博士論文

高品質ソフトウェア開発を実現する成功要因に関する研究

～ソフトウェア品質会計の構築，実践，および意義～

2013 年　1 月

誉田　直美

目　　次

第1章　序論

1.1　本研究の背景

「ソフトウェア危機」とは，1970 年代にソフトウェア生産の需給のギャップを示す言葉だった．おそらく現代は，第 2 のソフトウェア危機に瀕しようとしている．社会を見渡すと，いまやほとんどすべての製品にソフトウェアが使われている．テレビや冷蔵庫などの家電製品のみならず，電車，飛行機，銀行などの社会基盤を支えるシステムの信頼性を担うのはソフトウェアである．便利な機能や魅力的な機能はソフトウェアで実現するのが当り前になった．ソフトウェアなしでは，我々の社会は成立しないと言ってよい．しかし，企業はその責任と期待に応えるだけの品質を確保したソフトウェアを提供できているだろうか．

「ソフトウェア」という用語が世界で初めて使われたのは 1958 年，「ソフトウェアエンジニアリング」が提唱されたのは，その後 10 年経過した 1968 年のことである．日本で初めて「ソフトウェア」という用語が新聞に登場したのは 1969 年で，朝日新聞の記事が最初であるという[1-1]．すなわち，2013 年の今年は，ソフトウェアが誕生してから 55 年目，ソフトウェアエンジニアリングが生まれてから 45 年目であり，日本で初めて「ソフトウェア」という用語が新聞に登場してから 44 年しか経過していない．このわずか 50 年足らずの間に起こったソフトウェアの社会浸透は驚異的である．

これほどまでの短期間に，ソフトウェアの重要性が増大したのに対応して，ソフトウェアの品質は向上しただろうか．開発途中において失敗プロジェクトの予兆を見逃して出荷時期に影響する事態や，出荷後にトラブルが多発する事態は，依然発生している．結果として，経済活動へ影響を及ぼした例は枚挙にいとまがない．今や企業にとって，ソフトウェアの品質向上はビジネス上だけでなく社会的責任として必須の重要課題である．

ソフトウェアの品質確保が難しい原因は大きく 2 つ考えられる．ソフトウェアはその誕生後，50 年足らずの短期間に社会を席巻してしまったために，産業界に高品質ソフトウェアを提供できるだけの仕組みの整備が追いついていないのである．例えば自動車は，一般消費者が購入できるようになるまでに 100 年以上の年月がかかった．一方，ソフトウェアは，50 年程度の期間で社会のあらゆる場面で使われるようになった．ソフトウェアは単独で動くわけではなく，必ずハードウェアが必要である．そのハードウェアの驚異的な価格性能比の向上により，広く一般のユーザが利用できるようになり，同時にソフトウェアが一気に広まったのである．不幸なことに，ソフトウェアの品質確保の難しさは社会に理解

されていない．ソフトウェアは容易に作成し修正することができるものというのが一般社会の認識である．

2 つ目は，ソフトウェアそのものがもつ特性によるものである．ソフトウェアは目に見えない，論理の塊であるために複雑さが他の比でないなど，ハードウェアと比べて大きく異なる特性をもつ．この類がない特性に，まだ完全に対応する技術が揃っていない．それが，高品質ソフトウェア開発を大きく阻んでいると考えられる．

「第 2 のソフトウェア危機」とは，利用者にその自覚がないまま，品質確保が不十分なソフトウェアに社会が依存していることをいう．それは，ソフトウェアが社会のあらゆる場面に使われ，社会基盤を支えるシステムの信頼性を担っているにもかかわらず，提供側企業がソフトウェア品質を確保するのに苦慮しているのに加えて，ソフトウェア品質確保の難しさが社会に理解されていない状態を指す．ソフトウェアの社会浸透だけが急速に進み，技術や産業としての整備が追いつかない．しかも，ソフトウェアを取り巻く現代社会の状況は，既に待ったなしの状況まできている．

1.2　本研究の目的

本著者は，NEC で最初に設立されたソフトウェア開発部門という歴史をもつ組織に配属され，以来，20 年以上その組織の開発するソフトウェア製品の品質保証を担当してきた．当該組織は，現在約 5000 人のソフトウェア技術者を擁しており，本著者はその 5000 人の技術者が開発したソフトウェア製品の品質保証の責任を持つ．当該組織は，自らの出荷後バグ[1]が多いという問題を解決するために，「ソフトウェア品質会計」という品質管理技法を考案し，20 年以上かけて構築・適用してきた．さらに，品質会計の効果を引き出すために，品質会計を取り囲む仕組みを整備してきた．これらを現場に適用することによって，当該組織の出荷後バグ数を 1/20 以下に低減し，継続して維持するなどの成果をあげた．本著者はソフトウェア品質会計の構築および適用に中心的に関わってきた．

ソフトウェア品質会計の基本的な考え方やそれを取り囲む仕組みのうち，幾つかの重要な項目は，奇しくも，他の幾つかの日本企業が，1970 年代頃からソフトウェア品質問題に対して改善に取り組み，構築した仕組みと共通していた．ソフトウェア産業がソフトウェアの品質確保に苦慮しているとはいえ，効果が実証された幾つかのプラクティスが出てきているのである．

一方，ソフトウェア品質を向上するための技術はさまざまなものが提案されてきている．CMMI(Capability Maturity Model Integration：ソフトウェア能力成熟度モデル統合)[1-2]は，その代表的な技術である．本著者の所属する品質会計考案組織は，CMMI の最高レベルであるレベル 5 を 2000 年代初頭に達成しており，本著者はその CMMI 達成のリーダーを務めた．CMMI を提唱する米国 SEI(Software Engineering Institute)のレポートによる

[1] バグ：ソフトウェア内に潜在する欠陥や誤り（第 2 章において議論する）

と，2000 人以上の組織によるレベル 5 達成は，世界中で CMMI に挑戦する組織のうちわずか 0.6%しかない．

これらの経験と研究を踏まえて，本論文では，ソフトウェア品質会計技法および品質会計の効果を引き出す仕組みを論述する．さらに，ソフトウェア品質会計の工学的価値を論ずるとともに，高品質ソフトウェア開発の成功要因とその全体像を論述する．これにより，第 2 のソフトウェア危機に対して，高品質ソフトウェアを開発できる技術と仕組みを整備する道筋を示す．

1.3　本論文の構成

本論文は，本章を含め全 10 章から構成されている．

第 2 章では，高品質ソフトウェア開発を阻む課題を議論する．本論文の重要な用語である「品質」と「バグ」についてさまざまな研究や規格などによる定義を述べるとともに，品質とバグの関係について議論する．これらを踏まえて，品質マネジメントについて議論する．さらに，ソフトウェアの特性およびそれがソフトウェア産業に及ぼす課題を論ずる．同時に，高品質ソフトウェア開発を阻む課題を引き出すために，日本の CMMI レベル 5 組織への調査結果を基に，その課題を整理する．

第 3 章では，本著者が 20 年以上かけてその技法の構築と定義にかかわってきた「ソフトウェア品質会計」についてその技法の特徴と適用方法を論述する．ソフトウェア品質会計は，出荷後バグの多さに悩む組織が，自ら問題を解決するために考案した技法である．高品質ソフトウェア開発を実現する鍵になる技術と考える．

第 4 章では，ソフトウェア品質会計を構成する技法の 1 つである「バグ分析と 1+n 施策」の技法の特長と適用方法を論述する．「バグ分析と 1+n 施策」を個別に取り上げて説明する理由は，バグ分析が組織の改善活動に大きな影響を与える重要な技術と考えるためである．

