NAIST-IS-DT9861009

博士論文

実プロジェクトデータの分析に基づく
ソフトウェアプロセス改善と品質向上の研究

坂本　啓司

奈良先端科学技術大学院大学

情報科学研究科　情報システム学専攻

２０００年６月

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士(工学)授与の要件として提出した博士論文である。

坂本 啓司

審査委員：井上 克郎 教授
小山 正樹 教授
関 浩之 教授
松本 健一 助教授

実プロジェクトデータの分析に基づく
ソフトウェアプロセス改善と品質向上の研究*

坂本 啓司

内容梗概

ソフトウェアは社会システムの隅々にまで組み込まれてきており、その品質問題は２０００年問題に代表されるように社会的な課題となっている。ソフトウェアの些細なバグが大規模な停電事故や電話回線事故を引き起こしたり、極端な事例では人身事故まで報告されている。ソフトウェア品質については、ＩＳＯ－９１２６で六つの品質特性が定義されているが、停電事故や電話回線事故事例に見られるように問題の大きさは圧倒的にソフトウェアが期待通りの動作をしないという古典的とも言える品質問題に集中している。いわゆるバグをどうやって低減するかが、ソフトウェア開発に関わっている人だけではなく社会的急務であるといえる。

本論文では、まず第１章でソフトウェアの課題解決の重要性について述べ、次に第２章で、ソフトウェア品質を議論するベースとして、まずソフトウェア開発の持っている特有の難しさと、それを克服するためのプロセス改善とはどういうものかについて述べる。さらにその中でも品質管理が特に難しい理由は何なのかを良く理解した上で改善の方向を決めていく必要がある事を述べる。第３章では実プロジェクトデータの分析からデザインレビューが品質向上にいかに有効であるかについて述べる。第４章でも実プロジェクトデータの分析から、プロジェクト計画の重要性を述べ、開発プロジェクトの安定が品質および生産性の向上に寄与することを述べる。第５章では、現状プロセスの分析に基づく改善計画により、ほぼ計画どおりの生産性向上が図れたプロセス改善の事例を紹介する。第６章では、プロセス改善はなぜ難しいのか、またプロセス改善を成功させるためにはどのような取組みをすべきであるかを述べる。第７章では、本論文の全体のまとめと考察を行う。

キーワード)

ソフトウェアプロセス改善、プロジェクト管理、メトリクス、ソフトウェア品質管理

---

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DT9861009, 2000 年 6 月

A study of software process improvement and quality control based on analyses of actual project data.*

Keishi Sakamoto

Abstract

While ISO-9126 defines six characteristics on software quality, most of the faults are caused by lack of the most traditional characteristics “functionality” and “reliability”. How to reduce so-called bugs is a big problem to be solved not only for software engineers and organizations but also for human society.

In Chapter 1, importance to solve the software problems is described. In Chapter 2, difficulties of software development and process improvement to overcome the difficulties are described as a base for the following discussions on software quality. In the chapter, also described is necessity to decide a direction of improvement by understanding why software quality control is so difficult. In Chapter 3, described is effectiveness of design review for quality improvement by analyzing process data of actual projects. In Chapter 4, described are how important a project plan is and that stability of development project contributes to improving quality and productivity. In Chapter 5, a case of process improvement is presented in which productivity was much improved. In Chapter 6, described are why the process improvement is difficult and how we will achieve process improvement successfully. In the last chapter, consideration of the paper is summarized.

Key Words

Software Process Improvement, Project Management, Metrics, Software Quality Control.

---

* Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9861009, June 2000

目次

実プロジェクトデータの分析に基づくソフトウェアプロセス改善と品質向上の研究

図表一覧

１．はじめに

米国上院通商、科学、及び輸送委員会で、当時アメリカ三菱研究所長であったベラディ氏はソフトウェアについて次のように証言している。

「ソフトウェアはなぜ問題を起こすのでしょう？それは、ソフトウェアが今まで産業が経験しなかった、まったく新しいものだからです。それは奇妙な性格を持っています。それは、あるときは製品に見えますが、あるときは本のように、あるときはレシピのようにも見えます。世界中の智恵を集めても、この無形の工業製品という奇妙な特性を持つものを、うまく扱えないでいるのです。」[1]

ソフトウェアの問題は世界的規模で人類が初めて出会う難しい問題であるにもかかわらず、この事を理解している人は少なく、問題を矮小化し個別事情として処理してしまう人が多い。ソフトウェアの開発はハードウェアの製造のように同じ物を繰り返すと言うことは有り得ず、個々の事情は全て異なっているため、そこで起きる問題も個別事情として処理されてしまう可能性が高い。しかし全く異なると見える個々のソフトウェア開発もモデル化し抽象度を上げて議論をすることにより共通点も多くなり共有化できる知識も多くなる。ソフトウェアの問題を解決するには、ことさら個別事情に立ち入るのではなく、成功例、失敗例の中から他のプロジェクトや他の組織でも活用できる知識を纏めて交換し合い、その知識を活用する段階で個々の事情に応じてカスタマイズし活用することが必要になる。つまり、世界的規模でソフトウェア開発に関する知識の抽象化、その知識の交換、個別プロジェクトへの対応のための具象化という活動が必要となる。

ソフトウェアで問題が発生すると「また、彼（ら）はこんな馬鹿な事をしてしまった」という言葉が良く聞かれるが、これはソフトウェアが持っている本質的な難しさを理解せずに誰かの失敗として片づけてしまう事になり、再発防止への活動をおろそかにしてしまう恐れがある。そこで、人間はどんな振る舞いをするのかも含めて、ソフトウェア開発の難しさはどこから来るのかを理解した上で、それを前提とした改善策を考えていく必要がある。その改善策は個人レベルの技術力や規律の向上だけではなく、組織文化の改善を含めたトータルとしての組織力の向上が必要である。

本論文では、効果的なプロセス改善でトータルな組織力を向上させるための取組みの方向性を示すと同時に実プロジェクトデータ分析に基づくプロセス改善の事例を紹介する。

２．プロセス改善と品質向上の研究の位置付け

ソフトウェアシステムの大規模化、複雑化にともないソフトウェア開発における生産性、及び、品質向上の実現はソフトウェア工学における研究の主要な目標に位置付けられてきている。ソフトウェアの品質や生産性を向上させるためには、開発されたソフトウェアプロダクトだけでなく、その開発プロセスを対象として作業の改善を行うことが必要である[2]。

本章では、まずソフトウェア開発の持っている特有の難しさがどこから来るのかを整理し、その難しさを克服する唯一の方法であるプロセス改善とはどういうもので、プロセス改善を成功させるためにはどんな知識が必要かについて述べる。さらにプロセス改善のゴールの中でも品質管理が特に難しい理由は何なのかを良く理解した上で改善の方向を決めていく必要がある事を述べる。

２．１ ソフトウェア開発の特質

ソフトウェア開発で多くの問題が発生するのは、決してそれを担当する人達の能力に問題があるのではなく、対象とするソフトウェア開発が今だかつて人類が経験したことの無い特有の難しさを持っているためである。このような問題に取組むには、その特有の難しさの内容をよく理解した上でそれぞれに対する解決策を考えていかなければならない。ソフトウェア開発の難しさの現象をまとめると次のようになる。

①　ソフトウェアは直感的に見えず、プロジェクトの状況も直感的に見えない。
②　客観的に評価しにくい。
③　全体像がはっきりつかめず、常に要求は変化する。
④　人の知的創作物ゆえ能力差の影響が大きい。
⑤　人と人との関わりが大きく、コミュニケーション上の問題が多い。
⑥　一般に他の工業製品より複雑である。対象物は物理実体ではなく、人間の論理的思考内容であるため、多次元の属性をもっている。
⑦　工学としての歴史が浅く、経験ある管理者や高度な専門家が少ない。
⑧　後で発見されたシステム・ハードの問題をソフトで解決することが多い。
⑨　ソフト担当外からは「ソフトは簡単」と思われながらソフトの問題に一歩踏み込む

人がほとんどいない。

⑩ ソフト開発に対する要求は常に拡大しており技術力向上が追いついていない。

⑪ 個々のソフト開発はすべてパターンが異なる。すべてが応用問題で、個別に工夫が必要である。

⑫ プロセスとプロダクトの因果関係は非常に複雑で、確率的な現れかたをするため、問題分析・対応が難しい。

これらの現象は複雑に絡み合って悪影響を与え合い、更に悪い結果につながっていくが、その大元は次の四点から出発すると考えられる。

（・） 開発規模の増大

（・） 工学としての歴史の浅さ

（・） プロセスは人間の知的活動

（・） 開発要件は人間の思考内容

これらの関係を図２．１に示す。

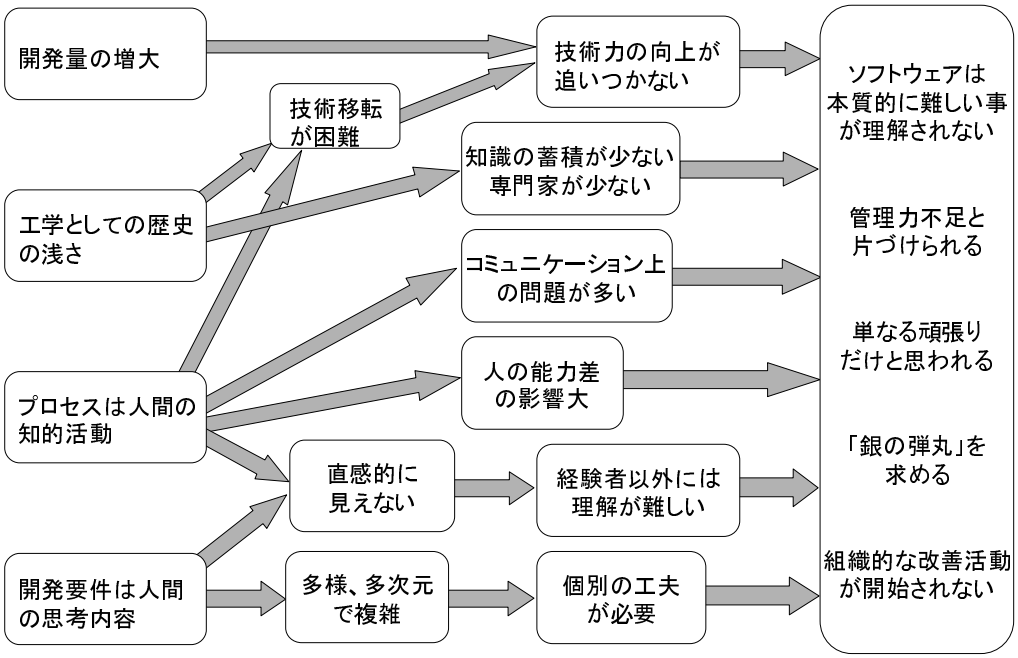

図2．１ ソフトウェア開発の難しさの原因

⟫ ソフトウェアで問題が発生した時にこれらの本質的な難しさがソフトウェア担当外の人、特に上級管理層や経営層に理解されないだけではなく、ソフトウェア担当者自身もこのことをよく理解できておらず、ましてや自分の失敗の原因を客観的に説明する事などできてはいない。このためソフトウェア担当者はますます悪い立場に追い込まれ、自社のソフトウェア担当だけが上手くやらないから、がんばらないから、管理が下手だから、他社では旨くやっているのに自社だけが失敗を繰り返していると思われてしまう。その結果、ソフトウェア担当者を責める事ばかりしたり、一発逆転の新技術（銀の弾丸[3]）を捜し求めたりし、この問題を解決する唯一の方法である組織的なプロセス改善が疎かになってしまうことになる。⟪

⟫ 以下で、上記四つの根本原因についてもう少し詳しく見てみたい。⟪

⟫ （１）⟫ 開発規模の増大⟪

⟫ １９４０年代後半にできた、現在の形をした最初のコンピュータは約１０００ステップのプログラムが書けたと言われている。その後、一つのシステムで作られるプログラムの量は文字どおり指数関数的に増え、各年代で作られた世界的な大規模ソフトウェアはほぼ１０年で一桁の増大を示している。ブルックスの「人月の神話[4]」に書かれているＩＢＭ３６０の開発は１９７０年前後のことで１００Ｋｓｔｅｐ程度の開発であったと思われる。当時は大変な苦労をして開発をしたと書かれているが、現在では１００Ｋｓｔｅｐ程度の開発プロジェクトはいたるところに存在し、たいていのところはそこそこ上手くやっている。現在の世界的レベルでの大規模プロジェクトは数十Ｍｓｔｅｐ程度になっているのではないかと思われる。⟪

⟫ また、マイクロコンピュータの組込み系ソフトの領域では普及の立ち上がりが遅かった事もあり規模増大のスピードは更に早く、たいていのメーカでは２０年間で３桁の開発規模増大に悪戦苦闘をしている。⟪

⟫ ソフトウェア以外の領域では規模が一桁大きくなるという事の大変さはよく認識されている。よく言われる喩えの「平屋の家を建てるのと１０階建てのビルを建てるのでは、自ずと技術・方法は異なってくる」というは万人の認めるところだと思うが、ソフトウェアの領域でも同じような事が起きる事を理解している人は少ない。ソフトウェアの規模が増大してくるとプロセス・プロダクトとも分割して管理をしていく必要があるが、どのように分割するか、そしてそれぞれをどのように管理するかがキーポイントとなってくる。⟪

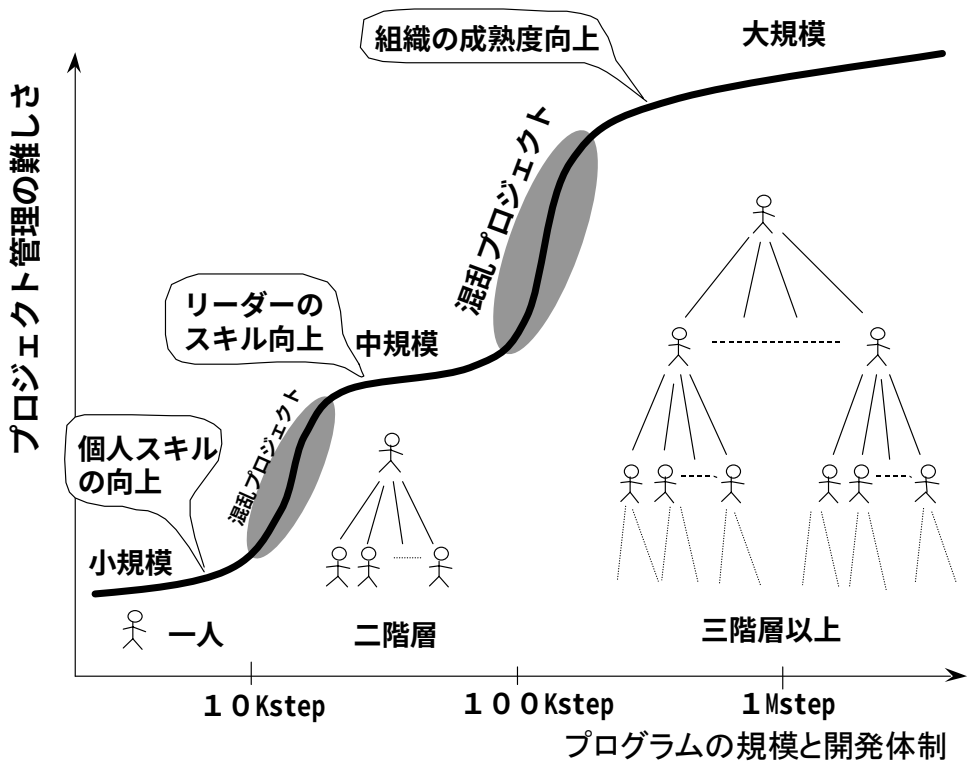

図2．2　開発規模とプロジェクト体制、プロジェクト管理の難しさ

般的にはこれらの分割されたものは階層構造の形になるが階層の数によって管理の難しさが飛躍的に増大する。図２．２に開発規模とプロジェクト体制、そしてプロジェクト管理の難しさの関係を示す。急激に難しくなるところを従来と同じと勘違いして、しかるべき対策を無しに取組むと、回復不能な混乱プロジェクトとなってしまう。中規模までのプロジェクトであれば個人またはリーダの頑張りで解決される事もあるが、大規模プロジェクトになると組織全体としての企業文化を含めた成熟度向上がないと根本的解決ははかれない。管理階層が三階層以上になると、もはやリーダ一人の頑張りでは解決できないレベルに問題は大きくなっている。見えにくいプロジェクトの状況を末端まで見えるようにし、問題が顕在化すればいち早く対処できるように組織としてのルールが有り、それが実際に浸透しているという組織としての成熟度が求められる。

〉（２）〉工学としての歴史の浅さ〉

〉良く知られている事であるが、ソフトウェア工学という言葉が初めて使われたのは１９６８年のＮＡＴＯの国際会議からであると言われている。以来、約３０年が経過しているが一般的な他の工学の領域が何百年という歴史を持っているのに対し圧倒的に浅いと言わざるを得ない。さらにソフトウェアの持っている他の特質からソフトウェア工学は、「物理・化学・数学などの理論を工業生産に応用するための学問・技術」という、工学としての定義のレベルに至っていないという意見まである。〉

〉メリーランド大学のバシリー教授は〉

「世界には二種類の難しい問題がある。〉

・深い問題：問題はよく定義されているが深く考えないと分からない問題。〉

・広い問題：何が問題なのかわかっていない問題。〉

〉〉〉〉〉〉〉問題が表面化してしまえば一つ一つはあまり難しくない。〉

〉〉〉〉〉〉〉これがソフトウェアの問題である。」〉

と言われた。一般の工学の対象は物理実体を伴った物についての「深い問題」であり、人間の活動を中心とした「広い問題」であるソフトウェア対しては、一般の工学と同じアプローチ方法を取ったり、他の工学分野のような理論の工業的応用を期待する事は元々無理な事とも言える。ソフトウェア工学の成果もソフトウェアの持っている特質をそのまま反映したものとならざるを得ず、漠として捉えどころが無く、理論の応用もそれぞれの工夫が必要になってくる。ソフトウェア工学を従来の工学の枠に入れることは本当はあまり適切ではなく、むしろ人文科学の領域に入れるべきではないかとも思う。いずれにしろ、プロセス・プロダクトとも人間の活動が大きく関わるため今後の研究には認知科学や社会心理学といった分野の知識の活用が必要になってくる。〉

〉（３）〉プロセスは人間の知的活動〉

〉ハードウェアの品質を決定する要素として、要件内容・設計プロセス・製造プロセス・物理的現象の四つのものが考えられる。物理的現象とは摩耗であったり落雷のような自然現象といったもので、設計段階で確率的に発生する事が想定されていたものと、全く想定外のものがある。いずれにしろ１００％防ぐ事は人知の及ばない領域であると考えられ、発生確率を低くする事が品質向上となる。この確率を表す指標にＭＴＢＦ（Mean〉Time〉Between〉Failure）とかＭＣＢＦ（Mean〉Cases〉Between〉Failure）といったものがある。い

ずれにしろ、ユーザ、メーカともある一定の確率で問題が発生する事を覚悟しているわけである。》

》 一方、ソフトウェアの品質を決定する要素を考えてみると、要件内容・設計プロセスの二つだけである。ハードウェアに比べて要素が少ない分、簡単のように見えるが実はこれがソフトウェアの問題を難しくしている本質である。最近の一部のパソコンソフトを除いては、何回かに一回はトラブルが発生する事を承知でソフトウェアを使うユーザはまず居なく、ＭＣＢＦといった確率の数値でソフトウェアの品質に折り合いをつける事は、まず不可能である。これら要件内容・設計プロセスの二つの品質を決定する要素はどちらも人間の知的な活動からのみ成り立っており、問題が発生した時はすべて人間の何らかのミスが原因となっている。このためミスを犯した担当者が責められて終わる事が多く、人間はミスを犯すものだという前提に立った改善や歯止め策が検討される事は少ない。また、ほとんどの開発がチームで行われるためメンバーの能力の差、メンバー間のコミュニケーションの問題が出てくる。ある程度大きな開発になると必ずしも十分な能力のある設計者ばかりでチームを編成することはできなくなり、そのような設計者への教育や設計結果の品質保証は大きな問題となってくる。さらに、たとえ優秀な設計者ばかりが集められたとしても、ソフトウェアの設計という知的作業について誤解無く、効率的にコミュニケーションをすることは大変難しいことである。》

》 このような人間の特質や能力の限界を認めた上で、プロセス改善によって組織としての歯止め策を講じていくことが大切である。》

》 （４）》 開発要件は人間の思考内容》

》 ソフトウェアに対する開発要件は物理実態による制約を受けることが無いため、無制限に多次元の複雑なものとなっていく。また、その複雑な要件を考える人間と、それをソフトウェアで実現する人間は一般的に別人であるために、正確に要件を伝えることはきわめて難しい。さらに要件を考える人間がソフトウェアで実現すべき要件をすべて定義しきるということも極めて難しいことである。要件定義者は頭の中で要件内容をシミュレートしながら要件定義をしていくため、要件定義者は設計者の知らないドメイン知識を、設計者は当然知っているものとして定義しなかったり、要件定義者が気になる主だった機能のみの定義で終わってしまって、異常処理に代表されるような細かい、しかしソフトウェアで実現するためには絶対に必要な機能の定義を省くことが多々ある。特に異常時の処理プロ

グラムは制御系のソフトウェアでは８０％を占めるといわれるほど重要なものであるが、軽く見られがちである上に、種々の異常パターンを考えて最初からすべてを定義しきるというのは大変難しいことである。ある業務をコンピュータでなく人間系で処理をする場合は、異常処理まで事前に考えておくことはまれで、異常が発生してから処理者の常識に従って処理方法が決定される（つまり仕様が決定される）ことになるが、コンピュータで処理をするには事前に発生しうる異常をすべて想定した上で（つまり仕様を定義した上で）プログラムとして組込んでおかなければならない。このようなことになる原因は要件内容を人間の頭の中でシミュレートすることの大雑把さと、コンピュータにより要件内容の処理を実行することの厳密さとのギャップから来るもので、ソフトウェア開発の持っている本質的な難しさの一つと言えるであろう。

２．２ プロセス改善の定義

前節でソフトウェア開発特有の難しさがどこから来るのかを述べたが、この難しさは何か特定の技術を導入すれば解決すると言ったものではなく、それぞれの組織の開発

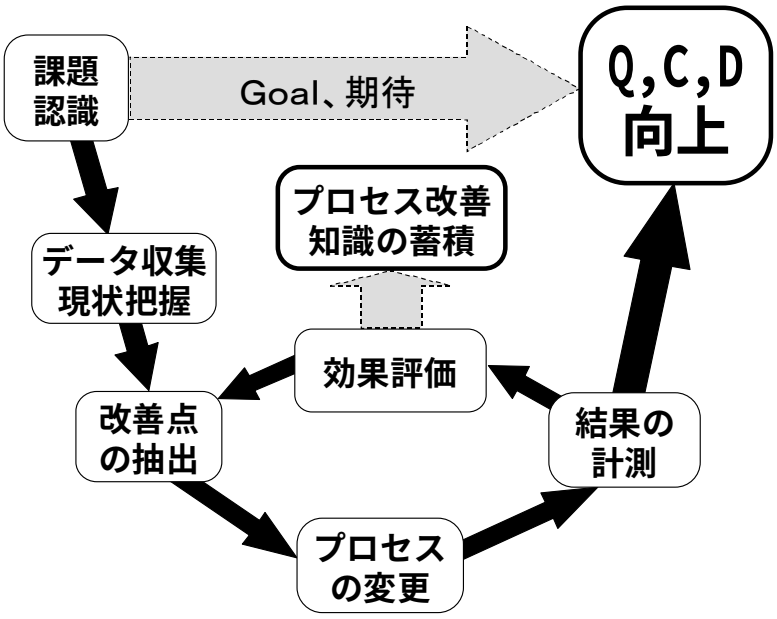

図2．３ プロセス改善サイクルのモデル

プロセスを分析しそれぞれが持っている問題を解決するプロセス改善で克服していかなければならない。》

》ソフトウェアプロセス改善とは、ソフトウェア開発のQ，C，D（品質・生産性・納期）の向上のために設計プロセスを変更することである。そのためには開発プロセスの分析、改善点の抽出、プロセスの変更、結果の計測・評価を行う必要がある。この事を示したプロセス改善サイクルのモデルを図２．３に示す。》

》プロセス改善がスタートするには、まずありたい姿としてのプロダクトのQ，C，Dがあって、それと自部門の現状とのギャップとして課題認識が必要である。その課題解決のためにプロセスデータを収集して現状把握し問題点の整理を行う。整理された問題点の中から優先順位をつけて具体的な改善点を抽出する。改善点を新しいプロセスとして定義し、その変更されたプロセスにしたがってプロジェクトを実行する。そして実行されたプロジェクトのプロセスとプロダクトのデータを計測し、プロダクトデータからプロセス改善のゴールであるQ，C，Dの向上を確認する。また、プロセスの変更量とプロダクトの改善量との関係を調べ、プロセスの変更がどの程度有効であったかを定量的に評価する。プロセス変更量とプロダクト改善量間の相関をプロセス改善の知識として蓄積する。そしてさらなる改善を目指して改善サイクルを継続的に回していく、というのがプロセス改善である。》

