第5回

組込みソフトウェア特性に基づくプロジェクト構築

野中　誠　東洋大学経営学部／
（独）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
nonaka-m@toyonet.toyo.ac.jp

組込みソフトウェアの大規模・複雑化が進んでおり，従来のように開発現場の努力で問題解決できる範囲を超えている．日本の組込みシステムおよび組込みソフトウェア産業の競争力を強化するためには，開発プロセスを測定・分析・評価して改善するといったエンジニアリング・アプローチをソフトウェア開発に適用し，その基盤の上に，開発対象ソフトウェアの特性に適合したプロジェクト体制を構築することが求められる．多様な組込みソフトウェアの特性を記述し，その特性に対して効果が期待できる開発技術を適用することで，プロジェクトの効率改善が期待できる．本稿では，エンジニアリング・アプローチの必要性と，ソフトウェア特性を明確化することの必要性を議論し，ソフトウェア特性に適したプロジェクトを構築する方法を紹介する．

組込みソフトウェア開発プロジェクトの課題

組込みソフトウェアの大規模・複雑化に伴い，組込みソフトウェアの品質確保は困難度を増してきている．一般に，組込みソフトウェアは不特定多数のエンドユーザに使用され，高い信頼性が求められるアプリケーションドメインで利用されることが多い．市場においてソフトウェア品質の問題が顕在化すると，開発元および関与する企業だけでなく，社会生活に対しても大きな影響を及ぼすことになる．また，組込みシステム製品のライフサイクルが短期化しているため，新製品の市場投入時期を早期化したいというニーズが強い．さらに，製品開発費に占めるソフトウェア開発費の比率が高まっており，これを必要最小限に抑えたいというニーズもある．

そのため，組込みソフトウェア開発プロジェクトには厳しいQCD（Quality, Cost, Delivery）制約が要求される場合が多い．この傾向は，経済産業省が実施した産業実態調査[1]の中でも顕著に現れている．この調査結果によると，QCDの各項目および機能性能の強化について，いずれも約90%以上の組織が重要課題として認識しており，中でも品質向上に対する課題意識は特に高い（**図-1**）．組込みソフトウェア開発プロジェクトにおけるQCDの向上は，多くの組織が抱える重要課題といえる．

マネジメント上の一般的課題

しかしながら，組込みソフトウェア開発ではQCDを向上させることが容易ではない．その原因として，組込みソフトウェア開発における開発技術上の課題だけではなく，マネジメント上の課題が挙げられる．すなわち，ハードウェアとソフトウェアの並行開発によってもたらされる，次のような外乱要因に起因する課題である．

(1) 開発初期の時点でソフトウェア要求が確定することは稀であり，開発の途中で機能追加などの仕様変更が発生する．これは，ハードウェアとの協調設計のためにソフトウェアの仕様が変更されるだけでなく，他社の競合製品の仕様が公開されると，競争上の理由により，開発の途中であっても仕様変更が求められるなど

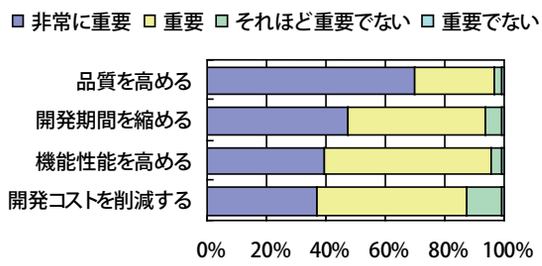

図-1　組込みソフトウェアの改善するべき領域[1]

の理由による．

(2) ハードウェア製造後に判明したハードウェア仕様上の欠陥を，ソフトウェアによって対処しなければならない．短期的な修正コストを考慮すれば必然的にソフトウェアで対応すべきだが，ソフトウェア開発プロジェクトを混乱へと陥れる原因となる．