第 5 章は，ソフトウェア品質会計を支える技術として，レビュー技術および幾つかの品質確保のプラクティスを論述する．ソフトウェア品質会計は，レビューを重視する品質管理技法であるため，レビュー技術は重要である．また，ソフトウェア品質会計は単独ですべての効果を出したわけではなく，品質会計の効果を引き出すために品質会計を取り囲む仕組みを整備してきたことが品質向上に大きく寄与している．その仕組みは，他の幾つかの日本企業が 1970 年代から実施してきたソフトウェア品質向上への取り組みの施策と，結果として同じ実装方法であった．そこに，高品質ソフトウェア開発を実現するための大きな鍵があると考える．

第 6 章は，ソフトウェア開発を支える開発方法論・ツール・開発環境を統合したシステムとしてのソフトウェアファクトリについて議論する．これは，ソフトウェアの品質・生産性向上の基盤となる．その最新の取り組みについて述べる．

第 7 章は，ソフトウェア品質会計の適用による品質向上の実例を 3 つ紹介する．これら 3

事例は，いずれも本著者がリーダーとして品質向上活動を推進した事例である．3 事例に共通するのは，ソフトウェア品質会計を徹底して適用したことと，品質会計を支える仕組みを同時に整備したことである．

第 8 章は，第 7 章までを踏まえて，ソフトウェア品質会計の工学的価値を論述する．さらに第 9 章では，高品質ソフトウェア開発の成功要因とその全体像について論述する．

最後に第 10 章では，本論文のまとめを行い，今後の課題について述べる．

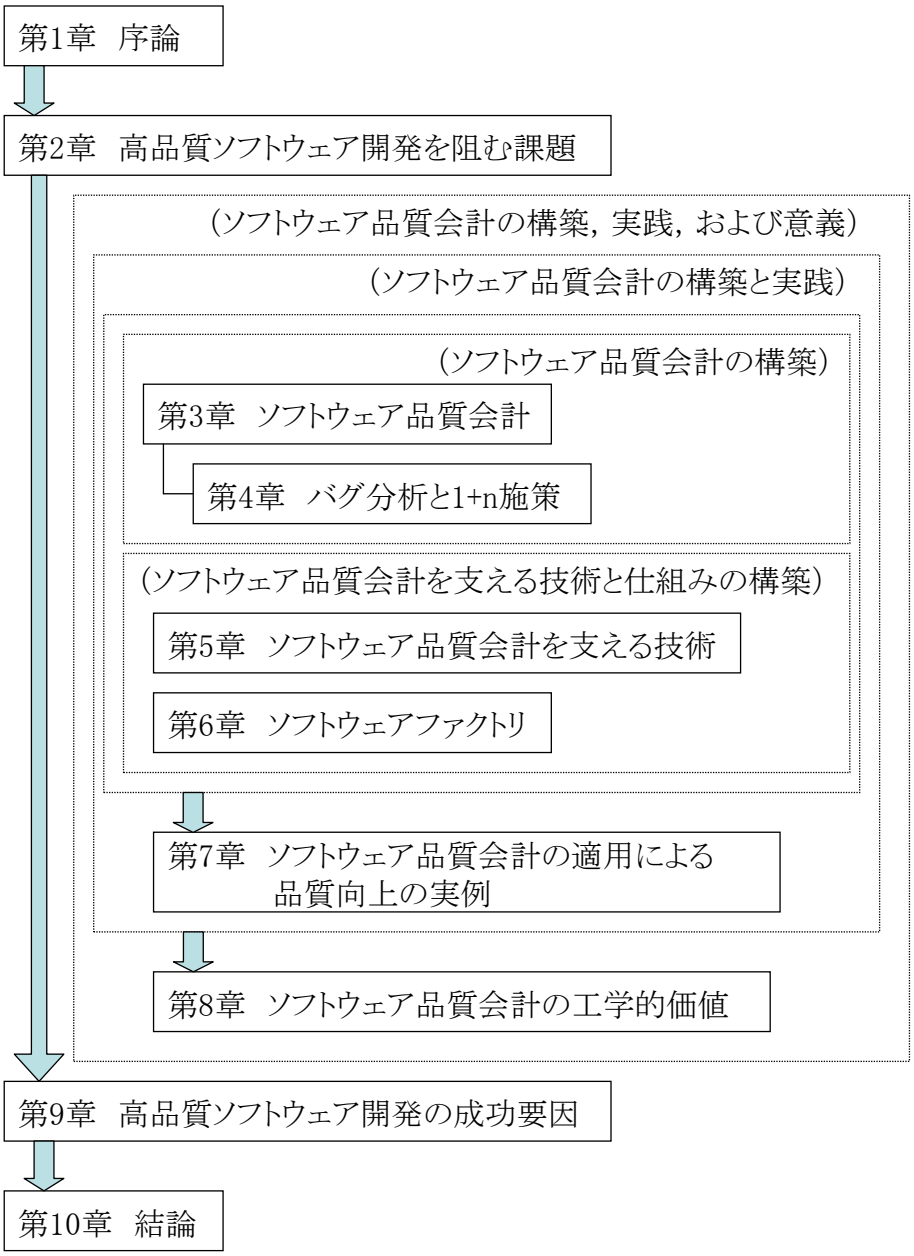


図 1-1　本論文の構成

第2章　高品質ソフトウェア開発を阻む課題

2.1　はじめに

本章では，高品質ソフトウェア開発を阻む課題を，ソフトウェア開発を囲むさまざまな観点から論述する．

はじめに，重要な用語である「品質」およびソフトウェア開発において重要な「バグ」という用語について，さまざまな規格や研究成果を紹介しながら，その定義を述べる．次に，品質マネジメントの定義とその管理対象について論じる．さらに，ソフトウェアの特性を説明し，ソフトウェアの難しさが，ソフトウェアのもつ本質的な難しさに起因するものと，本質的な難しさが引き起こす副次的な難しさに起因するものに分類できることを述べる．ソフトウェア産業の視点で，ソフトウェアの難しさが引き起こす高品質ソフトウェア開発を阻む課題についても解説する．現場における高品質ソフトウェア開発を阻む課題を導出するために，CMMI レベル 5 組織に対する調査結果に基づき，プロセスの能力成熟について考察するとともに，CMMI により解決できること・できないことを述べる．CMMI レベル 5 組織の調査結果を使用する理由は，ソフトウェアプロセスの機能欠落の影響がない条件のもとで，高品質ソフトウェア開発を阻む課題を抽出することが目的であるためである．

2.2　品質とは

「品質」というのは，実にさまざまな意味で使われる用語である．実際に発生した障害のように直接把握できるものを指す場合もあれば，魅力的というようなつかみどころが難しいものを指す場合，機能や性能面で明らかに優位にあることを指す場合などがある．「品質」は，規格や研究者によってさまざまな定義がなされており，統一した定義があるわけではない．ただし，品質の最終ゴールが「顧客の満足」であることは，世界的に合意が得られていると言ってよい．これは，ソフトウェア製品に限らず，ハードウェア製品やサービスなどすべての領域で共通した見解である．

本章では，規格や研究者による品質の定義を引用しながら，「品質」について議論する．表 2-1 にさまざまな品質の定義を示す．特に，一般的な品質とソフトウェア領域における品質との違いについて考察を加える．