》つまりプロセス改善とは、「ソフトウェア開発プロセスに関わる課題が整理され、開発プロセスの変更とプロダクトのQ,C,D向上の因果関係が説明でき、それが知識として蓄積され、必要な経営資源を投入して継続的に行われる改善活動」のことであるといえる。》この定義に若干の説明を追加したい。》

》まず、「ソフトウェア開発プロセスに関わる課題が整理され」というのは、事業の観点から見るとソフトウェアのビジネスを成功させるためには開発プロセスの問題だけではなく、会社の運営方針に関わることや、商品企画、販売戦略といった様々なことがある。ところが往々にして「ビジネスが成功しないのはソフトウェア開発が問題だから」と、短絡的にソフトウェア開発の問題だけにされてしまいやすい。ソフトウェアの担当者が努力をして改善できることに対して期待されるのはいいが、その努力の及ぶ範囲外についてもソフトウェアの問題だといわれるといつまでたっても改善はされず、みんなが不幸になってしまう。それゆえ、ソフトウェアプロセスで改善できる範囲を明確にし、それ以外のビジネス上の問題はそれぞれの担当に割振る必要がある。》

》次に、「開発プロセスの変更とプロダクトのQ,C,D向上の因果関係が説明でき」というのは、プロセス改善と言うものを定められた標準に従うことだとか、今まで作っていなかったドキュメントを作るようになることだと単純に思っている人が多いので、ゴールはプロダクトのQ，C，D向上であることを明確にするために言っている。プロセス改善の成果はプロセスを評価するだけで測ってはいけないと言うことである。いろいろな人が「プロセスはこうあるべき」と言った確信をたくさん持っている。しかしそれら確信のほとんどは証明された知識ではなくあくまでも仮説である。良いと思われるプロセスに従うことによってプロダクトのQ，C，Dが向上したと言う因果関係の説明があって始めてプロセス改善の成果があったと言える。この因果関係の説明の無い標準を作り、開発現場に守らせていくというのはほとんどが失敗するケースである。プロセスを変更することによってプロダクトのQ，C，Dが向上すると信じられなければ開発現場のプロセス変更への同意が得られず、標準の浸透は難しくなる。同意を得られないまま強権発動で標準を守らせてもプロダクトのQ,C,D向上の因果関係が説明できていないものなので、開発現場への負担が増えるだけで何の成果も出ない可能性が高い。》

》そこでプロセス改善成功事例のプロセスとプロダクトの因果関係を整理し、「知識として蓄積」して広く技術移転をさせていく必要がある。この因果関係の整理と知識としての蓄積は開発現場の個人が簡単に出来るものではなく、しかるべき「経営資源の投入」が必要である。またこれらの活動は常に新しい問題が発生するソフトウェア開発においては「継続的に行う」事は当然である。》

》

》２．３》プロセス改善成功のメカニズムと知識》

》図２．４にプロセス改善成功のメカニズムを示す。プロセス改善というのは経営の効率を上げるためのものであるから、まず経営としての課題認識が大前提となる。もちろん開発現場の個人が課題認識をして改善をスタートさせることもあるだろうが、それだけでは改善成果には限界がある。前節で述べたようにきちんとしたプロセス改善にはしかるべき経営資源の投入が必要である。また経営としては経営資源の投資をするわけであるから、それによるリターン、つまり効果を何で計るかを定義しなければならない。この定義されるものには生産性・品質指標であったり計画精度であったりリワーク工数であったりするが、この定義された効果評価用の指標が経営から開発現場まで、縦通しで共通の目標とし》

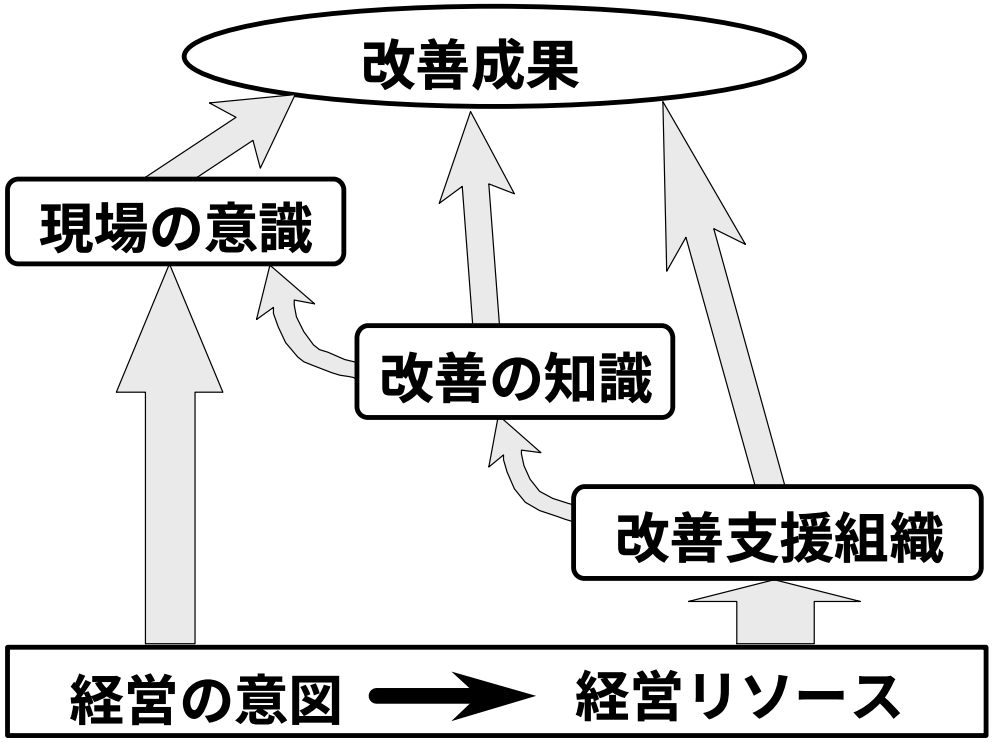

図2．4　プロセス改善成果のメカニズム

て改善活動の拠り所となるものである。プロセス改善のモデルは2．4節で述べるが、どのようなモデルをどのように自組織で適応していくのかを判断するのも組織としての取組み方法に関する知識として重要なものである。さらにプロセス改善は組織文化の変革と組合わされなければならないが、これをどのように有機的に結合させていくかも組織としての重要な知識である。

経営資源の投入によって組織化されるのが改善支援組織である。このメンバーのことを一般にＳＥＰＧ(Software Engineering Process Group)と呼ぶ。プロセス改善の成否はこのＳＥＰＧがいかに効果的にプロセス改善を推進するかにかかっているといっても過言ではない。プロセスチャンピオンといわれるような強力なプロセス改善推進役のＳＥＰＧになるには高いヒューマンスキルと同時に次のような知識が必要である。

⟫ まずプロセス改善の標準的なプロセスに関する知識と自部門への改善プロセスのカスタマイズ方法である。プロセス改善はやみくもに思いついた問題に対して取組んでいけばよいのではなく、その進め方にも標準的なプロセスが存在する。図２．３に示した「プロセス改善サイクルのモデル」も非常に大きなレベルでの改善プロセスの標準形を表している。世界的に有名なプロセス改善の標準プロセスとしてはカーネギーメロン大学ソフトウェア工学研究所の提唱するＩＤＥＡＬ(SM)[1][11]モデルがある。ＩＤＥＡＬは Initiating,⟫ Diagnosing,⟫ Establishing,⟫ Acting,⟫ Learning⟫ の頭文字を取ったもので改善活動の立ち上げから継続的な改善サイクルまでを表している。これら改善サイクルのもっとも重要なところは現状プロセスの問題分析で、このためには現状プロセスの把握方法、その内容を記述するための表記方法、表現されたプロセスから改善点の抽出方法、その改善点の取組み優先付けといった知識がＳＥＰＧには要求される。また改善点抽出のベースとなり、さらに改善成果評価のために必須となる開発管理データの収集項目と収集方法についての知識がＳＥＰＧには要求される。つまりメトリクスに関する知識である。⟫

⟫ ＳＥＰＧはこれらの知識だけではなく、次のような取組み姿勢が必要である。⟫

⟫ （・）ソフトウェア課題と改善成果状況について常に上級管理職や経営層に適確に報告⟫
⟫⟫⟫⟫ し、プロセス改善についての支持を常に維持していく。⟫
⟫ （・）開発現場と一体となって改善を推進し、単にプロセス変更のサポートをするだけ⟫
⟫⟫⟫⟫ ではなく結果としてのQ，Ｃ，Ｄの改善に責任を持つ。⟫
⟫ （・）開発現場の同意を得られないプロセス変更を強引に進めない。プロセス変更とプ⟫
⟫⟫⟫⟫ ロダクトの改善の因果関係をデータで証明し、開発現場の同意を得る。⟫
⟫ （・）自分達の改善成果を知識としてまとめると同時に、世界中から改善に関する知識⟫
⟫⟫⟫⟫ を吸収する。⟫
⟫ （・）ＳＥＰＧはプロセス改善の知識で開発現場からの信頼を勝ち取らなければならな⟫
⟫⟫⟫⟫ い。また開発現場の対象ドメイン知識に対して敬意を払わなければならない。⟫

⟫ このようにＳＥＰＧには、非常に能力が高く改善に対して情熱を持って取組む人を割り当てなければならない。⟫
⟫ ＳＥＰＧは設計者に対して知識の提供をし、その知識を生かして設計者が改善成果を上

---

[1] (SM) IDEAL is a service mark of Carnegie Mellon University.

げるようにするのを主たる役目としているが、場合によっては模範例として特定作業を分担し実行したり、設計者と共同作業を行い、直接的に改善成果を生み出すこともある。
　ＳＥＰＧが提供するのは、３章のデザインレビューによる品質向上や４章のプロジェクトの安定とソフトウェア品質・生産性向上で述べるような、プロセスとプロダクトのQ，C，Dとの因果関係の知識や、設計技法や開発環境に関する知識で、この知識を活用して設計者は効率的に設計を進めることが出来る。プロセス改善の成果は大半がここから得られるものである。
　改善成果を上げるためには企業文化の改革、設計者の意識改革も大切な要素である。ソフトウェア開発は大変高度な知的作業であるので設計者の意欲、意識が生産性と品質に大きく影響してくる。つまりモラールの向上が改善成果に大きく影響するということであるが、このためにはまず経営からのプロセス改善に対する明確な方針と指示が必要である。しかしこれだけであると設計者は改善の方法が分からないまま追いつめられる形になり、改善に対する意欲がそがれてしまうことになる。これを助けるのがＳＥＰＧの提供する改善に関する知識で、この知識を活用すれば設計者自身も良い仕事が出来、さらに経営の要求を満足させることが出来、設計者の改善に対するモラールを向上させることが出来る。

２．４　プロセス改善のモデル

　これまでに、様々なプロセス評価・改善手法が提案されている[5][6][7]。ISO9001[6]は二者間契約におけるソフトウェア品質保証システムの規格であり、品質の高いソフトウェアを開発するためのガイドラインがプロセス全体にわたって提示されている。また、各種の文書を用意するという管理的な側面が重視されている。一方、ハンフリーによって提案された能力成熟度モデル[5]はソフトウェア開発組織が信頼性の高い安定したプロセスをどうすれば作っていけるのかをモデル化したものである。その目的はソフトウェア開発組織の状況をプロセスの観点から分析し、その組織に合ったプロセス改善の目標を設定することにある[8]。このモデルはその後、カーネギーメロン大学ソフトウェア工学研究所（ＳＥＩ）により能力成熟度モデル Ver.1.1（CMM: Capability Maturity Model®[2]）[9][10]

---

[2] ®Capability Maturity Model and CMM are registered in the U. S. Patent and Trademark Office.

として纏められ、ソフトウェアの品質・生産性向上のためのプロセス改善（SPI:Software Process Improvement）を進めるモデルとして米国内のみならず欧州でも実質的なプロセス改善指針となりつつあり、「プロセスに着目した組織の能力および体質の改善が結果として組織全体の生産性ならびに品質の向上に寄与する。」という考えを提唱している。

CMMは組織としての成熟度に焦点を当て、Q，C，Dの不満足な結果を出す未成熟な組織と、満足な結果を出す成熟した組織を５段階に分類し、各段階を上がっていくための注力点としてＫＰＡ(Key Process Area)を定義している。以下にCMMの概要を簡単に述べる。

図２．５に成熟度の向上とQ，C，Dプロセス実績の関係を示す。図の横軸はQ，C，Dの満足度を表しており、右に行くほど不満足な結果を表している。縦軸はある組織が複数のプロジェクトを実行した時のプロセス実績の分布を表している。未成熟な組織においては達成不可能な計画値で開発をスタートし、結果は分散が非常に大きく、計画の２倍３倍というプロジェクトが存在し実績の平均値は計画から大きく乖離したものになる。

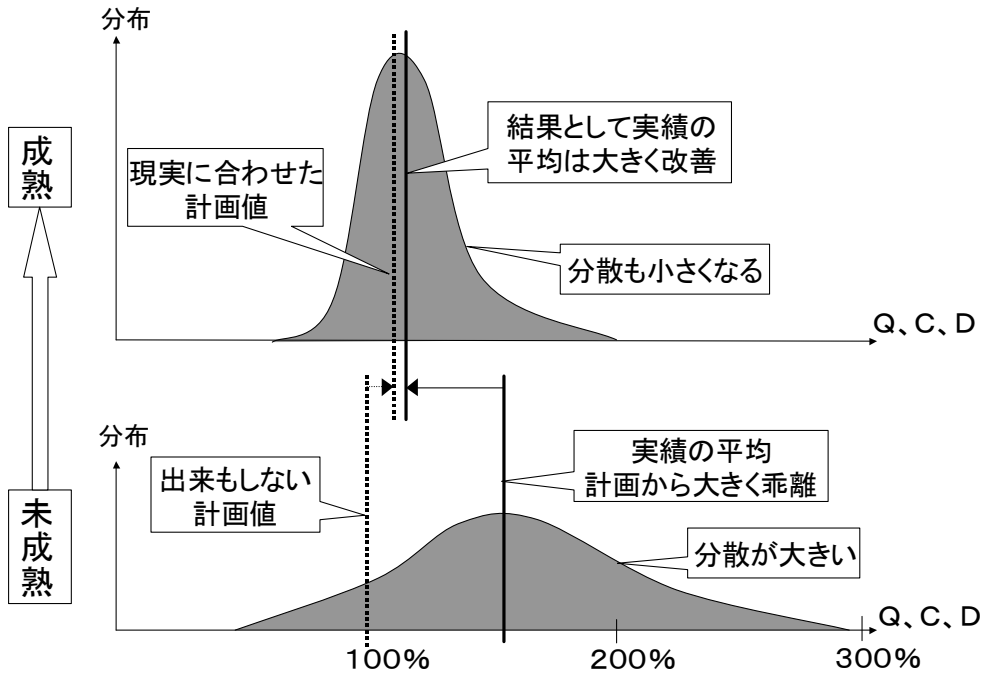

図2．５ プロセス成熟度の向上とQ，C，Dプロセス実績

プロセス実績分布の分散と平均値および計画との乖離の３点はプロセス実績から見ての成熟度評価の大切な要素となる。》

》一方、成熟した組織では現実に合わせた計画を立てるようになるために、未成熟な組織が立てる達成不可能な計画値より最初は悪い計画値になる。しかし、実績は計画に非常に近いものとなり結果的には未成熟な組織に比べてQ，C，Dはずっと改善されたものになる。また、プロジェクトによるばらつきも少なくなり、プロジェクトが制御可能になっていることを示している。》

》プロセス成熟度が上がっていくと、この分散と平均値および計画との乖離の３点が同時に良くなっていくと言われている。成熟度を上げていくためには段階を追って上げていく必要があり、途中の段階を省略することは出来ない。各段階から次の段階に上がるために特に注力すべき項目としてＫＰＡが定義されている。図２．６に５段階の成熟度レベルと各レベルのＫＰＡを示す。このＫＰＡはそれだけを行えば次のレベルに上がれるというものではなく、あくまでも各レベルにおける特に注力すべきポイントを述べている。したがって各レベルのＫＰＡはそのレベルになって始めて実行するものではなく、それより以前から取組んでおくべきものであるが、各レベルを達成してしっかりと歯止めをかけるためにはそれぞれのＫＰＡが必須であることを示している。》

》プロセス成熟度が上がっていくとその組織が実行するプロジェクトの可視性がどのように変化するかを示したのが図２．７である。ソフトウェア開発は見えないと良く言われるがそれは成熟度レベルが低いからであって、成熟度が上がっていくとリアルタイムにプロジェクトの状態が数値で把握でき、プロジェクト管理へのフィードバックが可能になってくる。》

》典型的なレベル１の組織ではプロジェクトの開始と終了が分かるだけで、プロジェクトの実行中の状況は外部からは全くといってよいほど分からない。プロジェクトを実行している人達自身にも状況が分からないのがほとんどである。これがレベル２になると工程単位程度のプロセスが定義され、この単位でのQ，C，Dの確認がされながらプロジェクトが実行される。レベル３になると各工程の中の作業レベルが定義され、この単位でプロジェクトの状況が定量的に把握できるようになる。レベル４になると各工程の作業レベルの進捗状況が定量的に把握され、当該作業制御のためのフィードバックとして使われる。さらにレベル５になるとプロセス定義も固定的なものではなくそれぞれのプロジェクトにとって最適な状態にダイナミックに管理されるようになる。》

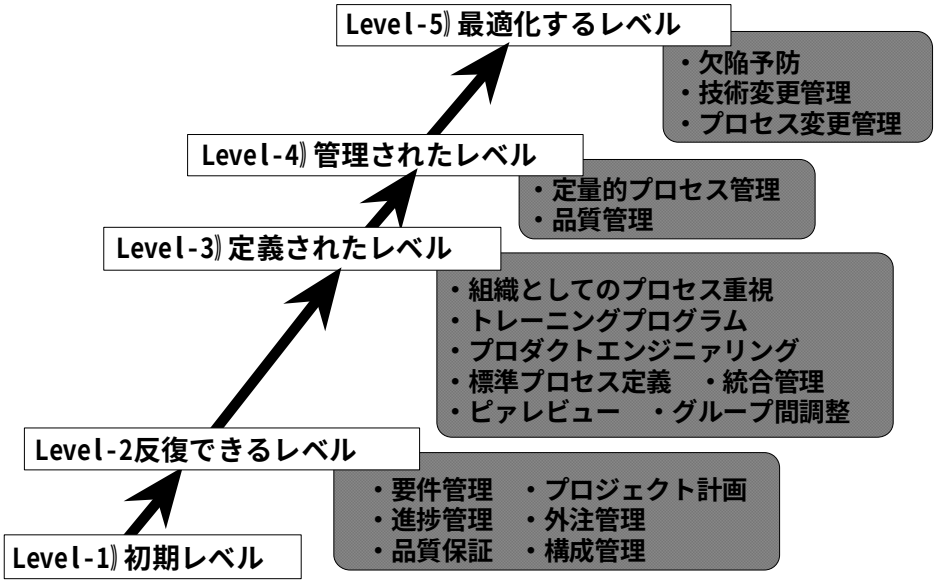

図2．6　各プロセス成熟度レベルでの注力点　KPA(Key Process Area)

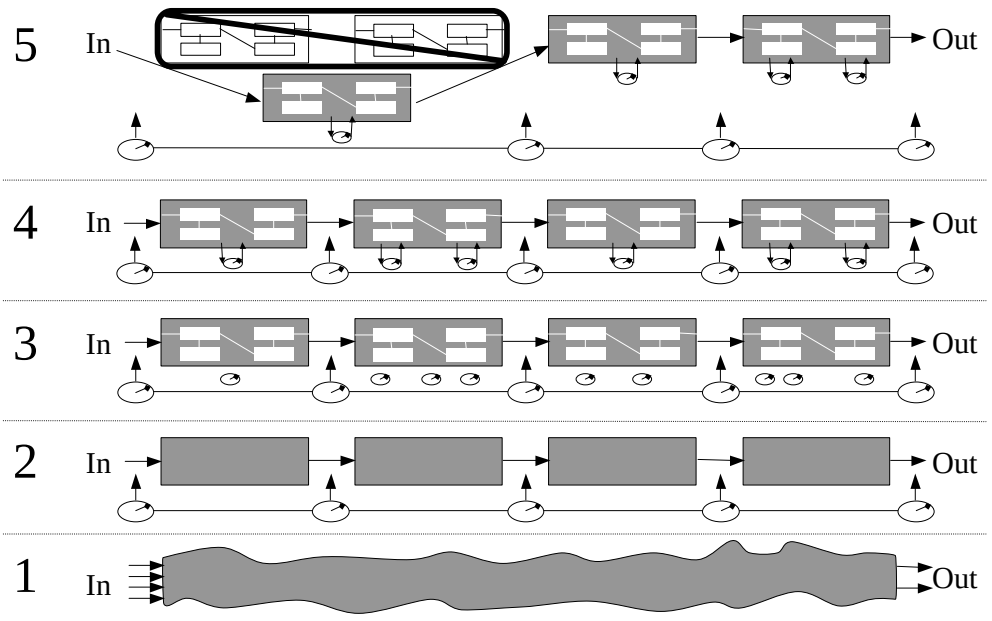

図2．7　プロセス成熟度とプロジェクトの可視性

⟫ ＳＥＩはCMMを活用するためのフォーマルアセスメントの実行プログラムも用意している。ただこれはあくまでもプロセスの評価であって、図２．５に示したようなQ，C，Dプロセス実績の評価は含んでいない。CMMはプロセスの改善点を発見しそれを改善していくためのモデルであるので、その結果であるQ，C，Dプロセス実績の評価は必要ではないという意見もあるが、その改善がどの程度有効であったかを評価し知識として蓄積するためにもプロセス実績の評価は必要であると思う。またCMMは組織の成熟度を１から５の５レベルに判定するため、元々は良いプロセス実績を得るためのプロセス改善のはずが、高い成熟度レベルの判定を得るためのプロセス改善に陥りやすい。CMMはあくまでモデルであり、すべての組織にそっくりそのまま当てはまるとは限らないもので、各組織に適応した工夫が必要であるが、ＫＰＡの表面的な項目のみを改善して高い成熟度レベル判定をも得ても、良いプロセス実績が得られなければ無駄な努力をしただけで、決してプロセス改善をしたとは言えない。アセスメントとかレベル判定というと常にこのようなレッテルをもらうための活動となりやすいため、経営トップから改善推進者まですべての人が十分心すべきところである。⟪

⟫

⟫ ２．５⟫ 品質管理の難しさ⟪

⟫ ソフトウェア開発の持つ難しさの中でも品質管理は最後まで残る難しさである。「Q, C, Dの向上」というように、品質・開発コスト・納期をならべて表現することが多いが、品質は何故他の二つに比べて制御が難しいのかをその特質から考えてみたい。⟪

⟫ （１）⟫ １００%確かでないことによる難しさ⟪

⟫ ソフトウェアの品質はその作り込み過程、つまりプロセスのみによって決定される。よって、品質向上のためには良いプロセスでソフトウェアを開発すべきということになる。しかし、プロセスというのはプロダクトという結果を生み出す原因であるので、プロセス単独での善し悪しの評価はできない。良い品質のプロダクトを生み出すプロセスが良いプロセスということになるが、このプロセスとプロダクトの因果関係を明確にして関係者全員で良いプロセスを共有化することは、因果関係に関わる要素が大変多いことから非常に難しい問題である。品質に関わるプロセスの要素として、工程毎の工数比率とかデザインレビューの工数比率といったものがあることは分かってきているが、これらはあくまでも良い結果を生み出す確率の高いプロセスであることが統計的に証明されているというだけ

である。例えば、デザインレビューの工数比率を２０％にしたからといってフィールドでのバグ件数がゼロになるという保証はどこにも無く、単にゼロになる可能性が高くなるということを言っているだけである。不具合が発覚した後であればどこかにバグが存在するのは確かで、プログラムの見直しという努力もバグ修正ということで報われることになるが、バグが存在するかもしれないというプログラムを、発見できるかもしれないという期待だけで、設計者は１００％確かな成果が出る保証の無い努力を強いられることになる。》

》 開発コストや納期についてはプロセスとの因果関係は明確で、例えば、デザインレビューの工数比率を２０％とれば、その部分については確実に開発コストと納期に影響を与える。それでもデザインレビューを充実させなければならないというのは、単に品質向上だけでなくそれによって手戻りを減少させ、結果的にトータルの開発コストの低減と納期の確保を期待するからである。プロセスの努力によって明確に増える開発コスト・納期への負担と、確率的に得られる品質向上に伴うメリットとの間で、コストと時間の投資をし続けるというのは大変難しいことである。》

》 また、バグの残ったソフトウェアがリリースされてしまっても、ユーザのところでたまたまバグのルーチンを通らなければ問題は表面化しないわけで、これもまた確率的である。》このように確率的であることから「自分だけは大丈夫」と根拠の無い期待を抱く設計者が多く、必要なプロセスの努力がおざなりになり勝ちとなる。》