(3) ソフトウェア結合テストにおいて，エミュレータなど独自のテスト環境を構築しなければならない場合が多く，その準備工数が必要になる．これは，テストの時点でハードウェアが完成していない，ハードウェアの破壊を避けるためにソフトウェアの欠陥が収束するまでハードウェアが使用できない，などの理由による．

(4) 高い信頼性が求められるために膨大なテスト項目を必要とし，テスト工数も多くなる．また，テストにより発見された不具合の原因究明や，不具合の再現が困難であることが多く，欠陥修正を含めたテスト工数はさらに増加する．

プロジェクト体制の問題

先に挙げた課題のうち，特に (1) と (2) は事前の予測が難しい外乱要因に依存した課題であるため，プロジェクトマネジメントを一層困難にしている．しかし，さまざまな開発現場の実状を調査した限りでは，組込みソフトウェア開発における問題は，これらの外乱要因だけに原因があるわけではなさそうである．むしろ，次に挙げる3点に示されるような，適切な開発プロジェクト体制が構築できていないことも大きな原因であると思われる．

(1) 開発プロセスを測定・分析・評価して改善するといった意味でのエンジニアリング・アプローチが十分に浸透しておらず，プロジェクトの成否は開発メンバのスキルに依存する部分が大きい．

(2) 開発対象ソフトウェアの品質要求が定義されていなかったり，製品が利用される環境が十分に認識されていなかったりするなど，開発対象ソフトウェアの特性が明確化されずに開発が行われている．

(3) 開発対象ソフトウェアの特性に適した開発技術が適用されていない場合が多く，無駄な開発工数が費やされたり，要求機能や性能が達成できていない製品が出荷されたりするなどの事例が起きている．

組込みソフトウェア開発プロジェクトのQCD制約が厳しいからこそ，その制約を合理的に満たすことができるような，効率的かつ効果的なプロジェクト構築・運営の体制が求められる．上記の問題を解決した上で，外乱要因に対するリスクマネジメントが実施できれば，大きな効果が期待できる．

以降では，上記の問題点について，各問題の意義，問題解決による効果，解決方法の指針などについて述べる．その際に，内容を組込みソフトウェア開発に限定せずに，ソフトウェア工学分野の，特にソフトウェア開発管理に関する研究・実践の蓄積に基づいて議論する．

エンジニアリング・アプローチによる効果

前述の通り，ここでのエンジニアリング・アプローチとは，開発プロセスおよびプロダクトを測定・分析・評価し，その結果に基づいて改善を行うことを意味する．ここでは，こうしたアプローチを開発現場に浸透させる手法とその効果，ならびに国内の組込みソフトウェア開発の現状について述べる．

日本型ソフトウェア品質管理

1970年代から1980年代にかけて，国内の汎用機メーカを中心に，TQC（Total Quality Control）として知られる日本型品質管理がソフトウェア開発に適用され，ソフトウェア製品の信頼性向上において大きな成果を収めた．ここで日本型ソフトウェア品質管理とは，「品質を追求すればコストはついてくる」といった考え方に基づき，開発工程を明確に定義し，レビューの徹底により工程ごとに品質管理を行うといった活動への組織的な取り組みを意味する（詳細は文献2）が詳しい）．実際には，工程標準の徹底や，開発現場での改善事例発表の奨励などの活動により，品質管理を開発現場に浸透させたことの効果が大きいようである．このような品質管理面での成功は“Software Factory”として欧米などにも紹介され，日本のソフトウェア品質の高さが国際的にも注目された．

1990年代になってISO 9000やCMM（Capability Maturity Model）が盛んに取り上げられるようになってからは，スタッフ主導の改善活動に重点が置かれたためか，開発現場への浸透の度合いがかえって低下してしまった感がある．しかし，Cusumanoらが実施した調査[3]によると，国内には，依然として世界的に高い品質水準を達成できている組織が存在する．この調査によれば，ソフトウェア製品の出荷後1年以内に発見された欠陥件数を，新規ソースコード1,000行あたりに換算した数値（欠陥密度）で比較すると，日本の調査対象企業では中央値が0.020，米国では0.400，インドでは0.263であった☆1．直感的に解釈すると，5万行のソースコードが新規に開発されたソフトウェア製品の場合，日本製だとおよそ