表 2-1　さまざまな品質の定義

カテゴリ	規格/研究者	品質の定義内容	備考
広く産業全般として品質を定義	ISO9000	本来備わっている特性の集まりが，要求事項を満たす程度	ISO9000で定義する要求事項とは，「明示されている，通常，暗黙のうちに了解されているもしくは義務として要求されている，ニーズまたは期待」をいう.
	Joseph M. Juran	・プロダクトの特性が顧客のニーズに応えることで満足を提供する. ・不備（障害や誤り）から免れる.	左記2つの観点（下記）から品質を定義している.
	Philip B. Crosby	要求に対する適合	
ソフトウェア領域での品質の定義	Gerald M. Weinberg	品質は誰かにとっての価値である.	ユーザ，システム管理者，プロジェクト管理者など，立場によって「価値」は異なる（品質の相対性）としている.
	James Martin	システムが本稼動するとき，どこまで真のビジネス(ユーザ)ニーズに合っているかということ	開発期間の短縮化と，時間とともに変化する顧客要求への対応に注目している点が特徴で，この考え方に基づき，RADの必要性を説いた.
	Roger S. Pressman	機能および性能に関する明示的な要求事項，明確に文書化された開発標準，および職業的に開発が行われた全てのソフトに期待される暗黙の特性に対する適合	開発標準が守られない場合は，ほぼ確実に品質の欠如が発生すると説明している.
	ISO/IEC25000	指定された特定の条件で利用する場合の，明示的または暗示的なニーズを満たすソフトウェア製品の能力	
	IEEE Std 610	システム，コンポーネント，またはプロセスが顧客またはユーザのニーズまたは期待を満たしている度合い	システム，コンポーネントのソフトウェア製品に関する品質だけでなく，プロセスの品質にも言及している.
日本における品質の考え方	石川薫	・狭義の品質：製品の品質 ・広義の品質：仕事の質，サービスの質，情報の質，工程の質，部門の質，人の質，システムの質，会社の質等，全てを含めた「質」をとらえるべき	品質を「質」としてとらえ，狭義・広義という視点で以下のように説明している. 広義の質を管理するのが品質管理の基本姿勢とし，日本的品質管理へ大きな影響を与えた.
	狩野　紀昭	・当たり前品質要素：それが充足されれば当たり前と受け取られるが，不十分であれば不満を引き起こす品質要素 ・一元的品質要素：それが充足されれば満足，不充分であれば不満を引き起こす品質要素 ・魅力的品質要素：それが充足されれば満足を与えるが，不充分であってもしかたがないと受け取られる品質要素	「当たり前品質」，「魅力的品質」という視点で品質を説明している.
	飯塚　悦功	ニーズに関わる対象の特徴の全体像	「ニーズに関わる」という表現で，受け取り手が抱くニーズの特性を対象とすることを示し，品質が受け取り手という外部の価値基準によって決まると説明している.

2.2.1　明示的なニーズと暗示的なニーズ

品質マネジメントシステムの国際規格 ISO9000 シリーズ（ISO9000　品質マネジメントシステム　―基本及び用語）では，品質を「本来備わっている特性の集まりが，要求事項を満たす程度」と定義している[2-1]．さらに，ISO9000 では，「品質」の定義の後に，要求事項について「明示されている，通常，暗黙のうちに了解されている若しくは義務として要求されている，ニーズまたは期待」と定義している．その要求事項には，明示的なニーズと暗示的なニーズがあることを示している．これら両方の定義をあわせると，品質とは明示的なニーズと暗示的なニーズの両者を満足している程度ということになる．暗示的なニーズとは，そのニーズが当事者にとって慣習または慣行であることをいい，「使いやすさ」などはまさに暗示的なニーズに該当する．なぜなら，機能的な満足は製品として必須であるとともに，それが使いやすいことは暗に当り前の条件として要求されているためである．

ISO9001 は，ソフトウェアに限らず製品やサービスを提供するすべての組織を対象とした規格である．したがって，明示的だけでなく暗示的なニーズも考慮するというのは，ハードウェアなども含む製品やサービスの提供において共通した「常識」と言える．一方，ソフトウェア開発の現場ではどうだろうか．暗示的なニーズを考慮することが現場で当り前になっているとは言えない状況である．そのような段階にあるからこそ，暗示的なニーズを読み取り考慮することは，品質の良いソフトウェアとして重要と考える．

2.2.2　プロセスで作り込む

Roger S. Pressman は著書「実践ソフトウェアエンジニアリング」のなかで，品質について「機能および性能に関する明示的な要求事項，明確に文書化された開発標準，および職業的に開発が行われた全てのソフトウェアに期待される暗黙の特性に対する適合」と定義している[2-2].

ここで注目してほしいのは，「明確に文書化された開発標準」について言及している点である．Pressman は，上記の品質の定義に続き，「開発標準が守られない場合は，ほぼ確実に品質の欠如が発生する」と述べている．

同様の指摘は，IEEE Std 610 という国際規格にも見られる．IEEE Std 610 では，品質を「システム，コンポーネント，またはプロセスが顧客またはユーザのニーズまたは期待を満たしている度合い」と定義している．単に「システム，コンポーネント」だけでなく，「プロセス」にも言及している点に注目すべきである．つまり，プロセスがユーザのニーズを満たしていることが，品質に大きくかかわると述べているわけである．IEEE Std 610 はソフトウェアエンジニアリングの用語定義集であり，上記の定義はソフトウェア領域を意識した定義と言える．

このように，開発標準などのきちんとしたプロセスに則って開発されなかったソフトウェアの品質に欠如が発生するということは，国際規格で定義されるほど，世界の共通認識となっている．ソフトウェア開発の現場では，「出来あがったモノさえ良ければどんな作り方をしてもいい」という主張を聞くことがあるが，これは間違いである．品質はプロセスで作り込むという考え方は，ソフトウェアにおいても「常識」である．

2.2.3　ソフトウェアに軸足を置いた定義

Gerald M. Weinberg は，品質を「誰かにとっての価値である」と定義している[2-3]．これは，立場によって「価値」が異なることを意味している．ソフトウェアを直接使うユーザ，その上司，運用者など，１つのソフトウェアに対する関係者は複数存在する．その各々の人たちにとって，「価値」は異なるという意味である．これを品質の相対性と呼ぶ．例えば，そのソフトウェアを直接使うユーザはマニュアルを読まなくても直感的に使えることを望む．しかし，その上司は出力結果のわかりやすさを，また，運用者は運用に手間がかからないことを望む．このように，そのソフトウェアに関与する人々を層別して，各々の層の「価値」を考えることは非常に重要である．

RAD(Rapid Application Development)の提唱者である James Martin の品質の定義は，「システムが本稼動するとき，どこまで真のビジネス(ユーザ）ニーズにあっているかということ」[2-4]であり，「時間」の要素を強く意識している点に特徴がある．IT 業界の変化の速さは，今さら解説するまでもない．その変化の速さを意識するからこそ，時間の経過とともに変化するニーズに対して，タイムリーに対応できることの価値を指摘しているわけである．わかりやすく言えば，決めた期日に確実にソフトウェアを提供できること自体に大きな価値があるということである．その期日だからこそ価値がある．品質・生産性の良いソフトウェア開発が課題である現状では，期日どおりに所定の要求を満足するソフトウェアを開発できることこそ品質であると言うべきかもしれない．

2.2.4　日本における品質の考え方

日本の品質に対する考え方の大きな特徴は「顧客指向」という点である．これは，製品の特性から品質を説明しようという欧米の考え方に対して大きな影響を与えたと言われている．

日本的品質管理に大きな影響を与えた石川馨は，品質を「質」としてとらえた点に特徴がある[2-5]．製品の品質を狭義の品質と定義し，仕事の質，部門の質，人の質など製品に影響を与えるすべての質を広義の品質と定義して，広義の品質を管理するのが品質管理の基本姿勢であるとした．

