を達成した日本人大リーガーは、その華やかな実績の裏に冷静な自己分析と絶え間ないトレーニングや工夫を積み重ねていることは周知の事実である。実はソフトウェアの分野でも華麗な、あるいは着実な仕事を達成するためには、こうした冷静な自己分析と地道な努力が求められているのではないだろうか。

日々の忙しい時間の中で、こうした努力を欠かさずに続けることが、恐らくシステム開発が陥っている3Kの問題を解決する唯一の解法なのかも知れない。ここでは、システム開発にかかわる技術者が自らの作業を見直し、より良いソフトウェア作りを進めていくためのアプローチについて考えてみたい。

2. 今の仕事の進め方を振り返ってみよう

さて、それでは、まず冷静な自己分析ということについてもう少し考えてみよう。例えば「あなたの仕事の進め方を振り返ってください」と言われたら、皆さんはどういう行動を取るだろうか。きっと「毎日の仕事に追われ、仕事の進め方を自分で考える余裕など無いよ」「どうせ、振り返ってみたところで、誰かが何とかしてしてくれるわけではないし」といった消極的な反応をする方もいるかも知れないし、逆に「この1カ月で実施したミーティングの議事録をひっくり返してみました」「この間のレビューの参加者を集めてレビューの進め方の問題点を洗い出してみました」という方もいるのではないだろうか。間違いなく開発の担当者や技術者がいて開発が進んでいくということは、その中で様々な仕事が流れていくことを意味している。そして、そうした仕事の進め方を振り返る場合、どのようにしたらよいだろうか。ここでは筆者の経験をもとに仕事の振り返りのポイントを整理してみよう。

【その1】
プロジェクトに設計やコーディングのためのルールに従っているか確認する

例えばソフトウェア作成の中で必須となるプログラミングを考えてみよう。経済産業省の組込みソフトウェア産業実態調査によると、多くの開発の中ではC言語が利用されている。しかし、C言語の場合、記述の自由度が高く、技術者の経験の差が出やすいと言われており、思わぬコーディングミスが多いようである。実際、技術者の技量や経験の差によって、作られるソースコードの出来に差が生じてしまうと、レビューがやりづらくなったり、保守性が低くなったりするなどの問題が生じ、結果としてプログラム、ひいてはソフトウェア全体の品質が大きく低下することにつながりかねない。こうした状況を解決するためのひとつの手段として、SECが策定した「ESCR※2：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］」（以下、ESCR）などを参考に自身のソースコードの書き方を見直したり、グループ内のコーディングルールを決めたりすることで、ソースコード上の誤り混入を未然に防ぐことができるのである。

さて、皆さんのプロジェクトでは、コーディングルールが決められているだろうか、少し振り返ってみていただきたい。中には「そもそもあることを知らない」「コーディングルールはあるがルール数が多く、また忙しいので守りきれない」などの問題があって利用されていない場合もあるかも知れない。もちろんチームとして開発を進めるうえでの基本はチーム内で決められたルールを守ることであり、その基本原則を遵守（じゅんしゅ）できているかどうかを振り返ってみると良いのではないだろうか。

※1 3K：「きつい」「帰れない」「給料が安い」。ソフトウェアの開発現場の厳しさを揶揄した表現。

※2 ESCR：Embedded System development Coding Reference

【その2】

プロジェクトのゴールがはっきりしているか確認してみる

皆さんは、プロジェクトが始まるときには、自分が何を担当するかを知りたいと思うであろうが、その前にどんなものを作ってどんなことに使われるだろうかと、知りたいと思うのではないだろうか。しかし実際のところは、プロジェクトのゴールがはっきりしていないまま、検討などの作業に着手した経験をお持ちの方が少なくないと思う。最近のソフトウェア開発ではスケジュールがタイトであることも多く、忙しさに任せてプロジェクトのゴールがはっきりしないまま進んでしまうこともあるようだ。しかし、ゴールがはっきりしないまま作業を進めると、計画とその達成具合の差異がチェックできず、結果として後戻りが発生しムダな作業をすることになってしまう。まず、皆さんにおすすめしたいのは、ぜひ、ご自身がかかわっているプロジェクトについて、もう一度そのプロジェクトのゴールを列挙し、反すうすることである。例えば、

①どのようなソフトウェアを作るのか
②どのようなユーザがどんな環境で使用するのか
③開発するための納期・品質目標・予算は

といった具合であるが、皆さんのプロジェクトでは、こうしたゴールがはっきりしているであろうか。プロジェクトに参加したならば、最初にそのプロジェクトのゴールを確認することは、ある意味、プロジェクトメンバ全員に共通する基本事項である。もちろんプロジェクトのゴールには、ビジネスとして直接的に関係する品質、コスト、納期といったものから、参加者の技術スキルの習得、あるいは、組織の開発プロセス定着まで様々な要素が考えられる。その中でも開発のベースという意味では、開発プロセスや開発計画といった要素は一担当者であっても十分に理解しておくことが必要であろう。ソフトウェアの開発プロセス関係、開発管理関係についてはSECでもかなり重視し、様々な参考情報を公開している。こうしたものを一読いただくのも近道かも知れない。

【その3】

作業項目と入力文書や成果物がはっきりしているか確認してみる

ソフトウェア開発では、要求定義に始まり、最終的なシステムテストに至るまで様々な作業を手際良くこなしていくことが求められる。必要な作業が抜け落ちていたり、入力文書や成果物が抜けていたりすると、決められた作業手順が実施できずに、開発のムダ・ムリが発生し、後戻りが発生したり品質悪化を招くことになりかねない。

さて皆さんのプロジェクトでは、作業項目を明確にし、その作業手順とその作業の入力や出力を整理して、仕事を進めているであろうか。

中には会社や組織としての標準的な工程や開発プロセスが決められているところもあるかも知れない。それはともあれ、こうした作業標準の有無にかかわらず、あるいは実施している実施していないにかかわらず、まずは、自分たちが開発の中で実施している作業を少し整理して考え直してみる価値はあるのではないだろうか。恐らくそれが現場発のソフトウェア開発力強化を進める上での第一歩になるのだと思う。もっとも実際の開発の中で行われている作業を整理しろと言われても、どのように整理したらよいかわからない方もいらっしゃるかも知れないが、その場合には、ぜひ、後述するSECの「ESPR※3：組込みソフトウェア向け開発プロセスガイド」（以下、ESPR）なども参考にしていただけるとよいかも知れない。

【その4】

プロジェクトに品質指標があるか確認してみる

近年、ソフトウェアの大規模化によって発生し続けているソフトウェア不具合やシステム障害を最小限に抑えていくためには、開発作業の中で、品質目標を定め、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）の実践が求められている。

ソフトウェアを開発する担当者にとっては、自分の作業や成果物が、品質指標を用いて可視化されれば、自分の仕事の良し悪しを周りからアドバイスしてもらうことができるであろうし、計画時には、定量的な作業見積もりが可能となるのは間違いない。

さて、そうは言っても、皆さんがソフトウェア開発作業を進める中で、「ドキュメントのボリュームは適正ですか」「レビュー時間やテスト項目数は適正ですか」と問われたとき、答えられるであろうか。実際問題として、ソフトウェアの品質をどのように定量化して考えるかは非常に悩ましい問題でもある。そもそも、どのような指標を使うか、そして、その値をどのようにして測り、どのようにして値の妥当性を確認し開発にフィードバックしていくかなど、様々な課題があることは容易に想像できるのではないだろうか。これらに関しては世の中にも様々な手法や考え方が提示されており、それらをぜひ参考にしていただくとよいかも知れない。もちろん、SECでもソフトウェア品質定量コントロールを実施するためのガイドとして後述する「ESQR※4：組込みソフトウェア開発向け品質作り込みガイド」（以下、ESQR）を発行している。これらも参考図書のひとつになると思う。

ここでは「仕事の進め方を振り返り」をテーマに幾つかの

※3 ESPR：Embedded System development Process Reference
※4 ESQR：Embedded System development Quality Reference

ヒントを列挙してみた。ここまでにも紹介したように、SECのESxR※5シリーズをはじめとして様々な参考図書が発行されているし、それ以外でも直接的にソフトウェア開発の作業(=プロセス)について整理されたガイドや国際標準なども発行されている。もちろん、こうしたものを参考に作成された社内のルールや規約などもあるかも知れない。まずは、これらの情報をフル活用して、振り返るにあたってのあるべき姿との差分を正しく認識することが必要なのかも知れない(図1参照)。

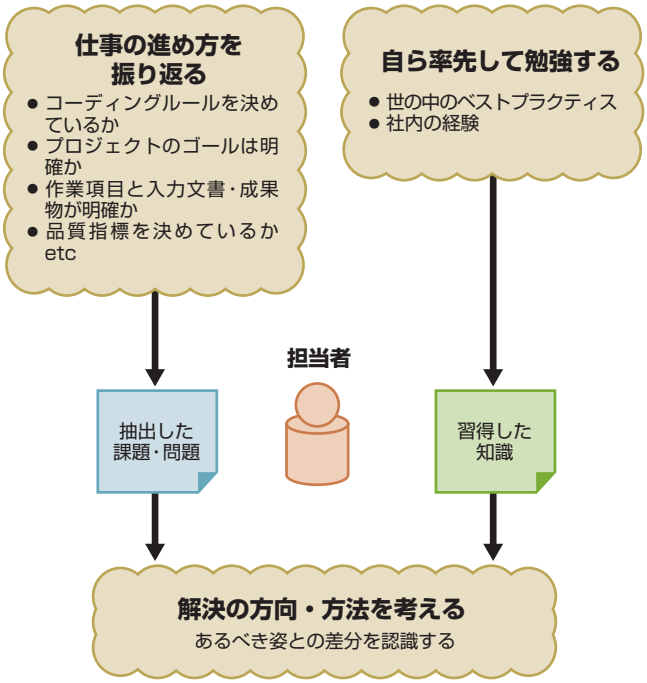


図1 改善に向けて担当者が始めるアプローチ

3. 一人で考えるのは限界がある

さて今の仕事の進め方を振り返ると、いろいろな良い点、悪い点などが見えてくると思う。多くの技術者はここで、見えてきた悪い点に着目し、どうしたらそれらが解決できるかを考えるのではないだろうか。問題の中には、恐らく冷静に分析し、考えることで解決策が思い浮かぶものもあると思う。その一方で、いくら考えても解決策が思い浮かばなかったり、幾つも解決策があったりして決めきれないなど、ぐるぐると考えが回ってしまう場合も少なくない。そうした場合、ぜひ、周りを見回してはどうだろうか。同じような悩みを持っている同僚がいるかも知れない。解決策を考える上でのヒントを先ほどと同じように筆者の経験をもとに列挙してみよう。

【その5】

問題はチームの問題として共有する

皆さんは、作業中に何らかの問題に気が付いたとき、自分で何とか解決できる問題なのか、そうでない問題なのか判断すると思う。自分で何とか解決できるレベルを超えた問題であれば、あきらめて、誰かが解決するだろうと考えてはいないだろうか。しかし、問題の中には、直接自分に関係なくても、プロジェクト全体の問題であれば、遅かれ早かれ、自分の作業にも影響してくるものがある。プロジェクト全体に影響する問題に気付いたときは、各人はプロジェクトの問題として共有し解決していく義務があると考えてみてはどうだろうか。とくに仕事の進め方というような大きな問題は、担当者であっても、時にはプロジェクト全体の目線で問題を捉え、解決に向けて検討する姿勢がほしい。マネージャや経営者だけでは、解決できる問題ではないことは理解できるのではないだろうか。担当者自らが先駆けて、チームとして問題の解決に臨んでいくことで、組織としてのソフトウェア開発力向上の糸口が現れると思う。

【その6】

解決策は、複数挙げてみる

大きな問題であればあるほど、問題の解決策を検討するときは、解決案を複数挙げてみて、良い点／悪い点を評価した上で、最良の解決策を選択した方がよい。複数の解決案をきちんと評価して解決策を実行していくと、そのときには多めに時間がかかるだろうが、あとになって問題が解決しないことがわかり、もう一度、別の解決策を講じるようなことは避けられるであろう。留意すべきことは、解決策を誤ると、あとになってうまくいかないことがわかり、作業工数を浪費しただけで終わってしまうことがあるということだ。

また、問題の解決策を複数挙げておけば、関係者に説明する際にも、スムーズに理解を得られるものだ。蛇足であるが、一般的に、説明する順番は、最良の解決策から説明すると良いと言われている。どうも日本人の説明は、前置きから入ったり、言い訳から入ったりと、なかなか本題や結論が見えない場合も少なくないようだ。説明の仕方ひとつ取っても、相手の理解を得るための工夫が求められるのは言うまでもないことであろう。

【その7】

解決策を考えるときは制約条件を確認する

一般に課題が見つかり、その解決策を考える場合、常にばら色とは限らない。どのような解決策であっても、その前提となる条件や制約条件は多かれ少なかれ存在すると考えておい

※5 ESxR:Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。

た方が間違いはないのではないだろうか。例えば、代表的な制約条件としては、時間、人、環境、コストなどを考えることができる。

①時間：
問題を解決するには、まず、いつまでに解決しなければならないのか、解決に向けた検討・活動にどれだけ時間を割くことができるだろうかといった時間的制約を確認してみよう。

②人選：
次に、問題を解決するために、人的制約は無いか、誰がキーパーソンとして必要なのか、また、そのキーパーソンはその問題解決のためにどれだけの時間を確保できるのか、その他に必要な要員は確保可能なのかなどを確認してみよう。

③環境・道具：
問題を解決するためのツールや、部材や場所などの環境的制約は無いかなど、環境や道具は意外に確認漏れが多い項目のひとつであり、注意が必要である。

④コスト：
もちろんビジネス上の解決策が求められる以上、コストや金銭的な側面の制約はどのような場合にも考慮しておかなければならない。たとえ、どんなに優れた解決策であっても、コストが想定外にかかってしまっては、ビジネスとして成立しないといったことも十分に起こり得るのである。

このようにそれぞれの解決策についての制約条件を整理し、その効果などとの天秤（てんびん）にかけて最終的な判断をすることが重要である。もちろん、こうした解決策の検討やフィジビリティ・スタディを早めに実施しておくことにより、実現性の低い解決策の検討に時間を費やさなくても済むし、解決案を説明する場合も、制約条件と共に解決案を説明すれば、理解してもらいやすいのは言うまでもない。

とにもかくにも、皆さんが開発の中のいろいろなシーンで問題に気付いたときは、自分一人だけで考えて、ムリだと考えてしまうのではなく、考え方を切り替えて、チームで共有することが重要であると考えてみてほしい。問題に早く気が付いて、チームで早く共有することが、プロジェクトに貢献するものだとポジティブに考えられるようになると、きつい仕事もやりがいを感じるのではないだろうか。

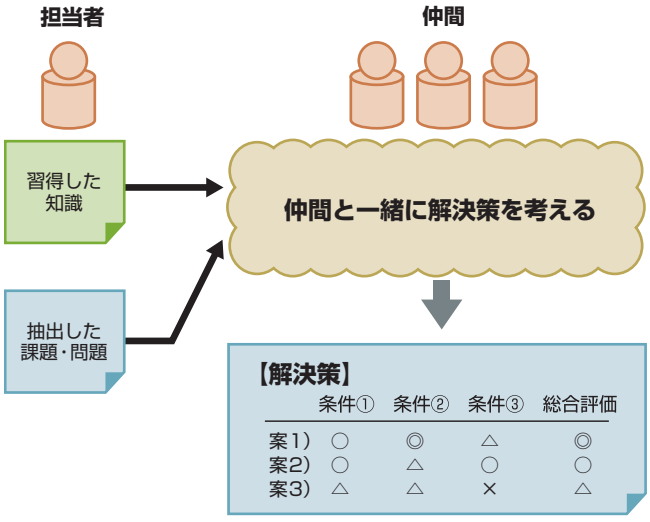


図2 解決策を考えるためのヒント

4. 知識が無ければ解決策は見えてこない

世の中のベストプラクティスを勉強しよう

前項までに、自らの作業を見直す際のポイント、問題を解決する際のヒントを紹介してみたが、これらを実践するだけで本当に解決策は見えてくるのであろうか。登ったことのない初めての山に仲間と共に挑戦しようとするときは、ガイドブックを探したり、インターネットで経験者の情報を漁ったりして、どういう装備をして、どこから登るか、どこで泊まるかなど、できるだけ必要な知識を得てから、自分たちの条件に合った登り方をするのではないだろうか。

現状の仕事の進め方を良くしようとするならば、ソフトウェア開発に関する技術情報を自ら求めて、自分の仕事に取り込んでいくための勉強をおろそかにしてはいけないのだと思う。ソフトウェア開発に関するノウハウや方法論を経験から知識として体系化することへの取り組みやその研究をソフトウェア・エンジニアリングというが、周りを見渡せば、世の中にはそのような参考になる情報がいっぱいあふれているはずだ。社内にも恐らく様々な参考になる情報が埋もれているであろう。そのような情報を地道に習得していけば、必ずや仕事の進め方の参考となる知識が得られるはずである。

現状の問題を認識し、何とか改善しなければならないと思っているならば、ここは、自分が先駆けて、情報を収集し、役に立つ情報は賛同する仲間に紹介し共に議論していくという姿勢も大切かも知れない。収集した情報をもとに、本気になって勉強し習得した知識であれば、必ず耳を傾けてくれるはずだ。仲間は、個人のベストプラクティスに引き付けられるものである。

参照技術体系ESxRシリーズ

さて、前項をひと言で言うと「もっと勉強してみたらどうでしょうか」ということに尽きるのであるが、実際に勉強しようと思ったらどのようなものを参考にしたらよいのだろうか。一概にどれが良いということはここでは触れないが、SECでもそうした方々の参考にしていただくべく、ソフトウェア開発に関する様々な参考図書や参照情報を公開している。例えば、組込みソフトウェア開発に関して紹介すると、我が国の組込みソフトウェア開発力を強化するという目標の中で我々が様々な企業を訪問し、

各社での取り組みについて議論させていただき、産業界の知見を様々な形で咀嚼（そしゃく）する中から、ソフトウェア開発における品質向上に向けた参照技術体系としてESxRシリーズを整備している。

ESxRシリーズはソフトウェア開発に実際にかかわっている方々に直接参考にしていただき、利用していただける良質の道具を提供するというコンセプトで整理されている。ぜひ、将来を見据えている開発担当者の方にご一読願いたい。恐らく今の自分のソフトウェア開発作業の中で、自分たちが欲しかった情報の幾つかは見つけることができるのではないだろうか。以下に、このESxRシリーズがソフトウェア開発に携わる担当者の方にとって、どのように役に立つかを簡単に紹介したい。

【ESCR：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］】

このガイドは「いかに品質の高いソースコードを作るか」をメインのテーマとして、実際の開発現場でのコーディング規約などを策定する際の参考情報を整理したものである。中には実際のコーディングの良い例／悪い異例などもC言語のコードで例示してあり、なぜそのように記述するのが良いのか悪いのかといった説明を含めてわかりやすく解説されている。この書籍は主にプログラミングなどに携わるやコードレビューをされる方などを対象にしている。

【ESPR：組込みソフトウェア向け開発プロセスガイド】

ソフトウェアを開発するためには様々な作業の手順を追って実施していくことが求められる。このガイドではソフトウェア開発に必要となる作業をその入出力情報などと併せて体系的に整理し、ソフトウェアプロセスの概念からわかりやすく解説している。今まで経験した各作業工程の作業内容に漏れや手抜きが無いか、関連する工程と照らし合わせて、工程で実施する作業と工程のインプット・アウトプットを評価し、自身の開発において求められる作業の見直しの際に参考とすることができる。

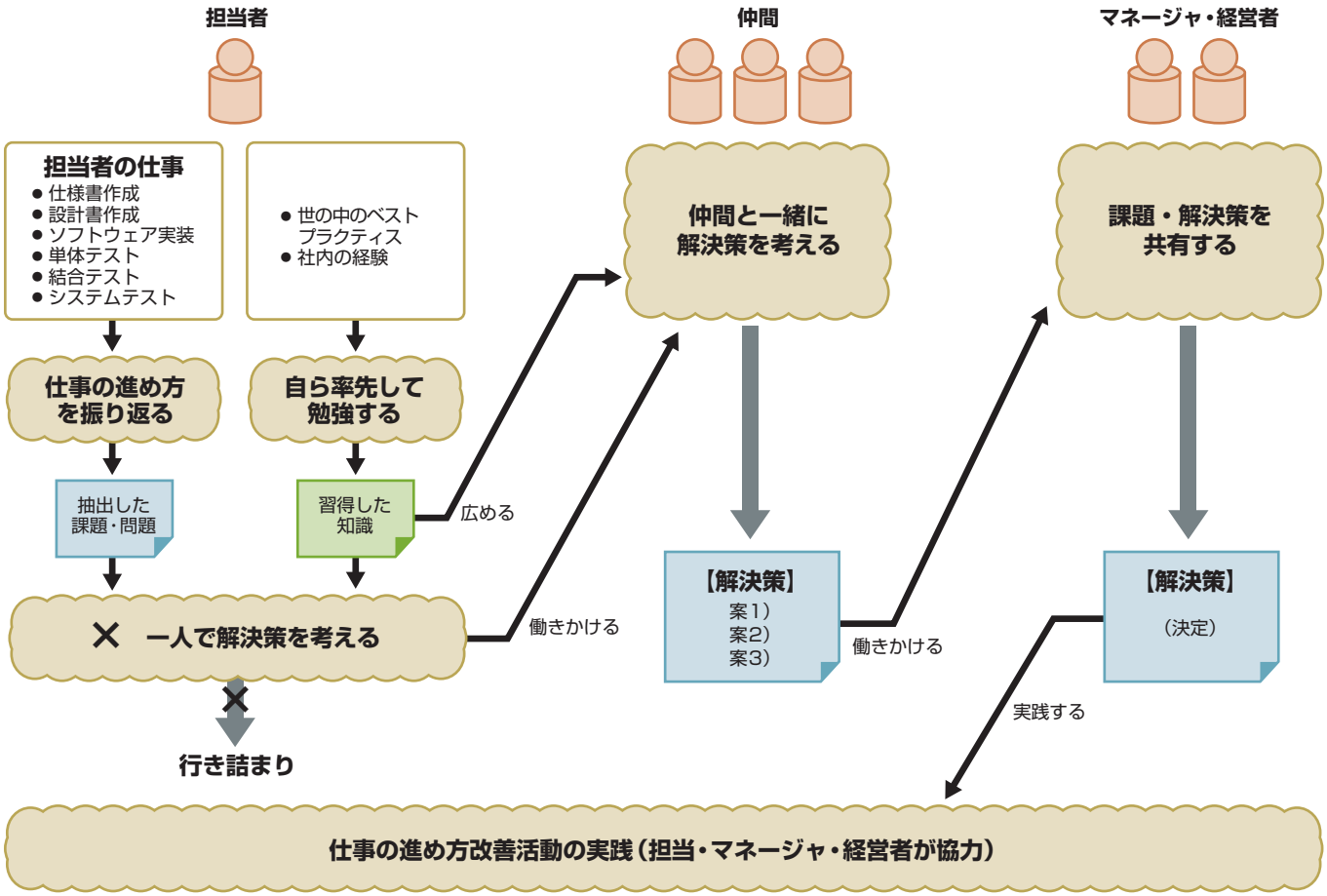


図3 担当者が率先して進める改善イメージ

【ESMR: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］】

今まで、途中からプロジェクトに参加したときに、渡されたプロジェクト計画書があまり役に立っていないと感じたり、プロジェクト開始時に、決めるべきことが決められていないと感じたりした方も少なくないのではないだろうか。ソフトウェア開発の中でもとくに問題が多いのが、このプロジェクト計画と呼ばれるものである。このガイドはソフトウェア・プロジェクトを始めるにあたり、どのようなことをどう決めておけばよいかという点について、計画書に盛り込む内容と共に整理したものである。計画書というと管理者が作るものという印象を持つ方もいらっしゃるかも知れないが、担当者としても自身の作業の位置付けや計画といった視点で十分に目を通しコミットしておかなければならない。その意味でも担当者としても、仲間と共に自分たちのプロジェクトの計画がどうなっているのか、そしてどうあるべきかを考えて、ぜひ、現場のマネージャに提案していくという役割を持ち合わせていると理解していただければと思う。