☆1 ただし，同質の企業およびプロジェクトを比較していない，ユーザ数によって報告される欠陥数が異なる，回答企業のデータの精度に問題がある，サンプルサイズが小さいなど，必ずしも妥当性の高い調査とはいえない．

	TSP	典型的
システムテストでの欠陥密度	0.4 (0～0.9)	15
出荷後の欠陥密度	0.06 (0～0.2)	7.5
システムテスト工数比率 (%)	4 (2～7)	40
システムテスト期間比率 (%)	18 (8～25)	40

括弧内は範囲を表す

表-1　TSPプロジェクトと典型的プロジェクトの比較[6)]

1件，米国製だと20件，インド製だと13件の欠陥がソフトウェア製品の運用中に顕在化する，といった具合である．

この数値を見て「自社ではもっと悪い」「データがないので比較さえできない」といった反応を示す向きも多いことであろう．各企業の実状はともかく，少なくともこの調査の範囲では，欠陥の少なさという意味でのソフトウェア品質に関して，日本には国際的な競争力を持った組織が存在しているようである．

TSP（Team Software Process）

その一方で，米国などでTSP（Team Software Process）[4)]というエンジニアリング・アプローチに基づく開発プロセスを適用し，日本の組織と同等またはそれ以上の品質水準を達成している事例が報告されている．TSPは，レビューを重視した開発手順が記述されたプロセス定義と，データ収集や品質計画に利用するための一連の書式用紙から構成される．TSPを適用した開発プロジェクトでは，プロセス定義に忠実に従ってプロジェクトを進め，開発プロセスの実績データを収集し，そのデータに基づいて開発計画と品質目標を立て，実績データに基づいてプロセスを改善していく．TSPは，技術者がPSP（Personal Software Process）[5)]という個人レベルでのプロセス管理手法および厳密性の高いソフトウェア設計手法を習得していることを前提としている．これにより，あらかじめ開発メンバにエンジニアリング・アプローチを浸透させておくことが，TSP導入のポイントである．

表-1は，TSPを導入したプロジェクト20件のデータと，CHAOS'94と呼ばれるStandish Groupが調査した典型的なプロジェクトを比較した結果である[6)]．表にある通り，TSP導入プロジェクトでは出荷後の欠陥密度は平均で0.06であり，先に示した日本の0.02とほぼ同等の品質水準を達成している☆2．しかも，システムテストの工数比率がCHAOS'94の典型的なプロジェクトに比べて圧倒的に少ないことが大きな特徴である．国内の組込みソフトウェア開発の場合でも，テストおよび検証の工数比率が30%を超える組織が約半数という状況である[1)]．やはり，TSPを適用したプロジェクトのテスト工数の少なさが際立っている．

このように，日本の組織が持つ高品質ソフトウェアを開発できる能力は，もはや圧倒的な競争優位とはいえなくなってきている．しかも，テスト工数比率の少なさを考えると，生産性についても競争優位を確保することが困難になってきている．

テスト工数比率にこれだけの差異が生じている最大の理由として，次の2点が考えられる．第1の理由は，TSPでは「テストは欠陥が存在しないことを確認するための工程」と位置づけ，レビューを徹底的に重視していることが挙げられる．TSPでは，結合テストの前までに開発プロセス全体で作り込んだ欠陥の97.5%，システムテストの前までに99%の欠陥摘出を目標値として掲げている[4)]．この目標値の達成に向けて，開発チームが計画的にレビューを実施し，データに基づくプロセス改善に取り組んでいることが，テスト工数の削減に結びついている．