狩野紀昭は，「魅力的品質」，「当り前品質」，「一元的品質」という視点で品質を説明した

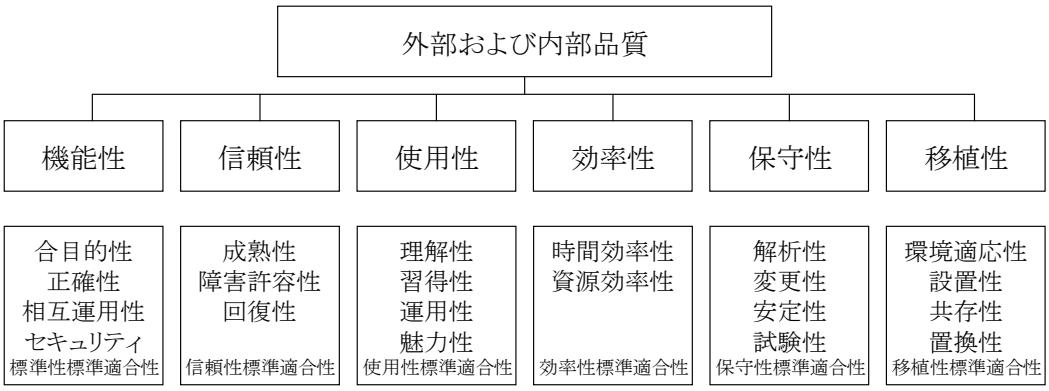
[2-6]．これは，顧客の心理的満足感と製品・サービスの性質がもたらす物理的充足状況から説明した概念である．「魅力的品質」とは，ある物理的な性質が充足されれば心理的満足感を与えるが，物理的な特性が不充分であっても仕方がないと受け取られる特性，「当り前品質」とは，物理的な性質が満たされていれば当り前と受け取られるが，物理的な性質が不十分であれば不満を引き起こす特性，これに対して一元的品質とは，物理的な性質が充足されれば満足，不充分であれば不満を引き起こす特性であり，物理的従属状況と心理的満足感が比例関係にあるような特性である．当り前品質が欠落するとクレームにつながる．魅力的品質の優れた製品・サービスはヒット製品につながる．

飯塚悦功は，品質を「ニーズに関わる対象の特徴の全体像」と定義している[2-7]．「ニーズに関わる」という表現で，提供する製品・サービスの受け取り手である顧客が抱くニーズの特性を対象とすることを示した．さらに，品質は，この受け取り手という外部の価値基準によって決まると説明している．製品・サービスの提供側ではなく，提供側にとって外部の顧客の価値基準で決まることは，品質を考える上で非常に重要である．

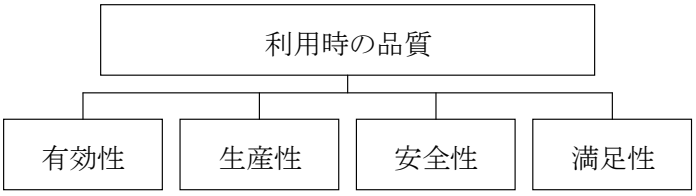
日本の特徴である「顧客指向」の考え方は，非常に重要である．企業の持続的発展のためには，顧客による支持が不可欠だからである．企業が提供する製品やサービスを，顧客が満足して購入するからこそ，企業はその対価を得ることが出来る．それを再投資して，企業は新しい製品・サービスを生みだす．この好循環を繰り返すことによって，企業は持続的発展を実現する．企業が長期的かつ安定的に存続するには，幅広い顧客に長期的に満足してもらうことが必須である．このように，顧客指向は，本質的に長期的な姿勢が必要と考える．

2.2.5　品質特性による定義

国際規格である ISO/IEC9126 では，図 2-1 に示すソフトウェアの品質特性を定義している[2-8]．外部および内部品質のための品質モデルと，利用時の品質のための品質モデルの 2 種類から構成されている．この品質モデルの考え方は，上述した品質の定義とは異なり，ソフトウェアを解析的にとらえたものである．外部および内部品質のための品質モデルでは，外部と内部品質を，機能性，信頼性，使用性，効率性，保守性，および移植性という 6 つの品質特性に分解して説明している．ソフトウェア開発では，機能性や信頼性が主体になりがちだが，その他の品質特性も同様に考慮することが重要である．使用性や効率性は顧客から直接求められる品質特性のため，見逃されることは少ないが，特に新規開発において保守性や移植性は軽視されがちである．改造が主体で，新規ソフトウェア開発の比重が軽くなった現在，ソフトウェアライフサイクルを長期的な視野でとらえ，新規開発時点から保守性や移植性を考慮する姿勢は必須である．保守性や移植性は，短期的には効果を実感しにくいが，ソフトウェアライフサイクルが長期化すればするほど効果が現れる特性である．



＜外部および内部品質のための品質モデル＞



＜利用時の品質のための品質モデル＞

図 2-1　ソフトウェアの品質特性[ISO/IEC 9126 (JIS X 0129)]

2.2.6　まとめ

品質にはさまざまな定義があることを確認した．本論文では ISO9000 の定義である「本来備わっている特性の集まりが，要求事項を満たす程度」を品質の定義として使用する．企業において品質を考えるとき，「要求事項」と「本来備わるべき特性」という対比の考え方が重要と思うからである．加えて，「本来備わるべき特性」という表現は，暗に「実際に実現している特性」との対比を示唆している．本来備わるべき特性と実際に実現している特性との差異を認識することは，製品・サービスの提供側である企業にとって非常に重要なことである．なぜなら，ソフトウェアにおいて「本来備わるべき特性」と「実際に実現している特性」に差があるのが実態だからである．また，「要求事項」とは，提供する製品・サービスの受け取り手である顧客が抱く特性であることに注意しなければならない．要求事項は，提供側である企業ではなく，受け取り手という外部の価値基準によって決まるのである．すなわち，品質を確保する目的は，顧客満足を得ることに尽きる．長期的かつ安定的に顧客満足を得ることにより，企業は持続的発展を実現するのである．

ソフトウェア領域において品質を確保するポイントは，表 2-2 にあげるとおりである．特に，的確な要求事項の把握とプロセスで品質を作り込むことが重要である．要求事項の把

握では，暗示的ニーズを考慮すること，利用者を層別して層毎のニーズを考慮すること，ソフトウェアライフサイクルを意識して 6 つの品質特性を考慮することなどが重要である．プロセスで品質を作り込むという面では，期日通りにソフトウェアを提供できるプロセスに価値があることと，そのプロセスを確実に守り，プロセスで品質を作り込むことが重要である．

表 2-2　本論文における品質の定義とソフトウェア品質を確保するポイント

品質の定義	本来備わっている特性の集まりが，要求事項を満たす程度[ISO9000の定義を引用]． 品質の最終ゴールは，顧客満足である．
ソフトウェア品質を確保するポイント	・明示的ニーズだけでなく，暗示的ニーズを考慮する． ・開発標準などのプロセスを確実に守る（守られない場合は，ほぼ確実に品質の欠如が発生する）． ・立場によって関係者を層別し，層毎のニーズを考慮する． ・決めた期日にソフトウェアを提供できること自体が価値である． ・機能性，信頼性，使用性，効率性，保守性，および移植性という6つの品質特性を考慮する．

2.3　バグとは

ソフトウェア開発の現場では，ソフトウェアが予定したとおりに動作しないことを，「バグ」「不具合」「欠陥」「ディフェクト」などと呼ぶ．本章では，これらの用語について定義する．