【ESQR:組込みソフトウェア開発向け品質作り込みガイド】

このガイドはソフトウェア開発途中での品質作り込みという視点から、設計した資料が大丈夫か、あるいは、実施した作業が十分かといった視点で確認するためのヒントを整理したものである。どうもソフトウェア開発において数値を測るというと管理的な印象が強く、開発者はあまり好ましく思わない方が多いようであるが、こうした品質計測評価は何も管理者や品質保証担当者のためだけにあるのではない。自身が作成した成果物や作業を測り、客観視するということは、結果として自身の作業を効率的に、そしてより高い品質を生み出すことにつながると考えることができる。最近は指示されたことのみ行う若者が多いとも言われているが、ぜひ、今まで、指示された通りにだけ作業を行っていた方は、自身の作業や成果物を測り客観的に評価していくことで積極的にプロジェクトのソフトウェア品質作り込みに参加してみてはどうだろうか。

5. まとめ
～チームとしての作業の最適化を目指して～

改善に向けて、ESxRシリーズなどの参照技術体系から一連の知識を習得することができたら、チームメンバ、現場のマネージャ、経営者に働きかけ、改善内容のイメージを具体化してみよう。ESxRシリーズガイドから習得できる知識は、様々なプロジェクトや開発するソフトウェアを搭載するシステムに共通して利用できるように記述されているため、これら習得した知識を、自分のチームやプロジェクトにあてはめて、改善できそうかどうか、改善したほうが良いかどうか検討を進めていくとよい。検討を進めていくうちに自分のチームやプロジェクトに合った利用方法がわかり、ソフトウェア開発作業の進め方の改善イメージが見えてくると思う。

具体的に改善を進めるためには、パイロットプロジェクトを選定する。このとき、ESxRシリーズからの情報を最初からすべて、モデルプロジェクトに適用させていくのは大変なので、改善テーマを決めて、的を絞るとよい。チームメンバ、現場のマネージャ、経営者それぞれがイメージする改善内容はバラバラにならないようにきちんと共有することが大切だ。そのためにも、最初に、プロジェクト計画書を作成するときに、プロジェクトの目標の中に、改善するテーマも含め、きちんと関連者で共有することから始めるべきであろう。

適用するテーマが決まれば、各人の目線で、意見交換しながら改善項目を議論し、改善計画書を具体的に作成していく。この段階では、既に、改善活動に対する経営者の同意が得られ、改善に向けたモデルプロジェクトの現場マネージャがアサインされているはずなので、開発担当者としては、改善計画に沿って、改善のために習得した知識・技術を自分たちの業務にあてはめて試行していく。

改善モデルプロジェクトが実行に移されて終了したときには、必ず、試行した結果を蓄積し、次のプロジェクトで利用できるように整備しておかなければ、改善活動が止まってしまうことに注意してほしい。

さて以上、本稿では、担当者というソフトウェア開発の中でもの中心的な役割を担う方々に対して、ソフトウェア開発力強化を進めるためのヒントを紹介してみた。いろいろなアプローチ法が考えられるが、図3のような現状のソフトウェア開発の仕事の進め方を改善していくひとつのアプローチもご理解いただけたのではないだろうか。冒頭でも述べたが、地道な努力なくしては、現状は良くならないことと、自ら情報を収集し、勉強して、良い知識を得ること、もうひとつは、一人で考えているのではなくて、同じ問題意識を持った仲間を作っていくことをしっかり意識し、ソフトウェア開発力強化を進めていただきたい。

（IPA/SEC　松田　充弘）

Part 1-6

改善を支える技術者教育

1. 技術者育成の必要性

技術を習得するということ

物事を行うには何ごとも技術の習得が必要である。例えば、カレーライスを作る場合においても、野菜の切り方、火加減の強弱、レシピの見方など、ひとつひとつを知識として学んで実践していかなければ美味しいカレーライスを作ることはできない。一方ソフトウェアの開発現場はどうだろう? ソースコードを書く技術、計画的に設計やテストを行う方法、対象ドメインの知識など、ソフトウェアを作るには欠かせない知識・技術はたくさんあるが、開発技術者はそれらをどのように習得しているだろうか?

これまでソフトウェア開発の現場・人を中心に、あるべき姿を論じてきた。本稿では開発現場の特に教育の場面に焦点を当て、技術者のスキル向上の手だてについて、自分の持つ技術を把握するにはどうすればよいか、また組織の開発力向上も視野において目標をどのように決め、進めていくべきかなどについて順を追って説明する。

開発力強化のための技術者教育

本特集号の随所で述べられている通り、ソフトウェアが年を追うごとに指数関数的に複雑化、肥大化しているのは周知の事実である。そして肥大化していくソフトウェアを支えるために開発現場では様々な技術や手法を取り込もうとしている。

ところがいくら技術の進歩があったとしても、その技術を使いこなす人がそこについていけなければ結果を出すことは難しくまさに宝の持ち腐れである。つまり日々進化していく新しい技術を取り込むにはそれに対応できるスキルや経験が必要であり、このために技術者はいかに新たな技術を獲得するか、スキルを向上させるかが課題となる。しかし、開発現場の技術者は日々開発業務に追われている。そのような中で、場当たり的に教育を行っても効果を得ることはなかなか難しい。したがって目標を定め、計画的・効率的にスキルを向上させることを念頭に置いた技術者教育を考える必要がある。

2. 技術者個人のスキル把握

「技術」と「スキル」

人材育成を考える上で、私たちはよく「技術」や「スキル」という言葉を使用する。例えば優秀な技術者や企業に対して「あの人は技術がある」「あの会社はスキルが高い」などと評することがよくある。このときの『技術』『スキル』という言葉は、それぞれ知識としての「技術」を意味しているのか、あるいは開発能力としての「スキル」を意味しているのか。恐らくこれらのことばの違いについて意識して使用している人は少ないであろう。まずは、組込みスキル標準「ETSS」[※1](以下、ETSS)での定義を参照してみよう(図1)。

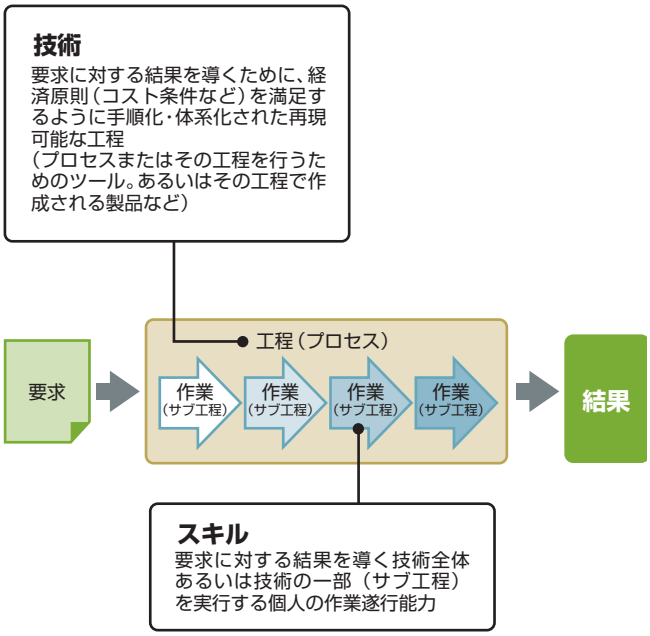


図1 「技術」と「スキル」の定義

このように「技術」は開発対象物や製品を設計・開発・製造といった一連の工程（手順）や、その工程を実現するためのツール、あるいは工程を通じて作られた製品自体と位置付けている。それに対し「スキル」は、個人が「技術」を実際にどれくらい使いこなせるかという期待値であると定義している（「～することができる」と表現できる）。

つまり、ある組込みシステムを開発するという要求に応えるためには、「技術」と技術を使いこなす「スキル」が必要で、これら両方がセットとなって初めて結果が出せるのである。従って、「技術」と「スキル」はセットで捉えることが重要となる。

「技術」と「スキル」の可視化

技術者個人の成長を考える上で、まず重要な要素となるのが「技術」と「スキル」の可視化である。製品開発にあたりどのような工程があり、そこでどのような技術を使っているか、またそれを使いこなすスキルはどの程度あるか、といった現状を可視化することがその第一歩となる。

例えばETSSでは対象となる人材のスキルを可視化し、目標となる人材像（あるべき姿）を定義する。そして目標とする人材像を実現するための教育プログラムを定義し実践する、という人材育成・活用のためのフレームワークを提供している（図2）。以降ではETSSを用いたスキルの可視化から教育までの一連の流れを紹介する。

（1）スキル基準の作成

「技術」と「スキル」の可視化にあたり、最初に実施すべきことは、まず現在保有するスキルや将来必要となるスキルなど、診断したいスキルの定義を整理することから始まる。

ETSSでは左側のスキルカテゴリとスキル粒度（第n層）で組込みソフトウェア開発に必要な技術を体系的に整理している（図3）。スキルカテゴリでは「技術要素」「開発技術」「管理技術」という三つの観点で整理をし、第2階層までのスキルカテゴリを提示している。なお下位階層を含むスキル項目の具体的な定義は、これらを利用する技術者個人や企業の組織などで

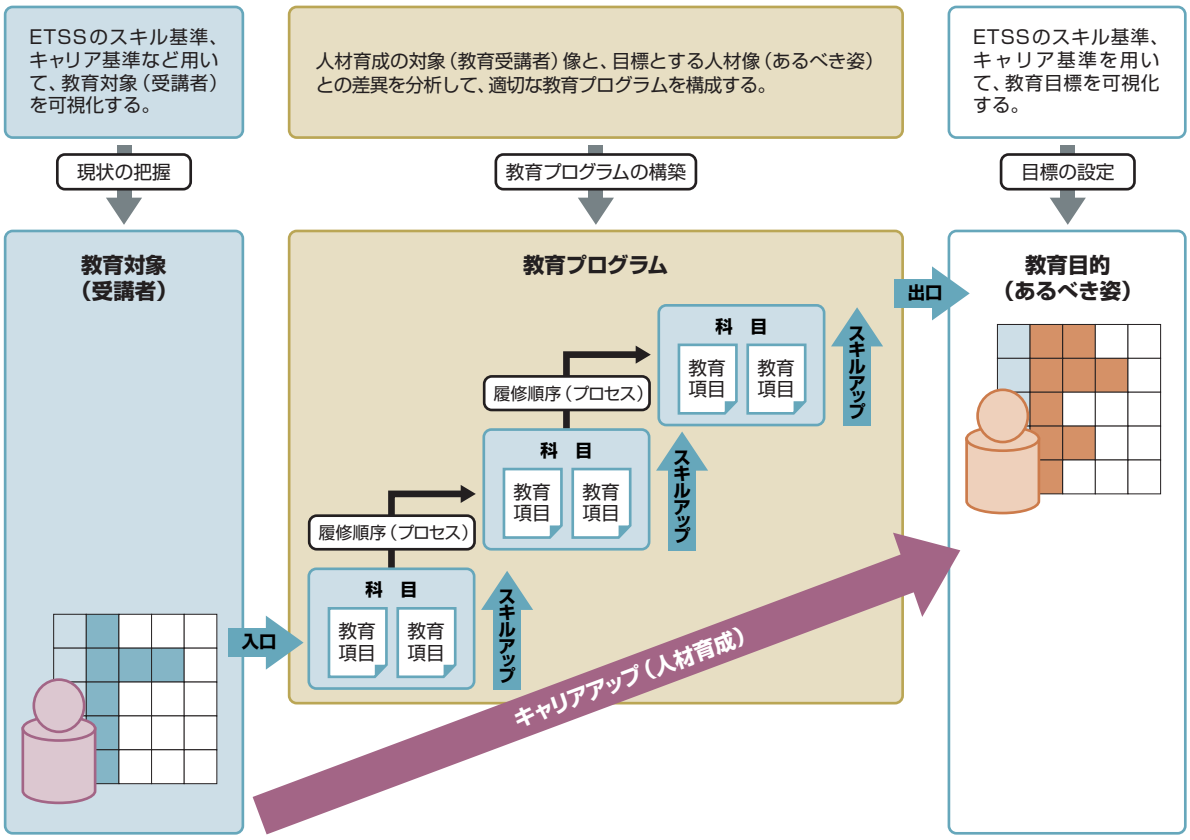


図2 人材育成・活用のフレームワーク

※1 組込みスキル標準「ETSS」：組込みソフトウェア開発力強化を目的に、組込みソフトウェア開発における「人材の育成」や「人材の有効活用」を実現するための指標や仕組みを規定している。2005年経済産業省 組込みソフトウェア開発力強化委員会により策定。Embedded Technology Skill Standards

定義する。つまり具体的なスキル項目の抽出は、利用する側のニーズにより異なるがその多くはどの工程（プロセス・手順）でどのような技術を使っているか、という業務の分析などを行うことで実施される。

(2) スキルの測定と分析

スキル項目の抽出・整理が行えたら、図3の左側で整理された技術項目それぞれに対し、右側のスキルレベルの部分で自身のスキルレベルの測定を実施する。それにより、スキルの分布が可視化され、強み・弱みを客観的に把握することができる。

なおETSSでは、スキルレベルの概念を共通化し、技術項目ごとに作業遂行能力の期待値（ポテンシャル）を4段階のスキルレベルで表現している。

- **レベル4：最上級 新たな技術を開発できる**
- **レベル3：上級 作業を分析し改善・改良できる**
- **レベル2：中級 自律的に作業を遂行できる**
- **レベル1：初級 支援の下に作業を遂行できる**

スキル可視化の具体例

《技術者Aさんのスキル可視化と特性分析》

それではある技術者Aさんを例に見てみよう。図3はAさんのスキル分布を表している。

［技術者：Aさんについて］

- **入社3年目のソフトウェア・エンジニア**
- **ある組込み機器のプラットフォーム開発に従事**
- **ソフトウェアの詳細設計からテストまでを担当**

［Aさんのスキル特性］

- **得意な技術領域：**
  プラットフォーム（技術要素）
  ソフトウェアコード作成とテスト（開発技術）
- **不得手な技術領域：**
  マルチメディア（技術要素）
  プロジェクト管理（管理技術）

このように個人のスキルを測定・分析することで、個々の技術者がどの技術をどのくらい持っているのか、また技術者としての強み／弱みを定量的に把握することができる。

3. 組織のスキル把握

上述のようにして個々の技術者のスキルは可視化できるが、これだけでは組織全体としてどの技術をどれくらいを持っているかは把握できない。このため図4に示すように、組織に所属する個々のメンバのスキル測定結果を集計し、組織としてのスキルを可視化していく工夫が必要になる。

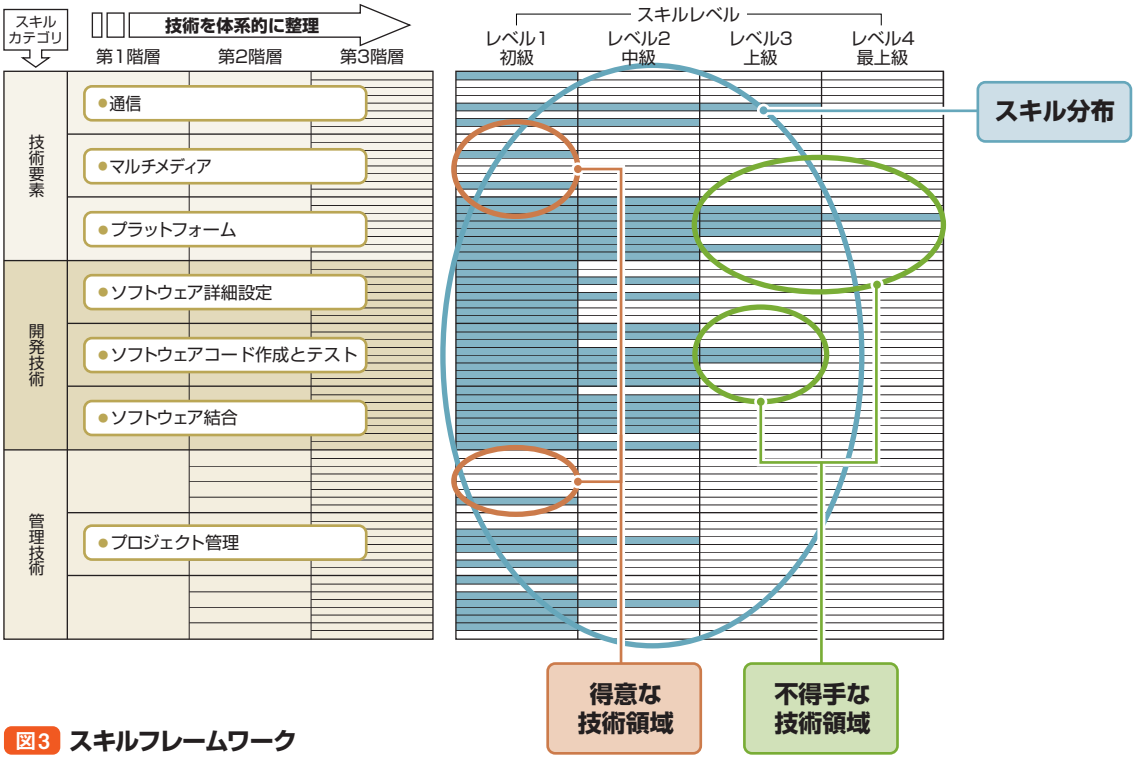


図3 スキルフレームワーク

スキル可視化の具体例

《ある組織のスキル可視化と特性分析》

技術者Aさんの所属する組織を例に具体的な例を見てみよう。図4は技術者Aさんが所属する組織のスキル分布を表している。

［Aさんの所属する組織］

- **ソフトウェア開発部門**

［組織のスキル特性］

- **得意な技術領域：**
  開発技術（全般）
  パーソナルスキル（コミュニケーションなど）
- **不得手な技術領域：**
  プロジェクト管理技術（全般）
- **その他：**
  管理技術の低さを開発技術とパーソナルスキルで補っている可能性が高い。組織全体として管理技術（全般）が低いため、プロジェクトマネージャなどのリーダー育成による管理技術向上が課題。

このように組織のスキルを集計・分析することで組織としての強み／弱みを把握でき、組織としての戦略的な人材育成、つまり組織力強化へつなげることが可能となる。

4. スキル目標の設定と教育

技術者個人のスキルが可視化できたら、可視化されたスキル分布と目標となる人材像（組織からの要求や方針、職種や専門分野：あるべき姿）とのギャップを把握し、どの技術をどれくらい獲得するか、伸ばすかというスキル目標を設定し、それを補完するための教育を検討する必要がある。もちろん、こうした個々の技術者のスキル向上は、前述の組織として求められるスキル像とリンクする形で考えていく必要がある。

スキル目標の設定

《技術者Aさんのスキル目標の設定》

例えば組織として不足するスキルを補完していくために、技術者Aさんのスキル目標を設定する場合を考えてみよう。その場合には下記のような目標を設定することができる。

［組織からの要求］

- 将来、プロジェクトマネージャを担って欲しい。

※ Aさんの所属する組織において、プロジェクトマネージャの育成が課題。

［Aさんの希望］

- 将来、上流工程（ソフトウェア要求分析・方式設計）を担当できるように現状の設計スキルを伸ばしたい。
- プロジェクトマネージャにも興味はある。

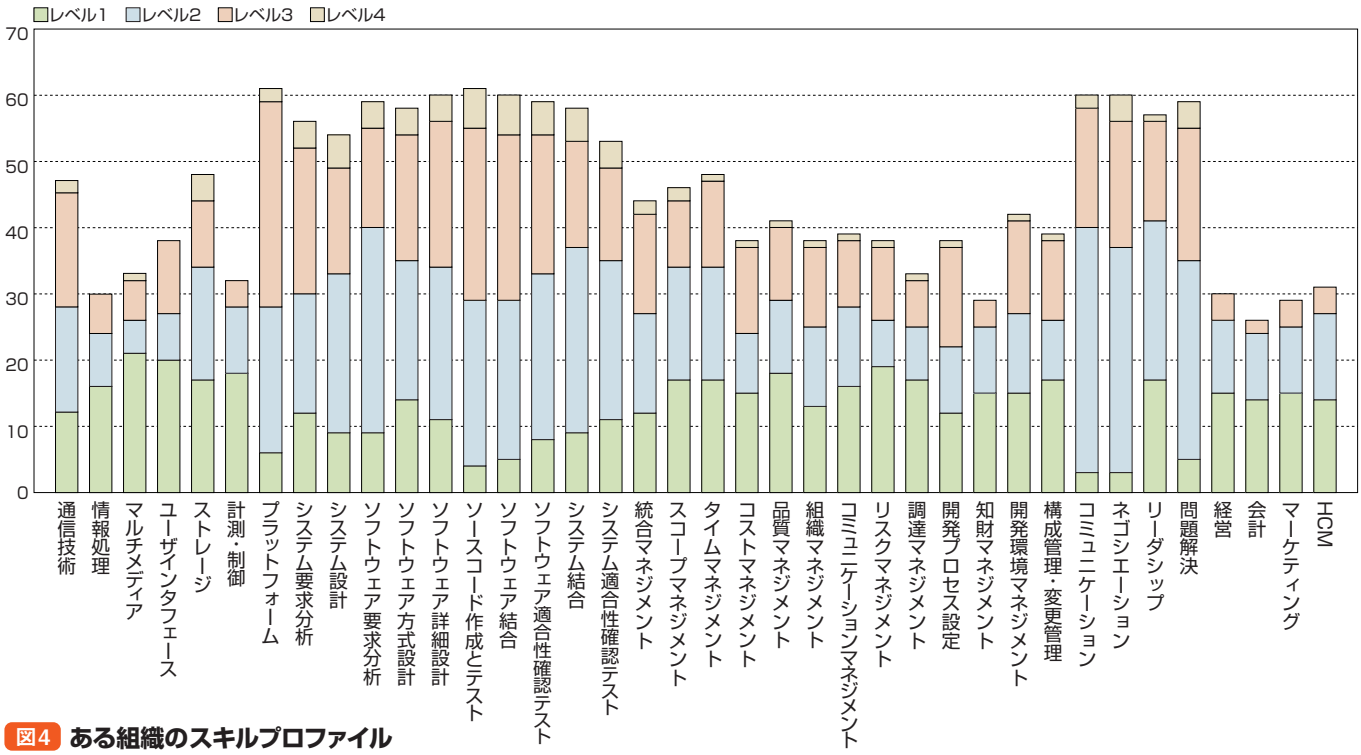


図4 ある組織のスキルプロファイル

［Aさんのスキル目標］
- プロジェクト管理（管理技術）：スキルレベル0→スキルレベル1
- ソフトウェアの詳細設計（開発技術）：スキルレベル1→スキルレベル2

教育計画の策定

上述のようにしてスキル目標が決まったら、次のステップとしてその目標に到達するための教育計画を作成する。そしてその教育計画を作成する上でまず重要となるのが、教育コンテンツの選定である。自社に既にあるコンテンツであればよいが、必ずしもすべてが社内提供されているとは限らない。その場合に備え外部から調達することも視野に入れておく必要がある。また教育コンテンツと関連し、教育の実施形態についても習得上の特性を考慮し検討する必要がある。例えば「技術」獲得を目的とする場合には講義型や自習型を、「スキル」習得を目的とする場合にはOJT※2や演習などの実習型を選択する。

《技術者Aさんの教育計画例（プロジェクト管理）》

図5は技術者Aさんのプロジェクト管理技術習得における教育計画（例）である。

この例に見るように教育計画では目標を明確にし、その目標を達成するための手段・方法を明確にすることがとくに重要となる。また、教育目標の達成度合いを測れるようにすることも忘れてはならないポイントである。

教育計画書	
目　　標	プロジェクト管理技術の習得（スキルレベル0→1）
科目名称	講師担当 『プロジェクト管理−初級編−』（講義）：社内教育部門 『プロジェクト実践』（OJT）：組込 太郎
実施形態	講義形式+プロジェクト実践（OJT） ※OJTではサブリーダーとしてリーダーと一緒に プロジェクトマネジメント実務を経験する。
実施時間	78時間 6時間×3回（講義） 60時間（プロジェクト実践）
日　　程	200x/04/01〜200x/06/30（3ヵ月）
実施の確認	毎月第4金曜日　17:00〜18:00 グループリーダ、部門長とミーティング実施
スキル判定	リーダーの支援のもとにプロジェクト管理実務を遂行できる ※スコープ・タイム・コストマネジメントの成果物で評価