》 （２）》 問題表面化までの時間後れによる難しさ》

》 品質に絡む問題を除けば、開発コスト・納期に関する問題の表面化は時間後れが無く、直裁的である。しかし、品質は直接的に見えないということから、問題が潜在化し先送りされてしまうことが多い。品質向上のプロセス努力の成果はその時点では直接的には見ることができず、後工程やリリースされた後にその結果が判明することになる。人間の性として問題先送り型の行動をとることが多く、明日の品質向上のために今日努力をしておくという投資的な行動というのはなかなか難しいものである。》

》 （３）》 絶対目標があることによる難しさ》

》 開発コスト・納期に関する目標は顧客の要求との相対的な関係によって決定されるもので、絶対的な目標値があるわけではない。顧客の無理な要求によって実現不可能な目標値が設定されるという危険性はあるが、それでも顧客との交渉の余地はある。ところが品質

についての顧客の究極要求は「バグゼロ」と非常に明確で、この絶対目標は交渉の余地の無いところである。ハードウェアの場合、ある一定の割合で発生する不具合は物理的な限界から来るものとして顧客も理解し、どの程度の確率であれば満足してもらえるかの交渉の余地はある。しかし、ソフトウェアで発生する不具合は全て人間系のミスによって発生するものであり、論理的には不具合はゼロであるべきと考える人が多い。実際には人間はミスを犯すものでありバグゼロは現実には非常に難しく、しかもこの絶対目標との間には交渉の余地が無いところから、どこまでも品質向上への努力が求められることになる。

（４）品質と開発コスト・納期の関係

開発プロジェクトによって作り出されたプロダクトのQ，C，Dを考えた時、本来の品質とは、開発が終わりプロダクトがリリースされた後に顧客に迷惑をかけたバグの数で評価すべきである。この意味において、開発中における品質問題というのは直接的には本来の品質問題と別物と考えるべきであるが、開発中の品質はリリース後の品質に大きく関係することと、開発中に明確になった品質問題は結果的に開発コストと納期に影響を与えることから、開発中の品質もここでの議論の対象とする。品質問題が明確になっているにもかかわらずそのまま開発完了してリリースすることはまずありえないことであり、品質問題があるにもかかわらず開発完了するのは品質問題が潜在化しているためである。開発中に品質問題が顕在化すれば、潜在化していた開発コスト・納期の問題が顕在化してくる。潜在している品質問題を不幸にして開発中に発見できなかった時は、一見、開発コスト・納期とも目標達成ができたように見えてプロジェクトは終了するが、リリース後に顧客のところで本来の意味の品質問題を起こすことになる。このリリース後の不具合対応コストも開発コストの中に含めて考えると結局は品質が開発コスト・納期もすべてを決定することになる。

よく「品質とコストはトレードオフの関係にある」という意見を聞くが、リリース後の失敗コストも含めて考えると一般的にはこの意見は的が外れていることが多い。

図２．８は品質レベルと品質コストの関係を示す概念図である。品質コストは大きく予防コストと失敗コストに分けられ、予防コストは品質レベルを高めるためには高くつく右肩上がりのグラフに、また失敗コストは品質レベルが高いと安くなる右肩下がりのグラフになる。この二つをたしたトータルコストの一番低くなるところが最適な品質レベルとなる。当然失敗コストの中には品質低下によってもたらされた顧客からの信用失墜に伴って

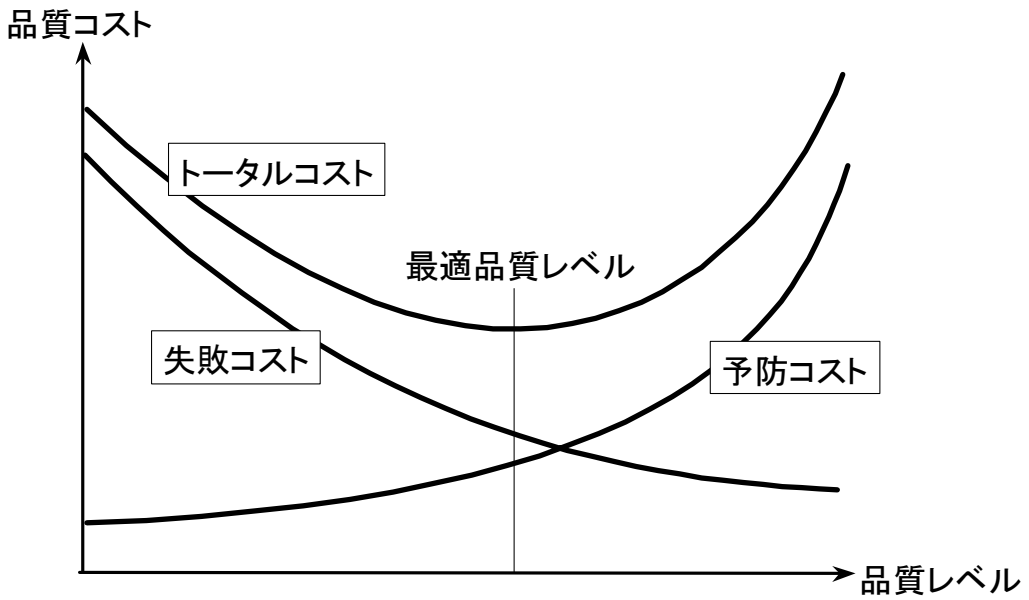

図2．8　品質レベルと品質コストの関係

発生するコストも含まれる。失敗コストは低い品質レベルにおいては大変大きいものになっているが少し品質レベルが上がると急激に低減できる。一方予防コストはある程度までの品質レベルであればあまり大きなものにはならないが、非常に高い品質を実現するためには急激に大きくなる。これらの曲線はそれぞれの対象ドメインが置かれている状況により変化してくる。例えば、パソコンのパッケージソフトであれば極端に品質が悪くない限り、一般的には失敗コストは低いものであるので結果的に最適品質レベルは左に移動する。また、例えば人工衛星の中のソフトウェアであればわずかな不具合でも衛星全体の機能に影響を与えることになれば究極の高品質にしない限り失敗コストは高いままの状態となり、最適品質レベルは大きく右に移動することになる。高品質レベルを実現するための予防コストについてもそれぞれの組織の持っている技術力によって増加曲線が異なってくる。非常に高い品質レベルを要求される対象ドメインにおいては、例えばフォーマルメソッドのような高品質のための設計技法を用いて予防コストの増加を押さえ、最適品質レベルを右にシフトさせなければならない。

ここで「品質とコストはトレードオフの関係にある」いう意見を振り返ってみると、この意見が正しいのは最適品質レベルを超えて高品質レベルを実現しようとした時である。

これはいわゆる過剰品質の状態ということになるが、トレードオフ論を聞く場合はこのような高品質の状態であることはまずありえず、ほとんどの場合、最適品質よりはるかに左側の位置にあることが多い。このような誤解が生じる原因としては次のようなものが考えられる。》

（・）》 失敗コストのことを忘れている》
品質コストが予防コストのことだけであると確かに単調増加となるが、トータルコストは図２．８に示すように決してそうはならない。これは全く論外の楽天家であるといえる。》
（・）》 品質は十分確保されていると思っている》
なんの客観的データも無いまま、最適品質レベルを達成していると誤解している状態で、ほとんどの場合、実際は最適品質レベルよりかなり低い状態にある。開発プロセスデータの蓄積により品質状況を客観的に把握する取組みが必要である。》
（・）》 予防コストは非常に大きなものであると思っている》
図２．８のグラフでいえば予防コストのグラフは急なカーブで立ち上り、最適品質レベルはかなり左にあると誤解している場合である。この誤解が生じるのは効率よく品質を向上させる方法を知らないためであり、自社内の開発プロセスデータの分析や、外部からの知識の吸収が必要である。》

》 （５）》 先憂後楽の精神論》

》 ソフトウェア開発は見えにくいといわれる中でも、品質は確率的であることと問題発生までに時間後れがあることより、状況を特に見えにくくしている。この見えにくく、不確かな状況というのは管理サイドから見ては当然のこと、設計担当者自身にとっても同じ事が言える。このような状況で、片や明確に状況が分かっている開発コストと納期のプレッシャに押し切られ、品質に対する甘い判断が下されてしまうことになる。明確なありたい状況と不確かなあってほしくない状況が人の心の中で葛藤を起こした時に、ありたい状況への期待が次第に確信になっていくというのは、認知心理学の領域で認知的不協和理論としてよく知られた話である。この場合でいえば、品質に対する不安があっても開発コストと納期を守りたいという願望から品質予防コストを削りたいという誘惑に駆られ、「品質は大丈夫であってほしい」から「多分品質は大丈夫だろう」、さらに「品質は大丈夫だ」と認識が変化していくのである。このような人間の心理から考えても最適品質コストのレベ

ルまではなかなか品質予防コストがかけられないものだという認識が必要である。⟫
⟫ このような状況を打破するのは開発コスト・納期に対する意識以上に品質確保への強い意志が設計者に求められる。まさに先憂後楽の精神が必要であるが、これは同じ負担の作業を後で実行するか先に実行するかだけではない。しかるべき品質予防コストをかけないと問題を先送りするだけではなく、後で手抜きをした以上の負担となって返って来るということを良く理解しておく必要がある。⟫
⟫ 品質向上には設計者の意識や意欲といった非常に人間臭い部分が大きく影響を与えるため、確実な歯止めをすることが非常に困難になるのである。このために設計者の自発的な意識を待つだけではなく、組織としての品質に対する方針を明確に打ち出し、常に意識付けをしていくのは勿論、開発プロセスデータの分析にプロセスとプロダクトの品質の関係を明確にし設計者に品質予防コストの妥当な値を知らしめること、さらには知識を提供するだけであるとプロジェクトの差し迫った状況の中ではその知識も隅に追いやられてしまう危険性があるため、必要な品質予防コストを掛けているか否かを組織としてチェックするしくみが必要である。⟫
⟫

⟫ ２．６⟫ 品質決定要因⟫

⟫ ソフトウェアがプロダクトとして出来上がりリリースされた後、ユーザのところで起こす不具合、つまりフィールド品質はどのような要因で決定されるかを図２．９に構造的に表してみた。⟫
⟫ これらの要因は、最初の設計でいかにバグの少ない高品質のプログラムを設計するかという作り込み品質と、作り込んでしまったバグをいかに徹底的に摘出するかのバグ摘出品質の二つに大別される。⟫

⟫ （１）⟫ 作り込み品質⟫

⟫ 品質の作り込み工程の後にはレビュー・テストのバグ摘出工程があるので設計の作り込み品質はフィールド品質に関係しないというのは間違いである。バグの摘出は作り込まれたバグに対して確率的に行われるものであって、１００％のバグ摘出率を期待することはできない。よって、フィールド品質は作り込み品質に（１－バグ摘出率）をかけたものとなり、作り込み品質に比例することになる。⟫
⟫ 図２．９では作り込み品質の要素を５つに分解しているが、これらはほとんどが従来か

ら言われている設計技術力そのものであり、プロジェクト実行中の管理面から強化できるものはあまり多くはない。

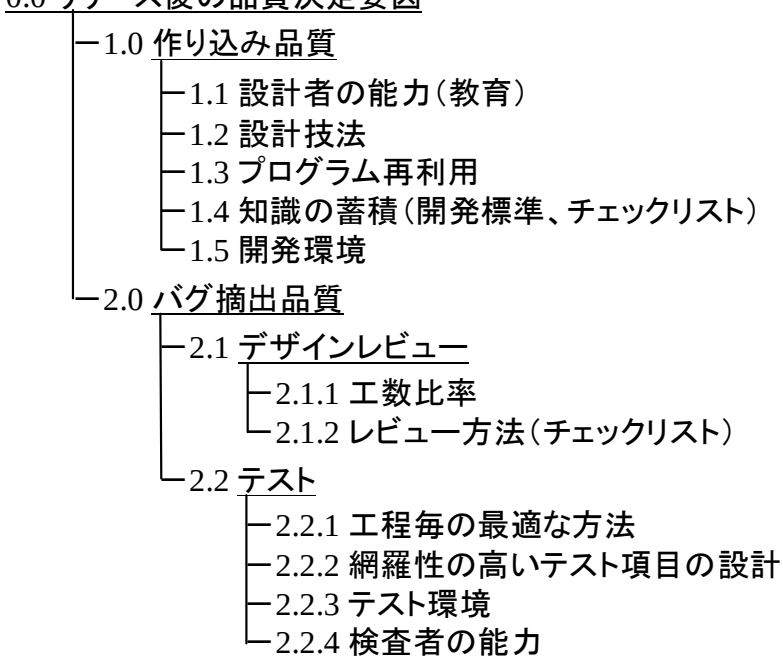

図2．９　品質決定要因の構造

①　設計者の能力

技術力というのは結局は設計者がその能力を発揮して初めて実現するものであるので、これがすべてといえなくも無い。高い能力を持つ設計者を確保したいというのはどの組織にとっても当然の要求であるが、これが思ったように行かないところにソフトウェアの問題の大半が有ると言ってもよい。そこで管理的側面として教育が行われることになるが、ソフトウェア設計と言う特殊な技術に対して設計者の適性というものもあり、特に最近言われているオブジェクト指向技術などについては設計者の向き・不向きを判断していく必要がある。何年か前はプログラム言語を知っていさえすればソフトウェア技術者と言われていたこともあったが、ソフトウェアに求められる要求が高度になるに従い技術者の選

別・淘汰もやむを得ないことと言える。》

》②》設計技法《

》従来から悪いプログラムの代表としてスパゲッティプログラムとか鰻の寝床プログラムとか言われているが、このような悪いプログラムは生産性を落とすだけではなく当然品質も悪いものになる。これを解決するために構造化設計の技法が広く使われて品質向上にも貢献している。いわゆる狭義のソフトウェア工学は生産性・品質を高めるための技法を追求してきて一定の成果を上げており、どのような技法を使うかは作り込み品質のために非常に大切な要素となる。特に、原子炉の制御ソフトに代表されるような、非常に高度の信頼性が要求されるミッションクリティカルにシステムにおいてはフォーマルメソッドのような格段に信頼性を高めるための技法を使っていく必要がある。また最近話題のオブジェクト指向を使ってテスト工程でのバグ件数を二桁も低減させた事例も報告されている。》

》こういった技法は難しいものが多く、世間で良いと言われているものを導入したからといって簡単に効果が上がるものではない。安易に新しい技法を導入して混乱した事例はたくさん有る。結局技法を使いこなすにはそのための設計者の能力が必要であり、技術者教育・人材確保と合わせて考えていかなければならない。》

》③》プログラム再利用《

》同じ条件で使っている限りソフトウェアにはハードウェアの摩耗に相当するような信頼性の劣化というものは有り得ないので、十分に使われて実績のあるソフトウェアを再利用するのは品質向上に貢献する。しかし実際には同じ条件で使われることは少なく、再利用に伴う問題もよく聞かれる。これはソフトウェアの仕様は非常に複雑なもので、再利用モジュールの仕様を全てドキュメントに表現しきれていなかったことによることが多い。この原因は一般的にはモジュールの独立性が低く他のモジュールと複雑に関係しあったものになっているためである。設計者は十分表現しきれていない仕様を何とか理解して使う努力が必要で、さらに再利用したモジュールにバグがあると設計者はブラックボックスとして使えばよかったはずのモジュールの内部を追いかける必要があり、極端に効率を落とすことになる。このような事から一般的に設計者は再利用をしたがらない傾向にあるが、これを解決するには設計技術面、管理面の両方からのアプローチが必要である。》

》設計技術面としては独立性の高い再利用モジュールを設計することと、その仕様をすべて定義しきることである。このためにはオブジェクト指向のような設計技法の導入が必要

になってくる。⟫

⟫ 管理面については、十分デバックして品質を高めたモジュールを再利用モジュールとして組織的に管理をする体制を作る。組織的管理の中には再利用モジュールの仕様面の問い合わせ対応やバグがあった場合の対応体制が含まれる。そして、再利用をしたがらない設計者の意識を変えていくために管理サイドからの方針の徹底と設計者への意識付けが継続的に必要である。⟫

⟫ ④⟫ 知識の蓄積⟫

⟫ これは設計者の能力向上を助ける一つの手段ともいえるが、一般的な設計技術ではなく自部門の経験から選られた設計ノウハウや対象ドメインに関する知識を開発標準とか設計マニアルといった形でドキュメント化し教育していくことが大切である。⟫

⟫ ⑤⟫ 開発環境⟫

⟫ 良い開発環境によって設計者は快適に作業ができ、設計に集中できることによって品質が向上するということが一般的には言える。しかしこのような一般論だけではなく、設計技法をサポートするツールであるとか直接品質を高めるチェックツール、静的解析ツールと言ったものが多くあり、活用をしていくべきである。⟫

⟫ （２）⟫ バグ摘出品質⟫

⟫ 作り込み品質を決定する要素はほとんどがその組織の持っている根源的な能力に関係するもので､重要であることは分かっていても一朝一夕には能力向上が図れるものではない。長期的な計画に基づいてこの能力向上を図っていく必要があるが、これとは別の視点で現状の組織能力のもとで作り込まれてしまったバグをいかに効率よく１００%近く摘出するかが重要となってくる。これをここではバグ摘出品質と言っているが、この摘出する方法として設計内容をチェックし直すデザインレビューと、実際にコンピュータでプログラムを実行してバグを摘出するテストの二つの方法がある。⟫

⟫ ①⟫ デザインレビュー⟫

⟫ ここで言うデザインレビューとはピァレビューともいわれるもので、設計者と同レベルまたは上位者といった同僚によって設計内容を詳細にチェックするレビューのことである。上級管理職も参加しての開発全体の状況把握をして今後の方針を判断するための公式デザインレビューはバグ摘出とあまり関係が無いので、ここでは議論の対象外とする。⟫

》デザインレビューは設計の初心者にとって手間のかかる面倒な作業としか映らないことが多いが、品質向上さらにはトータルの生産性向上にとって非常に大切なものである。デザインレビューで全バグの少なくとも５０%、うまくいけば８０%の摘出が可能でデバッグ・テスト工程に持ち込まれるバグの数を減らして品質を高めるだけでなく、バグ一件当りの摘出効率・修正効率の良さから生産性向上にも寄与する。但しこれはプログラムの対象ドメインによって一概に言えないことで、帳票への印字フォーマット編集プログラムや画面の編集プログラムのようにコンピュータで実行させれば容易に結果がチェックできるものはドキュメント上の細かなチェックを延々と続けるよりは実行させてチェックした方がはるかに早いというものもある。ソフトウェアの対象としている範囲は極めて広いため、全てに有効な画一的な方法というのは少なく、それぞれのドメイン毎に担当者が有効性を十分理解した上で取り組むべきである。》

》デザインレビューによるバグ摘出率を高める要素として、大きく分けてデザインレビューにかける時間とデザインレビューの方法の二つがある。デザインレビューにどの程度の時間をかけるべきであるかは対象ドメインによる違いはあまり無く、有効な知識と言えるので第３章で詳しく述べる。デザインレビューの方法については対象ドメインによる違いが大きく一般的な知識として共有できるものは少ない。》

列挙すると次のようなものである。》

（・）》レビュアの人数はせいぜい数人で、あまり多くない方が良い。》

（・）》特にコードレビューの場合は設計者とレビュアの一対一で行った方が良い。》

（・)》コードレビュー時には関連ドキュメントを全て準備しコードの正当性を確認する。》

（・）》コードレビューは設計者がプログラムロジックを説明する形で進める。》

（・）》設計ドキュメントのレビューは事前に説明会を開き、各自が個別にレビューをす》》》》る方が効率は良い。》

（・）》レビューの指摘結果は必ずドキュメントに残し、対応結果を最後までフォローす》》》》る。》

（・）》レビュー時に使用するチェックリストは有効である。》

》またこれらの他に、ボーイング社ではレビュアがＯＫを出したソフトウェアの品質責任はレビュアが持つという仕組みにしたところ飛躍的に品質が向上したという事例が有るとのことである。しかしこういった事例は国民性とか企業文化の違いに依存し、必ずしも成

功する方法とは思えない。また、チェックリストの内容についてはそれぞれのドメイン毎に自分達で過去の経験に基づいて作成しないと本当に有効なものとはならない。いずれにしろ各組織毎・ドメイン毎に定性的でも良いから有効と思われる方法をリストアップし共有化していく努力が必要である。》

》②》テスト》

》フィールド品質確保の最後の砦となるのがテスト工程である。フィールドと同じくコンピュータによってプログラムを実行させての確認となるが、フィールドで実行されるすべてのパターンを確認することは不可能なことで、フィールドではたまたまテストで実行されなかったパターンでバグが発覚することになる。すべてのパターンを確認できないことからバグ摘出も確率的となるため、フィールドでのバグ数はテストに入る時点でのバグの数に比例することになる。設計品質を高め、デザインレビューでのバグ指摘率を高めようというのは、テストに入る時点のバグ数を減らし、品質向上とテスト工程でのバグ対応工数の削減をはかるためである。》

》しかるべき設計およびデザインレビューが行われ、テスト工程に入力されるバグ数が決まった後は､いかに効果的かつ効率的にバグ摘出率を高めるかがテスト工程の課題となる。》まず、テストはコンピュータによってプログラムを実行するといってもいきなり一気に実行するのではなく、ステップを追って実行しないと効果的でないし、効率も悪くなる。これを無視して一気に実行させるのをビッグバンテストなどと言われるが、初歩的な効率の悪い方法の典型である。未成熟な組織においては各ステップでの行うべき事を省略してでも早く目に見える形での動作確認をしたがる傾向があるが、これは品質問題を起こす典型的なパターンである。やるべき事をやるべきステップで確実に実行するというのが品質向上のための大原則である。》

》このテスト工程の分割方法の詳細は各組織・ドメインの状況によって工夫をすべきであるが、最低限ホワイトボックステストとブラックボックステストに分けるべきである。ホワイトボックステストとはプログラムの内部のロジックを追いかけながら確認をしていくもので、動作は細切れになり全体の確認ができない中での手間のかかる作業となるため設計者は嫌がる傾向に有る。しかし実際は大変バク摘出効率の高い作業でテスト全体の８０－９０％のバグを摘出することが可能である。一方ブラックボックステストとはプログラムの内部構造を無視して外部仕様からだけテストパターンを決める方法で、実際のフィー

ルドでの稼動状況に近い形でのテストとなり、設計者の思い違いを見つけたり、全体を通しての不整合を発見したりすることができる。》

》通常はホワイトボックステスト・ブラックボックステストとも更に細分化されるが、これは設計工程での工程分割に対応した確認工程として細分化されるのが一般である。》

》同じテスト工数をかけても同じパターンばかりを実行していてはバグ摘出の効率は悪くなってしまう。そこでできるだけ異なったパターンを実行し網羅率を高めることが必要になってくる。この網羅率には全命令実行網羅率とか全分岐網羅率といったものがあり、よく使われている。ただしこの網羅率以外にもバグ摘出率を決める要素は有るため、これだけに頼ること無く、バグ成長曲線などの他の指標も使って総合的にバグの摘出状況を判断すべきである。いずれにしろ良いテストケースを設計することは生産性・品質ともに大きく貢献することなので各ドメイン毎の知識の蓄積が大切である。》

》最終のテスト時にはフィードとできるだけ同じ条件で実行させることがフィールド品質を高めることになるので、そのための環境というのは重要である。また、ブラックボックステストは誰がテストをしても同じであるというのは間違いで、優秀なテスト者はわずかな異常も見逃すことはなく品質を高める。さらに、実際のテスト時にはテスト仕様書に記載されたテスト項目以外にもその周辺のテストを思い付きで実行しバグを摘出することも多いので、テスト者の能力は重要な要素である。特に制御系のソフトウェアではタイミングが絡むこともあり、バグ摘出がテスト者の能力に関係することが多い。これらのことはドキュメント化して組織として蓄積すべきだという意見はもっともであるが、検査のプロと言われる人も居ることを認めて、そのような人を育成していくということも組織としては必要なことである。》

》（３）》バグ摘出工程とフィールド品質》

》設計で作り込まれたバグが、摘出工程でどのように摘出されていくかを図２．１０に示す。》

》これらの工数とバグ数の各々の因果関係を以下に説明する。》

（・）》本来必要な設計工数以下の場合は作り込みバグ数②は設計工数①と負の相関がある。つまり十分考えられていない設計では作り込みバグ数は多くなる。しかし必要以上に設計に時間をかけてもあるレベル以上の品質向上とはならず、むしろ余裕のありすぎる設》