2.3.1　バグの定義

JIS X 0014（情報処理用語－信頼性，保守性及び可用性）[2-9]では，バグに関連する用語を表 2-3 のように定義している．システムダウンなど人間が認識できる事象は「故障（Failure)」であり，故障の原因が「障害（Fault)」である．一方，「誤差・誤り（Error)」は理論値と計測値の差異をいう．「バグ」は，一般には障害（Fault）の意味で使用される．JIS X 0014 の故障や障害という日本語の用語は，現場の実感と合わないためになかなか定着しないのが実情である．例えば，障害という用語は，JIS X 0014 で定義する「故障」を想起させてしまうのである．こうした背景から，本論文では，現場で広く一般的に使用されている「バグ」という用語を，JIS X 0014 で定義する「障害（Fault)」の意味で使用する．すなわち，本論文において，「バグ」という用語は，「要求された機能を遂行する機能単位の能力の，縮退または喪失を引き起こす，異常な状態」（JIS X 0014:1999 で定義する「障害（Fault)」と同等）という意味で使用する．

表 2-3　さまざまなバグ関連用語の定義

用語	定義内容(JIS X 0014)	備考
誤差・誤り (Error)	計算，観測もしくは測定された値または状態と，真の，指定されたもしくは理論的に正しい値または状態との間の相違	
障害 (Fault)	要求された機能を遂行する機能単位の能力の，縮退または喪失を引き起こす，異常な状態	バグは，一般にはFaultの意味で使用される．不具合，欠陥，ディフェクトも同様である．
故障 (Failure)	要求された機能を遂行する，機能単位の能力がなくなること	トラブルと呼ぶ現象は，一般にはFailureの意味で使用される．

2.3.2　開発途中のバグの定義

ソフトウェア開発中におけるバグについて議論する．開発途中において，バグの定義は，実は自然に拡張して使用されていることが多い．実際に機能単位の能力の縮退または喪失を引き起こすような原因でなくとも，修正が必要と判断した場合はバグとして扱い，修正するのである．その典型的な例が，コーディング規則違反である．コーディング規則違反は，実際にプログラムの動作に影響しない場合であっても修正する．これはそのソフトウェアの保守性に関わるためである．仕様，規約や標準などとの差異も同様にバグと扱う．これらは，必ずしも実際に機能単位の能力の縮退または喪失を引き起こすとは限らない．このように実際に機能単位の能力の縮退または喪失を引き起こすような原因でなくとも，修正が必要と判断した原因をアノマリー（Anomaly）と呼ぶ[2-10]．開発途中においてはアノマリーをバグに含むこととする．

PSP（Personal Software. Process）では，「バグはプログラムの変更を行うたびに数える」と定義している[2-11]．この定義は，「変更が必要な箇所はバグとする」と読み替えることが出来る．変更が必要な箇所には，変更が必要となる理由があり，それをバグとするという意味である．変更が必要な理由には，実際に機能単位の能力の縮退または喪失を引き起こす場合と，必ずしも実際に機能単位の能力の縮退または喪失を引き起こすわけではない場合（アノマリー）があるということである．

設計段階のドキュメントのバグは，さらに現場を意識した詳細な定義が必要である．設計仕様書には，開発者の技術レベルに応じた適切な仕様記述の粒度が求められる．技術レベルが低ければ，詳細な説明が必要である．すなわち，関係者間の暗黙知の範囲を適切に設定して，仕様記述しなければならない．また，仕様記述は文章表現が避けられないために，文章表現の巧拙が仕様理解に大きく影響する．関係者全員が正しく理解できるような文章表現が求められる．

表 2-4 に設計およびコーディング段階のバグの定義例を示す．これは，アノマリーをバグとして含むことを前提として定義したものである．

表 2-4　設計およびコーディング段階のバグの定義例

設計仕様書のバグ	-標準に沿っていないものはバグとする. -前工程の仕様書に沿っていないものはバグとする. -記述がわかりにくいため，ほかの担当者が誤解する可能性の高いものはバグとする. 誤解する可能性の低いものはバグとしない. -他グループへの確認不足によるものはバグとする. -記述されていない部分はバグとする. ただし，誤字脱字（てにをは）などの記述ミスはバグとしない. -その仕様書の入力となった前工程の仕様書に問題があったためによるバグは，前工程のバグとする. -他グループの仕様書のバグによるバグは，他グループのバグとする.
プログラムのバグ	-コーディング規則に沿っていないものはバグとする. -コーディングミスはバグとする.
備考	重複バグは，全体で1件とする.

2.3.3　プロセスモデルとバグの関係

表 2-4 は，実は暗にウォーターフォールモデルを想定してバグを定義した例である．開発途中のバグの定義は，プロセスモデルとの関係を意識することが求められる．それは，バグと判断する時期を決定する必要があるためである．例えば，設計仕様書は，開発者が設計し，設計仕様書として表現されるものである．このうち，どの段階からバグと判断するかというのが本節での議論である．

図 2-2 に，表 2-4 で想定した開発プロセスを示す．これは V 字モデルである．V 字モデルとは，ウォーターフォールモデルのうち，対応する設計とテストを視覚的に同位置に表記することによって，設計とテストの対応関係を明確にしたモデルをいう[2-12]．V 字モデルは，対応する設計とテストを明確にすることによって，その設計内容をテストする段階を明らかにするという意味がある．

図 2-2 に示す V 字モデルは，設計およびコーディング段階の各工程において，常に設計（コーディング工程ではコーディング）とレビューがサブ工程として定義されていることに注意していただきたい．当該設計工程で作成した設計仕様書を，必ず同じ設計工程のレビューでレビューすることを前提としている．表 2-4 における設計仕様書のバグは，当該設計工程において設計が終了し，レビューを開始した段階から，バグとして認識されたものをバグとすると定義している．

この定義の考え方は，設計完了時以降からバグと判断するという意味である．設計完了時をどこに設定するかが考え方のポイントであり，それは適用している開発プロセスに応じて設定する必要がある．

開発途中の具体的なバグの定義や，バグと判断する時期は，ソフトウェア開発組織が各組織の特性に合わせて定義すべき内容である．バグを具体的に定義するためには，開発プ

ロセス，設計仕様書の記載方法，コーディング規則などの詳細な定義が求められる．これらを現場に利用可能なレベルで具体的に定義することは，実は高成熟した組織でなければ困難である．高成熟した組織は少なく，ほとんどの組織はバグの定義の段階で悩みを抱えている．これはソフトウェアの歴史の浅さに起因するソフトウェア業界の課題の１つと考える．