表5 技術者Aさんの教育計画書

教育の実施

以上の計画に基づき、Aさんのスキル向上のための教育活動を行う。教育を進めるにあたっては教育の過程を観察し、その結果に基づきフィードバックをかけていく（＝振り返りを行う）。つまり、現状把握・教育計画（Plan）、教育の実践（Do）、教育効果の測定と分析（Check）、改善事項の検討（Action）というPDCA※3を回して教育を実施していく。

当然のことながらプロジェクトは流動的なものである。当初予定した教育を受ける機会を作るのが難しくなるかもしれないし、OJTを行うプロジェクトの予定が大幅に変わってしまうかもしれない。また、中には計画はされたものの実践が伴わないままにプロジェクトが終了してしまったり、実践はされているものの教育の仕方に問題があり、意図したスキルが身に付かなかった、という例も耳にする。当たり前のことではあるが設定した目標に向かい確実に教育を実践していくためにも定期的な振り返りが重要となる。

5. おわりに

これまで技術者個人のスキルを向上するための教育について述べたが、上述のように、技術者個人のスキル向上の積み上げは、結果として組織の開発力強化につながる。従って、技術者個人が獲得すべき技術や向上すべきスキルは、技術者の個人的な目標だけでなく、組織からのニーズも含め決めていく必要がある。つまり、技術者個人のスキル向上を検討する上では技術者個人としてのスキル可視化だけでなく、組織としての可視化も非常に重要な要素となる。その上で、伸ばすべきところを伸ばす、あるいは足りないところを補うなど、状況に応じた戦略的な教育計画の策定と実践、またそれを支える教育のPDCAサイクル実践が技術者個人にとって、また組織にとっても重要となる。その過程において本特集号で紹介している、組織として開発力を強化するための開発リファレンス：ESxR※4やETSSが役に立つことであろう。技術者個人として、また組織としての開発力・実践力向上に少しでもお役に立てば幸いである。

参考文献
［IPA/SEC］組込みスキル標準ETSS概説書［新版］

（IPA/SEC　小林　直子）

※2 OJT：On the Job Training。実務を通じて、必要なスキルを身に付けさせる教育訓練。
※3 PDCA：Plan, Do, Check, Action。計画（P）、実行（D）、評価（C）、改善（A）の各ステップをひとつのプロセスとして回すマネジメント手法。
※4 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。

Part 2

ESxRを活用した品質・開発効率の改善

Part2ではSECが策定したソフトウェア・エンジニアリング領域でのリファレンス類：ESxRシリーズについて、概要から利用方法まで、具体的に詳細を解説する。ESxRに初めて接する方は入門書として、既に適用している方にもヒントとしての情報を提供している。新人から管理者、経営者に至るまで本内容に目を通していただき、ソフトウェア・エンジニアリングを遂行していく上で必要なこと、ヒント・注意点等をそれぞれの立場で読み取っていただきたい。

Part 2-1

ESxRとは
――狙いと全体像・関連――

ESxR※1は「良いソフトウェア」を組織で作っていくための仕組みや技術についてSECの組込み系エンジニアリング領域で産業界の知見をまとめ、整理しなおしたものである。これらは実践的、つまり開発サイドに実際に手に取っていただき、すぐに活用いただけるという大きな特徴を備えている。本節では、ESxRの全般について目的や全体像など、及び、各ESxRの連携について簡単に紹介する。また、利用上の注意点や利用の状況についても説明する。

1. ソフトウェアを作る ～エンジニアリングの整備

我々の身の回りには様々なコンピュータシステムがあふれており、コンピュータシステム漬けと言ってもよいような状態になっている。こうした中、システムのユーザからは、より高い品質のシステムをより安くかつ早く手に入れることが求められ続けている。ユーザからの要求を満たすためには、システムの中核を構成するソフトウェアの開発を高度化していく必要がある。世の中にはソフトウェア開発の高度化を目指した様々な手法や技術情報があふれているが、その方向性は大きく二つに分けることができるのではないだろうか。その第1は設計やテストといった、開発の中で直接的なものづくりにかかわる作業を高度化する方向性である。そして第2はプロジェクトマネジメントや開発プロセス整備などに代表される、開発の作業を側面支援する方向性である。

SECでは我が国のソフトウェア開発力を強化するという目標の中で様々な企業を訪問し、各社での取り組みについて議論させていただいてきた。この不況下でもビジネスを円滑に回している会社や組織では、この二つの方向性の活動がしっかりとかみ合っている場合が多いようである。つまり、直接的なものづくりと間接的にそれを支える活動とで、改善の歯車を回していくこと、それがソフトウェア作りにおいて品質や生産性向上を実現していく上での近道であると考えてもよさそうである。こうした産業界の知見を様々な形で議論する中から、SECではソフトウェア開発における品質向上に向けた参照技術体系としてESxRシリーズを整備している。ESxRはEmbedded=組込みという言葉が付いているが、ソフトウェアの本質としては組込み分野と、エンタプライズ分野とでそれほど変わるものではない。そのため、エンタプライズ分野でもESxRシリーズの多くの部分を適用することが可能である。ソフトウェア開発での各場面のイメージを作りながら本特集号を読んでいただきたい。

2. ESxRの全体像

ソフトウェアの開発工程は図1に示すように、要求定義からシステムテストまでをV字で表すことができる。

要求を定義し、アーキテクチャ設計、ソフトウェア設計を行い、

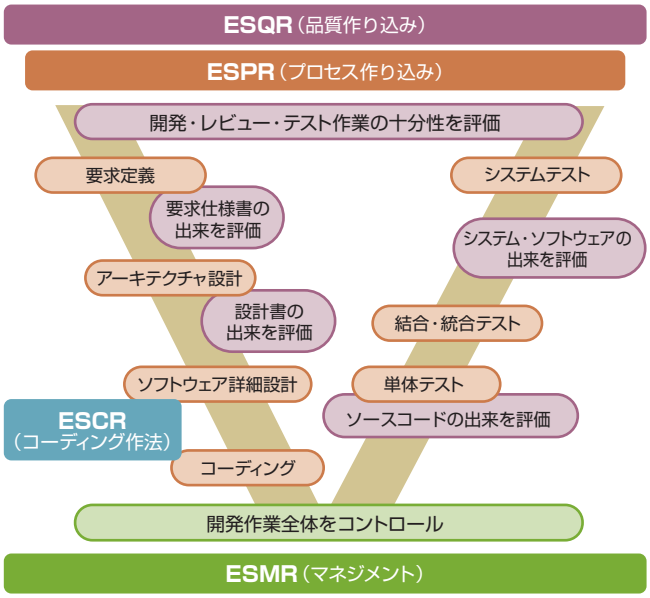


図1 V字モデルとESxR

※1 ESxR：Embedded System development exemplar Reference。組込みソフトウェア開発に関する各種開発技術リファレンスの総称。ESCR、ESPR、ESMR、ESQRなどで構成される。
※2 ESCR：Embedded System development Coding Reference。2006年5月発行。
※3 ANSI C：ISO（International Organization for Standardization：国際標準化機構）とANSI（American National Standards Institute：米国規格協会）が協同で標準化したC言語の規格。
※4 ESPR：Embedded System development Process Reference。2006年10月初版発行、2007年11月改訂。

実装を行う。これら各工程をお行儀良く行い、作り方を管理し、できたものを確認するという行為をきちんと実施することで効率的に質の良いソフトウェアを作り出すことができる。ESxRシリーズはこれらの活動を円滑に組織的に行うための四つのガイドを提供している。

【ESCR※2：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]](以下、ESCR)

ソフトウェアの根幹を成すのはソースコードである。多人数による開発形態が主流となっている現場には様々なスキルを持った人々が参画している。そのような中、高品質なソースコードを開発維持するために、ESCRではC言語の規範であるANSI C※3ではエラーとはならないが、トラブルを引き起こしがちな記述を避けるためのルールを作法という形で整理している。それらの作法を信頼性・保守性・移植性・効率性の四つの観点から整理分類しているのが大きな特徴となっている。また、ルールにのっとった例と、適合しない例を具体的なソースコードを用いて説明していることから、新人教育向けに利用されることも多い。

【ESPR※4：組込みソフトウェア向け開発プロセスガイド](以下、ESPR)

ソフトウェアを効率的に開発し、かつその品質を確実なものとしていくためには、お行儀良く作る、すなわち開発の各段階で適切な作業を適切な順序で実施することが必要である。

ESPRではソフトウェアの開発を円滑に進めるためのプラクティスを簡潔に整理し、そこで行うべき作業とその作業へのインプット／アウトプットを具体的に説明している。

【ESMR※5：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]](以下、ESMR)

一般にものづくりを行う上で、メンバがある一定以上の数になると、それを組織的に動かすための仕組みが必要となる。ソフトウェアを作る上で良いPDCA※6を回すには、計画を立てることが最初の一歩となる。ESMRは計画書に盛り込む内容について整理し、解説を行っている。また、実際に使うことのできるテンプレートも公開している。

【ESQR※7：組込みソフトウェア開発向け品質作り込みガイド]

ソフトウェアにはある一定以上の品質が期待されている。製品に品質を作り込むためには、品質を見えるように定義し、目標を定め、開発過程できちんとコントロールしていくことが肝要である。ESQRはとくに信頼性に重点を置いて、対象となるソフトウェアに求められる品質のレベルを分析・表現する手法を紹介している。更にそのレベルに応じてソフトウェアの開発過程で定量的、定性的に品質をコントロールする手法を整理している。

3. 四つのESxR：それぞれの適用方法

ESxRはそれぞれ独立して使うこともできるが、図2に示すように相互に関連させて使うこともできる。図2は、矢印の先がフレームであり、そこに対して情報を提供するものが矢印の元となっている。実線、破線はつながりの程度を表す。幾つかの例を使って考えてみよう。

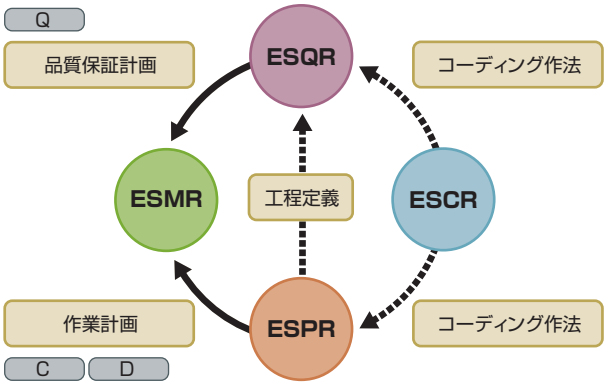


図2 ESxR関係図

プロジェクト詳細計画を立案する

【ESMR←ESQR、ESPR】

ESMRはプロジェクトの開発計画書について整理解説している。詳細開発計画の場面ではQCD※8の観点で品質保証計画、作業計画を考える(図3)。

〈作業計画〉 作業計画を立てる上で作業を洗い出し、個々

※5 ESMR：Embedded System development Management Reference。2006年11月発行。
※6 PDCA：Plan, Do, Check, Action。計画(P)、実行(D)、評価(C)、改善(A)の各ステップをひとつのプロセスとして回すマネジメント手法。
※7 ESQR：Embedded System development Quality Reference。2008年12月発行。
※8 QCD：Quality, Cost, Delivery。製造業の生産管理における基本要素。品質(Q)、費用(C)、納期(D)。

プロジェクト詳細計画	
Chapter4 リソース計画	4.1 開発規模と工数の計画 4.2 人員計画 4.3 設備、機器等の調達計画 4.4 プロジェクトの人員研修計画 4.5 予算計画書
Chapter5 作業計画	5.1 開発作業の洗い出し 5.2 開発作業の順序付け 5.3 開発作業担当者の割付 5.4 作業計画
Chapter6 品質保証計画	6.1 品質目標 6.2 品質保証の体制と仕組み 6.3 品質保証に関する主要なイベント
Chapter7 リスクマネジメント	7.1 リスクマネジメントの方針と仕組み 7.2 リスク一覧表

図3 ESMR：プロジェクト詳細計画表

5.1 開発作業の洗い出し

対象ソフトウェアの開発を進める上でプロジェクトとして実施する作業とその作業の結果作成される成果物を洗い出す。

● **記述すべき内容**

(1) ID（各作業を示す一連番号）
(2) 作業名称（内容が明確に理解できること）
(3) 作業標準（作業が従うべき社内ルール等）
(4) 開始条件
- 入力（仕様書、設計書、プログラム等。同一プロジェクト内での中間成果物とプロジェクト外から導入するものがある）および必要となる環境（ツール類、作業場所等）

(5) 終了条件
- 成果物
- 成果物の検証方法

(6) 備考
- 作業内容の説明、特に重要な作業項目、仕様未決等により後日変更の可能性があることを明確化する等で使用。

図4 ESMR：開発作業の洗い出し

の作業をどのように行うかということを決めるには、ESPRで定義している各工程が大いに参考になる。

ESMRでは、「5.作業計画　5.1　開発作業の洗い出し」で記載すべき内容を説明している。すなわち、作業名称・作業標準・開始条件・終了条件を明確にすべき、と記述がある（図4）。

これに対し、ESPRでは個々の工程で何を準備し、何を行い、何をアウトプットとするのか、ということが詳細に述べられている。例えば、コーディング工程は「4.1.2　プログラムユニットの実装」として下記のように入力及び出力をそれぞれ明確にしている（図5）。

〈入力〉
(SW305)　ソフトウェア詳細設計書
(SU1002)　ソフトウェア開発環境
(SU601)　不具合管理票（不具合修正の場合）

〈出力〉
(SW404)　プログラムユニット

作業内容、開始条件（入力）、終了条件（出力）が解説されているので（各入出力の頭にある記号は、その書類を作成するプロセスを示すものとなっているので芋づる式にたどることができる）、作業計画はこれらを参照しながら作ることができる。

4.2.1 プログラムユニットの実装

入力	概要	出力
(SW305) ソフトウェア詳細設計書 (SU1002) ソフトウェア開発環境 (SU601) 不具合管理票（不具合修正の場合）	プログラムユニットを実装する。	(SW404) プログラムユニット

参照情報　コーディング作法・規約

図5 ESPR：プログラムユニットの実装

〈品質保証計画〉 ESMRにおけるソフトウェアの品質保証計画の冒頭では、品質目標として記述すべき内容に、対象成果物や作業の明確化、品質指標の選定、目標値の設定を決めること、の3点を挙げている（図6）。

6.1 品質目標

対象ソフトウェアを利用するユーザやコンテキストを考慮し、製品としてどの程度の品質を目標に開発を進めるかを明確にする。

● **記述すべき内容**

品質目標の基本は、数値などで定量的にソフトウェアの品質を押さえることにある。このためソフトウェア開発の過程で作成される成果物ごとに何らかの評価指標を活用して、その目標値を設定することになる。

たとえば、対象がテスト工程のテスト成績書などであれば、そこに記載されるソフトウェアテスト段階での不具合数や不具合の修正率なども目標値として設定することができる。品質目標の設定では、

- 対象とする成果物や、対象とする作業（工程）を明確にする。
- それぞれをどのような品質指標（メトリクス）で押さえていくかを明確にする。
- 個々の品質指標の目標値を設定していく。

図6 ESMR：品質目標

これに対し、ESQRでは対象システム及びプロジェクトを分析し、品質指標を選択し、品質目標を設定するというプロセスを詳細に解説している。従って品質保証計画を立てる場面ではESQRで定義している手法をベースに考えるとよい（図7）。

コーディング作法を参照する

ソースコードの記述スタイルは人により様々であり、書き方のルールをとくに決めていない場合、同じ内容を記述したとしても100人の開発者がいれば100通りの書き方となる。これにより、次のような問題が起きてしまう。

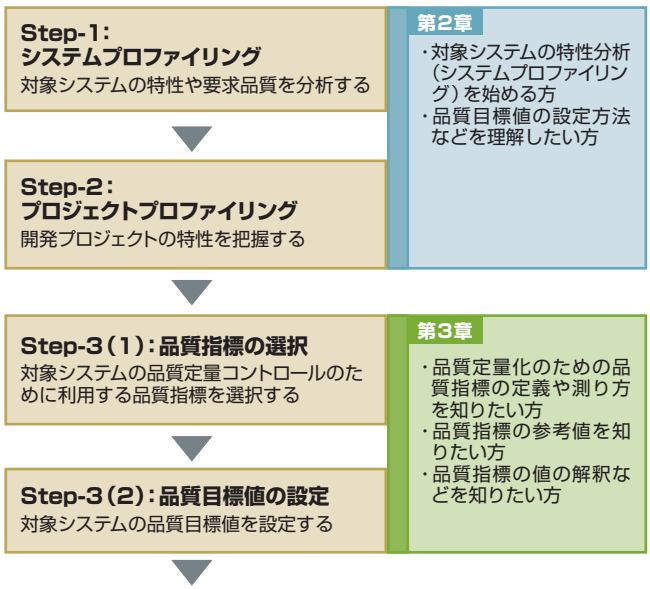


図7 ESQR：品質目標値設定の流れ

- **ソースコードをレビューしようとした場合、他人の書いたソースコードが自分の流儀と異なるために理解しづらく、不具合を見逃しがちになる。**
- **ソースコード行数を計測しようとした場合、空行などの入れ具合が人によって異なるため、同じ内容だとしても計測値に違いが現れてしまう。**

皆さんの組織ではこのようなことはないだろうか。組織内で共通のコーディングルールを決めておけば、ある程度一定の書きぶりになり、ソースコードの扱いが楽になる。また、ムラの無い計測、評価が可能になる。ESPR,ESQRでも以下のように提唱している。

〈ESPR〉「SWP4　実装及び単体テスト」の項で、コーディングについての記載があり、ここで以下のように記述している。

> 実装では、あらかじめ採用するコーディング作法・ルールを決めておく。

〈ESQR〉 ソースコードの品質を見る指標として、コードボリューム品質評価指標、コード特性品質評価指標があるが、そこへの説明として、以下のように記述している。

> 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。

これらに対し、ESCRでは一般的なコーディングスタイルのヒントとして保守性のルールの中に、「M4.1 コーディングスタイルを統一する。」がある。例えば、以下のような詳細な記述がある。

> （1）波括弧（{ }）の位置
> （2）字下げ（インデンテーション）
> （3）空白の入れ方
> （4）継続行における改行の位置

また、もう一歩踏み込んだ、コードの理解のしやすさという点では保守性の作法として下記五つが挙げられている。

> 保守性1：他人が読むことを意識する
> 保守性2：修正し間違えないような書き方にする
> 保守性3：プログラムはシンプルに書く
> 保守性4：統一した書き方にする
> 保守性5：試験しやすい書き方にする

例えば保守性のルールM1.4では「演算の優先順位がわかりやすいように記述する。」とあり、その解説が記述されている（図8）。すなわち、どのように記述すればよいかの例も掲載されている。

このように、ESCRを使うことによって、組織自身のコーディング規約を策定でき、安定した開発（ESPR）、ムラの無い測定と品質確保（ESQR）が可能になる。

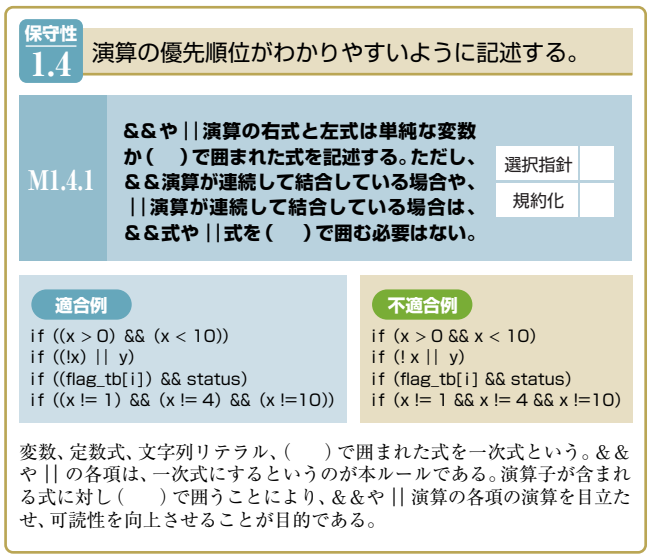


図8 ESCR：保守性ルールM1.4

4. ESxRの想定読者と活用シーン

更にESxRは役割により、また開発の工程により、様々な場面で使うことができる。役割ごとの使用場面を見てみよう。以下の図で、色の濃さはかかわりの深さを示している。

【経営者】

ソフトウェア開発の特徴を知り、
開発効率化、人材育成等に活かす

ソフトウェア開発の現場にはソフトウェア出身の上位層は少ないと、各企業からよく相談を受ける。皆さんの職場はどうだろうか？　このような悩みは、ソフトウェア・エンジニアリング固有の条件や悩みを上層部と共有できにくい、ということから発せられるようである。

ソフトウェアはハードウェアに比べると、設計から稼働に至るまでの時間が短い。かつ、人類史上初めてと言ってよいほどの複雑な系を手軽に作れてしまう。ところが、いったん不具合が潜んでしまうとそれを見つけ出すことは至難の業である。このように、非常に便利であり、なおかつ非常に危険な性質を併せ持つのがソフトウェアである。目に見えないソフトウェアを信頼性高く安全に作るために、開発者はいろいろな手段を尽くす。それらについて経営者の理解があることが非常に重要であることは言うまでもない。ESxRはソフトウェア開発における作業の進め方を理解していただくのに役に立つ。また、計画段階での開発承認の精査のための参考書として、ソフトウェア・エンジニアと会話するためのツールとしても役に立つのではないだろうか。

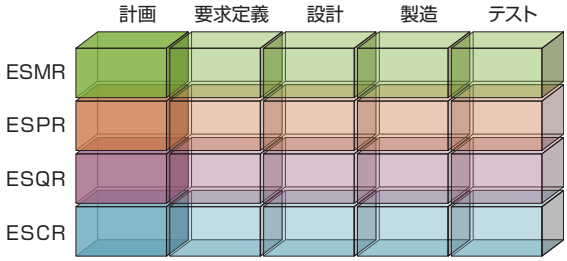


図9 経営者におけるESxR活用の場面

【管理者、グループリーダ】

ルール立案、計画立案の参考書として、
また開発段階でのマネジメント、品質コントロールに活かす

複数のメンバでソフトウェアを開発している組織では暗黙的／明示的にかかわらず何らかのルールがある。ルールは使う人がその意図を理解していないと、ただ従うだけの無為な作業となってしまう。場合によっては生産性や士気を低めてしまう原因にもなり得る。明文化されたプロセスやコーディングのルールがない組織ではESxRをヒントにすることによって自組織に役に立つルール作りをすることができる。また、既にルールのある組織ではESxRを参考にしながら、いったんそれらのルールを見直し、形骸化（けいがい）していないか、有効活用できているかなどを背景に潜む意味なども含めて見直すためのツールとして役立てることができる。

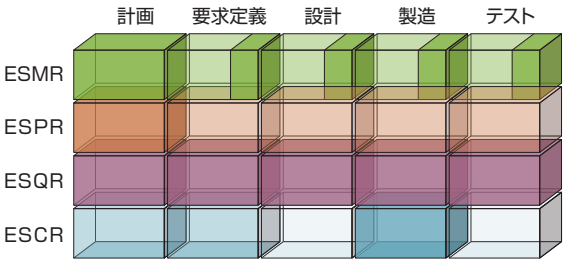


図10 管理者、グループリーダにおけるESxR活用の場面

【開発者】

作業のヒントを知り、効果的な作業のために活かす

展示会などでESxRの紹介をしていると、まず、どれから読んだらよいですか？　という質問をよく受ける。質問をする人は新人や入社数年目の方が多い。このような質問には、たいていこのように答えている。

「開発言語としてCをお使いであれば、まずコーディングのお作法を示したESCRをおすすめします。コーディングを行う際にすぐに役立てることができます。その次におすすめするのはESPRです。開発者であれば、言われたことそのままに作業し、ドキュメントなども作っているということが多いと思いますが、それらひとつひとつには皆、やらなければならない理由があります。ESPRからその理由の背景を読み取ることができます。同じように、ESMR、ESQRはマネージャさんが計画を出せとか、記録を取れなどの指示をしてくる背景を知ることができます。背景を知ることによって、本当は何を行わなければならないか、どのように見えるようにしたらよいか、工夫できますよね。」