もう1つの理由として，TSPでは形式手法の考え方を取り入れた，厳密性の高い設計手法を採用していることが挙げられる．これはPSPで扱っている設計手法であり，設計対象が取り得るすべての状態とイベントを直交させ，その交点ごとに，設計対象のアクションとその事前条件を検討する．その結果を論理記号により記述し，設計仕様を作成する．UML（Unified Modeling Language）などによる典型的な設計作業に比べると設計工数が多くなるが，その分，テスト工程での大幅な工数削減に貢献している．

テスト工数が削減されると，品質向上の効果だけでなく，プロジェクトマネジメントの観点からもメリットが得られる．一般に，テスト工程では，潜在欠陥数の予測や欠陥発見・修正の所要時間の見積りが困難であり，プロジェクトの進捗管理を難しくしている．プロジェクト全体に占めるテスト工数の比率が小さくなることにより，開発工数および開発期間の見積り精度向上が期待できる．

■日本の組込みソフトウェア開発の現状

翻って，日本の組込みソフトウェア開発の状況はどうであろうか．実態調査[1)]によると，約半数の組織で，開発案件の5件に1件以上の割合で工程間の手戻りが発生している．また，品質管理を，個人やチームで任意に行っている組織は約40%に上る．これらの結果や各組織の実態などから推察するに，日本の組込みソフトウェ

☆2 日本のデータとTSPプロジェクトのデータでは，プロジェクトの性質（たとえば予算規模や開発期間の制約など）が同一とはいえないため，比較の際には注意が必要である．

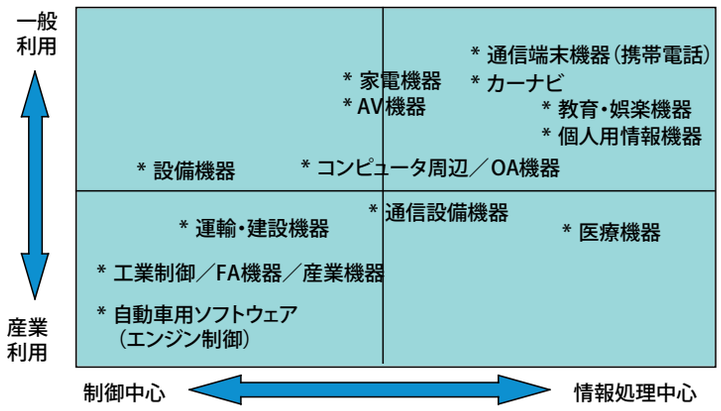

図-2　組込みソフトウェアのアプリケーションドメイン[1)]

ア産業は，エンジニアリング・アプローチに基づく開発が浸透しているとは言い難い状況にある．

それでも，品質問題が顕在化している一部のソフトウェア製品のケースを除き，多くの製品は実用上問題のない品質水準が達成できている．これは，スキルの高い技術者や，多くの開発工数を投入するなど「現場のがんばり」に依存するところが大きいためと思われる．しかしながら，厳しいQCD制約と機能性能の強化が求められている中で，エンジニアリング・アプローチ不在のプロジェクト体制では限界がある．開発作業の無駄を取り除き，開発の効率性を高めるために，開発現場へのエンジニアリング・アプローチの浸透が求められる．これは，以降に述べるソフトウェア特性に基づくプロジェクト構築のための基盤である．

ソフトウェア特性の明確化

ここでは，まず組込みソフトウェアの多様性を概観した上で，品質要求や製品が利用される環境など，開発対象ソフトウェアの特性を明確化することの必要性について述べる．また，特性の記述に利用できる国際規格や既存手法を紹介する．

組込みソフトウェアの多様性

これまでの連載でも再三指摘されてきた通り，組込みソフトウェアのアプリケーションドメインは広範にわたっている．特に日本で開発される組込みソフトウェアはその傾向が顕著であり，携帯電話・ディジタル家電・自動車など一般ユーザにも馴染みのあるドメインから，産業機器・医療機器・設備機器などに至るまで，実に多様なドメイン向けのソフトウェアが開発されている．**図-2**は，これらのドメインを，制御中心と情報処理中心の区別と，一般ユーザ向け利用製品と産業機器向け利用製品の区別で表した図である[1)]．しかし，1つの組込みシステム製品の中にも，制御中心と情報処理中心のソフトウェアシステムが混在している場合もある．もはや，製品のドメインによる区分では，組込みソフトウェアを必ずしも適切に分類できなくなってきている．