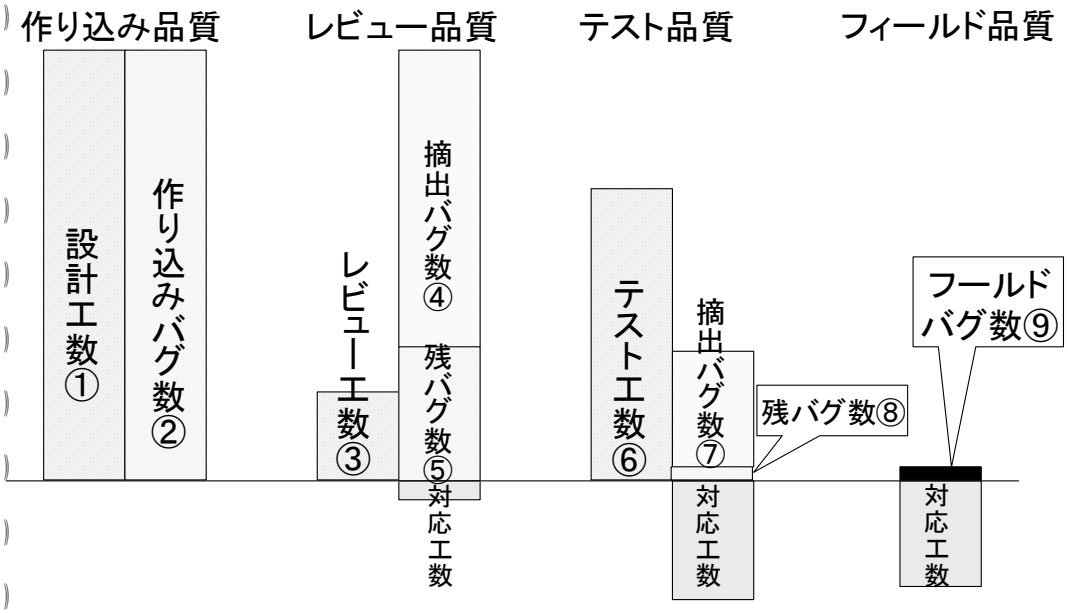

図2．10　バグ摘出工程とフィールド品質

計工数は設計者の緊張感を欠くことになり、品質を悪くする傾向がある。》

（・）》 作り込みバグ数②は以降の工程のすべての摘出バグ数④⑦・残バグ数⑤⑧⑨と正の相関がある。つまりすべてのバグ摘出は確率的にしかできないということで１００％摘出とか、ある絶対数値以下の残バグ数といったものは保証できないということである。》

（・）》 レビューによるバグ摘出数④はある一定レベルまでレビュー工数③と正の相関がある。逆にレビューで見逃された残バグ数⑤はレビュー工数③と負の相関を持つ。》

（・）》 テストによる摘出バグ数⑦はある程度テスト工数⑥と正の相関がある。ある程度という理由は、テストでのバグ摘出率はテスト工数以外にテスト設計や検査者の能力といった多くの要素があり工数との相関が低くなるためである。よってテストでの残バグ数⑧はテスト工数⑥とある程度の負の相関があることになる。》

（・）》 フィールドバグはテストでの残バグに対して確率的に発生するものであるがフィ

ールドでも発生しないバグは「神のみぞ知る」ものであるのでフィールドバグ数⑨＝テストでの残バグ数⑧と見なすべきである。》

》フィールドバグ数⑨と正の相関のあるものをまとめて見ると、作り込みバグ数②・レビューでの残バグ数⑤・テストでの残バグ数⑧となる。また負の相関のあるものは、設計工数①・レビュー工数③・レビューでの摘出バグ数④・テスト工数⑥・テストでの摘出バグ数⑦となる。つまりフィールド品質を高めるための注力点は設計・レビュー・テストに時間をかけるということになる。この事から「品質と生産性はトレードオフの関係にある」という単純な論が出てくるが、これは図２．８に示す予防コストのみを指していっているのであって、実際にはバグ対応工数による失敗コストを含めると単純なトレードオフの関係にはならない。当然、各工程でのバグ対応工数は摘出バグ数に比例するが一件当りのバグ対応工数は一つ工程が後ろにいくと一桁増えるといわれている。あるレベルのバグ摘出率まではレビューのバグ摘出効率は非常に良いため、その範囲における予防コストはあまり大きくならない。レビューにある程度の時間をかけた方が全体の生産性は向上するという理由はここにある。》

３．デザインレビューによる品質向上

本章では２．５節（３）項で述べたバグ摘出工程とフィールド品質のうち、デザインレビュー工程とフィールド品質の関係について述べる。

３．１ デザインレビューの目的

他の品質向上策に比べて、その気になればどんな組織ででも実行ができ、即効性があり効率も良いのがデザインレビューである。以下にデザインレビューの目的・利点について整理をしてみる。

（１）品質の向上

設計者以外のいろいろ観点からのチェックによって、設計者自身では見つけにくい設計者の思い違いやミスを発見することができ、組織としての英知を結集することができる。デザインレビューのために設計者が設計内容を他の人に説明する中で設計者自身がバグに気がつくことも多々ある。また、テストでは実行しにくいパターンもチェックが可能で品質向上に寄与する。テストはコンピュータでプログラムを実行しての品質確認であるが、デザインレビューはレビュアの頭の中でプログラム実行をシミュレートしての品質確認といえる。このため一度シミュレートして確認したはずでもバグを見落とすことがあるが、反面、実際のコンピュータではなかなか実行できないプログラムも頭の中では容易にシミュレートして確認することができる。この事は特に制御系のプログラムでタイミングが絡むような場合に威力を発揮する。

（２）生産性の向上

前節で述べたようにデザインレビューはバグの摘出効率が良い上に、摘出されたバグの修正工数はテスト工程やフィールドで発見されたバグの修正工数に比べて格段に少なくてすむために開発全体としての生産性は向上する。

（３）プロジェクトの安定

ソフトウェア開発のプロジェクトにとって一番恐ろしいことは、潜在化していた品質の問題がある日突然に顕在化しプロジェクトが混乱をすることである。２．５節で述べたがソフトウェア品質の状況を正確に把握することは特に難しいものである。そのため、うわべだけの進捗確認であると品質問題は潜在化し一見工程は順調に進んでいるように見えることが多い。これがテスト工程のようにはっきりと品質が確認できる工程になって始めて

品質問題が顕在化し、手戻り工数のためにプロジェクトが混乱することになる。進捗管理はプロダクトの品質を確認することとの組合せでないと何の意味も成さないものであるが、設計工程におけるプロダクトの品質確認とはデザインレビューそのもののことである。またプロジェクトを混乱させるような何らかの問題が発生した時も、レビューをしっかりとしておけば設計者だけが状況を把握しているということではなく、多くの人が状況認識を共有化できるため客観的な対策を考えることができる。このような事からデザインレビューはプロジェクトの安定に寄与する。

３．２ デザインレビュー実行のための前提条件

デザインレビューがいくら有効な品質向上策だといっても、人を集めて時間をかけさえすれば効果が上がるというわけにはいかず、当然効果的・効率的にデザインレビューをするための以下のような前提条件がある。これらの前提条件は、デザインレビューが組織としての英知の結集であることからも当然の帰結として出てくるもので、組織としては最低限整備しておくべきものばかりである。

（１）標準工程定義

標準工程定義はプロジェクト管理の一番のベースであり、関係者全員が開発の基本型について共通認識を持つためのツールである。ここで定義された一つ一つの工程はプロジェクトの進捗を管理する最低限の区切りを表しているはずなので、工程の切れ目がレビューのポイントとなる。それぞれの工程では標準的にはどこまで作業が進んでいるべきであるか、またどのような観点でレビューをするべきかの共通認識を得る事ができる。

（２）ドキュメント規約の制定

工程の単位で管理をしようと思えばその工程としての文書成果物があって初めてできることで、ドキュメント規約は標準工程定義との組合せで作られるべきものである。レビューはこのドキュメントに対して行われるので、規約の制定によりドキュメントの均質化がはかれ、多くの人によるレビューがやりやすくなる。

（３）個別プロジェクトの工程計画作成

個々のプロジェクトは全て事情が異なるため、標準工程とそれに伴うドキュメントがいくら定義されてもそれがそっくりそのまま実際のプロジェクトで使えるわけではない。こ

れらの標準形を参考にしながら各々のプロジェクトの事情に合った工程計画を立て、それぞれの工程での作成ドキュメントの定義を行う必要がある。ここで定義されたドキュメントが実際のレビュー対象となる。

３．３　データ分析結果によるデザインレビューの効果

本節ではデザインレビューの工数が品質とどのような関係に有り、デザインレビューがいかに品質向上に効果が有るのかを、オムロンにおける次のような特徴を持つソフトウェア開発から得られたデータの分析結果を基に示す。

(1)開発工数：プロジェクト当たり２～３００人月
(2)開発期間：プロジェクト当たり２～１８ヶ月
(3)対象ソフトウェア：現金自動入出金機、自動券売機、自動改札装置、キャッシュレジスター・ＰＯＳ等の制御ソフトウェア
(4)開発プロセス：ウオーターフォールモデル
(5)開発環境：UNIX ワークステーション
(6)プログラミング言語：Ｃ言語
(7)動作環境：専用ハードウェアとリアルタイムモニタ

（１）レビュー工数比率とフィールドバグ件数

ここでいう開発対象ステップ数、レビュー工数比率、フィールドバグ件数、レビュー指摘率とはそれぞれ次に示すものである。

開発対象ステップ数＝新規設計ステップ数
＋（一部修正して再利用したモジュールのステップ数）×α
＋（修正なしで再利用したモジュールのステップ数）×β
α、β：再利用するに当たって負担となる工数を勘案して決める係数。
通常はα＝０．２、β＝０．０１を使用。

レビューにかけた全工数
レビュー工数比率＝――――――――――――――
設計にかけた全工数

フィールドリリース後６ヶ月間に見つかった全バグ数
フィールドバグ件数＝――――――――――――――――――――――――――――
開発対象ステップ数

レビューで指摘された全バグ数
レビュー指摘率＝――――――――――――――――――
全開発工程の中で発見された全バグ数

図３．１は全工数が２人月以上の開発（保守のための一部修正を含む）について、レビュー工数比率とレビュー指摘率の関係を表したものである。かなりばらつきが大きくて二つの相関はなかなかいえないが、特徴的な次の二点のことが言える。

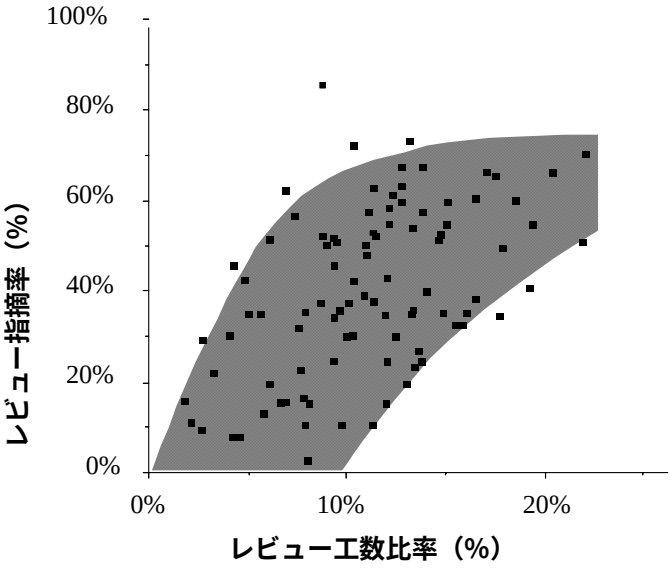

図3．１ レビュー工数比率とレビュー指摘率

一つはレビュー工数比率が１０％程度の時は指摘率のばらつきは大変大きいが、２０％位まで大きくなるとばらつきは少なくなり、指摘率５０％程度は確保されるようである。つまりグラフの右下の領域には分布が無いということである。これはレビュー方法の上手・下手に関係なく、２０％程度のレビュー工数をかければ一定の指摘率を確保できることを示している。

もう一つはレビュー工数比率が１０％を超えても指摘率はなかなか７０％を超えられないということである。これはレビューだけですべてのバグを摘出してしまうのは無理があることと、レビューの上手・下手で同じ指摘率を確保するにもかけるべき工数は違ってくるということを示している。

図３．２は前述の特徴を持つ対象プロジェクトで、１９９２年から１９９４年の間に開発着手、完了し、フィールド稼動後半年以上経過した開発プロジェクトの内、開発工数が２０人月以上で全ての収集実績データについて精度の妥当性が検証された３６プロジェクトを対象に、レビュー工数比率とフィールドバグ件数の関係を表したものである。

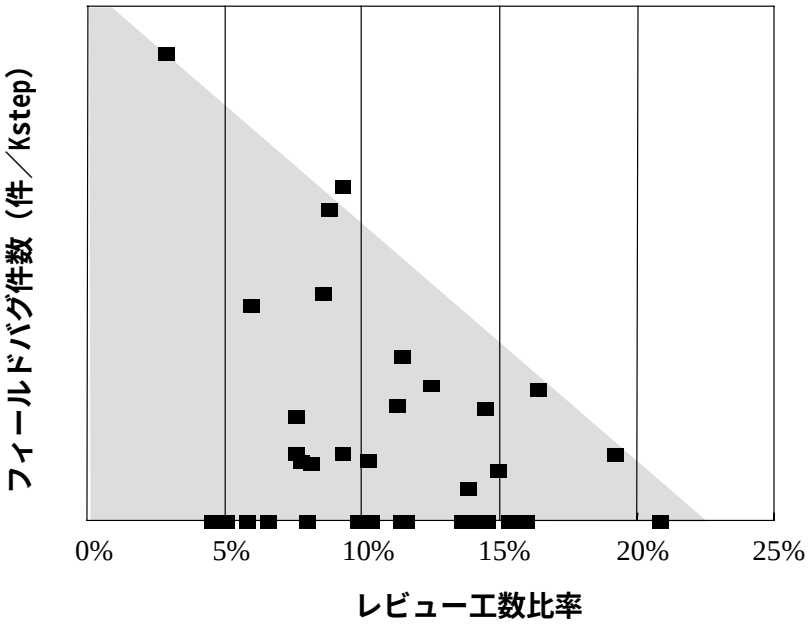

図3．２ レビュー工数比率とフィールドバグ件数

これも同じくかなりのばらつきがあるが、レビュー工数比率１０％を境にして１０％以下と１０％以上のプロジェクトでフィールドバグ件数に統計的優位差が有るか否かをマン・ホイットニーの検定法で検定したところ、レビュー工数比率１０％以上のプロジェクトはフィールドバグ件数が少ないという結果が出ている。このような厳密な統計的証明をしなくても、図３．２を見れば次の事がいえる。グラフの右上にプロットされるプロジェクトはなく、レビュー工数比率を２０％程度にすればほとんどフィールドバクをなくすることができると言える。

図３．１、図３．２の両方からいえることはとりあえずレビュー工数比率は２０％を目標にし、その後、品質を見ながらレビューの効率よい方法を探り、レビュー工数比率の最適値をきめていくべきである。

（２）コードレビュー工数比率とフィールドバグ件数

レビュー工数比率とフィールドバグ件数との相関はばらつきが大きいが、この理由の一つは各工程毎のレビューの重みとか有効性の違いを無視して、全行程工数の合計値で分析しているところにある。ここでは、コーディングを完了しテストに入る前のソースコードをチェックするコードレビューが品質向上に特に有効性であることを示す。ここでコードレビュー工数比率、コードレビュー指摘率は次の通りである。

コードレビューの工数
コードレビュー工数比率＝―――――――――――――――
コーディング工程の工数

コードレビューで指摘された全バグ数
コードレビュー指摘率＝―――――――――――――――――――――
全開発工程の中で発見された全バグ数

図３．３、図３．４、図３．５、図３．６の対象プロジェクトは図３．２のプロジェクトと同じものである。

図３．３はレビュー指摘率とフィールドバグ件数の関係を表したものである。図３．２

の横軸にレビュー工数比率をとったグラフと類似の分布をしているが、図３．３のグラフには左下の三角形の範囲から大きく外れたプロジェクトが三つほどある。図３．４は横軸をコードレビュー指摘率にして表したものである。図３．３と図３．４の丸印のついたものは同じプロジェクトを表している。図３．３で異常値を示している三つのプロジェクトはいずれもコードレビュー指摘率が０％であったことが分かる。コードレビューは実行すればまず何らかのバグ指摘があるのが当たり前であるので、これらのプロジェクトはコードレビューを実行しなかったか、または実行しても形ばかりのものであったと思われる。図３．４の分布を見るとコードレビュー指摘率が０％のプロジェクトは特に分散が大きく、品質が悪い状態になっている。また、左下の三角形に分布が集中するという特徴を示しており、コードレビュー指摘率はフィールドバグ件数に大きく影響していることが分かる。》

》 これらのことより、設計全体でのレビュー指摘率が高いだけではフィールド品質の安定は望めず、コードレビュー指摘率も同時に高くなければならないことを示している。つまりコードレビューは他の工程のレビューで補うことが出来るものではなく必須のレビューであると言える。》

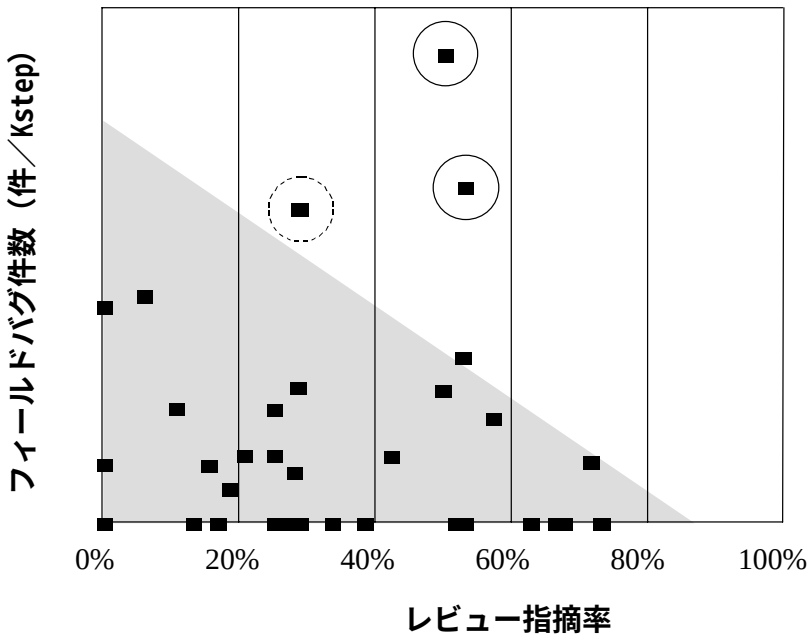

図3．３ レビュー指摘率とフィールドバグ件数

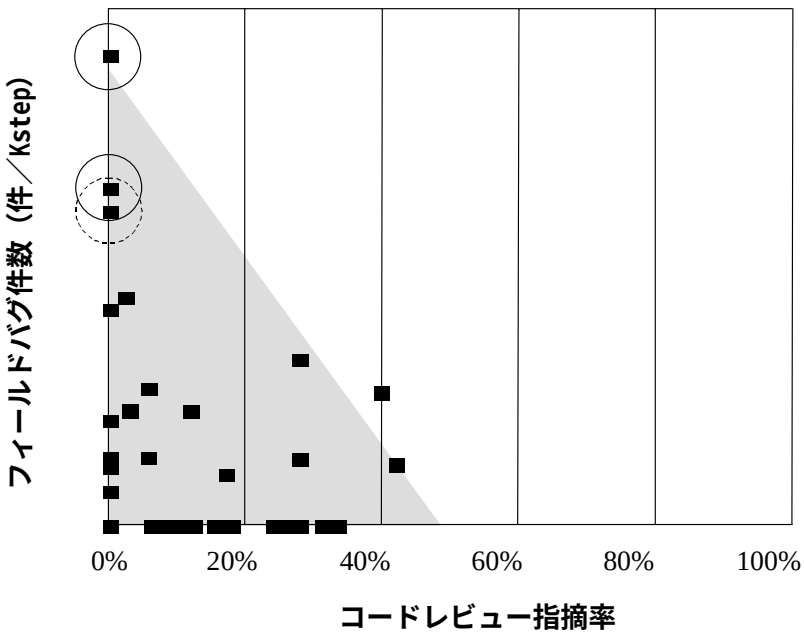

図3．4　コードレビュー指摘率とフィールドバグ件数

図３．５にコードレビュー工数比率とフィールドバグ件数の関係を、また図３．６にコードレビュー工数比率とレビュー指摘率の関係を示す。これらのグラフはコードレビューにおいても２０%程度のレビュー比率を確保すべきであることを示している。

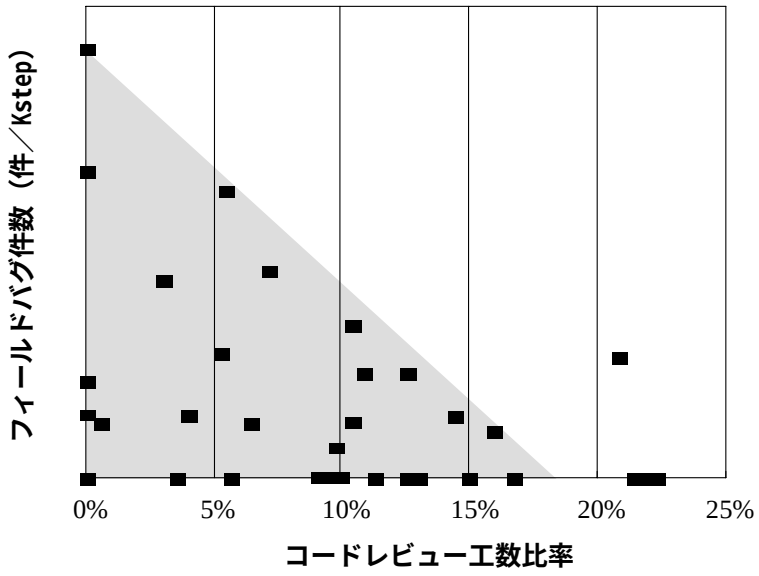

図3．5　コードレビュー工数比率とフィールドバグ件数

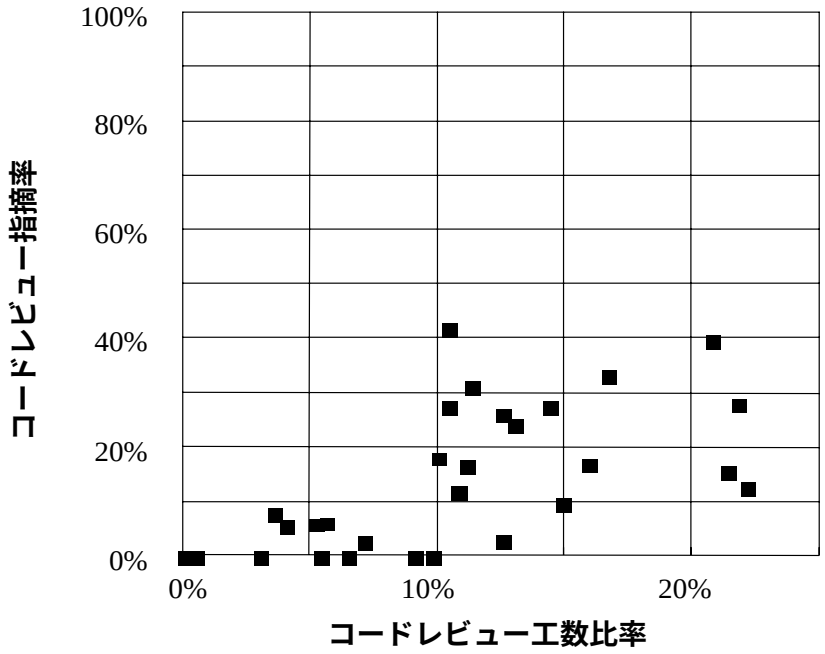

図3．6　コードレビュー工数比率とコードレビュー指摘率

４．プロジェクトの安定とソフトウェア品質・生産性

３章ではデザインレビューがフィールド品質の向上に大きく貢献することを述べたが、実際のプロジェクトの進行中はその優先順位を下げられ十分なデザインレビューが実行されないことが多い。これは２．５節で述べたように品質の問題は設計時点では見えにくいのに対し、開発コスト・納期の問題はそれぞれの時点で明確に認識され、せっぱ詰まった状態では品質に対する根拠の無い期待の心理が働いてしまうためである。そこで、プロジェクトが混乱して必要なデザインレビューが削減されてしまうことを防ぐためにプロジェクトを常に制御可能な状況にし安定させなければならない。以下にプロジェクトの安定と品質・生産性の関係の分析結果を述べる。

図４．１、図４．２、図４．４の分析対象のプロジェクトは、オムロンにおける次のような特徴を持つ１９９２年から１９９５年の間に開発完了したソフトウェア開発のうち、全ての収集実績データについて精度の妥当性が検証され、かつ開発計画と実績の比較対照が可能であった３１プロジェクトである。

(1)開発工数：プロジェクト当たり２０～数１００人月
(2)開発期間：プロジェクト当たり６～１８ヶ月
(3)開発対象容量：１０Ｋステップ～数１００Ｋステップ
(4)対象プロジェクト数：３１プロジェクト
(5)対象ソフトウェア：現金自動入出金機、自動券売機、自動改札装置、キャッシュレジスター・ＰＯＳ等の制御ソフトウェア
(6)開発プロセス：ウオーターフォールモデル
(7)開発環境：UNIX ワークステーション
(8)プログラミング言語：Ｃ言語
(9)動作環境：専用ハードウェアとリアルタイムモニタ