ESxRのすべてを熟読する必要はない。上手にESxRの良いところをつまみ食いして、真の品質向上に役立ててほしい。

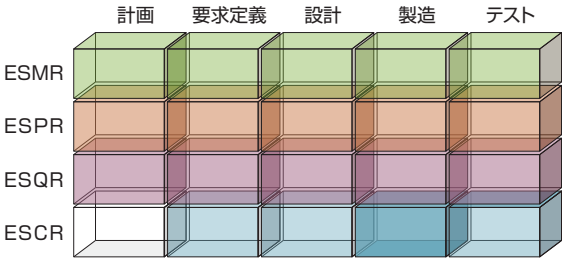


図11 開発者におけるESxR活用の場面

5. 利用上の注意点

「ESxRに準拠するにはどうしたらよいのですか」という質問をいただくことがある。ESxRシリーズはReferenceと称している通り、参照情報であり、所謂「従うべき標準類」とは若干性質を異としている。認証を行うことを目的としたものではなく、ソフト

ウェア開発作業のお手本や良いプラクティスを体系化、整理して紹介し、開発現場が少しでも良くなること、世の中のソフトウェアが少しでも良くなることを目的として作られている。

ESxRは開発を進める場面での参考資料として、別の場面では教材として、またある場面では自組織のルール作りを行う際のたたき台としてなど、自分の職場に合うようにどんどんカスタマイズして使ってほしい。有効に使うためには、マネージャあるいは専任者が開発者に押し付けるものではなく、使う人全員が理解していただきたい。きちんと理解することで開発者はルールの言外の意図まで読み取って自主的に動けるようになり、管理する側もより有効なデータを得られるようになるのである。

6. ESxRの普及の状況

ESxRについては、主催セミナーや組込み関連の展示会など様々な場で紹介させていただいており、既に実際の開発現場で利用していただいている企業もある。以下、我々がお聞きした中で典型的な利用法などを少し紹介する。

【ESCR】

ESCRは比較的導入が容易なこともあり、かなり多くの企業で利用されているようである。例えば、ある企業では派生開発品のソースコード再利用性を向上させるため、ESCRを参考に、保守性に特化してコーディングルールの見直しを行っている。これにより、従来よりもソースコードの品質が向上し、再利用効率を上げることができたと聞いている。また、この会社ではコーディングルール見直しに際し、コーディングルールを開発要員全員に教育することで、メンバの意識向上も図ることができたとのことである。

【ESPR】

ソフトウェア開発プロセスの整備はどの企業にとっても大きな関心事のようで、SECにはESPRに関する問い合わせも数多く寄せられている。ある企業では、主に設計などの開発上流工程の作業について、ESPRを参考に、開発プロセスの不足部分を再見直しし、併せてドキュメンテーションを整理することで、上流工程での品質可視化に挑戦していると聞いている。

【ESMR】

ESMRについてはSECがより積極的にかかわる実証実験として、自動車関連のJASPAR※9プロジェクトで適用評価をお願いしてきた。この実証実験ではプロジェクト参加企業の開発に関する標準的な計画書フォーマットを検討する際のたたき台として使われており、各社、計画書を作って相互にレビューしている。これらの活動を行うことにより、プロジェクト共通の計画書を策定し、計画的にプロジェクト全体の作業を進めることができるようになったという効果を生んでいる。

【ESQR】

ESQRは発表してから本稿執筆時までまだ1年を経過していないが、我々の予想を超えて広く利用され始めている。既にSECには多くの企業から自社で計測した数値とESQRの参考値との違いに関する質問をいただいている。また経済産業省とSECが共同事務局で取りまとめた「重要インフラ情報システムの信頼性」の報告書骨子でも、ESQRで提案しているシステムプロファイルの考え方が採用されており、エンタプライズ情報システムの分野でも関心を持っていただいているようである。

7. まとめ

ESxRシリーズはそれぞれ、セミナー、講演等の普及活動を通じて、また、企業への個別訪問によるヒアリング、実証実験等を通じて広くご意見を伺っている。

お問い合わせのやり取りをする中で、自社のデータは公開できないが、他社のデータは知りたい、という会社が非常に多い。ノウハウという壁に阻まれ、データを出すことにためらいがあるのは理解できるのだが、それではいつまでたっても業界全体が良くなることができない。社名が出ると困るという場合、SECではヒアリング後、値や事象のみを使わせていただき、社名は伏せた形で公開することも行っている。気軽にお話しいただけるよう願っている。

以上、簡単にESxRについて解説したが、興味を持たれたものがあれば本項以降で更に詳細な説明、具体的事例の紹介等を行っているので参照されたい。

本活動を通して、世の中のソフトウェアの品質が少しでも向上し、携わる人が少しでも幸せになることができれば幸いである。

（IPA/SEC　吉澤　智美）

※9 一般社団法人 JASPAR：Japan Automotive Software Platform and Architecture。自動車関連ソフトウェアの非競争領域を、国産メーカ各社が協調して開発する取り組み。

Part 2-2 実装品質の改善

「ESCR：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]」とは

プロダクトの実装品質向上のためには、開発者の暗黙の了解でコードを記述するのではなく、コーディング規約を定め、規約に基づいてコードを記述することが有効である。SECでは、コーディング規約を策定している方を対象にした「ESCR※1：組込みソフトウェア向け開発コーディング作法ガイド[C言語版]」（以下、ESCR）を刊行した。本ガイドは、品質特性に対応して分類した作法と、それに対応するルール群から構成され、ルール群の中から必要部分を取捨選択することにより、目的に見合ったコーディング規約を策定することが可能となった。本稿ではコーディング作法ガイドのあらましを紹介すると共に、必要性などを論じる。

1. ソフトウェアの実装品質とは

現在の情報処理技術を用いて計算機システムを構築する場合、計算機に指示を与えるためのプログラムの開発は必要不可欠である。通常、プログラムの多くはC言語、C++、Javaなどのプログラミング言語を用いて作成される。図1はプログラムの一部をC言語で記述した例である。ここで図1の(a)と(b)を見比べていただきたい。この二つのプログラムについて、幾つ違いや誤りを探していただけるだろうか。

(a)

```
if(x == 1){
    func1();  /*ＸＸ処理   */
    func2();
}
r = func3();  /*ＹＹ処理   */
if(r == 1) {  /*ＸＸに適合 */
    func4();  /*ＺＺ処理   */
}
```

(b)

```
if(x == 1)
    func1();
    func2();
r = func3();
if(r == 1)
func4();
```

図1 C言語プログラミングの例

実はこのような単純なプログラムであっても、誤りは入り込む。また、このプログラムを再利用しようとした場合、(b)のプログラムはコメントなどが入っておらず、極めて理解しづらく再利用しにくい。これらのことを読み取っていただけただろうか。このように計算機システムの中心的な要素であるプログラム、そしてそれを実現するソースコードは、極めて誤りが入りやすいという特性を持っている。

さてここで図2を見ていただきたい。この図は経済産業省が毎年実施している「組込みソフトウェア産業実態調査報告書」※2の2009年版から引用したものであるが、組込みシステムの開発において発生した不具合の原因工程を調べたものである。ご覧いただくとわかるように、不具合の約3割※3はソフトウェア実装・単体フェーズであるコーディングの段階でその原因を作り込んでいるのが読み取れるかと思う。

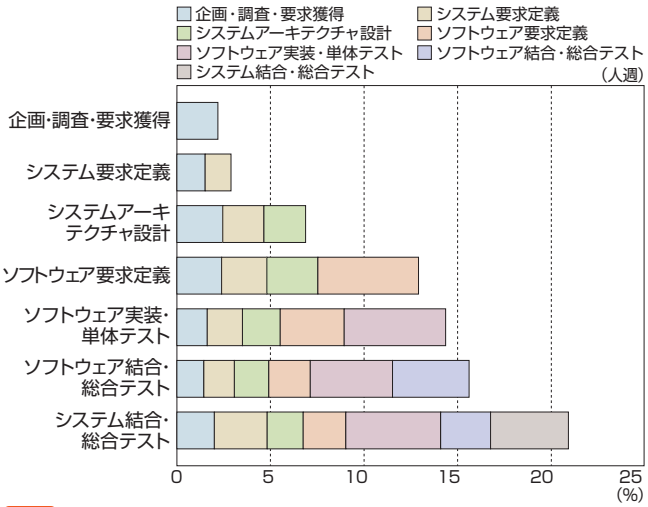


図2 ソフトウェア不具合発見工程別修正工数と発生工程の内訳
（出典「2009年版組込みソフトウェア産業実態調査報告書」プロジェクト責任者向け調査）

これは先に述べたソースコードの特性を如実に反映した結果であり、近年、ソフトウェア開発の世界では上流フェーズが注目を集めているが、実はシステム作りの中心的な作業であるプログラミングに大きな落とし穴が有ることがご理解いただけるのではないだろうか。

※1 ESCR：Embedded System development Coding Reference
※2 「組込みソフトウェア産業実態調査報告書」：2009年版は経済産業省のWebサイト（http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2009software_research/index.htm）からダウンロードできる。
※3 約3割：コーディングフェーズの対象となる「ソフトウェア実装・単体テスト」「ソフトウェア結合・総合テスト」の合計が約3割となる。
※4 ISO/IEC 9126：ISO（International Organization for Standardization：国際標準化機構）とIEC（International Electrotechnical Commission：国際

2. ソースコードの品質

さてこのような事実を目の当たりにすると、プログラムの品質、すなわちソースコードの品質を高めていくための施策導入が急務であることがおわかりいただけるだろう。しかし、こうした施策を考える前に、まずソースコードの品質とはどのようなものかを少し考えたい。

一般的にソースコードの品質というと、「バグが無い」など信頼性にかかわる側面が真っ先に思い浮かぶかも知れない。しかし、ソースコードの品質は必ずしもそれだけではなく、下記のように幾つかの視点を考えることができる。

- **ソースコードの保守のしやすさ（保守性）**
- **ソースコードの移植のしやすさ（移植性）**
- **ソースコードの誤りのなさ（信頼性）**
- **処理時間や資源に考慮したソースコードのつくり（効率性）**

通常、ソフトウェア・エンジニアリングの世界で、ソフトウェアのプロダクトとしての品質は、より広い概念で捉えられている。このソフトウェア製品の品質概念を整理したものが、ISO/IEC 9126[※4]であり、これをJIS化したものがJIS X 0129 [※5]である。

JIS X 0129では図3のように、ソフトウェア製品の品質にかかわる特性（品質特性）に関しては、「信頼性」「保守性」「移植性」「効率性」「機能性」「使用性」の六つの特性を規定している。このうち「機能性」と「使用性」の2特性については、より上流の設計段階以前に作り込むべき特性と考えられ、実装時、すなわちソースコード作成段階では、「信頼性」「保守性」「移植性」「効率性」の4特性が深く関係すると考えられる。

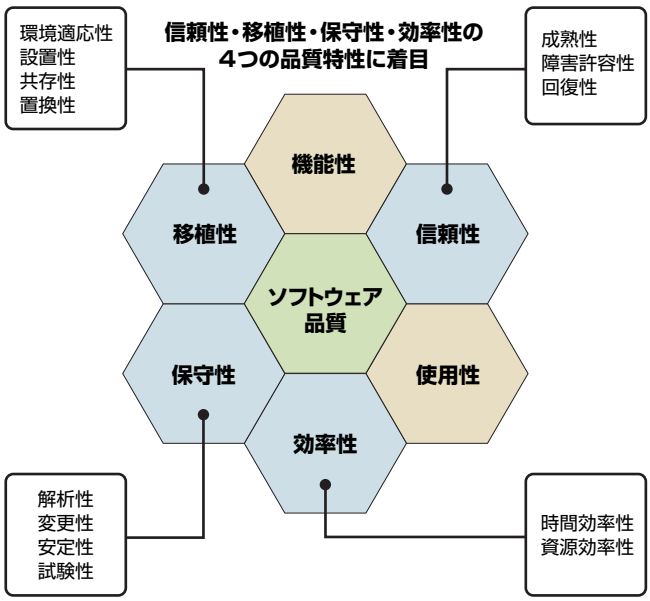


図3 コーディングの観点からの品質特性（JIS X 0129）

3. ソースコードの品質を向上するためには

このようにソースコードの品質と言っても、前述のように様々な切り口が有ることがおわかりいただけただろうか。このような多様な切り口を持つソースコードの品質を改善し、向上する近道はないのだろうか。SECではこうした課題を解決するために、様々な企業からのヒアリングや、委員会などによる多くの方々との意見交換の実施を行っている。ソースコードの品質に関して言えば、ソースコード・レビューやソースコード解析ツールを用いたコードチェックなどを行っている企業が多いようである。しかし、我々がヒアリングしたある企業ではコード解析ツールを導入したものの、ツールによって指摘されるワーニング（警告）が数多く出て困っており、こうした悩みの相談を受けたケースもあった。

このような様々な例をもとに考えると、コードレビューやコード解析ツールでソースコードの品質をチェックするということはできるが、まずそれ以前に、いかに品質の高いソースコードを実装していくかという側面にもう少し時間を割いてもよいのではないかと思われる。具体的には、

①社内やプロジェクト内で、誰が書いてもある程度、同じ品質レベルを維持できるようにするための基準を用意する。
②ソースコード作成の時点で①の基準をしっかりと守らせる
③作成されたコードを、①の基準との準拠性という視点でチェックしていく

という作業プロセスを確立していくことが近道ではないだろうか。

4. ソースコードの標準化

ソースコードの標準化が必要な理由

ソースコードは、C言語などのプログラミング言語を利用して記述される。プログラミング言語はいわゆる、言語のひとつであることに変わりなく、あるひとつのことを表現するにも、その言語を使う使い手によって様々な言い回し（表現）を取ることが可能である。

例えば、「単純に1から10までの整数を合計する処理」を取り上げると、

①for 文を利用した繰り返し処理で実現する。
②while 文を利用した繰り返し処理で実現する。
③if 文とgoto 文を利用して実現する。

など、様々な方法を考えることができる。このうち、①や②の方法を採れば、ソースコードの構造は比較的簡潔なものとなるが、③のようにgoto文などを利用すると論理構造はやや複雑になる。goto文で戻る先の指定を誤ったりすると、思わぬ不具

電気標準会議）によって、1991年に策定されたソフトウェア品質特性（Quality Characteristics）の規格。正式名は、Information technology software product evaluation:Quality characteristics and guidlines for their use。

※5 JIS X 0129:JIS（日本工業規格）が、1994年にISO/IEC 9126をもとに策定した規格。

合を誘発することがある。このような単純な例でも、ソースコードの書き方次第では、プログラムの複雑性などに影響することがわかる。

この事例でもわかるように、プログラミング言語という、明確に文法が決められている言語を利用する場合でも、開発する技術者個々人の癖や経験によって、ソースコードの記述には様々なバリエーションが生まれてしまう。企業活動の一環として開発されるソフトウェアが開発者個人の能力などによって、バリエーションが増えたり、品質にばらつきが生まれたりしてしまうのはできれば避けたい。

ソースコードの標準化のための方法

近年の組込みソフトウェアのほとんどは、複数人の技術者で開発されている。こうした場合、開発者が10人いれば10通りのソースコードの書き方が存在することになる。この状態を放置すると、前述したように、ソースコードの記述様式や品質のレベルはバラバラになってしまう。こうした事態を避けるためには、関係者間でソースコードの書き方などのルールを決め、守っていくことが重要である。このようなソースコードの書き方に関するルールをコーディング規約と呼ぶ。

5. ESCRは

前述のような背景の下、SECでは経済産業省組込みソフトウェア開発力強化推進委員会と連携して、ESCRを作成した。このガイドは、最終的なソフトウェアの品質に大きく寄与するソースコードの品質を確実に押さえることを念頭に、実際の企業の開発現場でソースコード作成の規範となる、コーディング規約を策定するための参考にしていただくことを目的としている。具体的には作法は「JIS X 0129-1ソフトウェア製品の品質第1部:品質モデル」に準拠した品質概念をもとに、作法概要、作法詳細に分類・階層化しそれぞれの作法にC言語に対応したルールをその必要性と共に参考情報として提示している。

さてここまで読んだ段階で恐らく多くの読者の方々はコーディングにおける作法とは何かという疑問を抱いているのではないだろうか。これについてはESCR策定委員会の中でも様々な意見に分かれたが、最終的に「コーディング作法とは経験に基づいた品質を意識したソースコードの記述方法」という定義を与えている。これは例えば、「保守性を考慮した場合に、内部の論理構造をできるだけシンプルにしておいた方がよい」といった抽象概念で整理されるものである。この点において作法は基本的にC言語などの個別のプログラミング言語には非依存なものとして捉えることができる。しかしながら実際のコー

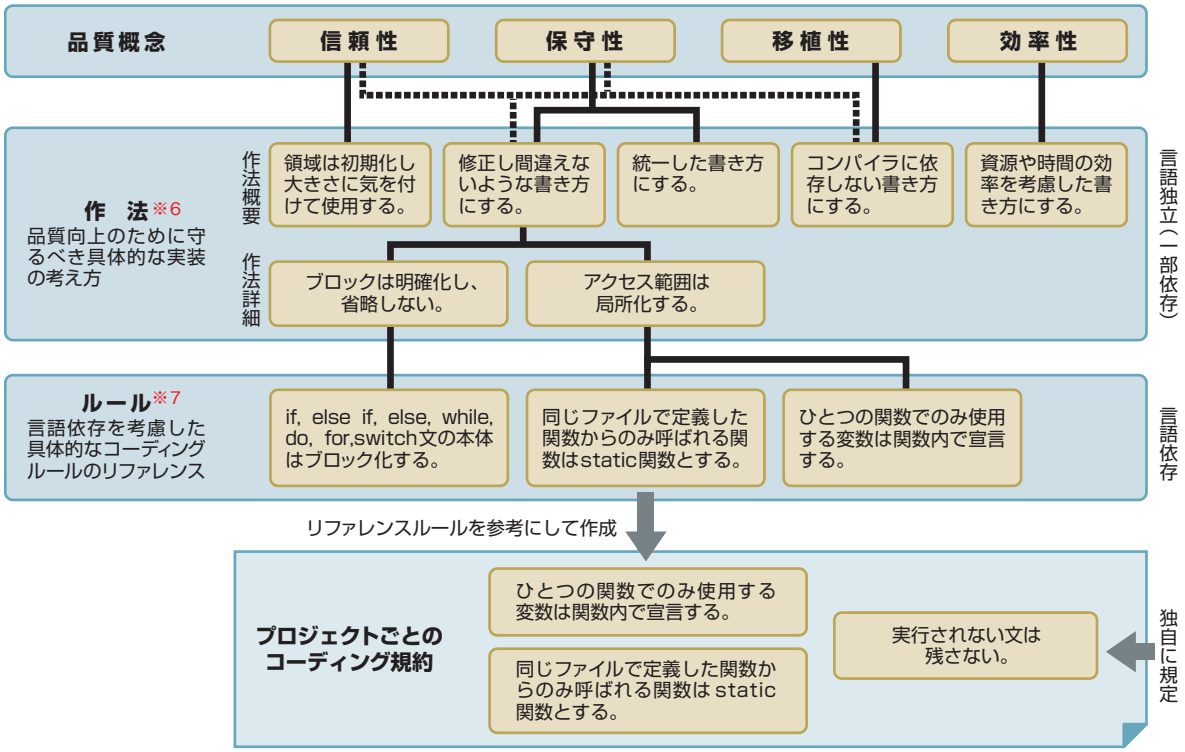


図4 品質概念、作法、ルールの関係

※6 作法:ソースコードの品質を保つための慣習、実装の考え方であり、個々のルールの基本概念を示す。作法概要、作法詳細に階層化して示している。

※7 ルール:守らなければならない、具体的なひとつひとつの決め事であり、コーディング規約を構成する。このガイドではリファレンスとして示している。なお、ルールの集まりもルールと呼ぶことがある。作法、ルールの多くは、複数の品質特性と関連するが、最も関連の強い特性に分類している。

ディング局面では、例えば「内部論理構造をシンプルにせよ」という作法を意識した場合に、どのようなコードの書き方をすべきかというより具体的な情報が必要となるのは言うまでもない。このため、個々の作法を達成するための具体的なコードの書き方をルールとして用意しておく必要がある。例えば、この例の場合、「goto文は利用しない」というソースコードの具体的な書き方を指定するルールを用意することができる。このようにESCRでは、品質概念-コーディング作法-コーディングルールという階層構造を持たせることで、体系的にソースコードの品質向上を実現するためのソースコードの書き方を整理している(図4)。

6. ESCRの読み方

作法から作法詳細、ルールへの展開

さてここまでの予備知識をもとにして実際にESCRの中身を眺めてみよう。

図5はESCRに掲載されている保守性作法の一部であるソースコードの保守性を向上させるための「プログラムはシンプルに書く」という作法の展開を示している。実際の開発者にこうした作法を提示したとしても、それでは「シンプルに書く」と言われても実際にどのように書けばよいのかという疑問には答え切れていない。このため、更にこの作法の具体的なところを明確にするために作法詳細という概念でもう一段の具体的な作法内容を紹介している。例えば、図5では、「シンプルに書く」という作法概要に関して「構造化プログラミングを行う」「1つの文1つの副作用とする」「目的の違う式は分離して書く」など複数の作法詳細でより具体的な指針が提示されている。更に個々の作法詳細については、例えば「構造化プログラミングを行う」という作法詳細を具体化したルールとして五つのルールがひも付けされている。

ルールの読み方

次に個々のルールについてもう少し詳しく眺めてみよう。図5のルールの二つ目は「goto文の使い方」に関するルールを示したものである。ここでは、ルールとして「(1) goto文は使用しない」、「(2) goto文は多重ループを抜ける場合とエラー処理に分岐する場合だけに使用する」という二つのルールが併記されている。このような場合、実際にコーディングルールを規定する際には後述するように、この二つのどちらかを選択して規定していくといった使い方が想定されている。

また、ESCRではこれらのルールについて、実際にC言語で記述した場合のルールに適合したコードの例、ルールに適合しないコードの例を示すことで、ルールがソースコード上でどのように反映されるかを明示している。ひとつ別の例を見てみよう。図6は図5の三つ目のルール「continue文を使用してはならない。」というルールの適合例／不適合例を切り出したものである。このルールはプログラム論理が複雑にならないようにするた

【不適合例】

```
for(i=0; ループの継続条件 1;i++){
    繰り返す処理 1;
    if(継続条件 2){
        continue;
    }
    繰り返す処理2;
```

【適合例】

```
for(i=0; ループの継続条件 1;i++){
    繰り返す処理 1;
    if(!継続条件 2){
    繰り返す処理2;
    }
}
```

図6 「continue文を使用してはならない。」不適合例・適合例

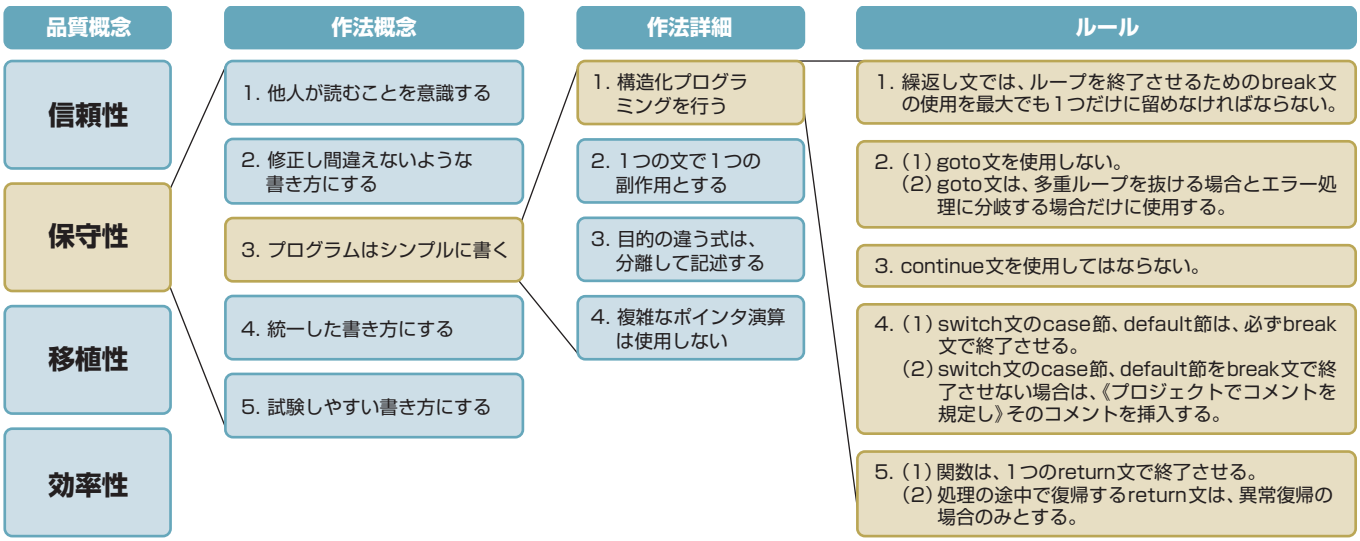


図5 コーディング作法の展開例

めの工夫であり、continue文を無くすことが目的ではなく、プログラムが複雑になる(上から下へ読めなくなる)ことを避けるための手段としてのルールである。実際にはcontinue文を用いることにより、可読性が向上すると考えられることもあり、論理をシンプルに表現できるかを考えて、ルールの採用可否をプロジェクトで判断する選択制ルールとしている。