アプリケーションドメインの多様性に加えて，組込みソフトウェア開発プロジェクトの規模や開発体制も多様である．**図-3**では，新規に作成される組込みソフトウェアのソースコード行数の分布[1)]を示している．その多くは5万行以下だが，中には大規模なコードが新規に作成されるプロジェクトもある．また，ここでは図示していないが，再利用率も0%から90%以上まで万遍なく分布している．これに対して，平均的な組込みソフトウェアの開発期間の分布は，6カ月未満と1年未満がほぼ同率，それらの合計で80%に上る．新規に開発されるコード行数にばらつきがありながら，短期間の開発が求められている状況が読み取れる．

ソフトウェア特性に適した開発技術の適用

このような多様な組込みソフトウェアに対して，同じ開発技術を画一的に適用することは，効率性および有効性の観点から不適切である．ここで開発技術とは，ツール，開発・管理手法，プロセスモデル，マネジメントレベルなどの総称を意味している．

たとえば，規模の小さなソフトウェア開発にTSPを適用すると，管理オーバーヘッドが大きくなる．また，信頼性が求められるソフトウェア開発にXP（eXtreme Programming）そのままのアジャイル手法を適用すると，プロジェクトのリスクが大きくなるなどのミスマッチが生じる．開発対象のソフトウェアおよびプロジェクトの特性に適合した開発管理手法やツールを適用することが

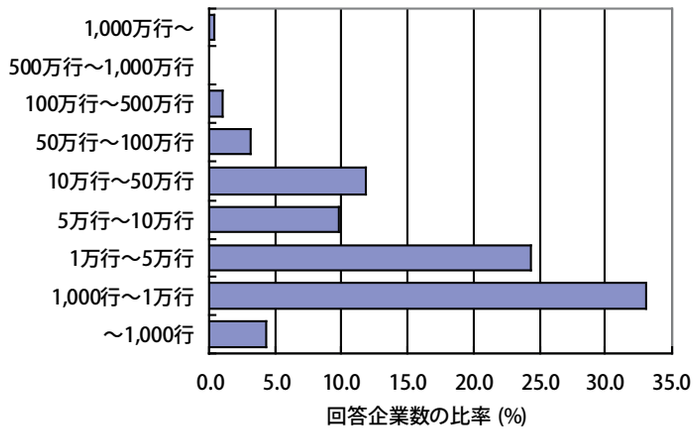

図-3　新規に開発する組込みソフトウェアのソースコード行数[1)]