⟩ ４．１⟩ 開発コスト計画精度と品質⟩

⟩ 図４．１に開発コスト計画精度と品質の関係を示す。開発コスト計画精度ＲＥは⟩
⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩ 開発コストの実績値－計画値⟩
開発コスト計画精度ＲＥ＝――――――――――――――――⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩ 開発コストの計画値⟩

で表され、実績が計画からどの程度ずれたかを示す指標である。ＲＥが１０％とは実績が計画の１１０％であったことを示し、ＲＥが－１０％とは同じく９０％であったことを示す。プロジェクトが混乱してくるとほとんどの場合開発コストに影響を与えるため、ここではＲＥをプロジェクトの安定度合いを示す指標とする。縦軸のフィールドバグ件数は３章で述べた定義と同じく次の通りである。⟩
⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩ フィールドリリース後６ヶ月間に見つかった全バグ数⟩
フィールドバグ件数＝――――――――――――――――――――――――――――⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩ 開発対象ステップ数⟩
⟩
開発対象ステップ数＝新規設計ステップ数⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩ ＋（一部修正して再利用したモジュールのステップ数）×α⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩ ＋（修正なしで再利用したモジュールのステップ数）×β⟩
⟩⟩⟩⟩⟩⟩⟩ α、β：再利用するに当たって負担となる工数を勘案して決める係数。⟩
⟩⟩⟩⟩⟩⟩⟩⟩⟩⟩ 通常はα＝０．２、β＝０．０１を使用。⟩

⟩
⟩ ここでの分析では対象プロジェクトを、ＲＥ≦－１０％、－１０％＜ＲＥ＜＋１０％、ＲＥ≧＋１０％の三つのグループに分けて、それぞれのグループ間でフィールドバグ件数に統計的優位差が有るかを調べている。－１０％＜ＲＥ＜＋１０％のグループはプロジェクトを良くコントロールでき安定していたグループで、ＲＥ≧＋１０％のグループは予算をオーバーし厳しい状況の中でのプロジェクト運営となり安定性にもかけていたと思われる。ＲＥ≦－１０％のグループは予算以下で開発ができているので一見プロジェクトは安定しているように見えるが、このグループは明らかに見積りミスをしており、その意味で⟩

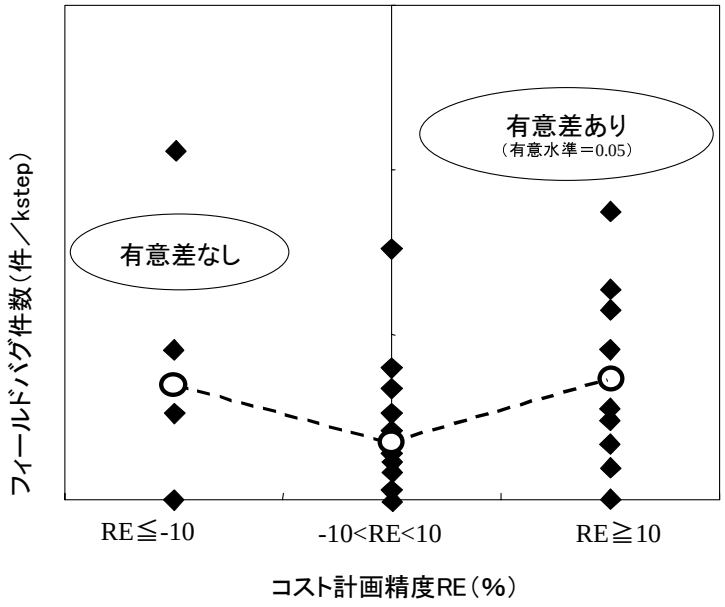

図4．1　開発コスト計画精度とフィールドバグ件数

不安定要素を持っているといえる。

図４．１において、一つ一つのドットは個々のプロジェクトを表しており、点線で結ばれた白抜きの丸印はそれぞれのグループの平均値を表している。

まず、平均値で見ると－１０%＜ＲＥ＜＋１０%のグループが一番品質が良く、計画精度がプラスマイナスどちらに振れても品質は悪くなっている。さらにそれぞれのグループ間で品質に有意差が有るか否かを有意水準０．０５で検定したところ、ＲＥ≧＋１０%のグループは－１０%＜ＲＥ＜＋１０%のグループに対して品質が悪いということについて有意差ありという結果が出た。ＲＥ≦－１０%のグループと－１０%＜ＲＥ＜＋１０%のグループの間には有意差が認められなかったが、これはＲＥ≦－１０%のグループのサンプル数が少なかったため統計的な検定ができなかったためで、ＲＥ≦－１０%のグループの方が品質は悪くなる傾向に有ることは認められる。

ＲＥ≧＋１０%のグループの品質が悪くなっている理由は次のことが考えられる。一つは、デザインレビューの重要性が十分認識できていないか、認識していても何らかの理由で進捗遅れ、開発コストオーバーが発生しプロジェクトを十分制御できなくなったかで、いずれにしろ結果としてデザインレビュー不足のために品質が悪くなったと考えられる。

もう一つは、元々の設計品質が悪く、デザインレビューに十分な時間をかけたにもかかわらずデバック・テスト工程での品質も悪く手戻り工数が増加し、結果的に開発コストオーバーとフィールド品質悪化をもたらしたと考えられる。いずれにしろ開発コストオーバーと品質悪化とは同時に発生するといえる。》

》 一方、ＲＥ≦－１０%のグループが－１０%＜ＲＥ＜＋１０%のグループに対して品質が悪いということについては直感的には違和感の有るものである。一般的には開発コストに余裕が有るので品質作り込みに十分な工数をかけるものと思われがちであるが、実際にはこのような見積りミスをおかすプロジェクトは管理能力も低いため品質についての取組みも弱くなりフィールド品質が悪くなったと思われる。》

》 プロジェクトをきちんとコントロールし必要な工程に必要な工数をかけることがフィールド品質向上に不可欠であるといえる。》

》

》

》

》

》

》

》

》

》

》

》

》

》

》

》

》

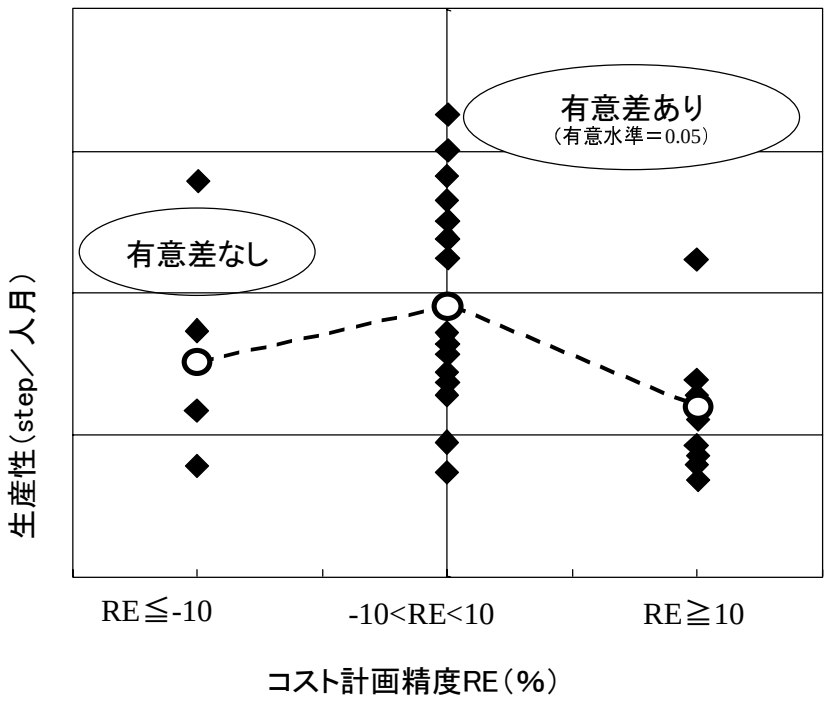

図４．２ 開発コスト計画精度と生産性

⟩ ４．２⟩ 開発コスト計画精度と生産性⟩

⟩ 図４．２に開発コスト計画精度と生産性の関係を示す。対象プロジェクトは図４．１と同じ３１プロジェクトで、横軸の開発コスト計画精度ＲＥは図４．１での定義と同じである。縦軸の生産性は⟩

⟩
⟩ ⟩ ⟩ ⟩ ⟩ ⟩ 開発対象ステップ数⟩
生産性＝――――――――――――――⟩
⟩ ⟩ ⟩ ⟩ ⟩ 開発全体にかけた工数⟩
⟩
開発対象ステップ数＝新規設計ステップ数⟩
⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ＋（一部修正して再利用したモジュールのステップ数）×α⟩
⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ＋（修正なしで再利用したモジュールのステップ数）×β⟩
⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ α、β：再利用するに当たって負担となる工数を勘案して決める係数。⟩
⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ ⟩ 通常はα＝０．２、β＝０．０１を使用。⟩
⟩

で定義されるものである。図４．１と同様に、一つ一つのドットは個々のプロジェクトを表しており、点線で結ばれた白抜きの丸印はそれぞれのグループの平均値を表している。⟩
⟩ 図４．１で計画精度の高いプロジェクトは品質が良かったのと同じように、－１０％＜ＲＥ＜＋１０％のグループが一番生産性が高く、計画精度がどちらに振れても生産性は低くなっている。また、それぞれのグループ間で生産性に有意差が有るか否かを有意水準０．０５で検定したところ、品質の場合と同じようにＲＥ≧＋１０％のグループは有意差あり、ＲＥ≦－１０％のグループはサンプル数が少ないために有意差が認められず、という結果が出た。⟩
⟩ ＲＥ≧＋１０％のグループの生産性が低いのは、標準より生産性の低いチームによる開発であったから結果的に開発コストがオーバーしたというよりも、見積りミスのためにプロジェクトが不安定になり手戻りや無駄作業が増えて生産性を落としたと考えられる。⟩
ＲＥ≦－１０％のグループの生産性が低い傾向にあるのは、品質と同様に直感的には違和感の有るものである。少なくとも見積り以上の生産性を上げたから開発コストが少なくてすんだのではないことだけは確かである。このグループは見積りが不正確であり、プロジ

ェクト管理力も低いと思われ、このためあまり効率の良い開発ができなかったといえる。また、設計者の心理として余裕のある計画のプロジェクトは緊張感に欠け、そのために作業効率を落とし生産性が低下したとも考えられる。

－１０%＜ＲＥ＜＋１０%のグループが一番生産性が高いのは、プロジェクトが安定しデザインレビューも十分実行され、手戻り工数が少なかったためと考えられる。

生産性を決定する要素は多くのものがあり、レビュー工数比率と生産性との相関はなかなか出難くいものであるが、レビュー工数比率以外の要素がなるべく似通ったプロジェクトで比較すると相関が認められる。図４．３は図４．２とは別のプロジェクトであるが、オムロンにおけるある商品のアプリケーションソフトウェアを客先ごとにカスタマイズするという類似のプロジェクトを抽出し、レビュー工数比率と生産性の関係を示している。これを見るとレビュー工数比率と生産性との間には弱い正の相関があることが分かる。レビューを十分に実行するとテストに入るまでの工数は増大するにもかかわらず全体の生産性が向上する理由は、レビューによって品質が向上しテストでのバグの数が減り、バグ対応工数の減少がレビューにかけた工数を上回るためである。

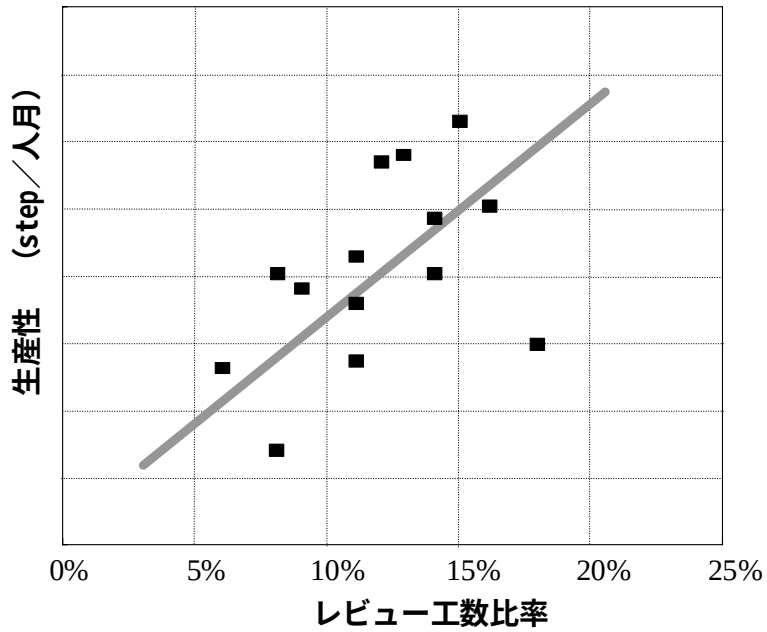

図4．３ レビュー工数比率と生産性

４．３　計画充実度と開発コスト計画精度

開発コスト計画精度の良いプロジェクトは品質・生産性とも良いことを示したが、計画を充実させることで開発コスト計画精度が良くなることを示したのが図４．４である。対象プロジェクトは図４．１と同じ３１プロジェクトで、横軸の開発コスト計画精度ＲＥは図４．１での定義と同じである。

計画充実度ＰＰは工程計画書の内容について以下の観点で５０点満点の点数をつけたものである。

（・）　ＷＢＳは作業の責任分担と対応して構造化されていること。
（・）　各作業の責任者、担当者が明確で、ただ一人の人が割当てられていること。
（・）　ＷＢＳ最下層の各作業（ワークパッケージ）の詳細化が十分で作業量の見積りが可能あること。また、極力２週間以下の作業に詳細化されていること。
（・）　作業手順の関連付けを示すネットワーク図や、これをスケジュール表として表現したガントチャート等との整合がとれていること。
（・）　各作業の文書成果物が定義されていること。

ＰＰの点数が高いほど計画書の内容は充実をしていたということを示している。図４．４はこのＰＰを四つのグループに分け、ＲＥのそれぞれのグループの中でどのように分布をしているかを示したものである。結果はＰＰの点数の高いプロジェクトは開発コスト計画精度が良いことを示している。－１０％＜ＲＥ＜＋１０％のグループでは大半のプロジェクトがＰＰ≧２５であるのに対し、ＲＥ≧＋１０％、ＲＥ≦－１０％のグループとも逆に大半がＰＰ＜２５となっている。ＰＰの評価項目となっている点を充実させることで開発コスト計画精度を上げることができ、ひいては品質・生産性ともを向上させることができる事を示している。

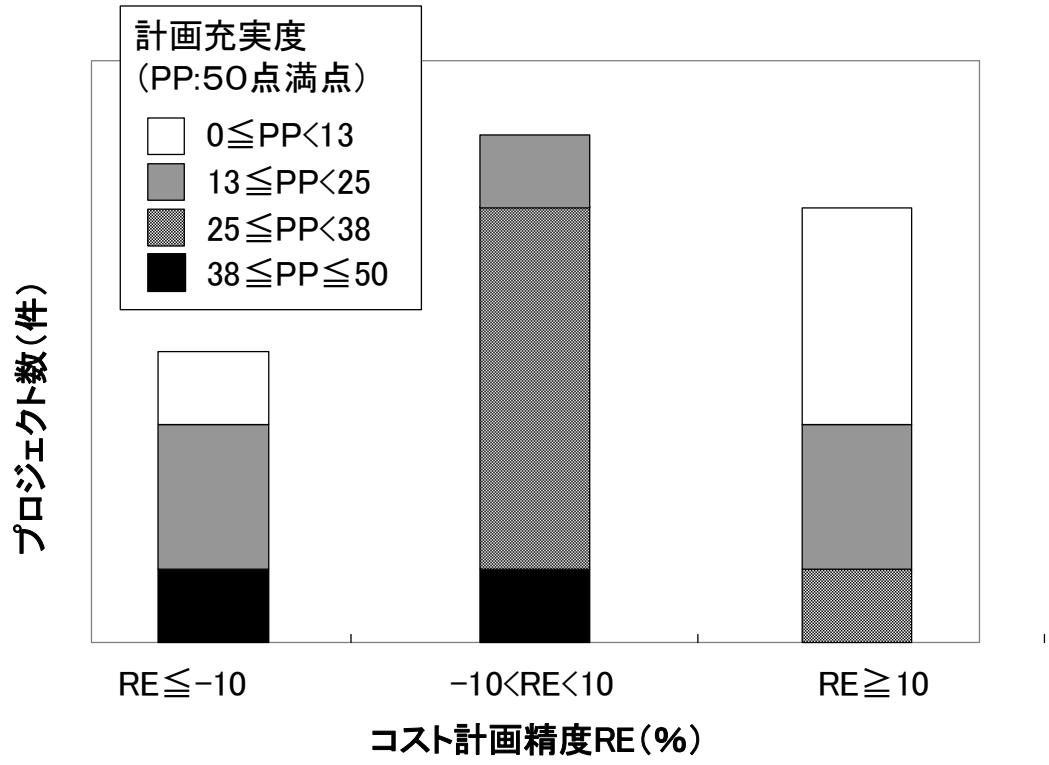

図4．4　計画充実度と開発コスト計画精度

５．利益予測に基づくソフトウェアプロセス改善の試み

ソフトウェア開発プロセスの改善は、通常、開発プロセスの現状把握と分析、分析結果に基づく改善策の作成と実行、に分けて実施される。これまでに多くの開発プロセス改善の提案があるが、それらはプロセスが具備すべき条件の評価に重点がおかれており、たとえ高い評価をうけても開発されるソフトウェアの品質がよくなるという保証はない。また、改善策の作成と実行においては、改善目標の抽象度が高く、開発現場に特化した改善計画をたてることは難しい。更に、トップダウン的に改善計画が与えられるため、現場の開発者に十分な動機付けを与えるのが難しく、結果として現場への導入を困難なものとしてしまう。

本章では、これらの問題点を解決する一つのアプローチとして開発者指向のプロセス改善の枠組みを提案する。提案する枠組みでは、現状の開発プロセスを形式的に記述し、その定量的分析結果に基づいて実行可能な改善計画を開発者に提示する。更に、改善計画に基づいてプロセス改善を行うことで達成される工数の削減量を定量的に予測し、それらを具体的な利益の形で提示することで、開発者にプロセス改善の強い動機付けを与えるところに特徴がある。提案した枠組みをある組み込みソフトウェアの開発を行う実際のプロジェクトに適用して評価実験を行った。具体的には、ある組み込みソフトウェア開発プロジェクトに対して、現状のプロセスの記述、問題点の把握、改善計画とその改善による利益予測の開発者への提示、改善計画の実行、実行結果の評価、という一連の手順を適用した。その結果、そのプロジェクトにおいてほぼ予測に近い工数削減が達成されていることを確認できた。

以降、５．１では、改善対象プロジェクトの概要と改善目標について述べる。５．２では、提案するプロセス改善の枠組みについて紹介する。５．３では、提案したプロセス改善の枠組みの適用について述べる。５．４では、適用結果に対する考察と枠組みに対する開発者からの意見をまとめる。最後に、５．５でまとめと今後の課題について述べる。

５．１　準備

５．１．１　対象プロジェクト

プロセス改善の対象となるプロジェクトは、ある一連の組み込みソフトウェア開発プロ

ジェクトの一つである。各プロジェクトでは、既存ソフトウェアに対してユーザの要求に応じて一部の機能だけを変更した類似ソフトウェアの開発がなされている。通常、これらのプロジェクトは少なくとも３年間続けられる。プロセス改善を始めた時点で、２つのプロジェクト(ＰＲ１とＰＲ２)が既に終了しており、次のプロジェクトＰＲ３が４ヶ月後に始まるという状況にあった。これら３つのプロジェクトの特徴を次の(1)～(8)にまとめる。

(1)開発工数：プロジェクト当たり２０～５０人月(平均３０人月)
(2)開発期間：プロジェクト当たり３～７ヶ月(平均５ヶ月)
(3)プロジェクトの数：年間５～１０プロジェクト
(4)プロジェクトメンバー：ほぼ同じメンバーが全てのプロジェクトを行う。
(5)開発環境：UNIX ワークステーション
(6)プログラミング言語：Ｃ言語
(7)動作環境：専用ハードウェアとリアルタイムモニタ
(8)開発プロセス：ウオーターフォールモデル

開発プロセスは、構想設計(ＣＤ)、機能設計(ＦＤ)、構造設計(ＳＤ)、モジュール設計(ＭＤ)、プログラミング(ＰＧ)、単体テスト(ＵＴ)、結合テスト(ＩＴ)、機能テスト(ＦＴ)、受入テスト(ＶＴ)の９工程から構成される。なお、ＵＴとＩＴでは開発者によるホワイトボックステストが、ＦＴでは開発者によるブラックボックステストが、ＶＴではテスト組織によるブラックボックステストが、それぞれ実施される。

プロジェクトで開発されるシステムの構造を簡単に説明する。システムはアプリケーション部と基本部の２つの部分から構成される。アプリケーション部は７つのモジュール(Ａ、Ｂ、Ｃ、Ｄ、Ｅ、Ｆ、Ｇ)で、基本部は２つのモジュール(リアルタイムモニタとＨ)で、それぞれ構成される。これらのモジュールの中で、リアルタイムモニタは全てのプロジェクトで共通に用いられ、変更はほとんど行われない。モジュールＢ、Ｃ、Ｄ、Ｅ、Ｆ、Ｇは一部が変更され、ＡとＨは大部分が変更される。従って、モジュールＡとＨの他のモジュールとの結合テストが各プロジェクトにおける重要な工程の一つとなっている。

５．１．２　改善目標

事業としての視点からは、品質、納期、開発コスト全てが重要であるが、現状では品質、納期で顧客の満足を得られており、今回の改善における目標としては開発コストを最小に

おさえるということが各プロジェクトに求められた。つまり、開発コストに重大な影響を与える問題を識別し、それを解決することが望まれた。この数年間コスト削減を目標として、管理を行ってきたが、その効果は現れていなかった。従って、５．３で述べるケーススタディでは、コストの削減を第一の目標と設定した。

技術的な視点からは、開発工程の遅れによるプロセスの混乱に着目した。既に述べたように、開発プロセスはウオーターフォールモデルに従っており、理想的には各工程は連続して実行されるはずである。しかし、これまでに継続して収集してきた開発データを分析すると、他の工程と並列して行われている工程が数多く存在することが確認された。この原因をつきとめ、対策をたてることがもう一つの目標である。

５．２ プロセス改善の枠組み

提案する枠組みでは、ソフトウェア開発プロセスの改善のためのガイドラインを提示する。類似の手法として Process Characterization がある[15]。

Process Characterization では、先ず、現状のプロセスを "As Is" map として記述し、対策をたてるべき問題点を明らかにする。次に、問題点に対する改善項目を明かにして、優先度をつける。更に、プロセスの改善度合を計測するためのメトリクスを定義する。また、改善されたプロセスを "Should Be" map として記述する。しかし、プロセス改善を行う上で最も重要な開発者への動機付けということは考慮されていない。

そこで、我々は改善計画の導入によって生じる利益とその計画の妥当性を開発者に定量的なデータを用いて示し、更に、(単に計画を提示するだけでなく)改善作業そのものを開発者自身と協力して実施することができれば、開発者のプロセス改善に対する動機付けを効果的にできると判断した。

提案する枠組みは次の６ステップから構成される。

(1)現状プロセスの記述
(2)現状プロセスの分析
(3)改善計画の作成
(4)利益予測
(5)改善計画の実行
(6)改善計画の評価

⟫ 本手法の特徴は次の(a)〜(c)にまとめられる。⟫

(a)現状のプロセスと改善されたプロセスを記述する。⟫
(b)改善計画を導入した場合の利益予測を行う。⟫
(c)改善活動を開発者と協調して行う。⟫

⟫ なお、実施においては、プロセス改善活動の中心的な役割を果たすＳＥＰＧ(Software⟫ Engineering⟫ Process⟫ Group)を設立した。⟫
⟫ 以降では各ステップでの作業内容について詳細に説明する。⟫
⟫

Step1(現状プロセスの記述):⟫⟫
⟫ ソフトウェア開発プロセスは、相互に関係し合った多くの作業から構成されている。従って、開発プロセスの現状を正確に把握するためには、プロセスを形式的に記述する必要がある。⟫
⟫ 提案する枠組みでは、現状のプロセスをペトリネットを用いて記述する[16]。⟫⟫開発作業をトランジションで、プロダクトをトークンで表す。プレースは次の作業実行の待ち状態を表す。⟫
⟫

Step2(現状プロセスの分析):⟫⟫
⟫ 現状のプロセスから収集したデータに基づいて、コスト、品質、納期に関する問題点を明らかにする。例えば、多くのコストを要した作業、品質に影響を与えた作業(フォールトを多く作りこんだ作業)を識別する。更に、開発者との議論を通じて、その原因を徹底的に究明する。⟫
⟫