7. ESCRの基本的な使い方

自組織のコーディング規約を策定する

ESCRはプロジェクトや組織のコーディング規約として直接利用するものではなく、コーディング規約を策定するためのガイドとして利用していただくことを念頭に編さんされている。実際に個々の組織におけるコーディング規約を整備する場合には、対象製品の特性・プロジェクトの性質を考慮し、作法・ルールの取捨選択やルールの具体化を行うことが必要となる。それでは、具体的に個々の組織内でESCRを利用してコーディング規約を策定していく流れを確認しておこう。全体としては、下記に示すように四つのステップを順に追うことで、組織のコーディングルールを策定していくことが可能となる。

【Step 1　作成方針の決定】

はじめにプロジェクトで作成されるコードの書き方を示すコーディング規約の作成方針を決定する必要がある。例えば安全性を重視し、便利であっても危険な記述方法は使用しないという書き方にするのか、危険な記述方法であっても注意して使用する書き方にするかなどが、この方針にあたる。

【Step 2　ルールの選択】

決定した規約作成の方針に従い、ルールの選択を行う。例えば、組込みソフトウェアの場合、異なるプラットフォームへの移植などを行う場合があるが、このような場合には移植性を重視し移植性に関するルールを多く選ぶなどの工夫をすることも必要になる。選択では各ルールの品質特性や各ルールの「選択指針」を参照することができる(図7)。

【Step 3　プロジェクト依存部分の定義】

既に具体的なルールの説明でも見たように本ガイドのルールには、以下の3種類のルールがある。

①規約としてそのまま使えるルール
②プロジェクトで規定する必要の有るルール
③プロジェクトの特性に合わせて、どのルールとするか選択する必要の有るルール

図5で紹介したgoto文の利用などに関する複数ルールの選択は③の典型的な例のひとつである。それぞれのプロジェクトの特徴などを考慮して、必要なルールの採用や選択を行っていく。

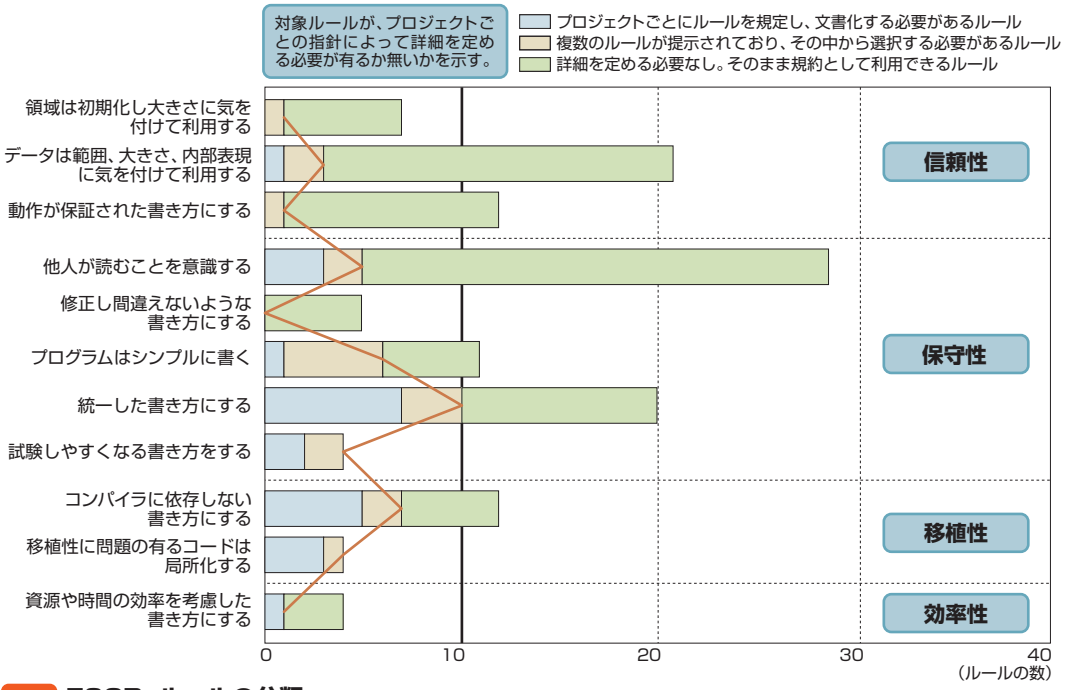


図7 ESCR ルールの分類

【Step 4　ルール適用除外の手順を決定】

実現する機能により、コーディング時に注目すべき品質特性が異なる場合がある（保守性よりも効率性を重視しなければならないなど）。この場合、定めたルール通りに記述すると、目的が達せられないなどの不具合になる可能性がある。このような場合に対応するべく、部分的に、ルールを適用除外として認めることを手順化しておく。

8.策定したコーディングルールの活用法

コーディング規約の活用場面

ここでは少し話を進めて、上記のような手順で策定したコーディングルールの組織内での活用・定着について考えてみよう。実際にコーディングルールが開発の中で利用される状況として、「開発者がコードを記述する局面」と「コードレビューの基準として利用する局面」という大きく二つの場面を考えることができる。

保守性 4.4　他人が読むことを意識する。

【作法】保1－1　使用しない記述を残さない。

4.1　**使用しない関数、変数、引数、ラベルなどは宣言（定義）しない。**

選択指針 ※8 ●
規約化 ※9

適合例

```
void func (int arg){
        /* ar未使用 */
```

不適合例

```
void func (void){
```

【備考】
使用しない関数、変数引数、ラベルなどは宣言（定義）は、削除し忘れたのか、記述を誤っているのかの判断が難しいため、保守性を損なう。
【関連ルール】
7.16

図8　コーディング作法・ルールの例

ソースコードを記述する際に利用

冒頭に述べたように、ソースコードを書くという作業は極めて属人性が高く、技術者の技量の差が出やすい作業である。一方で、工業製品の一部としてのソースコードは限りなく品質面での均質性を求められるという側面があり、どのような技術者がソースコードを書いたとしても、大枠、同水準の記述方法に従って記述されていることが必要となる。このため、様々な技術者が関与する開発において、前述の手順によって定められたコーディング規約を守ることにより、ソースコード作成時の属人性を排除することができる。また、これにより定められたソフトウェア品質を確保し、ソースコードの信頼性や保守性などを向上させることができると考えられる。これはソースコードの品質を直接的に向上させるという意味では非常に有効な利用の仕方と言える。実際にソースコードを書く際の参考としてコーディングルールを利用する場合には、プロジェクトメンバに事前に教育を行うなどの工夫をしておくことが望ましい。

ソースコード・レビュー時の基準として利用

小規模なソフトウェアだけでなく、大規模なソフトウェア開発では、一般にコードレビューやインスペクション※10を行うためのレビュー基準が必要になる。組織として策定したコーディング規約は、作成したソースコードを第三者の目で確認したり、品質面をチェックしたりする際のレビュー指針としても利用することができる。また、ESCRではルール以外にも、所謂コーディングミスに分類されるような凡例についても紹介を行っている。これらは一般的なコーディングルールとしては取り上げにくいものもあるが、ソースコード・チェックの際の確認事項としてチェック項目に含めて利用することもできる。

9. まとめ

以上、プロダクトとしてのソフトウェア品質の向上という視点から、ESCRについて概要を紹介した。ESCRは組込みソフトウェア開発をモチーフに、良い「ソースコードの書き方」のためのヒントを集めているが、組込みソフトウェア以外の開発でも参考にしていただける部分は多いと考えている。また、利用対象者という面でも、プログラミング経験の浅い方には、コーディングの入門書としてぜひご一読いただきたいと共に、経験豊富な方々にも組織としてのソースコードの標準化を意識しながら、そのエッセンスを読み解き、プロジェクトの仲間に広めていただきたい。ぜひ、皆さんの関係するプロジェクトや組織で、このESCRを参考にしていただき、プログラムの品質向上を推進していただければと考えている。

（IPA/SEC　大野　克巳）

※8 選択指針：コーディング規約を作成する際のルールの選択指針。
※9 規約化：対象ルールが、プロジェクトごとの指針によって詳細を定める必要が有るか無いかを示す。また、選択や文書作成を指示するルールもこの欄で示す。
※10 インスペクション：inspection。第三者による成果物（仕様書・プログラムなど）の検証。実際の動作はさせないで行う。

Part 2-3 プロセスの改善

「ESPR：組込みソフトウェア向け開発プロセスガイド」とは

SECでは高品質なソフトウェアを効率的に開発することを目的として、「ESPR※1：組込みソフトウェア向け開発プロセスガイド」（以下、ESPR）を刊行している。このガイドではソフトウェア開発を進める上で必要な作業を開発プロセスの視点で整理している。本稿では開発プロセスガイドのあらましを紹介すると共に、利用する上での注意点などを論じる。

1. ソフトウェアの開発プロセスとは

最初に開発現場をちょっとのぞいてみよう。読者の多くの方は、ソフトウェア開発に携わったことがあると思うが、「設計作業を始めようとしたら要求仕様書が無く、仕様は誰かの頭の中に有る」といったことや、会社から「開発プロセスが重要である」と言われているがドキュメントのフォーマットが決まっておらず、部門ごとにそれぞれ違っていたり、「プロセス改善」という言葉をよく耳にするが、そもそも実施すべきプロセスが決められていなかったり、決められていても不正確だったりという経験をしたことがないだろうか。

図1は「工程標準の定義状況」を示したものである。かなりプロセスに関心を持っているものの、まだまだ手が回っていない会社も少なくないようだ。

ソフトウェアに限らず、モノをひとつの製品として完成させるためには、様々な作業を積み重ねていくことが必要となる。ソフトウェアという製品を作り上げる上で「どのような作業を実施していくか」について、その作業の入力や出力、作業内容などを整理したものをソフトウェア開発プロセスと呼んでいる。

ソフトウェアの規模がそれほど大きくなかった時代は、開発プロセスをあえて意識しなくても、何とか製品としての形を実現できていた。しかし、近年のソフトウェア規模の増大の中で、ソフトウェアに起因する様々なトラブルや問題が発生するようになり、その原因のひとつとして開発プロセスの質がクローズアップされてきている。

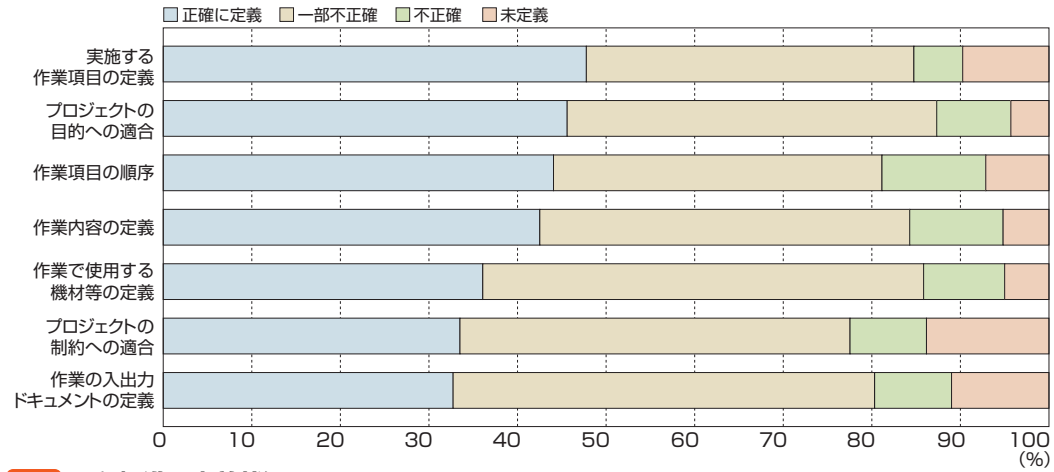


図1 工程標準の定義状況
（出典「2009年版組込みソフトウェア産業実態調査報告書」プロジェクト責任者向け調査）

※1 ESPR：Embedded System development Process Reference

ソフトウェア開発では、最初の要求獲得や要求定義に始まり、最終的なシステムテストに至るまで様々な作業を手際良くこなしていくことが求められる。こうした作業やその手順が人や組織によってまちまちだったり、必ずしも決められたやり方が存在しなかったりすると、結果として、作業の抜けや誤りなどが発生しやすく、非効率な開発を余儀なくされるといったことも少なくない(図2)。

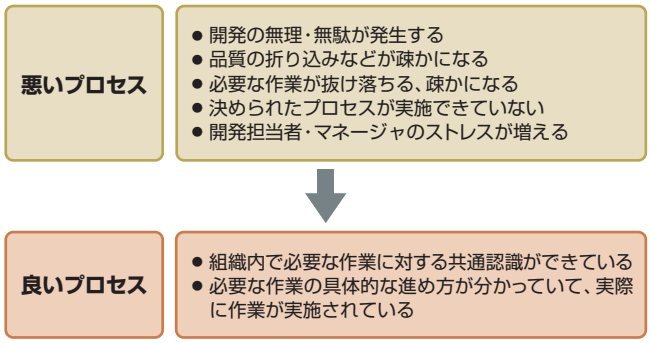


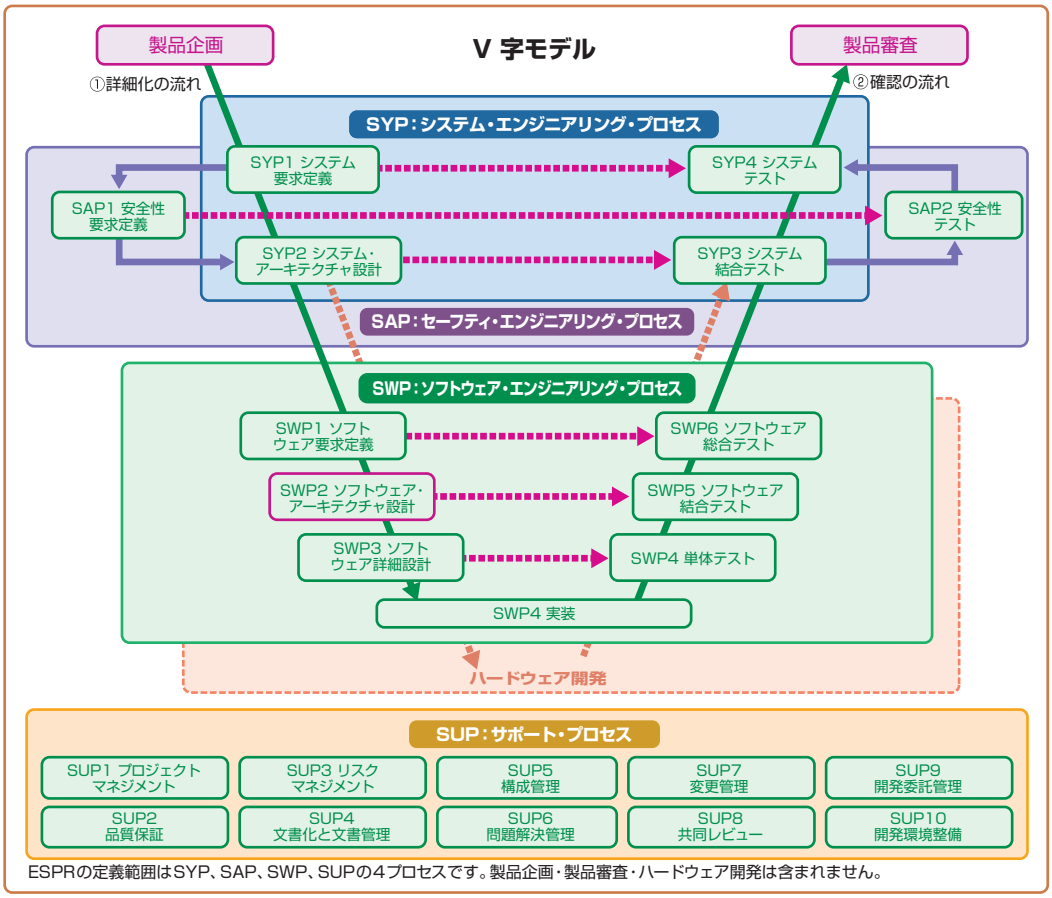
図2 悪いプロセスと良いプロセス

SEC組込み系エンジニアリング領域ではソフトウェア開発における開発プロセスの重要性を捉え、「高品質なソフトウェアを効率的に開発する」ことを目的に「開発プロセスガイド」を編さんした。具体的には、誰もが同じような手順と作業でソフトウェア開発をできるようにするため、より具体的な作業のレベルで開発プロセスを整理し、理解しやすい表現を用いている。

2.ESPRにおけるプロセス・作業の情報整理

それでは、開発プロセスガイドESPRの具体的なところを見てみよう。皆さんはソフトウェアの根幹部の構造を検討する際に、どのような情報をもとに、どのようなことを考えながら決めていくだろうか。ここでは「ソフトウェア・アーキテクチャ設計」を例に挙げて見てみよう。

まず、図3のV字モデル※2でこの作業が開発全体の中でどのような位置付けにあるかを確認しておこう。



ESPRの定義範囲はSYP、SAP、SWP、SUPの4プロセスです。製品企画・製品審査・ハードウェア開発は含まれません。

図3 V字モデルとESPR

※2 V字モデル：ソフトウェア開発のライフサイクルをV字で表現したモデル。

「ソフトウェア・アーキテクチャ設計」は、図3の中央にある「ソフトウェア・エンジニアリング・プロセス」において、開発するソフトウェアのアーキテクチャ（=動作［振る舞い］と構造）を決定していく作業、という位置付けになる。

ESPRは、より具体的な作業レベルを示すため、組込みソフトウェア（システム）開発に必要な作業を4階層（プロセス→アクティビティ→タスク→サブタスク）のフレームで整理し詳細化しており、それぞれ入力、実施内容、注意すべき事項、出力を掲載している。

図4は「ソフトウェア・エンジニアリング・プロセス」を詳細化した「ソフトウェア・アーキテクチャ設計」アクティビティを、ESPRから抜粋したものである。実際にアーキテクチャ設計を行う場合には、それ以前の作業で作成した成果物や検討した情報を参照する必要があるが、この場合には「ソフトウェア要求定義※3」（製品を実現するためにソフトウェアとして実現が必要となる要求を明確にする作業）で作成した「ソフトウェア要求仕様書」などをもとに行っていく。

そして、仕様書などの入力情報をもとに設計条件を確認した上で、ソフトウェア構造やソフトウェア全体の振る舞い、インタフェースなどを設計する、といった作業を実施していくことになる。

さて、これらの作業の中から「2.1.3 ソフトウェア全体の振る舞いと設計」を例に挙げて、更に詳しく見てみよう。これは、ESPRで最も詳細化したサブタスクのひとつである（図5）。

ESPRでは、作業として実施する事項を「実施内容」欄に具

図4 ESPRのプロセス

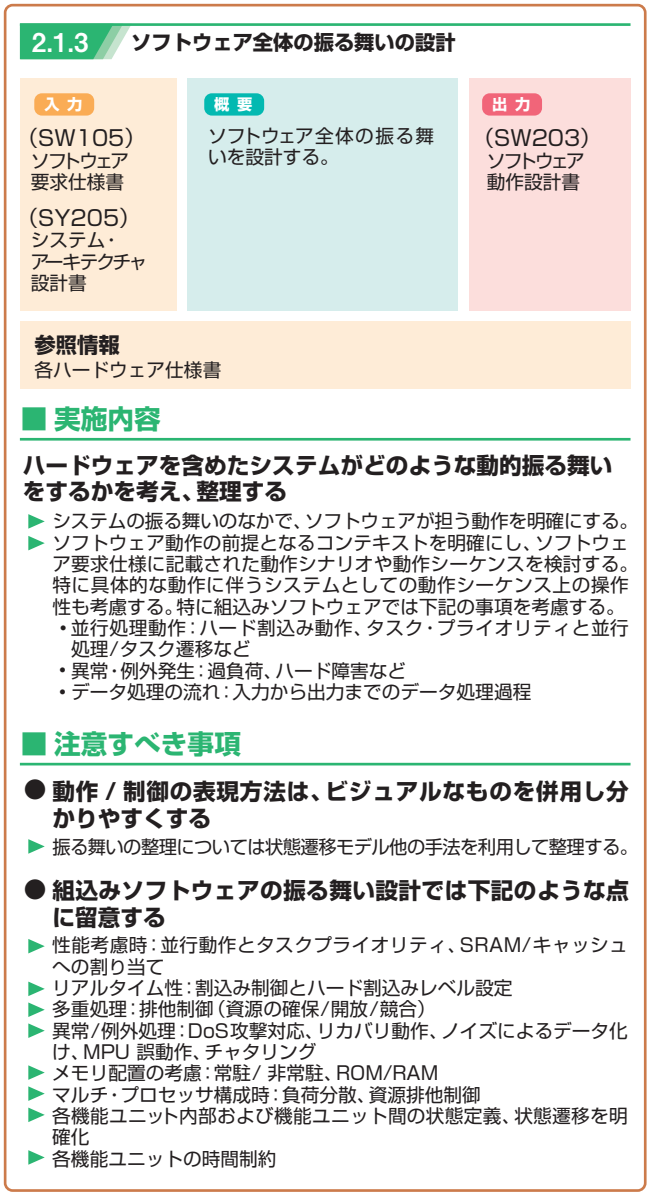


図5 ESPRのサブタスク

※3 ソフトウェア要求定義：製品を実現するためにソフトウェアとして実現が必要となる要求を明確にする作業。

体的に整理してある。この例では、ソフトウェア要求仕様書に記載された動作シナリオや動作シーケンスをもとに、ソフトウェア動作の前提となるコンテキストを明確にした上で、システムの振る舞いの中でソフトウェアが担う動作を明確にする、といった作業内容が掲載されている。

もちろん、これらの作業を行う上で、実際の組込みシステムを念頭に置くと、リアルタイム性やメモリ配置といった点に注意が必要となるが、これらの注意点についてもESPRでは図5の「注意すべき事項」欄に整理してある。

3. 作業成果物のテンプレート

また上記のような作業を実施した結果は、例えばソフトウェア・アーキテクチャ設計書といったドキュメントに整理される。しかし、ここで問題となるのは、このプロセスで検討した結果を実際にソフトウェア・アーキテクチャ設計書としてどのようにまとめていったらよいかという点ではないだろうか。

中にはソフトウェア・アーキテクチャ設計書という名称は知っていても、そこにどのような情報をどのような形でまとめていくかについては十分に理解できていない方も少なくないのではないだろうか。

このためESPRでは、これらのドキュメントにどのような内容を盛り込んでいくかという情報について、一部のドキュメントについてドキュメント・テンプレートを用意している。図6のソフトウェア・アーキテクチャ設計書のテンプレートを見ていただきたい。例えば「4.制御方式」の項目に、前述の「ソフトウェア全体の振る舞いと設計」の作業で検討した結果を整理するような構造になっている。

なお、ドキュメント・テンプレートはSEC Webサイトからダウンロード可能※4であるので、参考として活用していただければと考えている。

4. ESPRの基本的な考え方

さて、これまで見てきたように、ESPRは組込みシステムを開発していくための開発プロセスのリファレンス、ガイドであり、組込みソフトウェア（システム）の開発において実施すべき作業項目と注意すべき事項を整理している。また、これらの実施作業については、その作業の入出力を明示すると共に、作業内容についても具体的で理解容易な表現を用いている。また併せて、上述したように一部のドキュメントについての設計テンプレートなども用意し、プロセスの実行結果をどのように整理していくかといったヒントも併せて提示している。