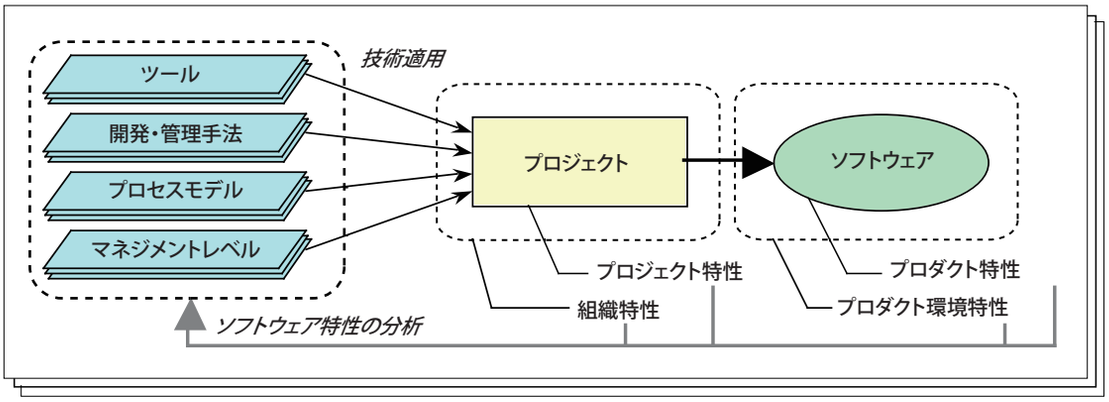

図-4　ソフトウェア特性分析に基づく開発技術の適用

求められる．これを概念的に表したのが**図-4**である．

適切な開発技術を適用するためには，まず，開発対象ソフトウェアの特性を明らかにしなければならない．本稿ではこれをソフトウェア特性と呼び，次の4つの観点から分類する．

(1) プロダクト特性：ソフトウェア規模，機能規模，複雑度，ソフトウェア品質要求，再利用可能性など，対象ソフトウェア自身の属性を表すもの．

(2) プロダクト環境特性：開発環境の制約，製品アーキテクチャ，要求の安定性，利用時の品質要求，連携する外部システムなど，対象ソフトウェアの開発時または運用時における環境の属性を表すもの．

(3) プロジェクト特性：コスト制約，開発期間，利用可能な人的資源，開発メンバのスキル，ステークホルダなど，開発プロジェクトの属性を表すもの．

(4) 組織特性：組織構成，組織文化など，プロジェクトを取り巻く組織の要因に関するもの．

組込みソフトウェアでの例を挙げると，製品ライン内で共通利用されるソフトウェアを開発する場合は，再利用性が特に求められる．また，品質要求に着目すると，一般的な組込みソフトウェアでは高い信頼性と効率性が求められるが，製品によっては要求される信頼性水準や割り込み処理のタイミングなどが桁違いに異なる場合がある．こうした特性の違いを認識した上で，ソフトウェア特性に適合した開発技術を適用することが，厳しいQCD制約を合理的に満たしていくために求められる．

ソフトウェア特性に関する既存手法

具体的なソフトウェア特性の記述方法を検討するにあたっては，以下に挙げるような国際規格や標準情報，その他の既存手法が参考になる．

- ISO/IEC 9126-1:2001（ソフトウェア製品品質のモデル）の品質特性および品質副特性：機能性，信頼性，使用性，効率性，保守性，移植性．
- ISO/IEC TR 12182:1998（ソフトウェア分類）：内部視点（規模，機能性，信頼性要求，性能要求，主要言語など），環境視点（適用領域，コンピュータ資源要求，クリティカルさなど），データ視点（データ表現，データ利用など）．
- ISO/IEC TR 14143-5: 2004（ソフトウェア機能領域の

決定）の機能領域の特性：制御・通信特性，データ特性，アルゴリズム特性．

- COCOMO II（工数・期間の見積り手法）のコストドライバ：製品特性，プラットフォーム特性，プロジェクト特性，要員特性．
- ISO/IEC 20926:2004（IFPUG法（ファンクションポイント法））の一般システム特性：データ通信，分散処理，性能など14特性．
- プロセスモデル選択の重要要因[7]：規模，重要度，変化の度合い，開発メンバのスキル，組織文化．

もちろん，ここに挙げたすべての特性を用いて対象のソフトウェアおよびプロジェクトを記述する必要性はない．開発技術に影響するソフトウェア特性を見極め，その特性に関してのみ記述すればよい．なお，ソフトウェア特性と開発技術との関係については，現時点では明確化できていない．今後，その関係を明確化していくことが課題として挙げられる．

ソフトウェア特性に適した開発プロセスの検討

以下では，ソフトウェア特性分析の結果に対して適切な開発技術を適用する方法の例として，プロセスモデルおよび開発手法に関する例を紹介する．具体的には，プロセスモデルの選択，品質要求に基づいた開発プロセスの設計，レビュー手法の選択について述べる．