Step3(改善計画の作成):⟫⟫
⟫ Step2 で発見した問題点に対する改善計画を作成する。改善計画は、改善されたプロセスの記述とプロセスを構成する各作業に対する詳細な対策から構成される。改善されたプロセスの記述にもペトリネットを用いる。詳細な対策は全て文書化される。⟫
⟫ Step3 では、ソフトウェア工学の技術(例えば、設計手法、レビュー技法、テスト手法)や過去のプロセス改善で得られた経験を改善計画に導入することを試みる。導入においては、対象となるプロセスへの導入可能性の検討を開発者と十分に行い、同意を得ることが

重要である。⟫

⟫

Step4(利益予測):⟫⟫

⟫ 利益予測では、改善計画の導入前と導入後における効果を定量的に評価する。複数の改善計画がある場合には、予測結果から最も適切なものを採用する。予測には様々な方法が考えられるが、過去の類似プロジェクトで収集したデータを用いて、改善プロセスのシミュレーションを行うことは、最も効果的な手法である。⟫

⟫

Step5(改善計画の実行):⟫⟫

⟫ Step4 で採用した改善計画を、改善対象プロジェクトへ適用し、実行する。SEPG は改善計画の実行状況を観察し、問題が発生すれば援助を行う。⟫

⟫

Step6(改善計画の評価):⟫⟫

⟫ 対象プロジェクトが終了した後で、改善計画の評価を行う。例えば、Step4 で予測した利益が実際に達成されているかどうかを検証する。⟫

⟫

⟫ ５．３⟫ ケーススタディ⟫

⟫⟫ ５．３．１⟫ プロセス改善の進捗⟫

⟫ ここでは提案した枠組みのケーススタディの進捗概要について述べる。このプロセス改善に参加したＳＥＰＧの作業は１０人月の工数を要した。⟫

⟫ １９９４年４月に最初の全体ミーティングを開催した。そこでは全てのプロジェクトメンバー(管理者、開発者(うち１名が開発リーダー))に対して、プロセス改善の必要性を説明した。プロジェクト管理者はプロセス改善の実施を承認した。⟫

⟫ Step1 は１９９４年４月から７月にかけて行われた。その結果、プログラミング工程とテスト工程に多くの工数が費やされていることを確認した。引き続き Step2〜Step4 を実施し、これらの工程における問題点を分析し、改善計画を作成した。更に、改善計画による利益予測を実施した。⟫

⟫ ８月に２回目の全体ミーティングを開催した。そこでは、現状のプロセスの問題点、改善計画、改善計画による利益予測をプロジェクトメンバーに説明した。管理者は改善計画

の実行を承認した。⟫

⟫ ８月から翌年の３月にかけて、改善計画に基づいて新しいプロジェクトＰＲ３が実施された。プロジェクト終了後、改善結果の評価を行った。以降では、Step1～Step5 についての詳細内容について述べる。⟫

⟫

⟫⟫ ５．３．２⟫ 現状プロセスの記述(Step1)⟫

⟫ ソフトウェア開発プロセスを理解するため、既に開発を終了している２つのプロジェクトＰＲ１とＰＲ２の開発者と議論を重ね、図５．１に示すプロセス記述を作成した。ＦＴ＆ＶＴのトランジションの詳細については、図５．２に示す。⟫

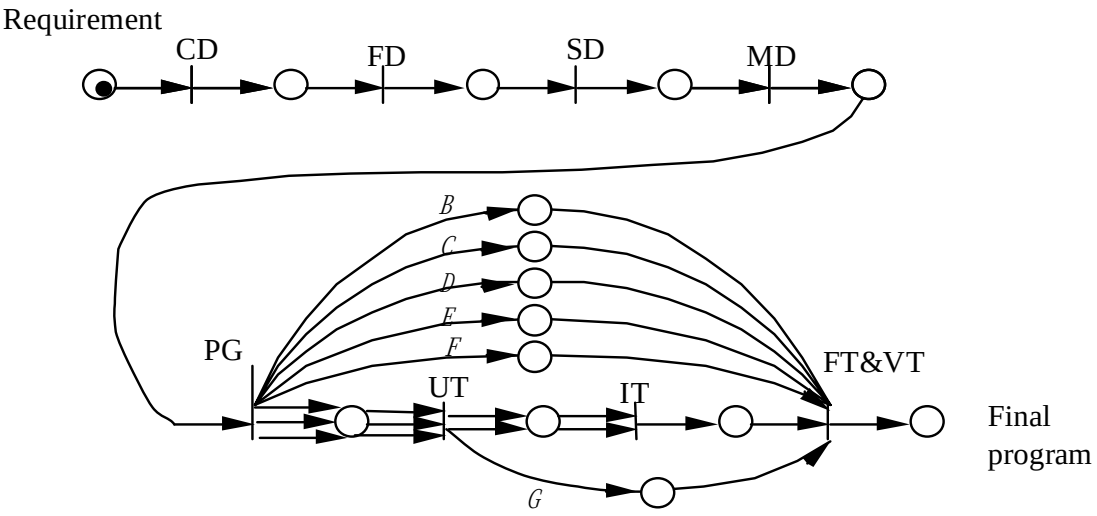

図５．１ 現状プロセスの記述

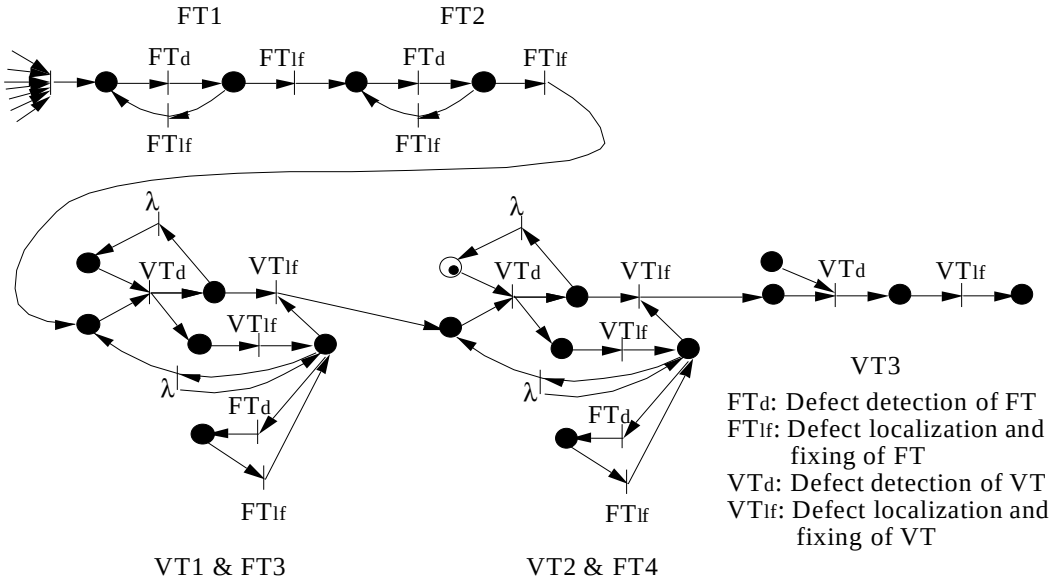

図5．2 FT&VTの詳細記述

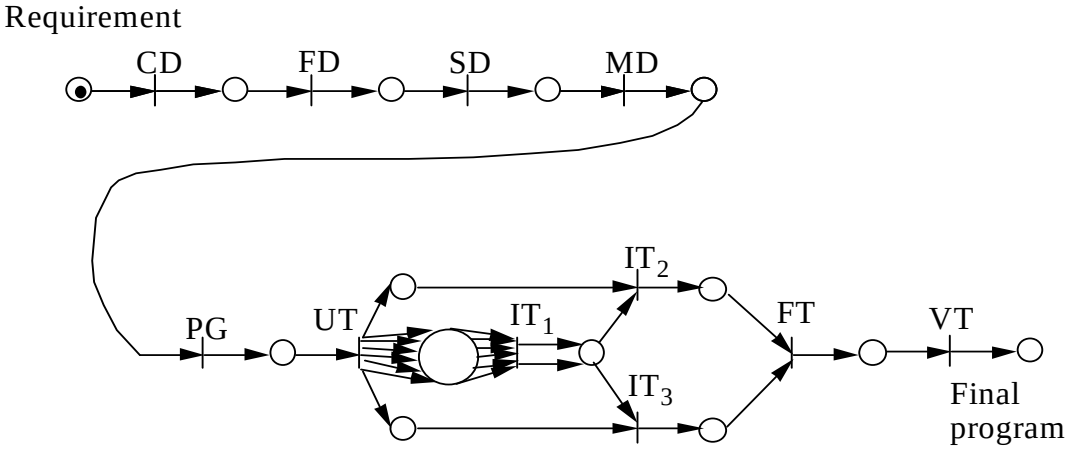

図5．3 改善されたプロセスの記述

⟫ しかし、最初から図５．３の記述が得られたわけではない。議論の最初、開発者は「何を実際に行ったか」というよりは、「何を行うべきであったか」という議論に終始した。これが、現状プロセスの理解に３ヶ月もの期間を要した理由である。そのため、ＰＲ１とＰＲ２で収集したデータ(例えば、各工程の実工数や各工程で発見されたフォールト数)を提示し、開発者が各工程で何をして、何をしなかったかということの事実確認を繰り返して行った。最終的に、図５．１と図５．２に示す現状プロセスの記述に成功した。⟫
⟫ 図５．１、図５．２からは、次の(1)〜(3)が認識できる。⟫

(1) ５つのモジュールB, C, D, E, FはＰＧ終了後すぐにＦＴ＆ＶＴを実施されている。⟫
(2)モジュールＧはＵＴ終了後すぐに、ＦＴ＆ＶＴを実施されている。⟫
(3) ＦＴとＶＴが並行して行われている。⟫
⟫
⟫⟫ ５．３．３⟫ 現状プロセスの分析(Step2)⟫

⟫ 現状プロセスの記述に実際のデータを割り当てて分析した結果、ブラックボックステスト(ＦＴとＶＴ)に多くの工数を要していることを確認した。これは、納入直前のＦＴとＶＴが混乱した状況となっていることを意味する。実際に、ＰＲ１とＰＲ２では、制御不能なほどに混乱した状態になり、そのためプロジェクトメンバーは多くの時間を費やしていた。これを定量的に確認するために、ブラックボックステストに対して次の３つのメトリクスＭ１、Ｍ２、Ｍ３を導入した。⟫
⟫
⟫
⟫⟫⟫⟫⟫ ブラックボックステストの工数⟫
⟫ Ｍ１＝⟫ ―――――――――――――――⟫
⟫⟫⟫⟫⟫⟫⟫ 全プロジェクト工数⟫
⟫
⟫⟫⟫⟫⟫ ブラックボックステストで発見されたフォールト数⟫
⟫ Ｍ２＝⟫ ―――――――――――――――――――――――⟫
⟫⟫⟫⟫⟫⟫ コードレビュー以降に発見されたフォールト数⟫
⟫
⟫⟫⟫⟫⟫ ＶＴ開始後のＦＴの工数⟫
⟫ Ｍ３＝⟫ ―――――――――――⟫
⟫⟫⟫⟫⟫⟫⟫ ＦＴの全工数⟫

⟫
⟫ なお、Ｍ２の定義式で、「コードレビュー以降に発見された」というのは、「コードレビューとテストで発見された」ということを意味している。⟫
⟫ Ｍ１とＭ２は、テストの効果を評価するために従来から用いていたメトリクスである。これまでに継続してブラックボックステストで発見されたフォールト数についてのデータを収集・分析していたため、これらＭ１、Ｍ２の値が出荷後の品質や生産性と強い相関を持つことが確認されている。一方、Ｍ３は２つの工程ＦＴとＶＴにおける制御不能状態を評価するために新しく導入した。⟫
⟫ 評価結果を表５．１のＰＲ１、ＰＲ２の部分に示す。これらの値と開発者へのインタビューから、以下の問題点(1)〜(3)を確認した。⟫

(1)Ｍ１:⟫開発工数の約半分がブラックボックステストに費やされている。これは、納入直⟫
⟫ ⟫ 前の数ヶ月に徹底してブラックボックステストを実施していたことと一致している。⟫
(2)Ｍ２:⟫ブラックボックステストでほとんどのフォールトが発見されている。この事実か⟫
⟫ ⟫ ら、プロジェクトＰＲ１とＰＲ２では、開発効率が非常に低く、テスト開始時点のプ⟫
⟫ ⟫ ロダクト品質も悪いことを示唆する。⟫
(3)Ｍ３:⟫ＶＴが開発者によるテストが終了する前に始められている。管理者と開発者への⟫
⟫ ⟫ 納期のプレッシャーによるあせりが表れていると考えられる。⟫

⟫ そこでＳＥＰＧはＰＧとテスト工程（ＵＴ，ＩＴ，ＦＴ，ＶＴ）における問題が明らかになるまで、開発者に対して以下の質問Ｑ１、Ｑ２、Ｑ３を繰り返し行った。⟫
⟫
Ｑ１:ブラックボックステストで発見されたフォールとは､どの工程で作りこまれたのか。⟫
Ｑ２：ソースコードのレビューをどのように行ったか。⟫
Ｑ３：ホワイトボックステストをどのように行ったか。⟫
⟫
⟫ その結果、以下にまとめる原因と思われる事実が明かになった。⟫
(1)ほとんどのフォールトはＰＧで作りこまれているが､コードレビューではほとんど発見⟫
⟫ されていない。これはレビュー方法が適切でなかったためである。⟫
(2)更に、ホワイトボックステストでは、ほとんどフォールトが発見されていない。これは⟫
⟫ ホワイトボックステストが明確に実施されていなかったことを意味する。⟫

表５．１　FTとVTに対するメトリクス

Metrics \ Project	*PR1*	*PR2*	*PR3*
(M1) $\frac{\text{Effort for black box test}}{\text{Total project effort}}$	47%	49%	44%
(M2) $\frac{\text{Number of defects removed in black box test}}{\text{Total number of defect+}}$	96%	94%	51%
(M3) $\frac{\text{Effort for FT after VT started}}{\text{Total effort for FT}}$	45%	34%	0%

+: After PG phase (including it)

(3)開発メンバーはコードレビューとホワイトボックステストの必用性を認識しているが、納期に対する強いプレッシャーが、これらの作業の実施を妨げていた。

５．３．４　改善計画の作成(Step3)

明かになった問題点の解消を目指して改善計画を作成した。改善計画は改善されたプロセスの記述と記述上の各作業に対する詳細な指示から構成される。

図５．３に改善されたプロセスの記述を示す。図５．３において、ＩＴ１はモジュールB，C，D，E，F，Gの結合テストに、ＩＴ２はモジュールA，B，C，D，E，F，Gの結合テストに、ＩＴ３はモジュールB，C，D，E，F，G，Hの結合テストに、それぞれ対応している。各作業に対する指示は次の通りである。

(1)より実用的で効率のよいコードレビューを実施する。

(1.1)レビューを2人で行う[14](1人がコードの説明をし、もう１人がその内容を確認する)。

(1.2)１時間当り１００行から２００行をレビューする。

(1.3)新規作成された部分と変更された部分は必ずレビューする。

(2)ＵＴに単体テストツールを導入すると共に、フォールト報告書の提出を求める。

(2.1)新規作成された部分と変更された部分は必ずツールを用いてテストする。

(2.2)ＵＴで発見・除去されたフォールトは必ず記録する。

(3)ソフトウェアの構造に合わせてＩＴを実施する。具体的には以下のようにＩＴを実施する。

(3.1)ＩＴ（ＩＴ１，ＩＴ２，ＩＴ３）を改善プロセスに記述された順番に行う。

(3.2)各ＩＴでのテスト項目は、テストを実施する前に決定する。

(3.3)ＩＴで発見・除去されたフォールトは必ず記録する(2.2と同様)。

(4)コードレビューとＵＴのスケジュールを制御可能にする。管理者は開発メンバーに以下のことを要求する。

(4.1)コードレビューとＵＴにおける、各モジュール毎の進捗と発見されたフォールト数を記録する。

(4.2)(4.1)の記録を定期的に管理者に報告する。

上述の様に、現状プロセスの把握には3ヶ月を要した。その作業によって現状プロセスを完全に把握することができたため、改善計画の作成には2週間しかかからなかった。

５．３．５ 利益予測(Step4)

５．１．２項で述べたように、事業としては主に開発コストに焦点をあてている。従って、予測される工数の削減で改善計画による利益を評価する。

ＳＥＰＧは改善計画をＰＲ１に適用した場合の総工数を予測した。以降、実際のＰＲ１をＰＲ１Ｏ、改善計画が適用されたＰＲ１をＰＲ１Ｉと表す。ＰＲ１Ｉに対する予測では、

次の(A1)～(A5)を仮定した。この内、(A2)～(A5)の数字は、対象組織での過去の成功したプロジェクトと類似の数値を挙げていた文献[17]から引用した。

(A1) ＰＲ１ＩのＰＧとテスト工程で発見される総フォールト数は、ＰＲ１ＯのＰＧとテスト工程で発見された総フォールト数と等しい。
(A2) ＰＧのコードレビューにおいて、総フォールトの５２．６％が発見される。
(A3) ホワイトボックステスト(ＵＴとＩＴ)で総フォールトの２６．４％が発見される。
(A4) ＦＴでは、総フォールトの１５．９％が発見される。
(A5) ＶＴでは、総フォールトの５．１％が発見される。

次に、各工程で必要とされる工数を算出するために、以下の式を用いた。

$$Ei = Eim + Eid + Eir + Eio$$
$$Eim = \alpha i \times s$$
$$Eid = \beta i \times ni$$
$$Eir = \gamma i \times ni$$
$$Eio = \delta i$$

ここで、Ei は開発プロセス中の i 工程(i＝ＰＧ, ＵＴ, ＩＴ, ＦＴ, ＶＴ)における総工数を表す。Eim は i 工程において、プロダクトを作成するために必要な工数を表す(但し、i がＵＴ, ＩＴ, ＦＴ, ＶＴの場合は、これらの工程でプロダクトを作成しないため、Eim=0 である)。Eid はフォールト発見に要する工数を、Eir はフォールト修正に要する工数を、Eio は環境設定、文書作成等の工数を、それぞれ表す。また、s はプロダクトのサイズを、ni は i 工程で発見されたフォールト数を、ぞれぞれ表す。

γi の値はＰＲ１Ｏでの実際の値を用いた。一方、αi、βi、δi の値は、幾つかの過去のプロジェクトのデータを参考に、プロジェクトメンバーとの議論を重ねて決定した。

表５．２にＰＲ１Ｏでの各工程毎に発見されたフォールト数と工数(人日)を示す。一方、表５．３にはＰＲ１Ｉでの予測結果を示す。

表５．２と表５．３から、提案した改善計画を適用することで、工数について１４．７％(1-1083/1270=0.147)の削減が予測される。更に、同様の方法をＰＲ２に適用した場合、１１．３％の工数削減が予測された。

結果として、改善計画を実施することで約１０%の工数削減が達成できると判断した。

表5．2　PR1Oでのフォールト数と工数

Phase	CD, FD, SD, MD	PG	UT, IT	FT	VT	Others	Total
Number of defects (distribution (%))		14 (4.5)	0 (0)	219 (69.7)	81 (25.8)		314 (100.0)
Effort (man-days) (distribution (%))	399 (31.4)	134 (10.5)	111 (8.8)	448 (35.3)	151 (11.9)	27 (2.1)	1270 (100.0)

表5．3　PR1Iでのフォールト数と工数の予測

Phase	CD, FD, SD, MD	PG	UT, IT	FT	VT	Others	Total
Number of defects (distribution (%))		165 (52.6)	83 (26.4)	50 (15.9)	16 (5.1)		314 (100.0)
Effort (man-days) (distribution (%))	399 (36.8)	155 (14.3)	199 (18.4)	227 (21.0)	76 (7.0)	27 (2.5)	1083 (100.0)

５．３．６　改善計画の実行(Step5)

提案した改善計画に基づいて新規プロジェクトＰＲ３を実行するための全体ミーティングを開いた。そこではＰＲ１、ＰＲ２で明かになった問題点、改善計画とそれを適用した場合の利益予測について説明を行い、管理者から承認を受けた。この開発メンバーは多くの会合を経て、改善計画の内容を完全に理解するのと同時に、プロセス改善の動機付けが十分になされていた。

５．４　改善計画の評価

５．４．１　結果

⟫ 表５．１に、ＰＲ３におけるＭ１、Ｍ２、Ｍ３の値も合わせて示している。⟫
⟫ 先ず、ＰＲ３におけるＭ１の値は期待したほど向上していない。これは、改善計画によってＰＲ３の他の工程の工数も同様に削減されたからである。一方、Ｍ２の値は大幅に改善された。半数のフォールトがブラックボックステストの前に発見されている。これは、プロダクトの品質が前工程でかなり保証されるようになって、結果としてフォールト修正の工数が削減されることを意味する。最後に、Ｍ３は０となった。ＶＴ工程が始まる時点で、ＰＲ３は完全に制御されている。その結果として、プロダクトの開発と納入は期限通りになされた。⟫
⟫
⟫⟫ ５．４．２⟫ 利益評価⟫

（１）ＰＲ１とＰＲ３の比較⟫
⟫ ＰＲ３との比較対象としてＰＲ１を選んだ。機能と開発規模の点で、ＰＲ１はＰＲ２よりも、ＰＲ３に類似度が高かったからである。⟫
⟫ 表５．２と表５．４にＰＲ１とＰＲ３の各工程における工数とフォールト数を示す。テスト工程のデータを比較すると、ＰＲ３のテスト効率と品質がＰＲ１に比べて非常に改善されていることがわかる。ＰＲ３のテスト工程での総工数は、ＰＲ１に比べて約１０％削減されている。この結果は、５．３．５項における利益予測の結果とほぼ等しい。⟫
⟫

（２）ＰＲ３と仮想的なＰＲ３の比較⟫
⟫ ここで、実際のプロジェクトＰＲ３を(改善されているという意味で)ＰＲ３Ｉと表し、改善計画が実現されていない仮想的なプロジェクトＰＲ３を定義してそれをＰＲ３Ｏと表す。⟫
⟫ 改善計画によって、削減された工数の大きさを評価するために、ＰＲ３Ｏの総工数を評価した。⟫
⟫ 評価に当たっては次の(1)～(3)を仮定した。⟫
⟫
(1)⟫ＰＲ３ＯのＰＧとテスト工程で発見される総フォールト数はＰＲ３ＩのＰＧとテスト⟫
⟫ 工程で発見された総フォールト数と等しい。⟫
(2)⟫ＰＲ３ＩのＰＧとＵＴで発見されたフォールトは全てＰＲ３ＯのＦＴで発見、修正さ

れる。

(3)コードレビューとＵＴはＰＲ３Ｏでは実施されない。

γｉ の値にＰＲ３Ｉの実測値を用いること以外は、５．３．５項で述べたものと同じ式を用いた。

表５．４と表５．５にＰＲ３ＩとＰＲ３Ｏの各工程毎のフォールト数と工数の値を示す。結果として、１１．８％(1-1051/1192=0.118)の工数削減が実現されている。この割合は５．３．５項で述べた利益予測の結果とほぼ等しい。

表５．４ PR3Iでのフォールト数と工数

Phase	CD, FD, SD, MD	PG	UT	IT	FT	VT	Others	Total
Number of defects (distribution(%))		141 (31.7)	78 (17.5)	83 (18.7)	93 (20.9)	50 (11.2)		445 (100.0)
Effort (man-days) (distribution(%))	303 (28.9)	75 (7.1)	5.4 (5.1)	127 (12.0)	215 (20.5)	244 (23.2)	33 (3.2)	1051 (100.0)

表５．５ PR3Oでのフォールト数と工数の予測

Phase	CD, FD, SD, MD	PG	UT	IT	FT	VT	Others	Total
Number of defects (distribution(%))		0 (0)	0 (0)	83 (18.7)	312 (70.1)	50 (11.2)		445 (100.0)
Effort (man-days) (distribution(%))	303 (25.5)	59 (4.9)	0 (0)	127 (10.6)	425 (35.7)	244 (20.5)	33 (2.8)	1192 (100.0)

５．４．３ プロジェクトメンバーからの意見

今回のプロセス改善を通して、開発メンバーと管理者は協力的であり、改善作業に対して非常によい印象を持っていた。主な理由を以下に示す。

(1)管理者はプロセス改善の必要性を完全に理解していた。
(2)多くの会合や議論を通じて理解し合うことで、ＳＥＰＧは開発メンバーと改善目標を共有することができた。
(3)開発メンバーと管理者は、ＰＲ３を実施している間、改善が実際に進んでいるということを実感できた。

協調作業がうまく行った一例として、コードレビューとＵＴ工程で使用される進捗報告書の書式を開発メンバーの１人が積極的に提案したということがある。
更に、次のような意見も寄せられた。

(1)納期直前の混乱が実際になくなった。
(2)多くのフォールトが早期に除去されるので、開発プロセスがより信頼性の高いものとなった。
(3)ＳＥＰＧは開発メンバーの一員であるかのごとく、プロセス改善に真剣に取り組んでいた。