ここで、ESPRのプロセス群について整理しておきたい。ESPRでは、組込みソフトウェア（システム）の開発に必要な作業を次の四つのプロセスで分類し、構成している。

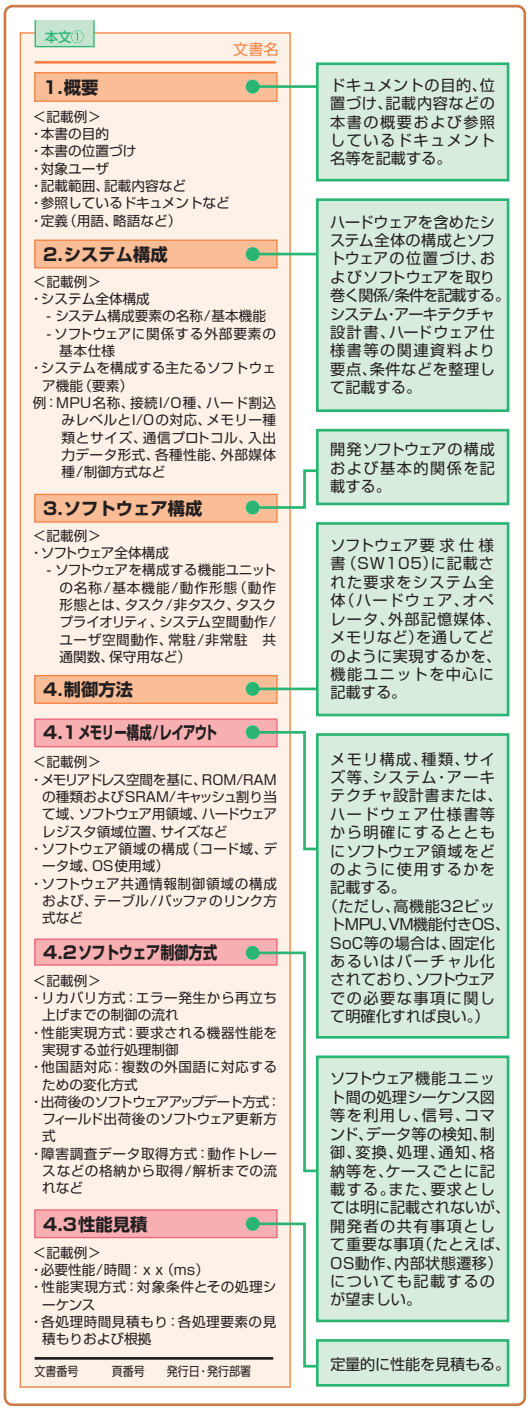


図6 ドキュメント・テンプレート

※4 http://sec.ipa.go.jp/index.html　ダウンロードには利用者登録が必要です。

①システム・エンジニアリング・プロセス（SYP※5）
組込みソフトウェアが組み込まれて動作する組込みシステムとして捉えた場合のシステム要求やシステムとしての動作検証などの作業を整理。

②ソフトウェア・エンジニアリング・プロセス（SWP※6）
ソフトウェアとしての要求定義からソフトウェア総合テストまでソフトウェアを作る際の直接作業を整理。

③セーフティ・エンジニアリング・プロセス（SAP※7）
安全・安心な組込みシステムを作り上げるために実施すべき作業を整理。

④サポート・プロセス（SUP※8）
ソフトウェア開発を円滑に進めるために必要となる支援作業や間接作業を中心に整理。

5. ESPRの活用について

プロセスと開発工程の設計

ところで、ESPRの中心はソフトウェア開発を進めていく上で、どのような作業が必要かといった視点から見たソフトウェアプロセスを中心に整理されている。これらのソフトウェアプロセスは、通常、ソフトウェア開発に求められる作業を整理しグルーピング化したもので、時間軸上の実施順序は言及しない。そのため実際に開発プロジェクトを進めていくには、代表的な幾つかのプロセスモデル（ウォーターフォール型※9やスパイラル型※10など）を参照して、自部門や開発プロジェクトで採用するプロセスを工程という形にしていく。つまり、開発プロジェクトにあてはめて、いつ、どのような作業を、どのような順序で、誰が行うかといった情報をまとめていく。ESPRではこれを開発工程の設計と呼んでいる。

開発工程の設計の目的は、マイルストーン※11、作業間の順序関係、作業期間、出力（成果物）を明確にすることにある。その際、製品のカットオーバはいつか、技術的に難易度の高い機能はあるか、どれくらいのスキルを持ったメンバをそろえるかなど、プロジェクトの特性を考慮する必要がある。

例えば、技術的に難易度の高い機能がある場合には、機能要求を検討する作業を重点的に行ったり、プロトタイプ作成を何回か繰り返したりするなど、工程を工夫することが求められるかも知れない。また、経験の浅い人がプロジェクトに参加する場合にはレビューを手厚く行って、成果物のでき具合を入念にチェックする必要があるかも知れない。

このように開発対象となるシステムやソフトウェアが持つ特性、あるいは、開発組織や開発プロジェクトの特性などを考慮して必要な作業を決め、開発工程の設計を行っていく必要がある（図7）。

ESPRは、ソフトウェアを新規に作成する場合を想定し、必要となる作業を整理したものであるが、対象ソフトウェアの性質によっては、この中から必要な作業を選定し、対象に合うようにテーラリング（調整）をするといった工夫が必要になる。

また、既存のソフトウェアの改造や流用などが中心となる開発の場合には、ESPRで整理してあるプロセスや作業の中から必要なものを取捨選択して利用していくといった工夫も必要になる。

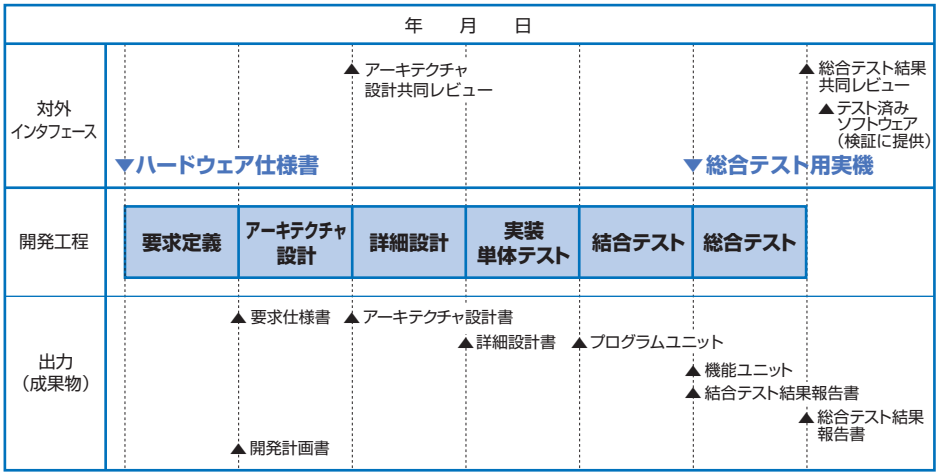


図7 工程設計の例

※5 SYP：SYstem engineering Process
※6 SWP：SoftWare engineering Process
※7 SAP：SAfety engineering Process
※8 SUP：SUpport Process
※9 ウォーターフォール型：ソフトウェア開発のプロセスモデルのひとつ。滝の水が落ちるように、後戻りすることなく開発を進めていく。
※10 スパイラル型：ソフトウェア開発のプロセスモデルのひとつ。らせん状（スパイラル）に開発を進めていく。

開発プロセスと品質作り込み

最初にも述べたが、ソフトウェア開発では要求獲得や要求定義に始まり、最終的なシステムテストに至るまで様々な作業を手際良くこなしていくことが求められる。一方で、ソフトウェアが大規模化し、開発期間も短くなってきており、作業の品質を確保することが難しくなっている。

例えば、要求の把握が不十分であれば、要求が不確定となり、続いて行う設計・実装作業では不十分な要求に基づいて行わなければならない。結果としてシステムの不具合が発生し、要求不備に伴う後戻り作業を行う破目になる。

これを改善していくには、個々の作業の過程で品質を作り込んでいく必要がある。例えば、要求定義で作成する要求仕様書には機能要求や非機能要求はもとより、要求仕様の決定事項と未決定事項も併せて記載しておくことも大切である。そして、レビューでは未決定事項が解決されたか、あるいは誰がいつまでに解決するのか、というように品質にかかわる事項を制御していくことが求められる(図8)。

6. ESPRを利用するにあたって

ESPRは開発の現場でプロジェクトマネージャやリーダ、技術者の方々が担当するプロジェクトで実施すべき作業や工程を設計する際に参考にしていただくこと想定している。また、開発作業の当事者以外でも個々の組織の作業標準や作業プロセスを整備する場合にも利用していただくことも想定している。

そのため、ESPRはソフトウェア、システムの開発プロセスに関する国際規格(ISO/IEC 12207、15288[※12])を参考に、我が国の組込みソフトウェア開発の現場で実践されてきたソフトウェア開発作業に関する知見をブレンドしており、プロセスの専門家ではない技術者やマネージャの方々が読んでもわかるように、ソフトウェア開発の世界で一般的に利用されている用語を用いて開発プロセスを整理している点も特徴のひとつである。

ESPRを参考に、それぞれの組織、プロジェクト、個人の作業を見つめ直していただき、それぞれに適したより効率的な開発スタイルを作り上げていただければと考えている。ESPRを机のかたわらに置いて、作業に漏れや抜けが無いか確認するためにパラパラとめくってみるところから始めてみてはいかがだろうか。

(IPA/SEC　山﨑　太郎)

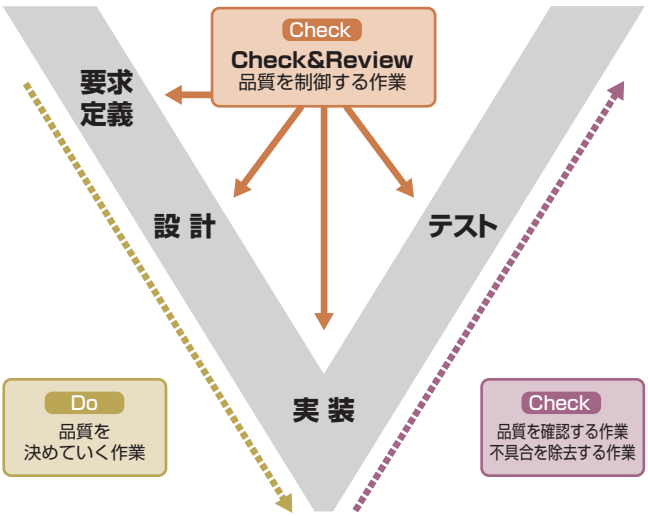


図8 開発プロセスにおける品質

※11 プロジェクトにおける大きな節目・チェックポイント。
※12 ISO/IEC 12207、15288:ISO(International Organization for Standardization:国際標準化機構)とIEC(International Electrotechnical Commission:国際電気標準会議)によって策定されたソフトウェア・ライフサイクル・プロセスのモデル規格。ISO/IEC 12207はソフトウェアを対象とし、ISO/IEC 15288はハードウェアを含むシステム全体を対象としている。

Part 2-4 プロジェクト管理の改善

「ESMR：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］」とは

SECでは開発プロジェクトを円滑に運営することを目的として、「ESMR※1：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」（以下、ESMR）を刊行している。このガイドではプロジェクトマネジメントのベースとなるプロジェクト計画書を作成するための基本的な考え方を整理している。本稿ではプロジェクトマネジメントガイドのあらましを紹介すると共に、利用する上での注意点などを論じる。

1. 開発プロジェクトのマネジメントとは

新聞などのメディアを見ていると「製品の発売延期」の記事を目にすることがある。延期の理由は様々かと思うが、中にはそもそもの開発計画にムリがあったり、計画に漏れや抜けがあったり、リスクを織り込んでいなかったり、あるいはプロジェクト計画書は作ったが計画と達成具合の差異を把握できていなかったりすることもあるのではないだろうか。

図1は「開発計画に対するプロジェクトの結果」を示したものであるが、機能・性能及び品質はかなりの会社が計画通りに達成しているようだが、開発費用や納期になると計画通りにいっていない会社も少なくないようである。

近年の大規模な組込みソフトウェア開発では、多人数によるプロジェクトを組織して開発にあたる場合が多くなってきている。こうしたプロジェクトによる開発ではいかにプロジェクト内の意思疎通を図り、円滑に進めていくかが成否の分かれ目となる。プロジェクトマネジメントはこうしたプロジェクトを円滑に進めるための技術として近年注目を集めている。

プロジェクトマネジメントの基本は、計画に対して実際がどのようになっているかを常にチェックし、プロジェクトを適切な方向に導いていくことにある。このことから、プロジェクトマネジメントの中でもとくに重要なのが、そもそもの目標値を決めるためのプロジェクトマネジメントのベースとなるプロジェクト計画書を作成する作業である。

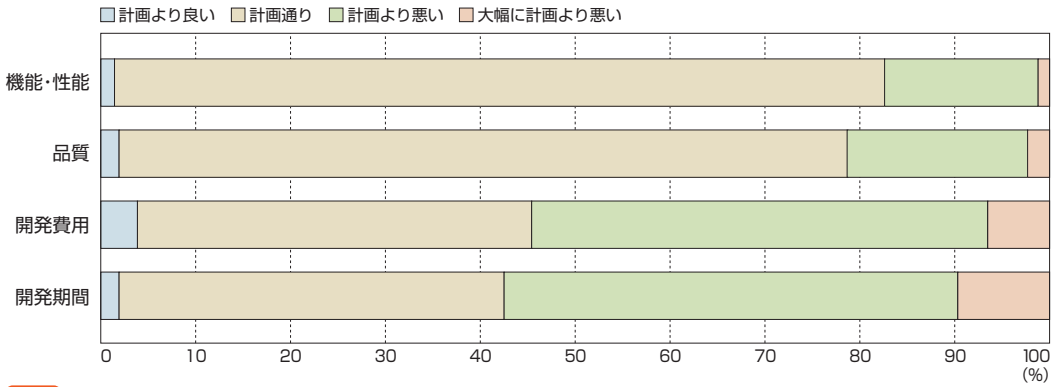


図1 開発計画に対するプロジェクトの結果
（出典「2009年版組込みソフトウェア産業実態調査報告書」プロジェクト責任者向け調査）

※1 ESMR：Embedded System development Management Reference

一般にプロジェクト計画書は、多くの企業やプロジェクトで作成されているが、そこにどのような情報を検討して盛り込んでいくかはまちまちであり、計画段階で必要な事項が検討・決定できていなかったりする。結果として、計画の漏れや抜けだけでなく、ムダな作業が発生してしまい、非効率的になっていることも少なくない(図2)。

SEC組込み系エンジニアリング領域では、開発プロジェクトを進めていく上でベースとなるプロジェクト計画書の重要性を捉え、「開発プロジェクトを円滑に進める」ことを目的にESMRを編さんした。具体的には、実際のソフトウェア開発プロジェクトでプロジェクト計画書を作成することを想定し、「実行可能なプロジェクト計画書」のための指針として、計画書フォーム例、記載例、項目の目的、なぜその項目が必要なのか、記載時の注意事項などを掲載している。

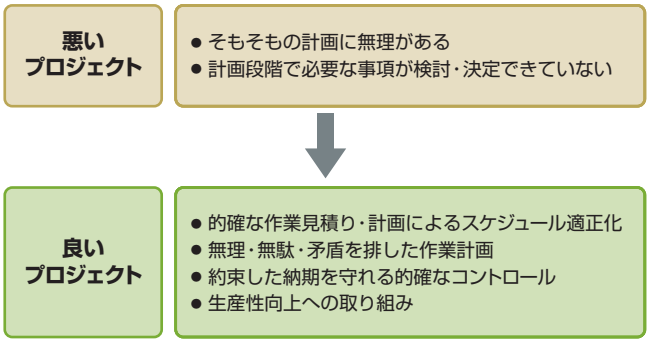


図2 悪いプロジェクトと良いプロジェクト

2. ESMRにおけるプロジェクト計画書に盛り込む情報の整理

さて、プロジェクト計画書を書くということは、プロジェクトとしてソフトウェア開発を進める上での出発点となる重要な作業である。しかし、「プロジェクト計画書を書く」という作業に直面したことのある方は、いろいろな点で悩んだのではないだろうか?

以下に考えられる問題点を幾つか挙げながら、ESMRの開発プロジェクトでの使い方について解説する。

プロジェクト概要情報	Chapter1 プロジェクトの概要		プロジェクトの概要情報
	1.1 プロジェクトの目的 1.2 プロジェクトの目標 1.3 目標達成のための方針・手段 1.4 プロジェクトの範囲	1.5 プロジェクトの前提条件 1.6 プロジェクトの成果物 1.7 スケジュールと予算 1.8 計画の更新	
参照情報	Chapter2 参照・定義		
	2.1 参照	2.2 定義	
プロジェクトの体制	Chapter3 体制		プロジェクトの体制
	3.1 製品開発プロジェクトの体制 3.2 外部インタフェース	3.3 ソフトウェア開発プロジェクト内部体制 3.4 役割分担	
プロジェクト詳細計画	Chapter4 リソース計画		Cost の計画
	4.1 開発規模と工数の計画 4.2 人員計画 4.3 設備、機器等の調達計画	4.4 プロジェクトの人員研修計画 4.5 予算計画書	
	Chapter5 作業計画		Deliveryの計画
	5.1 開発作業の洗い出し 5.2 開発作業の順序付け	5.3 開発作業担当者の割付 5.4 作業計画	
	Chapter6 品質保証計画		Qualityの計画
	6.1 品質目標 6.2 品質保証の体制と仕組み	6.3 品質保証に関する主要なイベント	
	Chapter7 リスクマネジメント		
	7.1 リスクマネジメントの方針と仕組み	7.2 リスク一覧表	

図3 プロジェクト計画書の構成

※2 QCD:Quality, Cost, Delivery。製造業の生産管理における基本要素、品質(Q)、費用(C)、納期(D)。

【問題点①】

計画書にどのような項目を書いてよいのかわからない

ESMRでは、プロジェクト計画書に盛り込むべき主な記載項目を掲載しているので、プロジェクト計画書の全体的な構成を把握することができる。プロジェクトの目的や目標などを記載する概要情報、体制やメンバの役割・責任といった運営に関する情報、またQCD※2としてソフトウェアが達成すべき情報などの項目である(図3)。

【問題点②】

計画書に書くべき項目はわかったとしても、具体的にどのような項目を埋めていくかがわからない

更にESMRでは、具体的にどのような項目を埋めていくべきか、またその項目に記述すべき内容や記述の際の注意／考慮すべき内容を掲載しているので、それに沿って項目を記述していくことができる。

例えば、図4の開発規模と工数の見積りではまず、見積単位を洗い出した上で、開発規模と開発の総工数を見積り、総工数を開発工程単位に振り分ける、といったように手順を示している。

そして、手順ごとの詳細な作業として、総工数の見積りの計算式など、具体的な方法も掲載している。

もちろん、これらの作業を行う上で、実際の組込みシステムを念頭に置くと、ハードウェア開発遅延、ハードウェアとの摺り合わせ結果によるソフトウェアによる実現部分の増加、といった点に注意が必要となるが、これらの注意点についてもESMRでは「記載の際に注意／考慮すべき内容」欄に整理してある。

【問題点③】

部門や組織として共通化された計画書のフォーマットが整備されておらず、計画書の書き方が異なっている

また上記のような作業を実施した結果をプロジェクト計画書としてどのような形でまとめるか、といった問題もある。

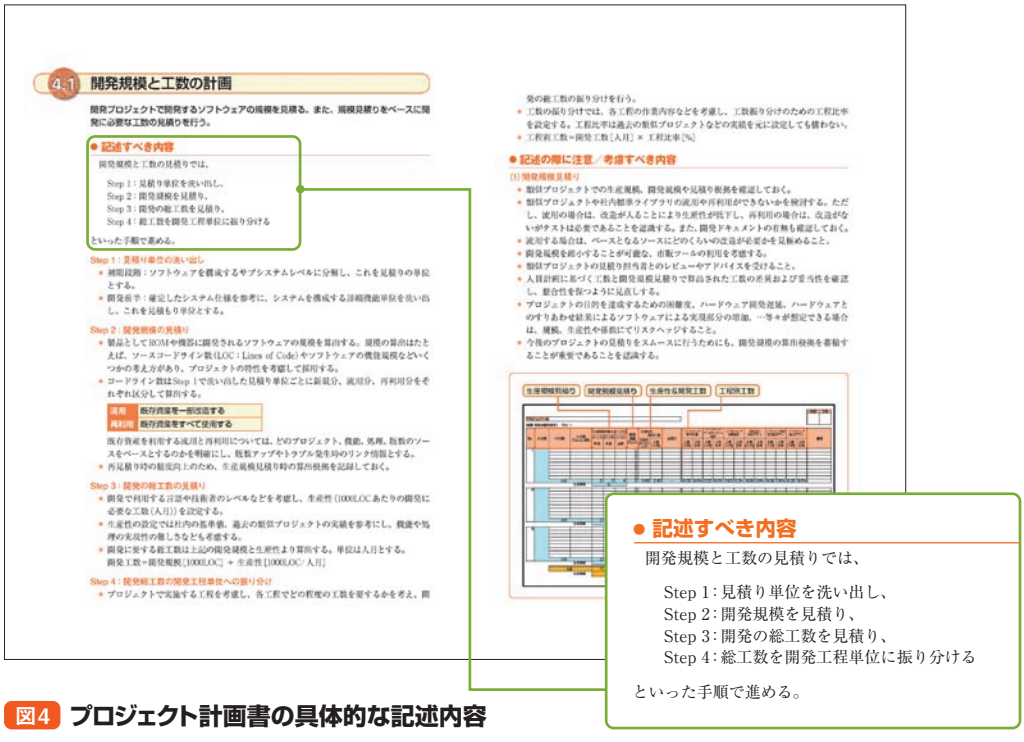


図4 プロジェクト計画書の具体的な記述内容

※3 http://sec.ipa.go.jp/index.html ダウンロードには利用者登録が必要です。

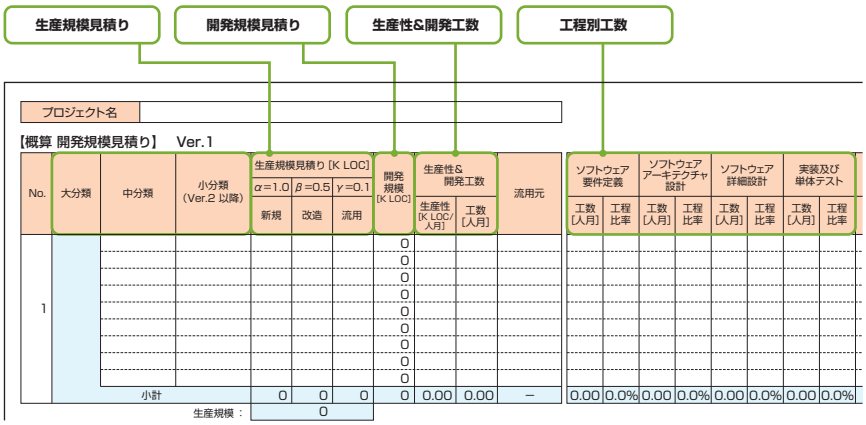


図5 プロジェクト計画書のひな形

ESMRには、プロジェクト計画書のひな形やテンプレートを掲載しており、部門や組織のプロジェクト計画書のフォーマットを整備する際に利用することができる。図5は、開発規模と工数を見積もった結果を一覧表としてまとめるためのひな形である。ここでは、生産性規模見積り、開発規模見積り、生産性&開発工数、工程別工数といった項目を整理できる一覧表のテンプレートを掲載している。

なお、ESMRに掲載しているテンプレートはSEC Webサイトからダウンロード可能※3である。参考として活用していただければと考えている。

【問題点④】

そもそも計画書をいつ、誰が書くかが決まっていない

一般に開発プロジェクトで実施する個々の作業は、開発対象となるシステムで実現する機能などの細部が明確にならないと、見えてこないものがある。このため、プロジェクト計画書に盛り込むべき各項目については、必ずしもすべての項目を一度に決めてプロジェクト計画書として整理することは現実的ではない。

ESMRのテンプレートの利用方法としては、段階的にその内容を詳細化し、最終的に実行可能なプロジェクト計画書として整理することを推奨している。

ESMRでは参考例として、プロジェクトの目的や目標といった概要情報はプロジェクト・キックオフ時に整理し、開発作業の順序付けや開発作業担当者の割り付けなどの詳細な項目はシステム仕様確定時に整理する、といったように項目の整理時期についても掲載している(図6)。