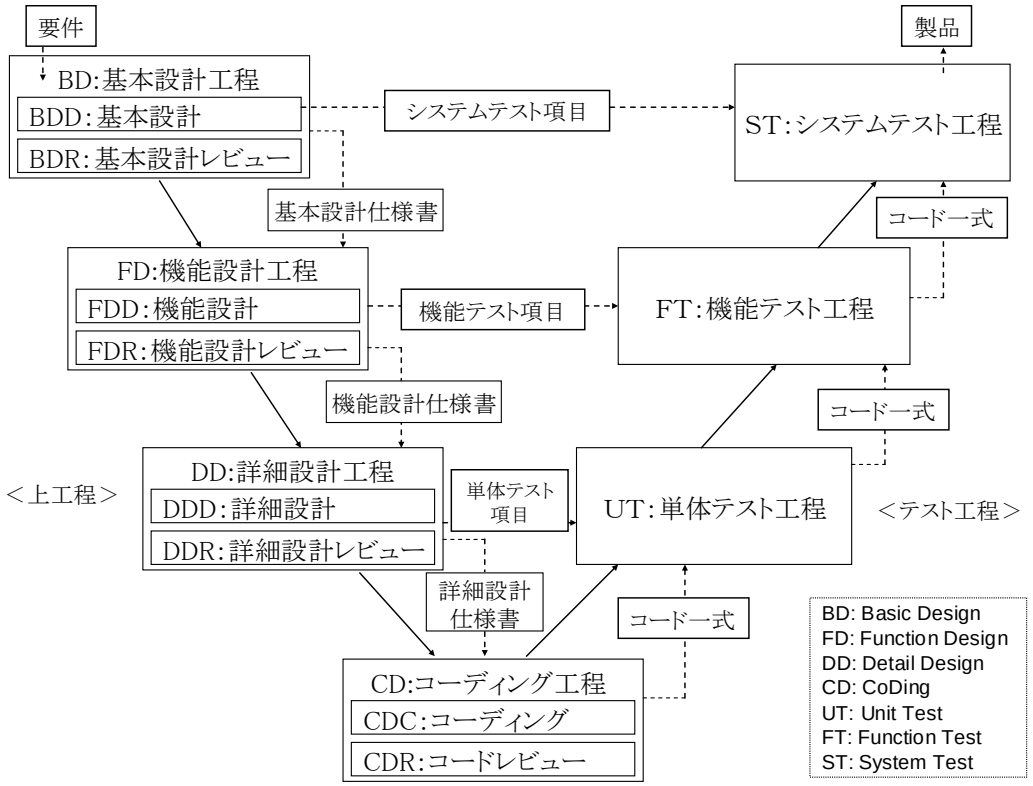


図 2-2　本論文で想定する開発プロセス（V 字モデル）

2.3.4　出荷後のバグ

出荷後におけるバグについて議論する．そもそも出荷とは何を指すかについて定義する必要がある．パッケージソフトウェアや組込みソフトウェアであれば，明示的に出荷日を設定し，出荷日以降に顧客が購入可能となるため，出荷の認識が比較的容易である．顧客と個別契約を交わし，当該顧客向けに固有のソフトウェア開発する場合に出荷の判断が難しいことがある．その理由は，ソフトウェアの内容に応じて顧客と共同でシステムテストを実施する場合や，顧客側の準備のために開発途中のソフトウェアを先出しするような場合など，さまざまな出荷形態があるためである．これらは個別に定義するしかない．１つの判断方法としては，開発組織で開催する出荷判定会議を区切りとして出荷前後を判断する方法である．これは，ソフトウェアの領域に関わらず出荷を判断できる方法である．

出荷後のバグは，当該バグが発生した場所によって分類できる（図 2-3 参照）．顧客にて発生したバグと開発組織にて摘出したバグの 2 種類である．顧客にて発生したバグには，アノマリーは含まないが，出荷後に開発組織にて摘出したバグにはアノマリーを含むことがある．出荷後に開発組織にてバグを摘出する契機となるのは，当該ソフトウェアの改造や，当該ソフトウェアにおいて顧客にてバグが発生したため対応が必要となる場合が考えられる．特に，当該ソフトウェアの改造の場合には，今回の改造内容や将来の改造に備えて，アノマリーを含めた修正を行うことがある．

図 2-3 の I の開発中のバグの件数は，ソフトウェアの開発規模に比例して増加する傾向がある．一方，Ⅲの顧客にて発生したバグの件数は，ソフトウェアの開発規模には比例しない．顧客の当該ソフトウェア利用状況や，汎用製品では顧客数などの影響を受けることが多い．例えば，当該ソフトウェアの利用環境が当該ソフトウェアの性能限界に近い状況で使用されるなど，さまざまな異常常態が発生しやすい利用環境の場合には，顧客でのバグは発生しやすくなる傾向がある．Ⅱの出荷後に開発組織によって摘出されたバグは，当該ソフトウェアの顧客数や改造予定の有無などの影響を受けることが多いと考えられるが，各々のケースにより事情が個別的であり，傾向は見られない．

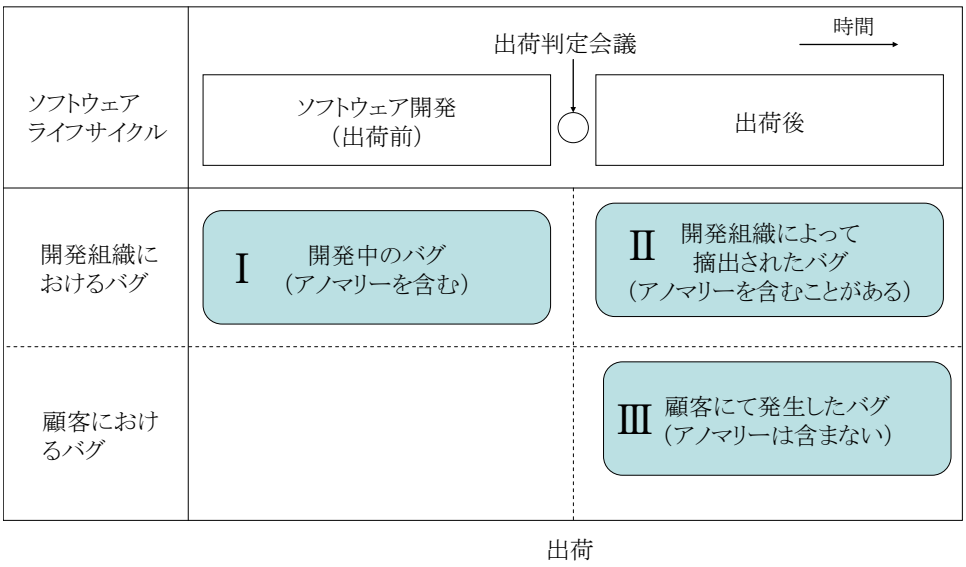


図 2-3　ソフトウェアライフサイクルとバグ

図 2-4 は，開発規模に対する I の開発中のバグとⅢの顧客にて発生したバグの散布図である．横軸の開発規模は左右とも同じ尺度で表示しているが，縦軸のバグ数は出荷前後で大きく異なる数値であるため，左右の尺度は異なる．開発規模と I の開発中のバグは相関係

数 0.887 と明らかに相関関係が認められるが，開発規模とⅢの顧客にて発生したバグは相関係数 0.149 であり，両者に相関関係がないことがわかる．

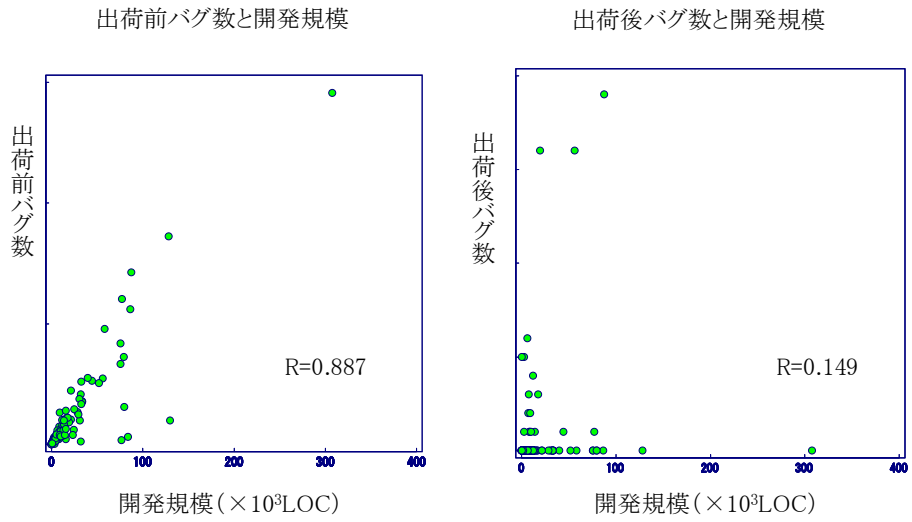


図 2-4　開発規模に対するⅠの開発中のバグとⅢの顧客にて発生したバグの関係

2.3.5　品質とバグの関係

品質とバグの関係について議論する．

2.2 において，品質を「本来備わっている特性の集まりが，要求事項を満たす程度」と定義した．この定義のキーワードは，「要求事項」，「本来備わるべき特性」，「実際に実現した特性」である．ソフトウェアにおいては，「要求事項」とは提供する製品・サービスの受け取り手である顧客が抱く特性，「本来備わるべき特性」とは開発者により設計された特性，「実際に実現した特性」とは当該ソフトウェアが実現した特性と考えることができる．