５．５ まとめ

本章では、開発者指向のプロセス改善のための枠組みを提案した。提案する枠組みでは、現状の開発プロセスを形式的に記述し、開発者にとって実行可能な改善計画を提示する。更に、改善計画に基づいてプロセス改善を行ったときに得られる利益を定量的に予測する。この予測結果を提示することによって、開発者に対するプロセス改善の動機付けを与えるところに特徴がある。更に、提案した枠組みを実際の開発プロジェクトに適用し、有効性の評価を行った。その結果、利益予測で得られた結果とほぼ同じ１０％の工数削減が達成された。

今後の課題としては以下のことがあげられる。
(1) より多くのプロジェクトに対するプロセス改善の枠組みの適用と評価
(2) 実際のプロジェクトをより簡単に記述するための手法の開発
(3) 利益予測をより系統的に行う手法の開発

(3)については､既に一般化確率ペトリネットに基づくプロジェクトシミュレータを開発しており、現場への適用について検討をはじめている[18]。》
》

６．プロセス改善の成功に向けて

ソフトウェア開発で問題が多発する理由について根本の部分で大きく認識にばらつきがあるが、プロセス改善への認識のばらつきはそれ以上に大きなものがある。プロセス改善を適切に実行し、しかるべき効果を上げている組織は多いが、一方で課題認識がありながらプロセス改善がうまく立ちあがらない組織が非常に多い。プロセス改善が一度うまく立ち上がるとその成果を示すことによりプロセス改善の有効性をアピールし、継続的な改善活動が可能となってくる。プロセス改善の一番の難しさは、しかるべき経営リソースを確保し、いかにプロセス改善を適切に立ち上げるかにあると言える。

本章ではまずプロセス改善はなぜ難しいのかの理由を整理し、次にこの難しさを克服しいかにしてプロセス改善を立上げるかに焦点を当て、さらにプロセス改善の成果を挙げていくためにどのような取組みをすべきかを述べる。

６．１ プロセス改善はなぜ難しいのか

「ソフトウェア開発プロセスもソフトウェアである」[12]というオスターワイル教授の有名な言葉があるが、プロセス改善もソフトウェア開発自身の持っている難しさと同じような難しさを持っている。プロセス改善とは人間の知的活動の効率を高めようというものであるから､担当者の意欲とか感情といった人間臭いところが難しさの本質であると思う。

以下に幾つかの観点から見てのプロセス改善の難しさについて述べる。

（１）知識としての整理が不十分。

プロセス改善ということが盛んに言われ出したのは世界的に見ても１９９０年代に入ってからである｡CMMはプロセス改善のモデルとして知識の整理に大いに貢献しているが、しかしこれでさえも万人に対してそのまま適用できるほどの整理ができているとは言えない。つまりプロセス改善は工学としてはまだ未成熟であるということである。この理由として次のようなことが考えられる。

工学の領域では特定の要素だけを変化させていくつかのパターンで繰り返し実験しその結果の確認で知識を整理することを行うが、この方法はソフトウェアの開発においてはまったく非現実的で有り得ないことである。ソフトウェア開発にしてもプロセス改善にしても、実験をするには対象が大きすぎて実験による莫大な投資とそこから得られるわずかな

知識とのバランスがとれない。開発プロセスについて実験のできるのは、現実のプロジェクトよりかなり小規模なプロジェクトであったり、開発行為のある部分のみに焦点を当てたもので、組織全体のQ，C，Dとの相関があまり大きくないものであることが多い。実験による確認ができない領域であるプロセス改善を他の工学の領域と同じように扱うのは元々無理なこととも言える。》

》 工学として扱うには計測・数値化が必須条件であるが、人間の知的作業およびその改善量を定量的に計測するのは大変難しいことである。またそれらが計測できたとしても、プロセスとプロダクトの因果関係が複雑で一義的な説明が難しく、そこから選られた知識の内容は統計的・確率的なものにならざるを得ない。さらにここで扱われる数値は各企業の実力をそのまま表してしまうことが多いため、オープンに議論をして知識を増やしていくための企業をまたがった情報交換の場は少ない。》

》 これらのことより、データに基づく議論が難しく、絶対的に正しいという知識を共有化することは困難である。そのためにプロセス改善はどのように取り組むべきか、どのような効果があるのかといった基本的なことに関しても個々人の信念、思い込みが多く働き、プロセス改善の方針について共通認識を持つことが難しい。》

》

》 （2）開発現場への意識付け（組織文化の改革）が難しい。》

》 ソフトウェア開発現場の多くの人たちは、自分達は十分に頑張っているのだから現状で良い、または仕方がないと思っている。実際、ほとんどの開発現場の人たちは大変良く頑張っているが、頑張りだけで結果に対して言い訳をしていることが多く、改善に対して問題意識が欠如していることが多い。さらにこのような人たちに外部から改善点の問題指摘をするのは当人達にとっては面白くなく反発をくうことが多い。自分の問題を自分が気づいて改善するのは意欲が湧くが、他人から指摘されて改善するのは面白くないというのは世の常である[13]。》

》 一方、開発現場の人たちに問題意識があっても改善に取り掛かっていないことが多くある。一つは問題の根が深すぎて解決しないと思ってあきらめてしまっている場合。あまりにも忙しすぎて、とても改善のための投資を自分からできないと思っている。または怠け心から、慣れ親しんだ今までのやり方を変えたくない、改善のための投資をするのはめんどうだ、という理由で改善に取り掛からないケースもたくさんある。》

》 このような状況では改善推進部隊は開発現場への改善取組みの働きかけに大変なエネル

ギーを必要とし、管理者からの働き掛けが無い限りそのエネルギーは浪費で終わることが多い。》

》

》（３）ゴールが何かを忘れやすい。》

》プロセス改善はプロセスを変更して結果のプロダクトを良くするというのがゴールであるが、改善推進部隊はプロセスを変更するための手順の制定・浸透やツールの導入の作業に没頭している間にプロセスの変更が目的と勘違いしてしまいがちである。こうなると手順の制定やツールの導入で満足してしまい、それによって結果のプロダクトがどう良くなったかを追わなくなる。プロセスの変更は必ずそれによる効果を確認して次のプロセス変更のためにフィードバックをかけていく必要があるが、これを怠ると効果の無いプロセス変更が増えて結局は改善と称して無駄な作業が増えることになる。》

》

》（４）外部からの理解が得られにくい。》

》ここでいう外部とはソフトウェア開発現場およびプロセス改善部隊以外の経営層や上級管理職であったり、営業や製造といった開発以外の部門の人たち、さらに開発部門でもソフトウェア以外のスキルを持った人たちのことである。ソフトウェアの持っている特質を理解しようとしない人たちや、ソフトウェアの問題はソフトウェア担当者がしっかりしていないからだと思っている人たちに、しかるべきリソースを投入してプロセス改善をすべきであることを理解してもらい、協力を得るというのは大変難しいことである。ソフトウェア担当者がしっかりしていないから問題が起きると思っている経営層や上級管理職は、改善は開発現場が身銭を切ってやるものだと思っていることが多く、このような状態であると改善のために必要なリソースが確保されることはない。またこのような人たちはマスコミのいいかげんな情報に振り回される事が多く、「一年以内に生産性２倍､開発期間半分を達成する」などという実現不可能な、また計測不可能な改善目標が設定される。スローガンと改善目標は別物であることが理解されていない。また一方、数値に対する過度の期待をすることもあり、ソフトウェアメトリックスに対して工場の製造ラインでの数値と同じような精度で議論できるものだと思っている人もいる。このような状態であると改善成果を数値で示しても数値の理解が不十分で成果が評価されない事が多い。》

》

》（５）改善のための知識、技術が不十分なまま取り掛かる。》

》改善というと小集団活動のＱＣサークルと同じで、現場の誰でもが取掛かれると思っている人が多い。確かにソフトウェアにおいても開発現場だけで解決できる小さな問題はたくさんあるが、しかし組織全体の成熟度を上げていく改善は実に奥が深く、専門性の高い知識、技術を必要とする。ここでの知識は工学に近いものもあるが、どちらかといえば認知科学や心理学とか社会学の領域に近いものが多い。また、プロセス改善のプロセスにも標準形のようなものがあり、それらを良く理解した上で自部門用の作戦を立てないと、我流の作戦だけでは効果的な改善は難しい。》

》

》（６）改善成果が出るまでには時間がかかる。》

プロセス改善の成果が確認できるのはプロセスを変更したプロジェクトの開発が終わってからである。成果が不十分であると別の改善点を探しプロセス変更をして別の開発サイクルを回すということが必要で、ソフトウェアの開発サイクルはそれほど短いものではないので必然的に成果確認まで時間がかかる。しかも、成果の出たプロセス改善を組織全体に広めて歯止めをかけるためには制度化を行い、組織文化の改革が必要で、これには大変時間がかかる。わずかずつの成果しかでないため、我慢しきれずあきらめてしまうことが多い。》

》

》（７）経営層、改善推進部隊の改善への意欲が成果を左右する。》

》プロセス改善というのは経営リソースを投入して進めていくものであるので、経営にとっては当然投資的行動であるが、開発現場にとっても何らかの投資的行為が必要である。これを引き出すのが経営層なり改善推進部隊の役割であり、これらの人たちがどこまで本気かは開発現場の人は敏感に感じ取るものである。改善成果が上がらなくても頬被りして成果が出たようにいったりすると誰もついてこない。また、とりあえず目の前の問題が解決し､喉元過ぎればというように思ってしまうと改善成果の歯止めがかからない｡経営層、改善推進部隊にはどこまでも貪欲な改善への継続的意欲が求められる。》

》

》６．２》どのように取り組むか》

》前節で述べたようにプロセス改善は大変難しいもので､簡単に取り組むところが多いが成功するところは少ない。アメリカの調査ではプロセス改善の７０％は失敗に終わっている

ということもいわれている。このような中で、プロセス改善を成功させるためにどのような取組みをすべきであるかを以下に述べる。》

》

》（1）プロセス改善への経営トップの絶えざる関心》

》プロセス改善はその組織の経営課題を解決するための投資をすることであるから、投資の権限を持っている人からのトップダウンの改善指示が絶対に必要である。これによって具体的に改善のリソースが確保されるだけでなく、全員参加型で開発現場の末端までプロセス改善に巻き込むためには不可欠なものである。》

》またこの時に、経営トップはプロセス改善によってどのような項目をどのように改善したいかの改善成果指標を明示すべきである。Q，C，Dのどの項目をどのような尺度で測っていくらにしたいかを明確にし､必ず改善成果を計測し実績評価をしなければならない。良くあるのが開発期間半分とか生産性２倍といった計測方法を決めないままの改善目標である。ソフトウェア開発は全てが別のもので同じ開発ということは有り得ないため、ばらばらの期間とか生産性といったものを計測・比較するには何かで正規化しなければならない。これが不充分であると経営からは言っただけで終わり、改善目標の達成も検証が出来ず、結局真剣にプロセス改善をしようという雰囲気が無くなってしまう。そこで、経営サイドから見てのソフトウェアの課題を明確にし、それが例えば品質であれば「年間のフィールド不具合総件数を何%削減」というように、改善が達成できたのか出来なかったのかの議論の余地が無いように、明確な計測方法を示した上で改善目標を設定しなければならない。》

》

》（2）プロセスチャンピオンの掘り出し》

》アメリカではプロセス改善の推進役の中心として、なくてはならない人のことをプロセスチャンピオンというそうであるが、プロセス改善の成果の可否はプロセスチャンピオンを見つけられるか否かにかかっているといっても過言ではない。プロセスチャンピオンは責任感のある情熱家で、改善に対して意欲が高く、説得力もあって人望も厚い、優秀なエース級の人材である。つまり仕事の出来る人ということになるが、仕事の出来るタイプには二種類あって、一つは良いソフトウェアの設計が出来るタイプ、もう一つは良いマネジメントが出来るタイプである。プロセスチャンピオンは後者のタイプで、しかも改善ということに非常に興味を持っており、現状を少しでも良くしようということに生きがいを感

じているような人である。このような資質を持った人は、育てるというよりも天性の資質というようなもので、どのような組織にも必ず何人かはいるはずである。如何に早くこういう人を自分達の組織の中から見つけ出し、プロセス改善推進役として専任化していくかが大切である。

（３）改善専任組織の設置

前項のプロセスチャンピオンを中心に改善対象組織の規模等に応じて、改善専任組織メンバーを割当てしていく。改善リソースは開発リソースの１－３％程度必要であるといわれている。

また、一般的にプロセス改善の投資対効果(ＲＯＩ:Return on Investment)は５～８倍と言われている。つまり５～８億円の改善成果を期待するのであれば１億円程度の改善リソースを準備する必要がある。改善成果に確信の持てない経営者にとってはなかなか決心のつかないものであるとは思うが、逆にいえば一桁小さい投資しかしなければいくら改善が成功しても一桁小さい成果しか期待できないと言うことである。本気で改善成果を期待するのであればしっかりと改善投資をすべきである。

（４）プロセス改善の勉強

ボトムアップ型で現場の問題解決型のプロセス改善は現場の事情を知っている人であれば誰でも出来るものであり、非常に間口の広いものである。しかし、組織的な改善を進めるには目の前の問題を解決するという改善だけではなくもっと系統だった取組みが必要で、まずそのための知識の吸収をすべきである。プロセス改善の知識は整理が不十分と前述したが、これは他の工学の領域と比べて、証明済みで万人が認めるような知識としての整理が不十分であるといったので参考になる知識が無いといったのではない。自分の組織でそのまま当てはまらないまでも少し工夫をすれば役に立つ知識や参考になる改善事例はたくさん発表されている。その代表というべきものがＣＭＭであり、ＩＤＥＡＬモデルといったものである。これらはプロセス改善プロセスの標準形とも言うべきもので、ソフトウェア開発プロセスでもまず標準プロセスを理解した上で個別のプロジェクト毎のプロセスを定義すべきであるのと全く同じことで、プロセス改善の標準プロセスをまず理解した上で個別のプロセス改善の作戦を考えるべきである。

またプロセス改善は人間の特質を良く理解した上で、人間が陥りやすい失敗パターンに

対して組織的な対策を打っていくものであるので、認知科学とか社会心理学といった社会学的アプローチもベースとしての知識として必要になってくる。

（５）経営から期待される改善成果指標の分解

（１）項で、「まず経営が期待する改善成果を計測可能な具体的数値として定義すべきである」と述べた。経営トップのこのような方針発表によって設計者の気持ちが引き締まり少しは改善成果が得られる。しかし気持ちだけで改善できる量というのはわずかなもので、本当の改善成果を得るためにはゴールとすべき成果指標を分解して、その成果と因果関係のあるプロセス要素のどこをどう注力すれば良いのかを示すプロセスの改善点を抽出し、設計者に示す必要がある。

例えば、フィールド不具合件数５０％削減がゴールであると、現状でのフィールド不具合件数をカテゴリー別に分類して、カテゴリー毎に改善目標値を割り振る。さらに各カテゴリー毎にプロセスの注力点を抽出しプロセスとしての目標値を示す。フィールドバグ数と相関のあるプロセスは図２．１０に示したが、例えばデザインレビュー工数比率を２０％にするとか、テストにおけるカバレッジをいくらにするといった、設計者がどうすべきであるかを示す目標数値への落とし込みが必要である。

（６）プロセス改善の短・中期計画策定

「プロセス改善は開発プロジェクトのごとく実行しろ」ということがよく言われる。プロセス改善はただ漫然と改善活動を行ってもあまり効果が上がらない。それは開発プロジェクトと同じように、ゴールを定め作戦を立てて実行していくものであるので計画策定は必須である。計画策定にはプロセス改善の標準プロセスから自社向けの工夫をし、自社向けの作戦が必要である。計画には３年程度でどの成熟度レベルを目指すかといった中期計画と、具体的な実行内容を決めた１年程度の短期計画が必要である。

（７）プロセス改善に熱心なマネージャとの協力関係

プロセス改善を開始する時や、その組織にとっては実証済みでない新しい改善項目を取り入れる場合は、どこかで改善の有効性を確認実証してから組織全体に広めていくことをする必要がある。これはソフトウェアの開発プロセスというのは実に様々な条件の下で様々な形を持っているため、標準的なプロセス改善の方法や他の組織で有効であった改善

項目がそのまま対象としている組織で有効である保証はないためである。プロセス改善が有効でないと思っているマネージャに無理矢理プロセス改善に参加をしてもらっても、熱心な取組みが無い状態では改善効果はあまり期待できない。》

》 そこでその組織にとってそのプロセス改善が有効であることをパイロットプロジェクトで実績作りし、その成果を示しながら組織全体に改善を広めていくのが有効である。パイロットプロジェクトは何としても成功させないといけないので対象プロジェクトの選定が非常に大切である。その第一条件は、対象プロジェクトのマネージャがプロセス改善に理解があり積極的に改善投資をする覚悟を持っており、改善推進部隊と友好的な関係にあり協力して改善に取組めることである。》

》

》 （8）データ収集・分析》

データ収集は実行中のプロジェクト管理に必要であることは当然であるが、プロセス改善においてもプロセスとプロダクトの定量的計測が絶対に必要である。改善前の現状分析・問題点抽出、改善成果の計測、プロセスとプロダクトの因果関係を解明し改善知識として再利用、といったことのためにデータが使われる。》

》 収集すべきデータはどの組織でも同じようなものは、工数、品質データ、プロダクトの量といったものであまり多くはない。ただしこれらのデータの粒度と精度が問題で、未成熟な組織に対してあまりに高い粒度と精度を求めると収集自体が出来なくなってしまう。まず、その組織にとって収集可能な粒度と精度で収集をすれば、その組織としてのそれなりに役に立つ分析は可能である。そして少しずつデータ収集の有効性を示しながら粒度と精度を上げていく。このためにはプロセス定義の粒度を細かくしていくこととデータの信憑性をチェックする仕組みが必要で、とりもなおさず成熟度レベルを上げていくということになる。》

》 データ収集・分析は成熟度向上に不可欠なものであり、また逆に成熟度が向上すればデータ収集がしやすくなり分析結果も更に有効なものとなってくる。》

》 データ収集は未成熟な組織においては大変抵抗の大きなものであるので、明確に目的を持ち項目を絞って収集し、分析結果はきちんと開発現場にフィードバックをしなければならない。》

》

》 （9）アセスメントの実施》

》ソフトウェアのプロセス改善がすべてCMMに基づいて行わなければならないことはないし、CMMに基づく場合もＩＤＥＡＬモデルに基づく改善サイクルを廻さなければならないということでもない。自部門での十分認識されている問題解決に当たり、解決策とその優先順位付けを決める際にCMMのＫＰＡを参考にするだけで十分なこともある。》
しかし一般的にはアセスメントは非常に有効な方法であるといえる。アセスメントには、顕在化している問題の解決方法の提示、つまりその組織にとっての改善項目の抽出という側面と、潜在化している問題点の抽出、つまり改善成果の歯止めとしての組織的制度化の確認という側面がある。》
》アセスメントは有効であるとはいえフォーマルアセスメントの実施は時間とお金のかかるものであるのでそんなに頻繁に出来るものではない。そこで顕在化している問題の改善項目抽出を主たる目的として数ヶ月に一回くらいの割合で半日程度のミニアセスメントを実施し、組織的制度化による改善の歯止めを確認する事を主たる目的にし２年に一回くらいの割合で実施するフォーマルアセスメントを組合わせるのが効果的であると思われる。》
》
》（１０）社外との交流》
》プロセス改善というのは力任せにがんばれば出来るものと思っている人も多いが、実際にはベースとなる知識はしっかりと持っておく必要がある。この知識はそれほど多くのものではないが自分の経験や自社内の経験だけから得るには限界がある。各社のおかれている状況は様々なので他社の事例がそのまま自社に当てはまることはあまり無いが、シンポジウムやワークショップへ参加し他社の状況を参考に自社のプロセス改善を振り返ってみるのは大変有効である。自社の状況を論文など形で発表するとその何倍も他社の情報が入ってくるようになる。これらの交換される情報は一般の工学分野の情報とは違い、情報を得たからといってそのまま自社の利益に繋がるものではなく、自分達なりの状況に置き換えた理解と改善実行の大変な努力があって始めて成果を生むものである。よって情報交換が即利害関係に繋がらないので、ＳＥＰＧの集まりは会社をまたがったものであっても非常に率直にかつ友好的に行われるのが常である。》
》
》以上述べたようにプロセス改善を確実に成功させるための知識は未整備で万人が共通認識を持つまでには至っていない。改善スタートには誰の目にも明らかな投資が必要である。ところがその投資が成果を上げるか否かの確信がないまま投資の決心が必要で、決心をす

る権限を持っている人が改善に対してどのような認識をしているかで改善が開始するか否かが決定される。》

》図６．１はプロセス改善の投資対効果の関係を示している。横軸は時間経過を表し、縦軸は組織全体のパフォーマンス・改善投資・改善効果を表している。組織全体のパフォーマンスは改善をしていないと急激に低下してくる。あるところで決心をして改善投資を行い改善を開始するが改善効果はすぐには出てこない。初期段階における改善投資は改善サイクルが定常的になった時よりも多く投資をしなければならない。改善効果は改善開始よりｔだけ遅れて現れてくる。このため初期投資量をａとするとｔの期間は全体のパフォーマンスがａだけ下がることになる。プロセス改善のもっとも難しいところはこのパフォーマンスの低下に耐えてどこまで改善投資が出来るかということである。投資権限を持った人がその気になるか否かが問題で、「改善をしようと本気で思った人の権限範囲が改善できる範囲」であるといえる。》

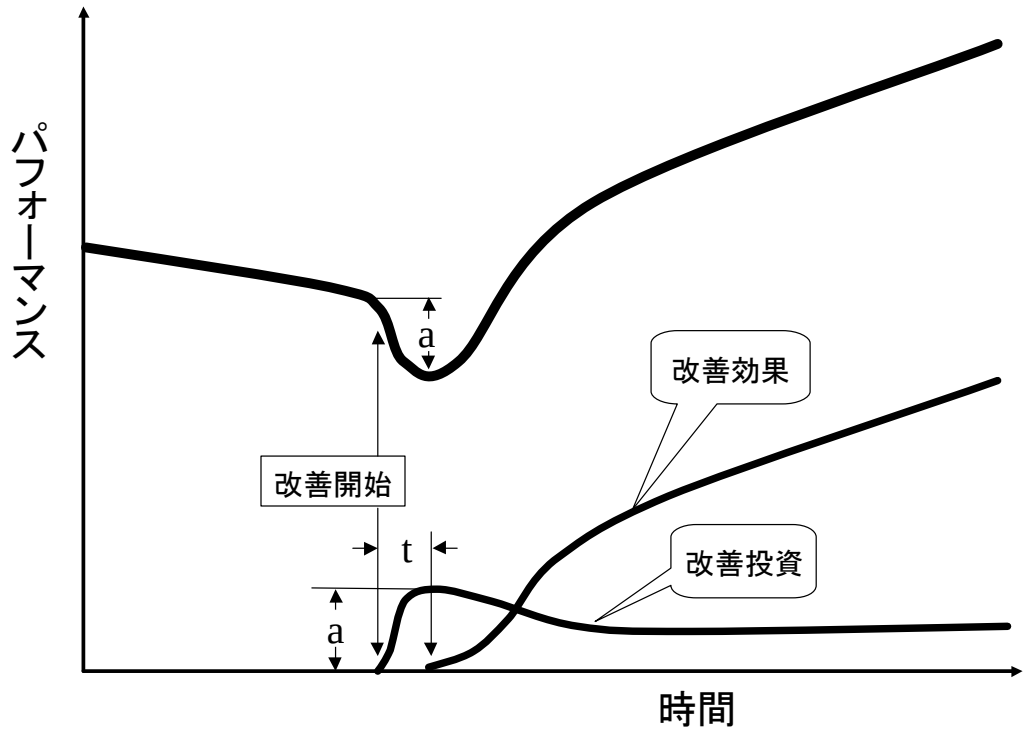

図6．１ プロセス改善の投資対効果

７．おわりに

本章では、本研究で得られた成果をまとめ、今後の課題について述べる。

本研究では、世界中でソフトウェア開発の問題が発生しつづけ、さらにその問題解決のための改善活動がなかなか効果を上げていないという現状を踏まえ、まずソフトウェア開発とそのプロセス改善の持っている本質的な問題点を整理し、その解決のために取組むべき方向を示した。プロセス改善は課題解決型の取り組みであり、それぞれの組織の課題認識の大きさにより、非常にミクロな末端レベルの改善で終わるものもあるし、組織全体として大きな成果を上げるものもある。このようになる原因として、プロセス改善とはどういうものであるかの共通認識が不足しているが考えられるため、非常にマクロなレベルでのプロセス改善のプロセスモデルを提示し、プロセス改善を成功させるために必要な知識の種類を整理した。また、プロセス改善の中でも最後まで課題として残る品質問題について、品質問題が持っている難しさはどこから来るのかを整理し、開発工程毎の品質決定要因と品質向上のための取組み項目について述べた。