	プロジェクト・キックオフ時	システム仕様確定時点
Chapter1 プロジェクトの概要		
1.1 プロジェクトの目的	●	
1.2 プロジェクトの目標	●	
1.3 目標達成のための方針・手段	●	
1.4 プロジェクトの範囲	●	
1.5 プロジェクトの前提条件	●	
1.6 プロジェクトの成果物	●	
1.7 スケジュールと予算	●	
1.8 計画の更新		●
Chapter2 参照・定義		
2.1 参照		●
2.2 定義		●
Chapter3 体制		
3.1 製品開発プロジェクトの体制		●
3.2 外部インタフェース		●
3.3 ソフトウェア開発プロジェクト内部体制		●
3.4 役割分担		●

図6 プロジェクト計画書の記述タイミング

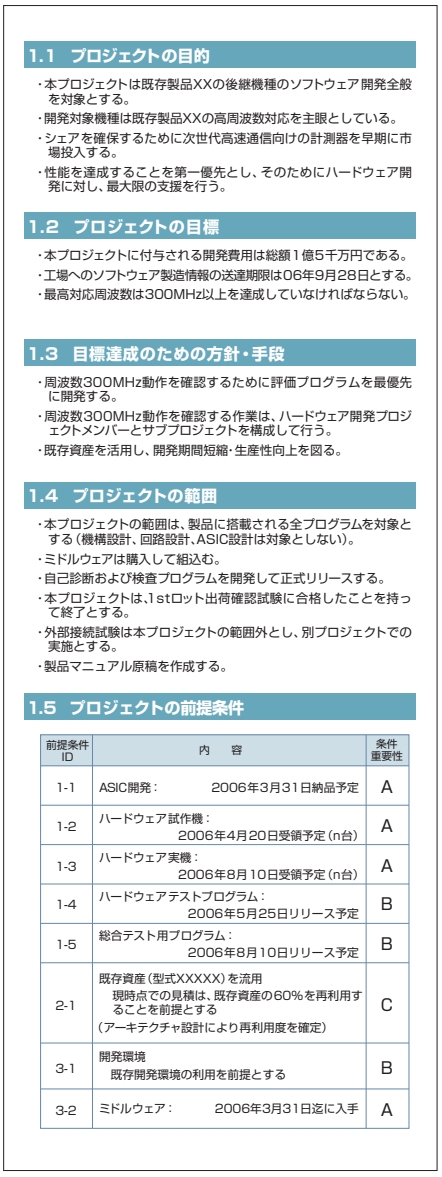
3. プロジェクト計画書の作成例

更にESMRには事例編として、プロジェクト計画書のテンプレートを活用して作成したサンプルを掲載している(図7)。

ここでは、一例として「プロジェクトの目標」の悪い記述例と良い記述例を挙げて見てみよう。

【記述例1】

①工場へのソフトウェア製造情報の送達期限は2007年9月とする。
②最高対応周波数を向上させる。
③製品化時残存バグ数を既存製品XXよりも削減する。



1.1 プロジェクトの目的

・本プロジェクトは既存製品XXの後継機種のソフトウェア開発全般を対象とする。
・開発対象機種は既存製品XXの高周波数対応を主眼としている。
・シェアを確保するために次世代高速通信向けの計測器を早期に市場投入する。
・性能を達成することを第一優先とし、そのためにハードウェア開発に対し、最大限の支援を行う。

1.2 プロジェクトの目標

・本プロジェクトに付与される開発費用は総額1億5千万円である。
・工場へのソフトウェア製造情報の送達期限は06年9月28日とする。
・最高対応周波数は300MHz以上を達成していなければならない。

1.3 目標達成のための方針・手段

・周波数300MHz動作を確認するために評価プログラムを最優先に開発する。
・周波数300MHz動作を確認する作業は、ハードウェア開発プロジェクトメンバーとサブプロジェクトを構成して行う。
・既存資産を活用し、開発期間短縮・生産性向上を図る。

1.4 プロジェクトの範囲

・本プロジェクトの範囲は、製品に搭載される全プログラムを対象とする(機構設計、回路設計、ASIC設計は対象としない)。
・ミドルウェアは購入して組込む。
・自己診断および検査プログラムを開発して正式リリースする。
・本プロジェクトは、1stロット出荷確認試験に合格したことを持って終了とする。
・外部接続試験は本プロジェクトの範囲外とし、別プロジェクトでの実施とする。
・製品マニュアル原稿を作成する。

1.5 プロジェクトの前提条件

前提条件ID	内　容	条件重要性
1-1	ASIC開発：　2006年3月31日納品予定	A
1-2	ハードウェア試作機： 2006年4月20日受領予定(n台)	A
1-3	ハードウェア実機： 2006年8月10日受領予定(n台)	A
1-4	ハードウェアテストプログラム： 2006年5月25日リリース予定	B
1-5	総合テスト用プログラム： 2006年8月10日リリース予定	B
2-1	既存資産(型式XXXXX)を流用 現時点での見積は、既存資産の60%を再利用することを前提とする (アーキテクチャ設計により再利用度を確定)	C
3-1	開発環境 既存開発環境の利用を前提とする	B
3-2	ミドルウェア：　2006年3月31日迄に入手	A

図7 プロジェクト計画書の作成例

記述例1は一見すると何の問題もないように見えるが、プロジェクトを進めていった際に、果たして目標の達成具合をチェックできるだろうか。問題点を指摘してみよう。①は送達期限が9月の初めなのか、末なのか定かでない。②は「最高対応周波数を向上させる」とあるが、どの程度向上させれば良いかわからない。③もどれくらいバグ数を削減すれば目標を達成したのか不明である。

それでは、上記の問題点をどのように修正すれば良いか次に見てみよう。

【記述例2】

①工場へのソフトウェア製造情報の送達期限は2007年9月30日とする。
②最高対応周波数は300MHz以上を達成していなければならない。
③製品化時残存バグ数の目標を既存製品XXに対し10分の1とする。
④本プロジェクトに付与される開発費用は総額1億5千万円である。

①は期日が明確になり、②は対応周波数の目標値が具体的になった。③も同様にバクの削減目標が具体的にチェックできるようになった。④は記述例1では納期と品質は記述されているが、コストが記述されていないので開発費用を追加した。

このように、プロジェクト計画書には計画とその達成具合の差異が明確にチェックできるよう、より具体的に記述することが肝要となる。

4. ESMRの基本的な考え方

これまで見てきたように、ESMRは組込みソフトウェアの開発プロジェクト着手時点で検討し決めておくべきことを整理し、計画書として標準的なスタイルを利用して文書化するためのリファレンス、ガイドであり、ESMRは以下のような特徴を持っている。

- **品質・納期・コスト等の側面を意識したプロジェクトの全体像を整理するための枠組みを用意している。**
- **組込みソフトウェア開発プロジェクトの計画を立てる場合に注意すべき点なども参考情報として掲載している。**
- **順次埋めていくことでプロジェクト計画書を作成できる標準的な計画書のテンプレートを用意している。**

また、ESMRで推奨するプロジェクト計画書は全体が大きく二つに分かれている。第1部はプロジェクト計画のサマリ情報として様々なステークホルダ※4への説明資料として位置付けている。一方、第2部はプロジェクト計画の詳細であり、品質、納期、コスト、開発のリスクといった四つの視点からそれぞれ詳細な計画を策定する枠組みを与えている。

※4 ステークホルダ:Stakeholder 企業の利害関係者のこと。

更に、計画の詳細部についてはWBS※5などを活用し、開発規模見積りから開発工数配分まで詳細な見積りを行うための表など、図表形式を活用している。なお、ESMRではこうしたプロジェクト計画書のひな形と共に、それぞれの図表を利用する上での注意事項や組込みならではの留意事項、具体的に計画を立てる際に利用できる手法事例なども併せて掲載している。

5. ESMRの活用について

ソフトウェアの開発プロジェクトでは、開発スケジュールなどが極めてタイトであったり、開発途中で要求仕様や開発の制約条件が変わったりすることが少なくない。こうした事象が頻発するプロジェクトでは、「計画書など作成しても意味がない」「書いてもすぐ変更になるので面倒」といったことを理由にプロジェクト計画書が作成されない場合もある。こういうケースでは、往々にして計画書不在の行きあたりばったりのプロジェクトになってしまい、プロジェクトはより悪い方向へ流れていってしまう。

プロジェクト計画書は、このようなプロジェクトこそきちんと立ち止まって考え、状況に応じて見直しをかけていくことで、プロジェクトを成功に導くための有効なツールであると言える（図8）。

6. ESMRを利用するにあたって

ESMRは実際の開発現場でプロジェクトを担当するマネージャやリーダの方々がプロジェクト計画書などを作成する際に活用していただくことを想定している。また、組織内でこうした計画書の標準形を整備する際に活用していただくことも想定している。

ESMRで示すプロジェクト計画書の記載項目並びにテンプレートは、組込みソフトウェア開発を進める上での一般的にプロジェクト計画書に欠かせない要件を絞り込んである。しかし、開発プロジェクトによっては、数人のお互いの情報交換が密にできるような場合もあり、こうしたケースではESMRの多岐にわたる項目をすべて整理しなくてもプロジェクト運営が円滑に進む場合もある。

例えばESMRの人員計画では、開発フェーズの進行に応じた投入人数を時系列的に整理するために山積み表形式※6も載せている。しかし、少人数プロジェクトでは投入人数の変動が（ほとんど）無いため、担当者の役割を整理するだけにとどめる、といった使い方もある。プロジェクト計画書を作成するのにムダな作業が発生してしまったら本末転倒にもなりかねない。

このように、ESMRで記載したプロジェクト計画書の記載項目や記述フォームについては、プロジェクトの特性などを考慮し、カスタマイズして利用してすることをおすすめする。

（IPA/SEC　山﨑　太郎）

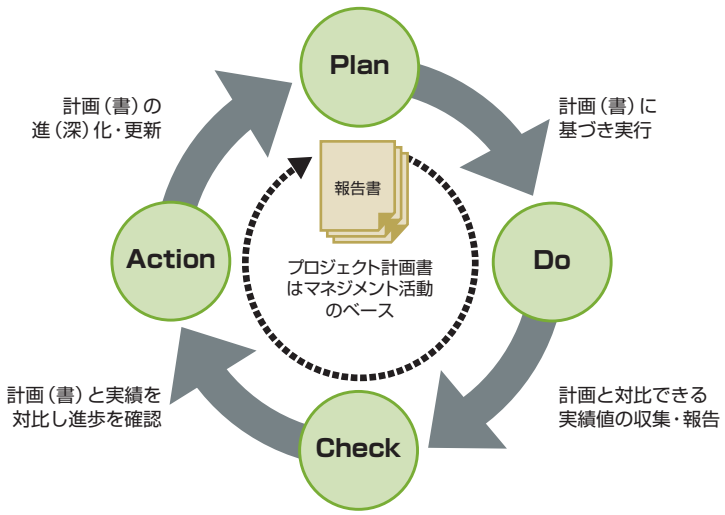


図8 開発プロジェクトマネジメントにおけるPDCA

※5 WBS：Work Breakdown Structure。プロジェクト全体を詳細な作業に分割する手法。

※6 山積み表：ソフトウェア開発の各工程においてどの程度の人・時間が必要となるのかを示した集計表。

Part 2-5 品質定量コントロール

「ESQR：組込みソフトウェア開発向け品質作り込みガイド」とは

ソフトウェアの品質は目に見えないと言われる。ソフトウェアはハードウェア・システムを動かすための仕組みであり、確かに直接見ることはできないものの、間接的にその品質を測る方法は幾つもある。「ESQR※1：組込みソフトウェア開発向け品質作り込みガイド」（以下、ESQR）はソフトウェアの品質を見える化、数値化して品質をコントロールするための概念、手法、品質目標及びその参考値、システム障害の程度を客観的に評価するためのスケールなどを提案している。更に、ソフトウェア開発は人的要因の品質への影響が非常に大きいことから、数値に表すことのできない定性的な観点として、開発を円滑に進めるためのヒント集も紹介している。

1. ソフトウェア品質は見えているか

ソフトウェア品質を見えるようにする

今や健康ブームである。「トマトが赤くなると医者が青くなる」という諺（ことわざ）がヨーロッパにあるらしい。その言葉を知ってか知らずか、八百屋やスーパーの青果物売り場では、かわいらしいミニトマトから昔ながらの大きいトマトまで、いろいろな種類のトマトがずらりと並んでいる。トマトを買う人はどのような基準でそれらの中から自分の好みのものを選ぶのだろう？　それらトマトの横には「リコピン豊富!」とか「糖度が抜群!」など、外から見ただけではわからないような情報を書いた魅力的なPOPが添えてあったりする。このように、今や目に見えるものでも、更に目に見えない特徴まで定量的な数字、要素を用いて品質を表そうとする時代である。

翻って我々の普段開発しているソフトウェアはどうだろう。一般に、ソフトウェアの品質は見えないと言われている。見えないが故に、捉えどころがないとか、開発管理が難しいとか、リリース後のトラブルが絶えないなどの問題があるということになっている。しかし果たして本当にそうだろうか？　トマトの糖度のように、目に見えないものを見えるようにすることはできないだろうか？　見えるようにできたとして、そこで捉えた品質はソフトウェア開発にどのように活かしていったらよいだろうか。

日本の現状 ～企業ヒアリング結果から～

ソフトウェアは生活のあらゆる場面でなくてはならない存在となっている。近年、組込みソフトウェアは需要の急拡大に伴い、品質や信頼性・安全性等が重視されるようになっている。エンタプライズ系ソフトウェアは組込み分野の数年先を走っていると言われているが、どちらも複雑さ、規模の拡大などの外的要因、またそれを解決する技術的課題などの内的要因に端を発し、様々な問題を抱えてしまっているようである。

研究員として各企業に品質確保のための手段のヒアリングさせていただいている印象では、どちらも実際のソフトウェア開発現場では、ISO 9000※2やCMMI※3などに実質的に取り組めている一部の企業（部門）を除いては、明確な品質目標を定められているところは少ない。品質目標を定め、それに基づいて品質の定量的なコントロール（品質制御）を行うためには、プロジェクトの各要素の計測が必要となる。ところが、何を測るか、どのように活かすかということでさえ明確に運用できていない企業が散見される。その結果として、さらなるソフトウェア不具合やシステム障害が発生し続けている（図1）。これはもはや

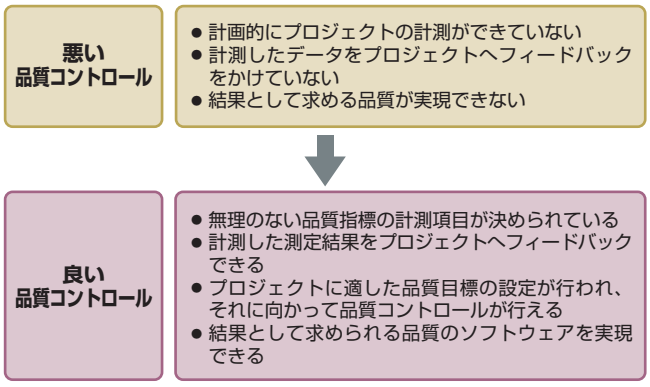


図1 悪い品質コントロールと良い品質コントロール

※1 ESQR：Embedded System development Quality Reference

※2 ISO 9000：ISO（International Organization for Standardization：国際標準化機構）によって策定された品質管理のための国際規格。

※3 CMMI：Capability Maturity Model Integration。能力成熟度モデル統合とも呼ばれ、開発プロセスを5段階の成熟度で定義している。

各企業の担当の努力が足りないというような個別の問題にしてしまうことでは済まされない。ソフトウェアの開発過程で行われる様々な作業・成果物の何をどのように測るか、ということを整理し、業界全体でベンチマークできるような仕組みを設けることが急務となっている。

SECでは開発を進めているソフトウェアやその開発作業を客観的に計測し、開発途中段階から品質を目標値に近づける（ソフトウェア品質作り込み）のための作業をあと押しするガイドとしてESQRを策定している。

2.品質作り込みの基本的な考え方

ソフトウェア品質作り込みPDCAサイクル

冒頭、「ソフトウェアを見えるようにする」ということについて述べた。ソフトウェアの品質をよりよいものにしていくためには、その品質を常に見えるように工夫し、その目標値に近づけていく努力が欠かせない。その点では、PDCA※4サイクルを開発のそれぞれの局面で回しながら進めていくことが重要であり、これがソフトウェア開発作り込みを意識した開発プロセスの実現につながってくると言える。具体的には次のような一連のサイクルを繰り返す。

①開発をしようとしているソフトウェア・システムに求められる品質レベル（目標）を明確に決める（Plan）
② ①で決めた目標を目指して開発を進める（Do）
③開発の過程で常に目標に合致しているかどうかを計測評価する（Check）
④目標に達していない場合には品質目標達成に向けた施策を実行する（Action）

我々が提案するESQRでは、こうした一連のソフトウェア品質作り込みに向けたPDCAサイクルを円滑に回すための様々な道具を用意している。

ソフトウェア品質を測る

●何を測るか

それでは、定量的な品質コントロールとはどのように行っていくのだろうか。皆さんは日ごろの開発の中でレビューはどのくらいやっているか、あるいはソースコードの行数はどのくらいかなど、実際のソフトウェア開発作業やその結果を測っているだろうか。レビューの時間数あるいは工数を測るということは、レビューをきちんとやっているかという、レビューの「作業」の十分性を評価することを目的としている。また、ソースコードの行数を測ることは「成果物」であるソフトウェアの規模やできがどの程度であるかということを把握することを目的としている。品質コントロールを行う上では、「作業」と「成果物」の大きく二つに着目して計測を行うということになる。

しかし、それらの計測項目や計測方法は会社によって異なる。レビュー工数ひとつを取ってみても、トータルの時間数のみを測っていたり、特定の工程のみを測っていたりと様々である。ソースコードの行数もコメントなどを除いた実行数を測るか、空行まで含めた全行数を測るかなど、いろいろな「流派」がある。同じ会社の中でも組織によって、プロジェクトによってまちまちであることも珍しくない。これでは、せっかく測ったデータをほかのプロジェクトやほかの会社と比較して議論をしようとしてもなかなか難しい。きちんと品質コントロールを行う上では、何を、どう測るかを定めることがひとつのポイントとなる。

ESQRでは「作業」「成果物」この二つの計測項目についてそれぞれ「プロセス品質評価指標」「プロダクト品質評価指標」と名付けている。計測すべき項目とそれらの計測の方法について整理することで、個々の開発において計測すべきポイントが明確になる。

●測った値をどう使うか

このESQRを策定する過程で非常に多くの企業の方々と品質定量コントロールについて議論させていただいた。その中には様々な定量化の試みがあったが、比較的多く見かけたのは、全社を挙げて様々な指標を駆使し、精細なデータを集めようとしているケースであった。ただそれらの中には、計測したデータを持て余してしまっており、その数値をどのように解釈したらよいか、どのように開発に利用していったらよいかといった極めて当たり前の事項が十分に練られていないケースも少なくなかった。ソフトウェアの開発過程において、品質をコントロールしていくためには、計測した値を開発途中や終了時にフィードバックしていくことが大切である。

例えば、レビュー工数であれば、レビューを行った際にその工数がソフトウェアの規模や開発にかかった工数に照らして十分かどうかということを見極めるということである。足りなければレビューが不十分であるとして不足している個所を見つけ、更にレビューを行うなどの手当てを行う。逆に多ければレビュー方法にムリやムダはないか、レビュー対象物のもともとの品質が悪すぎたのではないかなどの分析を行い、それぞれ対策を打つということである。

品質目標を考え、設定する

●品質目標は一意に決まらない

さてこのようにして計測したデータをもとにした開発へのフィードバックを行う上では、当然のことながら計測した値が良

※4 PDCA：Plan, Do, Check, Action。計画（P）、実行（D）、評価（C）、改善（A）の各ステップをひとつのプロセスとして回すマネジメント手法。

いのか悪いのかということを判断しなければならない。このためにはそれぞれの計測指標について、どの程度であれば大丈夫かといった判断基準となる値が必要となる。ESQRではこうした判断基準となる値のことを品質目標値と呼んでいる。それではこの品質目標値はどのように決めていけばよいのだろうか。

その出発点としてまず、開発対象となるソフトウェアにそもそもどの程度の品質が求められているかということを明確にしなければならない。例えばパソコンのマウスのコントローラと銀行のATM※5システムとで、同じだけのレビュー工数をかけるべきだと主張する人はあまりいないであろう。むしろ対象とするシステムが提供する機能やサービスあるいは、そのユーザや利用シーンなどによって、ソフトウェアに求められる品質レベルはかなり異なってくると考えるのが自然である。このためESQRでは後述するように「システムプロファイル」「プロジェクトプロファイル」という考え方を採用し、開発しようとしているシステムの特性、そしてそのシステムを開発する組織の特性を加味して品質目標値を設定できるように工夫している。

●目標値の目安

一方でこのようなプロファイルという考え方を導入したとしても、個別の指標、例えば「1KLOC※6あたりどの程度の設計レビュー工数を割くのが妥当であるか」といった悩みの答えはなかなか得られない。レビューやテストなどはどのソフトウェアでも同じだけ行えばよいというものではない。例えば、100KLOCのソフトウェアを10時間レビューすることと100万KLOCのソフトウェアを10時間レビューすることは同じではない。そこで、一定の規模あたりの行数でレビュー作業工数を表すこととなる。ソフトウェア品質に関する計測の問題は、実は、新しい問題ではなくこの数十年来、延々と繰り返されてきた問題のひとつであり、その都度、多くの研究者や実務家の手により様々な指標やメトリクスが提案され続けてきた。しかしこうした新しい指標の提案は実際の現場で生まれる上記の悩みの解決にはなっていないようである。

実際の開発現場では、具体的に目の前の設計書やソースコードに対して「何人でどれくらいの時間レビューすればよいか」を悩んでいる場合がほとんどである。こうした悩みを解決する唯一の方法は、個々の指標についてのおおよその参考値を用意することではないだろうか。逆にこうした具体的な指標目標値の議論を抜きにして、ソフトウェア品質の可視化やそれに基づく科学的な改善策の議論は難しいとも言える。こうしたことを受け、ESQRで各品質評価指標の明確な定義に加え、前述の品質レベル別に各指標の参考値を提示するという試みを行っている。

システム障害時の振り返りも重要

こうしたソフトウェア品質に関する作り込みなど様々な努力を重ねたとしても時には残念ながら障害が発生する。ソフトウェア以外の分野を見ると、例えば、深刻な航空運輸事故が発生した場合には、航空・鉄道事故調査委員会などが組織され、徹底した原因分析とその結果のフィードバックが確実に行われる仕組みが確立している。しかしながら、ソフトウェア・システムについては、システム障害発生時の徹底した原因分析やその結果のフィードバックの仕組みはいまだ確立されていない。昨今、話題となっているIEC 61508※7の中ではこうした状況に警鐘を鳴らすべく、事故発生時の事後安全計画の重要性に言及している。

ESQRではソフトウェア・システムに関するシステム障害の振り返りとフィードバックを確実に行い、次の開発時の品質目標に反映させるための仕組みとして、ST-SEISMIC Scale※8（システム障害震度）の概念を採用している。これは実際のフィールドで不幸にして発生した不具合の影響度を分析し、スケール（=ものさし）で表すことでシステムプロファイルの参考とする。

3. ESQR詳細の紹介

ESQRのベースとなっている品質定量コントロールの考え方について、やや概念的な説明を加えてきたが、もう少し詳細にESQRの中身を見ながら具体的な紹介をしてみよう。

ST-SEISMIC Scale（システム障害震度）

ここまでに紹介したように不幸にして発生したシステム障害の原因をきちんと分析し、その結果を次の開発に反映させ、ひいては品質コントロールループの中に取り込んで品質作り込みを行うことは極めて重要な活動である。実際問題として、フィールドでシステムに障害が起こった場合にその報道のされ方は必ずしもその深刻度に応じた大きさに沿ったものではなく、話題性といったものに左右されがちであるのは周知の事実ではないだろうか。そこで、ESQRではその被害度の分析を行い、8階級のスケールで客観的に表すことを提案している。このスケールを用いることでシステムに起こった障害を冷静に表し議論し、振り返ることが可能となる。具体的にはST-SEISMIC Scaleは表1のように、障害の程度をユーザの操作、健康被害、経済的被害あるいはシステム周辺への環境という四つの観点で評価分析し、8階級に分類している。

ST-SEISMICというのは地震の震度になぞらえており、0階級なら無震、8階級なら激震というように表す。

※5 ATM：Automated Teller Machine。銀行などに設置されている現金自動預け払い機のこと。
※6 KLOC：Kilo Line Of Code。プログラムの規模を表す単位であり、1KLOCはソースコード1,000行となる。
※7 IEC 61508：IEC（International Electro technical Commission：国際電気標準会議）によって策定された、プロセス産業における電気・電子・プログラマブル電子に関する機能安全の国際規格。