一方，バグは「要求された機能を遂行する機能単位の能力の，縮退または喪失を引き起こす，異常な状態」と定義した．この定義における「要求された機能」とは，狭義の意味では開発者が設計した特性である「本来備わるべき特性」であるが，本質的にはこれに加えて顧客の抱く「要求事項」を含めるべきである．一方，顧客が認識するのは「実際に実現した特性」である．

図 2-5 を参照すると，「要求事項」に対して「本来備わるべき特性」の合致する程度が品質である．「本来備わるべき特性」に対して「実際に実現した特性」の非合致箇所は，明らかにバグである．では，「要求事項」と「本来備わるべき特性」の非合致箇所はバグだろうか．顧客の抱く「要求事項」はバグに含めるべきと考える．すなわち，バグとは，単に設

計通りに動作しないことを指すだけでなく，本来あるべき特性を実現していない場合を含む．例えば，当該ソフトウェアの想定する利用環境において，主要なユースケースを満足していない設計は，本来あるべき機能の欠落であるためバグと判断するべきである．また，程度によるものの，使いにくい設計は，暗示的なニーズに対する欠落であるためバグと判断するべきである．

一方，「本来備わるべき特性」が，ある特性において「要求事項」を大きく超えるような場合が魅力的品質であり，このようなとき，いわゆるヒット商品が生まれるものと考える．

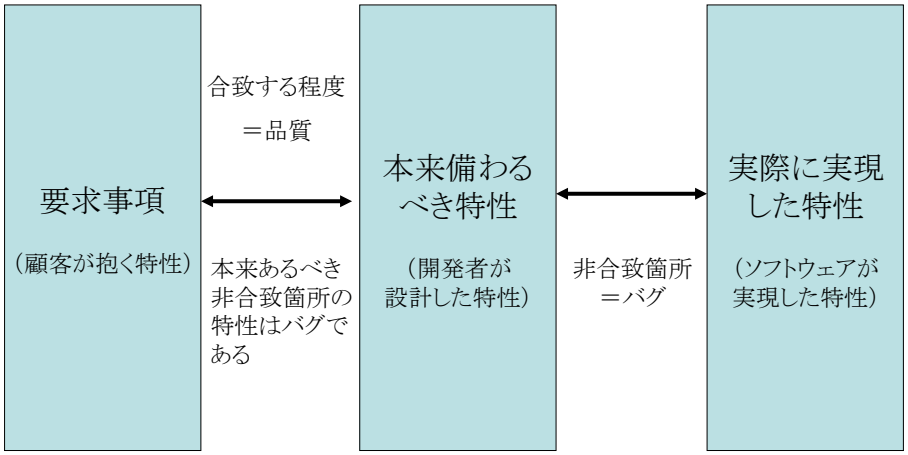


図 2-5　品質とバグの関係

2.3.6　まとめ

バグについて，さまざまな視点から議論した．バグの定義，開発途中のバグ，開発プロセスとバグの関係，出荷前後とバグの関係を考察した．さらに，品質とバグの関係について論じた．本論文では，バグの定義である「要求された機能を遂行する機能単位の能力の，縮退または喪失を引き起こす，異常な状態」の「要求された機能」を，「本来あるべき要求された機能」ととらえる．その意図は，通常想定するバグの対象範囲よりやや広く，本来あるべき特性を含めてバグの対象範囲と定義するという意味である．その上で本論文では，「品質」を「ソフトウェアの出荷後のバグの少なさ」に絞って，以降の議論を進める．出荷後のバグの少ないソフトウェア開発の実現は，ソフトウェア産業の根幹をなすとともに，ソフトウェアの重要性が増加した現在において提供側の社会的責任でもあるからである．

競争力のある魅力的品質の追求は，出荷後のバグが少ないソフトウェア開発を基盤として実現するものと考える．

2.4　品質マネジメントとは

2.4.1　品質マネジメントの定義

ISO9000 では「品質マネジメント」を，「品質に関して組織を指揮し，管理するための調整された活動」と定義している[2-1]．2.2 節において，企業が長期的かつ安定的に存続するには，幅広い顧客に長期的に満足してもらうことが必須であると述べた．この「幅広い顧客に長期的に満足していただく」ためのマネジメントが，品質マネジメントである．

マネジメントの 3 要素は QCD（Q:Quality，C:Cost，D:Delivery）と言われるが，この QCD の文脈での Q は非常に狭い範囲の品質を指している．一方，「品質マネジメント」でいう品質は，C や D を包含する非常に広い概念での品質を意味する．2.2 では，品質の良し悪しは，製品・サービスの受け取り手である外部基準によって決まると述べた．言い換えれば，外部基準に適合するという目的のために，品質マネジメントを含むすべての行動がなされるべきであるということが示唆されている[2-7]．その意味でも，品質は C や D を包含する非常に広い概念でとらえるべきである．

2.4.2　品質マネジメントの管理対象

品質マネジメントの管理対象は 2 つあると言われている．結果系に焦点をあてる考え方と要因系に焦点をあてる考え方である[2-12]．結果系に焦点をあてる考え方とは，プロセスのアウトプットが基準を満足しているかどうかを確認することにより，悪いものは外に出さないとする考え方である．この典型的な手法が検査である．検査とは，何らかの方法で試験・測定した結果を，あらかじめ設定した合否判定基準に照らして合否判定することである[2-12]．ソフトウェア開発の場合には，各工程での成果物（設計仕様書やテスト仕様書など）も結果系の対象である．検査のメリットは，確実に不適合品を排除できることである．しかし，結果として不適合品が多くなると収益を圧迫してしまう．

一方，要因系に焦点を当てる考え方とは，プロセス重視の考え方であり，初めから品質のよいソフトウェアを作り出すプロセス作りを基本とする．初めから良いものを作り出すプロセス作りを重視し，構築したプロセスに沿ってものを作ることを基本とする．したがって，常にプロセスに沿ってもの作りされているかをチェックすることが重要になる．プロセスを重視した品質のマネジメントのメリットは，良いものを作る確率を高めることができることである．しかし，不適合品をゼロにするのは困難である．どれほど低い確率であ

っても，不適合品が作られてしまう可能性を皆無にはできない．

結果系に焦点をあてる考え方によって確認する品質をプロダクト品質，要因系に焦点をあてる考え方によって確認する品質をプロセス品質と呼ぶ．このプロダクト品質とプロセス品質は，相互に関連しており，両者を融合してマネジメントすることが重要と言われている[2-12]．両者ともメリットとデメリットがある．それをうまく組み合わせて，プロダクト品質とプロセス品質の両面からバランスよく品質マネジメントすることが重要である．

2.4.3　組織とプロジェクトの視点の違い

ソフトウェア開発は，プロジェクト体制により開発されることがほとんどである．プロジェクトとは，「独自の成果物またはサービスを創出するための有期活動」[2-13]と定義される．この「独自」と「有期活動」という点に注目していただきたい．ソフトウェア開発は，常に一品ものの独自の成果物を作り上げる活動である．量産は単なるソフトウェアのコピーでしかない．ほとんどのソフトウェア開発はプロジェクト体制で行われるため，有期活動である．ある独自のソフトウェアを開発するために，開発者が集められ，開発が終了すればそのプロジェクトは解散する．プロジェクトは，ある組織内に発足され終了するまでその活動は継続する．組織からみると，組織内には常に複数のプロジェクトが進行しており，発足から終了を繰り返している．