次に、実際のプロセス改善の事例として、実プロジェクトデータの分析結果を三例紹介した。

まず第１の事例は、品質向上のためにデザインレビューが有効であることに着目し、全設計工数の中に占めるデザインレビュー工数の比率とフィールド品質の関係を分析した。その結果、レビュー工数比率１０％以上のプロジェクトは明らかにフィールド品質は良く、レビュー工数比率が２０％になるとほとんどフィールドバグはゼロであることが分かった。さらにコーディング工程の設計工数の中に占めるレビュー工数比率は大きくフィールド品質に影響を与えることを示し、全体のレビュー工数比率を上げるだけではなくコードレビュー工数比率を１５％以上確保すべきであることが分かった。

第２の事例は、プロジェクトが安定していないとロスが生じて生産性、品質とも悪化する傾向にあることを示した。プロジェクトの安定度合いを開発費の計画値と実績値の乖離度合いで評価し、乖離が±１０％以内のプロジェクトはそれ以外のプロジェクトに比べて生産性、品質ともに良いことを示した。また一般に生産性と品質はトレードオフの関係にあるとよく言われるが、デザインレビュー工数比率と生産性は正の相関があることを示し、プロジェクトを安定させ実行すべきプロセスを確実に実行させることにより生産性、品質

とも同時に向上させることができることを示した。さらに工程計画書の内容を評価しその充実度と開発費の計画値と実績値の乖離度合いは明らかに相関があり、計画段階の充実がプロジェクトを安定させ、さらに生産性、品質とも向上させることを示した。》

》第３の事例は、開発者指向のプロセス改善の枠組みを提案し、この枠組みによりほぼ計画通りの改善が行えることを示した。提案する枠組みでは、現状の開発プロセスを形式的に記述し、開発者にとって実行可能な改善計画を提示する。更に、改善計画に基づいてプロセス改善を行ったときに得られる利益を定量的に予測する。この予測結果を提示することによって､開発者に対するプロセス改善の動機付けを与えるところに特徴がある｡更に、提案した枠組みを実際の開発プロジェクトに適用し、有効性の評価を行った。その結果、利益予測で得られた結果とほぼ同じ１０％の工数削減が達成された。》

》最後に、開発の現場としてプロセス改善を定常的に行っていく上でのさまざまな障害を整理し、プロセス改善の立上げから組織的に定着させるまでの取り組み方を述べた。》

》

》今後の課題として以下の研究を進めていく必要がある。》

》ソフトウェアのプロセス改善は成功する組織がある一方、あまり成果を上げられていない組織も多い。成功する組織はたまたま偶然成功するのではなく、それなりの工夫や知識の蓄積があって成功している。しかしそれらの知識は大半が経験知、暗黙知の領域にとどまっており、多くの組織がその知識を共有するに至っていない。そこで次の二つの観点から、経験知、暗黙知を明示知としてまとめ、多くの組織で知識の共有化ができるようにしていかなければならない。》

》第１の観点として、本論文の第３章、第４章で事例紹介したような、開発プロセスのデータとその結果であるソフトウェアプロダクトのQ，C，Dの相関を数多く分析することである。開発プロセスのどこに注力することでプロダクトのQ，C，Dにどのように影響を与えるかを知ることにより､開発中のプロセスを制御し結果としてプロダクトのQ, C，Dが制御可能となる。開発プロセスデータには非常に多くの種類があるが、この中で開発中に制御が容易でしかもプロダクトのQ，C，Dとの相関の強いものを見つけ出し、明示知として知識の共有化を図っていかなければならない。》

》第２の観点として、本論文の第５章で事例紹介したような、プロセス改善を効果的に進める改善プロセスの分析である。プロセス改善で効果を上げた事例を分析し、改善推進組織であるＳＥＰGが次に挙げるような観点でどのような取り組みをしたから有効であった

かを明示知として纏めていく必要がある。》
・改善計画をどのように立てたか。どの程度の改善投資をしたか。》
・改善前の現状プロセスをどのようにして把握したか。》
・改善前、改善後のプロセスをどのような表記法であらわすのが効果的であるか。》
・現状プロセスから改善点を抽出するのはどのように行ったか。》
・開発プロセスデータはどのような項目をどのようにして収集したか。》
・改善による効果をどのような方法で見積もったか。》
・プロセス改善への積極的参加を開発担当にどのように意識付けたか。》
これらの知識は統計的な数値として纏めることは難しく、個々の成功事例の要素をたくさん集める必要がある。さらにこれらは人間の理解の仕方であるとか、共同作業を進める上での人の気持ちの問題といったこととも関係しており、従来の工学の領域だけではなく心理学、認知科学、社会学といった領域との共同研究が求められている。》

謝辞⟩

⟩

⟩　私が本研究を始めるきっかけとなったのは１０数年前にソフトウェア信頼性シンポジウムにて奈良先端科学技術大学院大学⟩ 鳥居⟩ 宏次⟩ 副学長とお知り合いになれたからで、以来、研究テーマの範囲に限らず幅広くご指導ご助言をいただきました。私の全ての研究実績の原点は鳥居副学長にあると言っても過言ではありません。さらに私の個人的事情で本研究の纏め上げに挫折しかけたとき、力強く励ましていただき、具体的に本研究纏め上げのご指導をいただきました。鳥居副学長なしには本論文の完成はなかったと思います。公私ともにご指導、ご支援をいただいた鳥居副学長に心より感謝の意を表したいと思います。⟩

⟩　本研究をまとめ上げるにあたり、適切なご指導とご助言をいただきました奈良先端科学技術大学院大学情報科学研究科⟩ 井上⟩ 克郎⟩ 教授に心より感謝いたします。⟩

⟩　大阪大学大学院基礎工学研究科⟩ 菊野⟩ 亨⟩ 教授には１０年近くもの間、本研究の全過程において適切なご指導とご助言をいただき、研究業績の大半の論文を菊野教授のご指導とご支援のもとに完成させることが出来ました。心より感謝いたします。⟩

⟩　本研究を進めるにあたり、終始適切なご指導とご助言をいただきました奈良先端科学技術大学院大学情報科学研究科⟩ 松本⟩ 健一⟩ 助教授に心より感謝いたします。⟩

⟩　本研究を進めるにあたり、終始適切なご指導とご助言をいただき、さらに本研究のまとめにあたり特段のご指導とご支援をいただきました大阪大学大学院基礎工学研究科⟩ 楠本⟩ 真二⟩ 助教授に心より感謝いたします。⟩

⟩　本研究をまとめ上げるにあたり、適切なご指導とご支援をいただきました奈良先端科学技術大学院大学情報科学研究科⟩ 飯田⟩ 元⟩ 助教授に心より感謝いたします。⟩

⟩　本研究を進めるにあたり、プロセス改善を知的な共同作業の観点から分析し貴重なご助言とご指導をいただきました奈良先端科学技術大学院大学情報科学研究科⟩ 中小路⟩ 久美代⟩ 助教授に心より感謝いたします。⟩

⟩　本研究を進めるにあたりデータ分析の統計的手法のご指導と研究業績論文作成のご指導およびご支援をいただいた大阪大学大学院基礎工学研究科⟩ 水野⟩ 修⟩ 助手に心より感謝いたします。⟩

⟩　本研究を進めるにあたり研究の方向付けについて、いろいろな議論を通じて多くの示唆を与えてくださったＳＲＡ株式会社⟩ 岸田⟩ 孝一⟩ 副社長に深く感謝いたします。⟩

⟩　さらに、本研究の機会を与えてくださったオムロン株式会社ソーシャルシステムズビジ

ネスカンパニー開発生産センター所長の田村⟫ 稔⟫ 常務に感謝いたします。⟫
⟫ また、オムロン株式会社ソーシャルシステムズビジネスカンパニーでの長年にわたるソフトウェアプロセス改善にともに取組み、本研究の共同研究者として協力していただいた田中⟫ 敏文君、高木⟫ 徳生君、新原⟫ 直樹君、稲垣⟫ 勝巳君に感謝します。⟫
⟫ 最後に、論文作成がなかなか進まない中で心配をかけ、さらに私の個人的逆境の中で心のささえとなり、論文完成のエネルギーを与えてくれた⟫ 妻⟫ 恵子に心から感謝します。⟫

参考文献

[1] 米国のソフトウェア産業の競合性に関するヒヤリング報告書(Nov. 13, 1991) : 米国 上院 通商、科学、及び輸送委員会報告書, 松原友夫訳.

[2] Basili V. R. and Rombach H. D.: "The TAME project: Towards improvement-oriented software environment", IEEE Trans. Software Engineering, Vol.14, No.6, pp.758-773 (1988).

[3] Brooks F. P., Jr.: "No silver bullet: Essence and accidents of software engineering", Computer, Vol.20, No. 4, pp.10-19 (1987).

[4] Brooks F. P., Jr.: "The Mythical Man-Month", Addison-Wesley (1975).

[5] Humphrey W. S.: "Managing the Software Process", Software Engineering Institute, Addison-Wesley (1989).

[6] 飯塚悦功編 : ”ソフトウェアの品質保証”, 日本規格協会(1992).

[7] ＳＰＡ：ISO/IEC TR 15504-1,2,3,4,5,6,7,8,9 : 1998.

[8] 山田, 高橋 : ”ソフトウェアマネジメントモデル入門”, 共立出版(1993).

[9] TR24 : Paulk M., C., Curtis B., Chrissis M. B., and Weber C. V., "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.

[10] TR25 : Paulk M., C., Weber C. V., Garcia S. M., Chrissis M. B., and Bush M. W., "Key Practices of the Capability Maturity Model, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-25, DTIC Number ADA263432, February 1993.

[11] IDEAL : http://www.sei.cmu.edu/ideal/ideal.html

[12] Osterweil L. :) "Software processes are software too", Proceedings of 9th International Conference on Software Engineering, pp.2-13 (March 1987) Monterey CA USA.

[13] Fox C. and Frakes W.: "The Quality Approach: Is It Delivering?", Communications of the ACM, Vol.40. No.6, pp.25-29 (1997).

[14] Bisant, D. B. and Lyle, J. R., "A two-person inspection method to improve programming productivity", IEEE Trans. Software Eng., Vol.15, No.10, pp.1294-1304 (1989).

[15] Koltun, P.: "Software process assessment: A first step to improvement", Proceedings of International Software Quality Exchange 92, pp.4B31-4B39 (March 1992) San Francisco CA USA.

[16] Peterson J. L.: "Petri Net Theory and the Modeling of Systems", Prentice-Hall (1981).

[17] Piwowarski P., Ohba M. and Caruso J.: "Coverage measurement experience during function test", Proceedings of 15th International Conference on Software Engineering, pp.287-301 (May 1993) Baltimore MD USA.

[18] Kusumoto S., Mizuno O., Kikuno T., Takagi Y. and Sakamoto K.: "A new software project simulator based on generalized stochastic petri-net", Proceedings of the 19th International Conference on Software Engineering, pp.293-302 (May, 1997) Boston MA USA.

[19] Takagi, Y., Tanaka, T., Niihara, N., Sakamoto, K., Kusumoto, S. and Kikuno, T.: "Analysis of review's effectiveness based on software metrics", Proceedings of the 6th International Symposium on Software Reliability Engineering, pp.34-39 (October 1995), Toulouse, France.

[20] Tanaka, T., Sakamoto, K., Kusumoto, S., Matsumoto, K. and Kikuno, T.: "Improvement of software process by process description and benefit estimation", Proceedings of the 17th International Conference on Software Engineering, pp.123-132 (April 1995), Seattle, WA, USA.

[21] Sakamoto, K., Niihara, N., Tanaka, T., Nakakoji, K. and Kishida, K.: "Analysis of software process improvement experience using the project visibility index", Proceedings of Asia-Pacific Software Engineering Conference '96, pp.139-148 (December 1996), Seoul, Korea.

[22] Sakamoto, K., Nakakoji, K., Takagi, Y. and Niihara, N.: "Toward computational support for software process improvement activities", Proceedings of the International Conference on Software Engineering 98, pp.22-31 (April 1998), Kyoto, Japan.

[23] Mizuno, O., Kikuno, T., Inagaki, K., Takagi, Y. and Sakamoto, K.: "Analyzing effects of cost estimation accuracy on quality and productivity", Proceedings of the International Conference on Software Engineering 98, pp.410-419, (April 1998), Kyoto, Japan.

[24] Abdel-Hamid, T. K.: "The dynamics of software project staffing: A system dynamics based simulation approach", IEEE Trans. Software Engineering, Vol.15, No.2, pp.109-119 (1989).

[25] Armenise, P., Bandinelli, S., Ghezzi, C. and Morzenti, A.: "Software processes

representation languages: Survey and assessment", Proceedings Of 4th Conference Software Engineering and Knowledge Engineering, pp.455-462 (June 1992).

[26] Bandinelli S. C., Fuggetta A. and Ghezzi C.: "Software process model evolution in the SPADE Environment", IEEE Trans. Software Engineering, Vol. 19, No.12, pp.1128-1144 (1993).

[27] Musa J. D., Iannino A. and Okumoto K.: "Software Reliability : Measurement, Prediction, Application", McGraw-Hill, 1987.

[28] Curtis B., Krasner H. and Iscoe N.: "A field study of the software design process for large systems", Communications of the ACM, Vol.31, No.11, pp.1268-1287 (1988).

[29] Furusawa K., Hirayama Y., Kusumoto S. and Kikuno T.: "Modeling and quantitative evaluation of software process based on a generalized stochastic Petri-net", Proceedings of 15th Software Reliability Symposium, pp.99-104 (1994) (in Japanese).

[30] Furuyama T., Arai Y. and Iio K.: "Fault generation model and mental stress effect analysis", The Journal of Systems and Software, Vol.26, pp.31-42 (1994).

[31] Hirayama Y., Mizuno O., Kusumoto S. and Kikuno T.: "Hierarchical project management model for quantitative evaluation of software process", Proceedings of the International Symposium on Software Engineering for the Next Generation, pp.40-49 (February 1996) Nagoya Japan.

[32] Kellner M. I.: "Software process modeling support for management planning and control", Proceedings of the 1st International Conference on Software Process, pp.8-28 (October 1993) Los Angeles CA USA.

[33] Lee, G., and Murata, T.: "A β-distributed stochastic Petri net model for software project time/cost management", The Journal of Systems and Software, Vol.26, No.2, pp.149-165 (1994).

[34] Liu, Lung-Chun, and Horowitz, E.: "A formal model for software project management", IEEE Trans. Software Engineering, Vol.15, No.10, pp.1280-1293 (1989).

[35] Matsumoto, K., Kusumoto, S., Kikuno, T., and Torii, K.: "An experimental evaluation of team performance in program development based on model --Extension of programmer performance model", Journal of Information Processing, Vol.15, No.3, pp.466-473 (1992) (in Japanese).

[36] Ousterhout, J. K.: "Tcl and the Tk Toolkit", Addison-Wesley, (1994).

[37] Raffo, D. M.: "Evaluating the impact of process improvements quantitatively using process modeling", Proceedings of CASCON93, Vol.1, pp.290-313 (1993).

[38] Sackman, H., Erickson, W. J., and Grant, E. E.: "Exploratory experimental studies comparing online and offline programming performance", Communications of the ACM, Vol.11, No.1, pp.3-11 (1968).

[39] Tvedt, J. D., and Collofello, J. S.: "Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling", Proceedings of COMPSAC95, pp.318-325 (August 1995) Dallas TX USA.

[40] Collofello, J. S., and Woodfield, S. N., : "Evaluating the effectiveness of reliability-assurance techniques", J. Syst. & Software, 9, 3, pp.191-195 (1989).

[41] Fagan, M. E. : "Design and code inspections to reduce errors in program development", IBM System Journal, Vol.15, No.3, pp.182-211 (1976).

[42] Frakes, W. B., Fox C. J. and Nejmeh B. A. : "Software Engineering in the UNIX/C Environment", Prentice-Hall (1991).

[43] Freedman, D. P. and Weinberg, G. M. : "Handbook of Walkthroughs, Inspections and Technical Reviews", Little Brown and Company (1982).

[44] Genuchten, M. V: "Why is software late? An empirical study of reason for delay in software development", IEEE Trans, Software Engineering, Vol.17, No.8, pp.582-590 (1991).

[45] Humphrey W.S. : "Characterizing the software process: A maturity framework", IEEE Software, Vol.5, No.2, pp.73-79 (1988).

[46] Johnson A. : "Software process improvement experience in the DP/MIS function", Proceedings of the 16th International Conference on Software Engineering, pp.323-329 (May 1993) Sorrento Italy.

[47] Knight, J. C. and Myers, E. A. : "An improved inspection technique", Communication of the ACM, Vol.36, No.11, pp.51-61(1993).

[48] Kusumoto S., Matsumoto K., Kikuno T. and Tanaka K. : "Improvement of software development process by using fault tolerant techniques", Journal of Computer Systems Science & Engineering, Vol.9, No.2, pp.83-88 (April 1994).

[49] Myers G. J. : "A controlled experiment in program testing and code walkthroughs/inspections", Communications of the ACM, 21, 9, pp.760-768 (1978).

)

[50]) Yourdon) E.) and) Constantine) L.) L.:) "Structured) Design:) Fundamentals) of) a) Discipline) of) Computer) Program) and) System) Design",) Prentice-Hall) (1979).)

)

[51]) Aoyama) M.) :) "Agile) software) process) model",) Proceedings) of) COMPSAC97,) pp.454--459) (August) 1997)) Washington) D.C.) USA.)

)

[52]) Basili) V.) R.,) Condon) S.) E.,) Emam) K.) E.,) Hendrick) R.) B.) and) Melo) W.) :) "Characterizing) and) modeling) the) cost) of) rework) in) a) library) of) reusable) software) components",) Proceedings) of) the) 19th) International) Conference) on) Software) Engineering,)) pp.283--291) (May) 1997)) Boston) MA) USA.)

)

[53]) Cimitile) A.) and) Visaggio) G.) :) "A) formalism) for) structured) planning) of) a) software) project",) International) Journal) of) Software) Engineering) and) Knowledge) Engineering,) Vol.4,) No.2,) pp.277--300) (1994).)

)

[54]) Fenton) N.) E.) and) Pfleeger) S.) L.) :) "Software) Metrics) :) A) Rigorous) &) Practical) Approach",) PWS) Publishing) (1997).)

)

[55]) Heemstra) F.) J.) :) "Software) cost) estimation",) Information) and) Software) Technology,) Vol.34,) No.10,) pp.627--639) (1992).)

)

[56]) Humphrey) W.) S.,) Snyder) T.) and) Willis) R.) :) "Software) process) improvement) at) Hughes) Aircraft",) IEEE) Software,) Vol.8,) No.4,) pp.11-23) (1991).)

)

[57]) Moller) K.) H.) and) Paulish) D.) J.) :) "Software) Metrics) :) A) Practitioner's) Guide) to) Improved) Product) Development",) IEEE) Press) (Chapman) &) Hall) Computing)) (1993).)

)

[58]) Poiaga) L.) :) "Operations) research) in) project) management) and) cost) engineering) :) An) outlook) for) new) operational) developments",) European) J.) Operational) Research,)

Vol.41,) No.1,) pp.1--14) (1989).)
)
[59]) Podorozhny) R.) M.) and) Osterweil) L.) J.) :) "The) criticality) of) modeling) formalisms) in) software) design) method) comparison",) Proceedings) of) the) 19th) International) Conference) on) Software) Engineering,) pp.303--313) (May) 1997)) Boston) MA) USA.)
)
[60]) Tausworthe) R.) C.) :) "The) work) breakdown) structure) in) software) project) management",) Journal) of) Systems) and) Software,) Vol.1,) pp.181--186) (1980).)
)
[61]) Yourdon) E.) :) "Death) March) :) The) Complete) Software) Developer's) Guide) to) Surviving) `Mission) Impossible') Projects",) Prentice) Hall) Computer) Books) (1997).)
)
[62]) Curtis) B.,) Krasner) H.) and) Iscoe) N.:) "A) field) study) of) the) software) design) process) for) large) systems",) Communications) of) the) ACM,) vol.31,) no.11,) pp.1268--1287,) 1988.)
)
[63]) Hirayama) Y.:) "Quantitative) evaluation) of) software) process) based) on) hierarchical) project) management) model",) Master) dissertation,) Osaka) University,) 1996.)
)
)
)

研究業績

論文

1. 坂本 啓司、田中 敏文、楠本 真二、松本 健一、菊野 亨、“利益予測に基づくソフトウェアプロセス改善の試み”、電子情報通信学会論文誌（採録決定）.

国際会議

1. Toshifumi Tanaka, Keishi Sakamoto, Shinji Kusumoto, Ken-ichi Matsumoto and Tohru Kikuno, “Improvement of software process by process description and benefit estimation,” Proceedings of the 17th International Conference on Software Engineering, pp.123-132, April 1995, Seattle WA USA.
2. Yasunari Takagi, Toshifumi Tanaka, Naoki Niihara, Keishi Sakamoto, Shinji Kusumoto and Tohru Kikuno, “Analysis of review's effectiveness based on software metrics,” Proceedings of the 6th International Symposium on Software Reliability Engineering, pp34-39, October 1995, Toulouse France.
3. Yasunari Takagi, Naoki Niihara, Katsumi Inagaki and Keishi Sakamoto, “Evaluation of process improvement activities using software metrics,” Proceedings of International Symposium on Future Software Technology '96, pp.125-130, October 1996, Xi'an China.
4. Keishi Sakamoto, Naoki Niihara, Toshifumi Tanaka, Kumiyo Nakakoji and Kouichi Kishida, “Analysis of software process improvement experience using the project visibility index,” Proceedings of Asia-Pacific Software Engineering Conference '96, pp.139-148, December 1996, Seoul Korea.
5. Shinji Kusumoto, Osamu Mizuno, Tohru Kikuno, Yuji Hirayama, Yasunari Takagi and Keishi Sakamoto, “A new software project simulator based on generalized stochastic Petri-net,” Proceedings of the International Conference on Software Engineering 97, pp.293-302, May 1997, Boston MA USA.
6. Katsumi Inagaki, Yasunari Takagi, Keishi Sakamoto and Osamu Mizuno, “Analyzing

the cost estimation accuracy in software project with respect to productivity and quality," Proceedings of International Symposium on Future Software Technology '97, pp.372-377, October 1997, Xiamen China.
7. Keishi Sakamoto, Kumiyo Nakakoji, Yasunari Takagi and Naoki Niihara, "Toward computational support for software process improvement activities," Proceedings of the International Conference on Software Engineering 98, pp.22-31, April 1998, Kyoto Japan.
8. Osamu Mizuno, Tohru Kikuno, Katsumi Inagaki, Yasunari Takagi and Keishi Sakamoto, "Analyzing effects of cost estimation accuracy on quality and productivity," Proceedings of the International Conference on Software Engineering 98, pp.410-419, April 1998, Kyoto Japan.
9. Osamu Mizuno, Tohru Kikuno, Yasunari Takagi and Keishi Sakamoto, "Characterization of risky projects based on project managers' evaluation," Proceedings of the International Conference on Software Engineering 2000 (In Press), June 2000, Limerick Ireland.

研究会、シンポジウム
1. 稲垣勝巳，高木徳生，坂本啓司，水野修，菊野亨、" ソフトウェア開発プロジェクトにおける開発計画の分析 －品質、生産性との関連性－ "、電子情報通信学会技術研究報告 SS97-27, pp.15-22, October 1997.
2. 水野修，菊野亨，稲垣勝巳，高木徳生，坂本啓司、" コスト見積り誤差評価の統計的仮説検定を用いた考察 "、ソフトウェアシンポジウム'98 予稿集、pp.77-82、June 1998.
3. 高木徳生、中小路久美代、新原直樹、坂本啓司、" プロセス改善技術移転に向けたプロセス改善活動の分析 "、ソフトウェアシンポジウム'98 予稿集、pp.24-30, June 1998.
4. 高橋光裕、菱谷淳、新原直樹、高木徳生、坂本啓司、" ファンクションポイント法の組込／制御ソフトウェアへの適用可能性に関する一考察 "、電子情報通信学会技術研究報告 SS98-2, pp.9-16, April 1998.
5. 高田 義広，坂本 啓司、" コンポーネントベースの組込みソフトの一開発で持った問題意識 "、電子情報通信学会技術研究報告 SS99-37, pp.17-24, November 1999.

書籍、雑誌

1. Keishi Sakamoto, Kouichi Kishida and Kumiyo Nakakoji, "Cultural Adaptation of the CMM: A Case Study of a Software Engineering Process Group in a Japanese Manufacturing Factory," Software Process, pp.137-154, John Wiley & Sons Ltd, 1996.
2. 坂本啓司、岸田孝一、中小路久美代、" プロセス成熟度モデルの活用によるプロセス改善 : 日本におけるSEPGの例 ", ソフトウェアプロセスのトレンド、pp. 211-237、海文堂、1997.
3. 坂本啓司、" ソフトウェアプロセス改善と品質向上 ", ソフトウェアテストと品質保証の実際、pp.70-109、日本テクノセンタ、1999.