■プロセスモデルの選択

プロセスモデルとは，要求分析や設計などの実施すべきアクティビティを，どのような開発戦略のもとで具体的に実行していくのかを表現したモデルである．典型的なモデルとして，ウォーターフォールモデル，スパイラルモデルなどが挙げられる．また，XPに代表されるアジャイルプロセスモデル，TSPのような計画駆動型モデルなど，さまざまなモデルが示されている．

プロセスモデルの有効性に関しては，特にアジャイルと計画駆動型との間でしばしば論争が起きる．しかし，それぞれ得意とする対象に適用するべきであって，必ずしも対立するモデルではない．Boehmらは，アジャイルと計画駆動型において，規模，重要度，変化の度合い，開発メンバのスキル，および組織文化の5つをプロセスモデル選択の重要要因としている[7]．これらの要因について対象を分析し，各モデルの有効性が最も期待される領域で適用することが求められる．

■品質要求に基づいた開発プロセスの設計

品質要求を定義することにより，その達成に見合った適切な開発プロセスを検討できる．たとえば，保守性や移植性といった品質特性が重視されるソフトウェアの場合，ソフトウェアアーキテクチャ設計において，ソフトウェアの柔軟性を高めるための設計作業を重点的に実施するなど，開発プロセスのテーラリングが求められる．

品質要求に基づいて開発プロセスを設計する方法として，品質機能展開（QFD：Quality Function Deployment）の利用が考えられる．QFDは，ユーザの品質要求を品質特性で表し，製品を構成する機能部品や開発工程にまで系統的に展開する手法である．ここでは，品質要求を開発プロセスの各作業項目へと展開するためにQFDを利用する．その際に，**図-5**に示したような品質要求と作業項目との対応関係ならびにその関連度を，組織の開発対応方針に基づいてあらかじめ定義しておく．図-5の例の場合，機能性の副特性である合目的性を高めるために，プロトタイピングとデザインレビュー充実化に重点を置いて開発を行うといった対応方針が示されている．

以下に，QFDを用いて品質要求に基づく開発プロセスを設計する手順を示す．

(1) 開発対象ソフトウェアの品質要求を，ISO/IEC 9126-1のソフトウェア品質特性に基づいて定義する．各品質特性の要求水準に基づいて，品質特性のウェイトを決める．

(2) 品質要求と作業項目との対応関係に基づき，品質要求水準を達成するために重点的に実施すべき作業項目を抽出する．

(3) 対応関係に示された関連度および品質特性のウェイトに基づいて，各作業項目で適用する開発手法を決定する．たとえば，レビュー充実化が求められる場合は，担当者によるレビューだけでなくインスペクションを取り入れるなど，より厳密性の高い手法を選択する．このようにして開発プロセスを設計する．

(4) 開発手法別の生産性実績データに基づいて，作業項目に必要な工数を見積もる．これを積算してプロジェクト全体の工数を見積もり，QCD制約を満たせる可能性が高いか否かを判断する．QCD制約を満たせる可能性が低いと判断した場合は，品質要求や機能要求の量と，コストや納期とのバランスをとるために，要求者または製品企画者などと調整する．

■レビュー手法の選択

レビュー手法と一口にいっても，レビュー時にドキュメントを読む方法によって，AHR（Ad Hoc Reading），CBR（Checklist-Based Reading），PBR（Perspective-Based Reading）などさまざまな手法がある．AHRはチェックリスト等を用いずにレビュー担当者任せでドキュメントをレビューする手法であり，レビュー担当者