ソフトウェア開発側ではこの結果を使って障害発生時の対応を決定できると共に、システムのタイプを改めて見直し、次回開発時のシステムプロファイルにフィードバックしていくことができるようになる。

品質目標レベルの設定

その一方で、実際の開発局面では開発しようとしているソフトウェア・システムがどの程度の品質を求められているかを適切に見極めて、品質目標を設定していくことが必要となる。ここで利用する道具として、ESQRでは「システムプロファイル」「プロジェクトプロファイル」という二つの考え方を用意している。

(1) システムプロファイル

●システムプロファイルの考え方

多くのソフトウェア・システムは最終的にそれを利用する様々なユーザが存在している。そしてこれらのシステムがひとたびトラブルを起こすと、多くのユーザに有形無形の様々な不利益をもたらす結果となる。

このため、ソフトウェア・システムの品質目標を考える上では、まず、こうしたユーザ目線に立ち、どの程度の品質が期待され、求められているかを議論することが必要ではないだろうか。これを踏まえてESQRではそのシステムが期待通りの動作を行わなかったときにユーザにどの程度の被害を与えてしまうかということを人的損失、経済損失の二つの観点から分析し、4段階のレベルのどこにあるかを明確にする。これをシステムプロファイルと呼ぶ。ここで具体的な例を考えてみよう。

例えばマウスのコントローラはたとえ不具合があって利用ができなくなったとしても最悪、近くの家電量販店に駆け込めば代替品はいくらでも手に入れることができる。一方、銀行に置かれているATMのシステムに不具合があり預金の残高表示に狂いが生じてしまったとしたらどうであろうか。これは間違いなく大問題であり、ほかの銀行に駆け込めばよいといった次元の問題ではないのは明らかである。こうして見ると、マウスに求められる品質レベル、ATMに求められる品質レベルはそれぞれ異なっており、例えば、マウスはNormalレベル、ATMはCriticalレベルといった程度と考えることができる。そしてこのようにして求められたプロファイルの結果をもとに、定量的コントロールを行う際に用いる品質指標の目標値を決定することができる。もちろんユーザがシステムに求める品質はシステムによって、あるいはシステムを構成する各部分によって異なるのは言うまでもない。

●システムプロファイルの手順

それではシステムプロファイルの手順をもう少し詳しく見てみよう。図2に示すようにシステムプロファイルではシステム利用・運用時に発生する可能性のあるシステム障害を想定し、人的側面及び、経済側面の二つの観点によって対象システムを四つのシステムタイプに分類する。

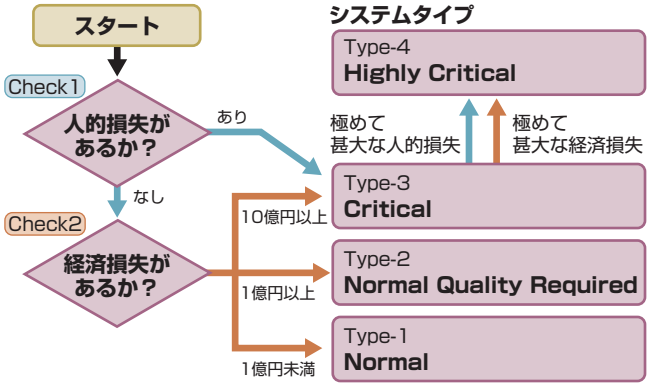


図2 システムプロファイルの手順

表1 ST-SEISMIC Scale（システム障害震度）

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	列		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	

※8 ST-SEISMIC Scale：System Trouble SEISMIC Scale。システム障害震度とも呼ぶ。システム障害の客観的な評価・判定を行うための指標。

(Check 1)まず、人的側面ではそのシステムが障害を起こすことにより、人が亡くなったり大怪我したりすることがあるか、という人的損失を検討する。被害があるシステムであればType-3以上、更に、人的・経済的に極めて甚大な被害が発生すると想定した場合にはType-4と判断する。

(Check 2)次に人的損失がない場合は経済側面を考える。経済損失というのは、そのシステムあるいはシステムを制御するソフトウェアに障害が発生した場合にユーザがどの程度の経済的被害を受けるか、ということを金額に換算して考える。被害額により、1億円未満(Type-1)、1億円以上(Type-2)、10億円以上(Type-3)という3段階に分類し、更に極めて甚大な被害が発生すると想定した場合にはType-4とする。ここで注意したいのは経済損失は使い手の立場で考える、ということである。作り手の都合、例えばリコールのための費用はここには含まない。作り手の都合は次のプロジェクトプロファイルで考慮する。

(2) プロジェクトプロファイル

●プロジェクトプロファイルの考え方

上記のようにユーザ視点から見て、ソフトウェア・システムに求められる品質レベルを考えることはご理解いただけただろうか。一方、セミナーなどでこのシステムプロファイルの説明をすると、必ずと言ってよいほど、こうしたソフトウェアを作る側の事情は考慮しなくてよいかという質問が寄せられる。つまり、成熟した経験豊富な組織で開発する場合と、そうでない組織で開発する場合では、例えば設計レビューひとつ取ってもある程度のさじ加減があるのではないかという質問である。こうした質問に答えるべくESQRではどのような条件でソフトウェアを作るのかといったことを作り手の視点で考慮し、システムに求められる品質レベルの調整を行うプロジェクトプロファイルの概念を加えている。これは具体的には開発プロジェクトの特徴、ソフトウェアの規模が今までの経験のものより大きいとか、プロジェクトに新人が多いといった10のファクターで分析する。これをもとに、システムプロファイルで得られたレベルに対する補正係数を算出し、品質目標値に補正をかけていく。

実際問題として開発対象となるシステムの性質がどのようなものであるかはプロジェクトを進めていく上で非常に重要なファクターとなる。また、システムを開発するプロジェクトメンバの構成がどのようなものであるかということも大きなファクターとなる。求める品質を実現させるためにシステムやプロジェクトの状況を分析し、開発時にどの程度の品質作り込みを行わなければならないかということを可視化するのがプロジェクトプロファイルである。

●PCP※9ファクターを使った補正係数の求め方

プロジェクトプロファイルを用いた補正係数は表2を使用して算出する。

ESQRではプロジェクトの進め方などに影響する10個のファクターを提示する。これらのファクターはプロジェクトを進める上で品質を確保しようとしたときに追い風あるいは向かい風になる条件である。例えば開発メンバ全員のスキルが高い場合、

表2 プロジェクト・プロファイル・ファクター算出表

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			

※9 PCP:Project Characteristic Profiling

作り込み時の品質が相対的に高くなると考えられるので、レビューやテストなどの不具合検出に要する時間を低く抑えることができると考えられる。

逆に例えば仕様が極めてあいまいな場合などはプロジェクトを進めていく上でレビューを頻繁に行い、かける工数を増加するなどの必要がある。このような観点でチェックを行っていき、補正係数を求める。

品質指標、参考値

このようにしてプロファイルを用いてシステムに求められる品質レベルを決める。実際の定量的品質コントロールを行っていくためには、このようにして決定した品質レベルを考慮した上で適切な品質指標を用いて開発作業や開発物を測り、評価していくことが必要となる。ESQRでは品質指標としてプロセス、プロダクトの二つの観点について品質評価指標とその参考値及び、それらを計測するための基礎指標を提示している(図3)。

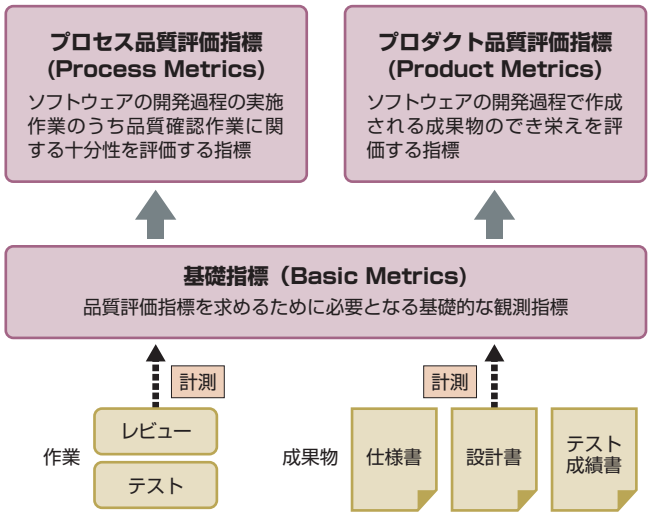


図3 品質評価指標

(1) プロセス品質評価指標

品質を確認する作業にはレビューとテストがある。

プロセス品質評価指標ではこれら二つの作業が要求定義、設計、実装、テストの各プロセスで十分に行われているかどうかということを評価する。各プロセスで行う作業については「ESPR※10 :組込みソフトウェア向け開発プロセスガイド」で詳細を記述しているので参照されたい。評価の観点としては、開発全体作業に対して適切な工数が費やされているか(作業充当率)と、開発規模に対して十分な工数が費やされているか(作業実施率)の二つがある。

(2) プロダクト品質評価指標

開発を行っていく上では、最終成果物としてソフトウェアそのものが、開発過程の中間成果物としてドキュメントなどが生成される。プロダクト品質評価指標はこれら開発時の成果物であるドキュメント、ソースコード、オブジェクト(ソフトウェアそのもの)の三つについて評価を行う。

ドキュメント品質評価指標では品質を確保するだけのボリュームがあるか、記述内容のバランスは適切かなどを見ることで質を評価する。ソースコード品質評価指標ではコードを構成する単位(ファイル、関数)が品質を確保するに足るボリュームに収まっているか、コードの内容が品質を確保するに足りる構造になっているかということを評価する。テスト品質評価指標ではオブジェクトを十分にテストできるようになっているか、検出した不具合がきちんと修正されているかなどを評価する。

(3) 基礎指標

基礎指標は品質評価指標を算出するために計測すべき、ソースコード行数や作業工数などの具体的な計測指標そのものである。これらの基礎指標については、

- **誰でも手軽に測れる(計測のために高価なツールを必要としない)**
- **計測者によって計測結果がぶれない**

ということを念頭にその定義や測り方を規定している。

(4) 品質目標値の算出方法

ESQRでは以上述べたプロセス品質評価指標及びプロダクト品質評価指標のそれぞれに対し、システムプロファイルで求めたシステムレベルごとの参考値を提供している。

この参考値をベースに、以下の式を用いることでプロジェクトごとに各品質評価指標の目標値を決定することができる。

品質目標値=
(システムレベルごとの参考値)+{(補正係数/10)×補正ベース値}

以上を行うことにより、ソフトウェアの品質を見えるようにし、開発過程で定量的にコントロールしていくことができるようになる。

定性的な側面からの品質評価

ESQRでは定量的に測ることのできない人間的な要因についても、定性的な観点として五つのチェックリストを用意している。

定量的品質コントロールを行っていく背景には健全なプロジェクト活動を行っていく必要がある。ソフトウェア開発は人間的な活動を多く含み、その中には定量的に測れないものもある。

※10 ESPR:Embedded System development Process Reference

ESQRではソフトウェア開発に関する活動のうち、コミュニケーション、ドキュメント、レビュー、テスト及び定量的品質コントロールの五つの活動について、定性的な側面からの評価と改善を行うためのチェック項目を解説と共に提示している。例として、コミュニケーションに関するチェック項目を見てみよう(表3)。

例えば、「2. コミュニケーションの相手を尊重しているか」という項目ではまず、メンバ間で尊重しあうことについて解説し、更に注意すべき点として「(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心がける」を示している。これらのチェック項目、解説は当たり前のことが書いてある。ただ、当たり前のことを当たり前にやる、ということが品質を作り込むのには何より重要である。そのようなことも読み取っていただきたい。

表3 コミュニケーションチェック項目

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、議論したか	□
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	□

4. ESQRの活用について

例えば仮に、比較的大型の情報機器の次期モデル開発をする場合を例に考えてみよう。

システムプロファイル及びプロジェクトプロファイル

ユーザにとっての価値を評価するシステムプロファイルは次のように考えることができる。情報機器であるので、人的損失はさほど考える必要はないであろう。次に経済損失を考える。この機器の稼働率は比較的高く、営業時間中はほぼ動いているような状態である。ユーザ数は出荷台数から約3万人と推定している。この機器にトラブルがあった場合、過去の障害事例も併せて考えると、部品交換を伴うので修理に約2日を要する。ちなみに、本情報機器を緊急レンタルした場合、1万円ほどであることがわかっている。従って、この機器のシステム障害時の損失額は次の式で算出できる。

損失額=損害額(10,000円)×影響率(0.8)×被害日数(2日)
×ユーザ数(30,000人)
=480,000,000円

損失額は4.8億円ということで、1億円から10億円の間であることからシステムプロファイル結果はType-2:NQに決定できる(図2参照)。

開発側の事情を考慮するプロジェクトプロファイルは次のように考えることができる。プロジェクト・プロファイル・ファクターは今回はESQRで示されたものをそのまま使用する。前モデルからの改造であるので、開発規模はさほど大きくなく、開発の条件などもある程度わかっている。ベースとなるソフトウェアの品質レベルもまぁ良い。ただ、改造開発ということで、勉強がてら、新人が多く割り当てられている。また、開発プロセスとして標準のものが決め切れておらず、ややあいまいに進めていることがわかっている。このように考えていくと、補正係数は次の式で求めることができる。

補正係数=Σプロジェクト・プロファイル・ファクター /10=2/10=0.2

品質評価指標の選択と目標値の決定

今回の開発では、前製品の開発で設計工程にやや不安があったため、今回は設計レビューをしっかり行おうということになった。そこで、設計レビュー作業実施率を計測することとしよう。設計レビュー作業実施率はESQRを参照すると図4のようになる。設計レビュー作業実施率は図4の下部に示すように設計のレビューにかかる工数をソースコード全行数で割ることで1KLOCあたりの設計レビュー工数はどのくらいかということを算出する。NQであるので参考値は9.6、これに補正係数0.2に補正ベース値を掛け合わせた値を加算し、10.08という目標値を得ることができる。これは1KLOCあたり10.08人時程度の工数をかけてレビューを行うという意味である。直接計測する基礎指標は設計レビュー工数及びソースコード全行数である。

ID	PR21		略称	ERDR	
名称	設計レビュー作業実施率		名称(英語表記)	Execution Ratio of the Design Review	
ファクター	N	NQ	C	HC	補正ベース値
参考値	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				
指標値の意味	●設計のレビュー プロジェクト規 ●安全性、信頼 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				

システムタイプ
NQ（Normal Quality required）
の場合
設計レビューは
1KLOCあたり9.60人時程度
をかけて行う

図4 設計レビュー作業実施率

品質評価指標を用いた開発作業の制御

実際の開発の場面では、設計レビューにどのくらいの工数をかけたかを計測し、目標値との差分を求めることでレビューの十分性を見る。設計レビューの時点ではソースコードはまだ存在しないため、ソースコード全行数は見積もり値を使う。

仮に、設計レビュー報告を受けた際に得られた値が10.0だったとする。この場合、目標値10.08と大きな差がないため、品質を確保するための設計レビュー工数が確保されたと判断し、作業を続行することができる。値が6.8だった場合、目標値10.08より極端に小さいので品質を確保するための設計レビュー工数が足りないと判断し、再度レビューを行う、あるいは後工程でフォローアップするなどの手段を考える。また、14.2という値が得られた場合、目標値10.08に対し、大きすぎるのでレビューに工数がかかりすぎているということが考えられる。この場合、レビュー作業の進め方に問題がないか、ベースとなるレビュー対象の設計書の品質に大きな問題がないか見直すなどの手だてを行う。このように開発の過程で品質指標の目標値と計測値とを対比することで開発作業の工程ごとに品質を確保できているかどうかの確認をすることができる。

●ESQRによるソフトウェア品質作り込みループ

ESQRでの定量的品質コントロールの様子を図5に表すと以下のようになる。

ST-SEISMIC、システムプロファイル、プロジェクトプロファイルを適用してシステム及びプロジェクトの分析を行い、何をどのように造るかを可視化する。その分析結果を用いて品質指標の選択と目標値の設定を行う。そこで定めた値を使用してソフトウェア開発時に品質をコントロールしていく。更にその結果を用いて次製品の分析をより詳細に行う、というように組織内でフィードバックをかけながらコントロールループを回していく。

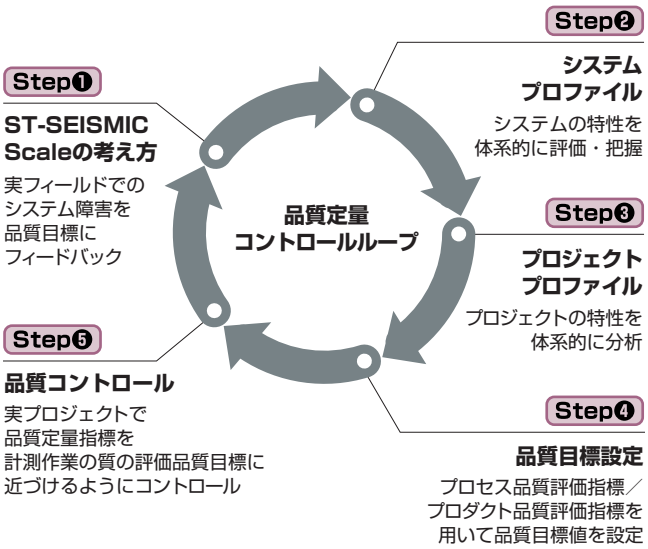


図5 ESQRによる品質定量コントロールループ

5. まとめ

以上、開発対象や開発側条件の可視化を行い、定量的品質コントロールを行う手法を示した。ESQRを用いることにより、適切な品質指標を選択し、開発するシステムに見合った目標値を定めることができる。

従来、ソフトウェアの品質コントロールは技法や考え方は示されていたものの、何をどのくらいやったらよいかということについては行間を読み取らねばならないことが多く、組織間で具体的な数値を用いて議論できることは少なかった。数値を過信すべきではないし、ESQRで示した品質指標、参考値等もまだまだ精査すべきものと考えているが、今後はソフトウェア業界全体でこれら数値を用いて、現実的な議論を展開し、ソフトウェア業界全体の品質の向上に寄与できれば幸いと考えている。

（IPA/SEC　吉澤　智美）

著者紹介

平山 雅之

IPA/SEC 組込み系プロジェクト領域責任者。ESxRシリーズ開発の全体統括を担当。所属は㈱東芝、ソフトウェア技術センター。情報処理学会監事及び組込みシステム研究会主査、ISO/SC7/WG10,20委員なども兼任している。専門はソフトウェアの品質・信頼性技術。工学博士。

吉澤 智美

IPA/SEC研究員。組込み系プロジェクト所属。組込みソフトウェアを開発するためのスキーム検討及び普及活動に従事、「ESQR：品質作り込みガイド」の策定及び執筆担当。現在はESQRの普及活動を中心に全国へ講演巡業中。 NECエレクトロニクス㈱所属。

大野 克巳

IPA/SEC研究員。組込み系プロジェクト所属。日本のソフトウェア開発力強化を目指してSEC立ち上げ時よりESxR策定に携わる。トヨタテクニカルディベロップメント㈱所属。

山崎 太郎

IPA/SEC研究員。組込み系プロジェクト所属。出自はエンタプライズ系だが、ソフトウェア・エンジニアリングの観点では組込み系と重なるところもあり、「ESPR：開発プロセスガイド」と「ESMR：プロジェクトマネジメントガイド」の策定と執筆を担当した。現在はESPRとESMRの普及活動に従事。日本ユニシス㈱総合技術研究所所属。

小林 直子

IPA/SEC研究員。組込み系プロジェクト所属。「組込みスキル標準(ETSS)」の普及促進活動に従事。主に地域に向けての講演活動や実証実験によるETSS導入推進活動を担当。所属企業ではETSS導入責任者として開発現場にETSSを展開中。アヴァシス㈱所属。

松田 充弘

IPA/SEC研究員。組込み系プロジェクト所属。担当は組込みソフトウェアのプロジェクトマネジメントをわかりやすく実践するための研究。所属は沖電気工業㈱。主に無線装置やキャリア系通信設備のソフトウェア開発に幅広く携わる。

編集後記

ESxR特集号はいかがだったでしょうか。

特集をお読みいただき、「ぜひ導入を検討したい」「もっと具体的なアドバイス が欲しい」などのご要望がありましたら、ぜひSEC Webサイトお問い合わせ画面（http://sec.ipa.go.jp/member/feedback.php）より、ご連絡ください。

担当研究員がアドバイスやご協力をさせていただきます。

また、演習を含めたセミナーなどを不定期に開催しておりますので、ご参加を希望される方は、Webサイトの利用者登録（無料）をしていただければと思います。

登録者の方には、メールマガジンで随時イベントやセミナーなどのお知らせをさせていただきます。

SEC journal® 別冊ESxR特集号

編集兼発行人 〒113-6591 東京都文京区本駒込2-28-8 文京グリーンコート センターオフィス16階
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 所長 松田 晃一
Tel.03-5978-7543 Fax.03-5978-7517
http://sec.ipa.go.jp/

本特集号の内容に直接関係する出版物

技術リファレンス集

【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 Ver.1.1
〔株式会社翔泳社〕

【改訂版】組込みソフトウェア向け 開発プロセスガイド ESPR Ver.2.0
〔株式会社翔泳社〕

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 Ver.1.1
〔株式会社翔泳社〕

組込みソフトウェア開発向け 品質作り込みガイド ESQR Ver.1.0
〔株式会社翔泳社〕

技術導入に向けた概説書

組込みソフトウェア開発における 品質向上の勧め［ユーザビリティ編］
〔株式会社翔泳社〕

組込みソフトウェア開発における 品質向上の勧め［設計モデリング編］
〔アイティメディア株式会社〕

組込みシステムの安全性向上の勧め (機能安全編)
〔株式会社オーム社〕

開発力強化を支える 技術者の育成を目指して

組込みスキル標準ETSS概説書 ［新版］
〔株式会社翔泳社〕

組込みスキル標準 ETSS導入推進者向けガイド
〔株式会社毎日コミュニケーションズ〕

組込みスキル標準 ETSS教育プログラム デザインガイド
〔株式会社翔泳社〕

その他のSEC出版物

経営者が参画する要求品質の確保 ～超上流から攻めるIT化の勘どころ～ 第2版
〔株式会社オーム社〕

共通フレーム2007 ～経営者、業務部門が参画する システム開発および取引のために～
〔株式会社オーム社〕

ソフトウェア開発見積りガイドブック ～ITユーザとベンダにおける 定量的見積りの実現～
〔株式会社オーム社〕

ソフトウェア改良開発見積りガイドブック ～既存システムがある場合の開発～
〔株式会社オーム社〕

ソフトウェアテスト見積りガイドブック ～品質要件に応じた見積りとは～
〔株式会社オーム社〕

ソフトウェア開発データ白書2009 2327プロジェクト 定量データ分析で分かる開発の 最新動向
〔株式会社日経BP〕

定量的品質予測のススメ ～ITシステム開発における品質予測の 実践的アプローチ～
〔株式会社オーム社〕

ITプロジェクトの「見える化」 ～上流工程編～
〔株式会社日経BP〕

ITプロジェクトの「見える化」 ～中流工程編～
〔株式会社日経BP〕

ITプロジェクトの「見える化」 ～下流工程編～
〔株式会社日経BP〕

ITプロジェクトの「見える化」 ～総集編～
〔株式会社日経BP〕

ソフトウェアエンジニアリングの実践 ～先進ソフトウェア開発プロジェクト の記録～
〔株式会社翔泳社〕

プロセス改善ナビゲーションガイド ～なぜなに編～
〔株式会社オーム社〕

プロセス改善ナビゲーションガイド ～プロセス診断活用編～
〔株式会社オーム社〕

プロセス改善ナビゲーションガイド ～ベストプラクティス編～
〔株式会社オーム社〕

プロセス改善ナビゲーションガイド ～虎の巻編～ 改善ゴールに一歩近づくために
〔株式会社オーム社〕

SEC journal 別冊ESxR特集号
第5巻第5号(通巻20号)
2009年11月16日発行

編集兼発行人
〒113-6591 東京都文京区本駒込2-28-8 文京グリーンコート センターオフィス16階
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
所長 松田 晃一
Tel.03-5978-7543 Fax.03-5978-7517
URL:http://www.ipa.go.jp/

IPA®
独立行政法人 情報処理推進機構

ISSN 1349-8622