品質要求	品質副特性 ＼ 開発特性	要求定義			システム設計									基本設計					
		プロトタイピング	用語の標準化	図表記載充実化	構造化設計	過負荷テスト設計充実化	プロトコルの標準化	用語の標準化	図表記載充実化	データ機密保護充実化	障害対策充実化	ユーザインタフェース標準化	デザインレビュー充実化	構造化設計	機能テスト設計充実化	データ形式標準化	用語の標準化	図表記載充実化	デザインレビュー充実化
機能性	合目的性	◎	△	△	△			△	△				◎	△	○		△	△	◎
	正確性		○	○	△			○	○				◎	△			○	○	◎
	相互運用性						◎						△			◎			△
	セキュリティ									◎			△						△
	標準適合性						◎					◎	△			○			△
信頼性	成熟性					○					△		△						△
	障害許容性										◎		△						△
	回復性										○								

図-5　品質要求と作業項目との対応関係（一部）

のスキルに大きく依存する．CBRはチェックリストを用いてドキュメントをレビューするという典型的な手法であるが，チェック項目が膨大になると効率が低下しがちである．PBRはレビュー担当者が利用者やテスト担当者などの観点別に，与えられたレビューシナリオに従ってレビューを実施する手法である．これらの有効性比較に関する研究は多数行われており，一般的にはPBRが最も有効で効率の良い手法と考えられている．

しかし，対象ソフトウェアの特性やレビュー対象ドキュメントの記述方式，レビュー担当者のスキルなどによっては，PBRよりもCBRの方が効果的となる場合がある．レビューでの欠陥見逃しによる工数の損失は，その内容によっては大きなものとなるため，ソフトウェア特性に対して効果的な手法を適用することが望ましい．また，制御フローの複雑なプログラムの場合は，レビューによる欠陥除去よりも，機能テストによる欠陥除去の方が効率的であるとの従来研究もある．このような差異を生じる原因をソフトウェア特性により記述すれば，レビュー手法の選択基準が明確化できる．

プロジェクト構築手法の確立に向けて

以上，エンジニアリング・アプローチをソフトウェア開発に適用することの必要性と，ソフトウェア特性に基づいてプロジェクトを構築する概念的方法を紹介した．こうした考え方をプロジェクト構築の際に取り入れることによって，開発効率の向上に対する示唆が得られれば幸いである．

なお，この方法は十分に確立された方法論レベルにまで至っているわけではない．現時点では，どの特性に着目すればどの開発手法の選択に役立つのかなど，プロジェクト構築方法が詳細レベルにまで検討できていない部分がある．今後，ソフトウェア特性と開発手法との関係分析が求められる．

現在，ソフトウェアエンジニアリングセンターでは，組込みソフトウェア向けの標準開発プロセスの検討を行っている．ISO/IEC 12207いわゆるSLCP（Software Life Cycle Process）の開発プロセスを参考にして，組込みソフトウェア開発における参照モデルとして利用できるような標準開発プロセスの提供を目標としている．標準開発プロセスを組織に取り入れる際には，本稿で紹介したように，ソフトウェア特性を考慮して標準プロセスをテーラリングすることが必要になる．標準プロセスの検討と併せて，テーラリング方法の検討を深めていきたい．

参考文献

1）経済産業省，2004年版組込みソフトウェア産業実態調査報告書（2004）．
2）森口繁一編：ソフトウェア品質管理ガイドブック，日本規格協会（1990）．
3）Cusumano, M., MacCormack, A., Kemerer, C. F. and Crandall, B.: Software Development Worldwide: The State of the Practice, IEEE Software, Vol.20, No.6, pp. 28-34（2003）．
4）Humphrey, W. S.: The Team Software Process（TSP），CMU/SEI-2000-TR-023（2000）．
5）Humphrey, W. S.: A Discipline for Software Engineering, Addison-Wesley（1995）（松本正雄監訳：パーソナルソフトウェアプロセス技法，共立出版（2000））．
6）Davis, N. and Mullaney, J. : TSP in Practice: A Summary of Recent Results, CMU/SEI-2003-TR-14（2003）．
7）Boehm, B. and Turner, R. : Balancing Agility and Discipline, Pearson Education（2004）（河野正幸，原　　幹 監訳：アジャイルと規律，日経BP社（2004））．

（平成17年5月10日受付）