組織レベルのマネジメントと，プロジェクトレベルのマネジメントは視点が異なる．プロジェクトレベルのマネジメントは，当該プロジェクトの所定の QCD を達成することが目的である．プロジェクトは，各々独自のソフトウェアを創出するためのものであるから，独自のソフトウェアを所定の QCD 目標を達成しながら開発することに集中すればよい．一方，組織レベルのマネジメントは，当該組織で実施するプロジェクトの QCD 目標達成の成功確率を向上させることが狙いである．そのためには，当該組織で実施するソフトウェア開発の特性を分析するとともに，成功するプロジェクトと失敗するプロジェクトの特性を分析し，繰り返し実行可能なソフトウェアプロセスを構築・運用することにより，プロジェクトの成功確率を高めることに集中する．組織からソフトウェアプロセスが提供されている場合のプロジェクトレベルのマネジメントは，提供されているソフトウェアプロセスを使って，当該プロジェクトの特性を分析し，そのソフトウェアプロセスをテーラリングしながら現場へ適用することが成功の鍵となる．

このように，マネジメントの対象が組織かプロジェクトかによって，狙いと活動内容が異なることを意識する必要がある．

2.5　ソフトウェアの特性

本節では，ソフトウェアがもつ特性について議論する．高品質ソフトウェア開発の実現のためには，ソフトウェアそのものの特性の理解が欠かせないとともに，ハードウェアを中心として発展してきた品質管理などの技術をソフトウェアへ応用するために必須なためである．

さらに，ソフトウェアがもつ特性によって引き起こされる問題として，ソフトウェア産業の課題を述べる．ここで，ソフトウェア産業の課題を述べる理由は，結果として高品質ソフトウェア開発の実現に大きく影響するためである．

2.5.1　ソフトウェアの特性

Frederick P. Brooks, Jr.は，ソフトウェアの難しさについて本質的なものと偶有的なものに分けて説明した[2-14]．本質的な難しさとは，ソフトウェアが本来もつ性質である．偶有的な難しさとはソフトウェアがもつ本質的な特性から導かれる副次的な性質であり，目下の開発にはつきまとうが本来備わっているものではない困難のことである．「偶有的な難しさ」という表現は，受け取り手にとって意図するところが理解しにくいため，本論文ではこれを「副次的な難しさ」という表現に置き換えて説明する．

ソフトウェアの難しさのなかで本質的な難しさとは，「抽象的なソフトウェア実体を構成する複雑な概念構造体の仕様作成と設計およびテストである」としている[2-14]．すなわち，顧客の要求を定義し，ソフトウェアが実現すべき機能として設計し，実装設計し，その結果の正しさを確認することである．これは，コンセプトを作り上げその正しさを実証することと言い換えてもよい．一方，副次的な難しさとは，ソフトウェアの実装過程を中心として起こる難しさである．設計内容を扱いにくいプログラミング言語で表現し，ハードウェアの制約のなかで実装していくことをいう．ここでは緻密さと正確さを要求され，1 箇所の間違いも許されない．また，ソフトウェア開発には支援的な活動が多く，構成管理や日々の成果物のバックアップ，開発環境やテスト環境の準備なども副次的な難しさに含まれる．これらの支援活動でも緻密さと正確さを要求されることに変わりはない．

ソフトウェアの本質的な難しさは，対応が非常に難しい．それは，コンセプトを創り出すことそのものだからである．一方，副次的な難しさは，困難ではあっても解決可能な部類の難しさである．ソフトウェアの歴史を見ると，ソフトウェアの難しさへの取り組みは，もっぱら副次的な難しさの解決に対するものであった[2-14]．高級プログラミング言語や開発支援ツールなどがその代表例である．したがって，副次的な難しさに対しては徹底して解決に取り組むべきである．本質的な難しさは，ソフトウェアの本来もつ特性から大きな影響を受ける．以下に，本質的な難しさとの関係を意識しながら，ソフトウェアの特性を述べる．

（1）ソフトウェアは複雑である

ソフトウェアは論理の塊である．そのため，規模の増大につれて複雑さは指数関数的に膨れあがる．Frederick P. Brooks, Jr.はこれを，「ソフトウェア実体の規模の拡大は，単に同じ要素を大きくすることを繰り返すのではない．必ず異なる要素の数が増える．たいていの場合，その要素は互いに非線形で影響しあうものだから，全体としての複雑性の増加は線形どころではない.」と表現している．

しかも，ソフトウェアの扱う問題の対象は，人間の習慣や行動，社会制度であるため，ソフトウェアの複雑性に拍車がかかる．ソフトウェアを利用するためのインタフェースは，人間の習慣や行動に大きく左右されるだけでなく，時間の経過とともに変化する．例えば，短期間のうちに，コンピュータの操作がコマンド主体の操作から画面上での操作に置き換わったことはその代表例である．社会制度も同様である．対面や電話やファックスが主体であった商取引が，短期間のうちにネットワークを介した取引に移行した．しかも，その商習慣は，業界，企業，部門によって微妙に異なる．

ハードウェアには，必ず従わなければならない物理・化学法則や空間的制約がある．しかし，ソフトウェアには，そのような制約はほとんどない．従わなければならないのは，人間の習慣や行動，社会制度であり，これらは常に変化していく．このような問題を扱うために，ソフトウェアは膨大な規模の開発を強いられる．ソフトウェアは変更しやすいという誤解から，ハードウェアで対応できない最終的な変更を強いられるのもソフトウェアである．結果として，その複雑度は計り知れないほどに膨れあがる．

ソフトウェアの複雑性という特性の結果として，次の（2）および（3）に述べる特性が，さらにソフトウェアの難しさへ影響を及ぼす．

（2）ソフトウェアは目に見えない

ハードウェアは，目に見えるし，触って感触を確かめることができる．テレビ，車，冷蔵庫など，どれも目で見て触って確認できる．しかし，ソフトウェアに直接触って感触を確かめることはできない．

ソフトウェアの本来もつ複雑性のため，ソフトウェアの構造を視覚的に表現するのは非常に困難である．１つの図で表現することは不可能であり，制御の流れ，データの流れ，従属関係，時間的処理順序などの用途に合わせて複数の図で表現するしかない．ソフトウェアの規模が増大すると，複数の図を使っても表現することは困難になる．近年において，大規模なソフトウェアを一気に開発することはまれであり，たいてい複数回の開発を重ねて開発する．時間の経過とともに，設計方針が微妙に変化し，仕様が変更になり，ソフトウェアの構造自体がゆがむことがしばしば発生する．こうなると，視覚化の困難さが一層増加する．

目に見えないという特性そのものは，ソフトウェアが本来もっている性質である．しか

し，その特性は解決できないものではないと考える．例えば，設計仕様書の書き方を工夫し，記述すべき図と記法を決めるなどの標準化を進めることによって，設計途中のソフトウェアであっても，ある程度目に見えるものにすることはできる．最近は，UML(Unified Modeling Language)といった世界共通の設計記法が出てきており，記述のためのツールも出回っていることから，設計仕様書の標準化は困難ではあるが不可能ではなくなった．設計の標準化によって，その大きさや特性を計量できるようになる．目に見えないソフトウェアの開発作業を計量できるように標準化し，定量的に把握可能とすることは，ソフトウェアを視覚化する第１歩である．

（3）ソフトウェアは人間の知的作業によってつくられる

ソフトウェア開発において，人間的要素の影響は計り知れない．ソフトウェアの本質的な難しさである，コンセプトを作り上げその正しさを実証することは，まさに人間の知的作業でしか実現できない．時間的な制約の中でこのような知的作業を行うことは，限られた優秀な技術者でなければ不可能である．これは，ソフトウェアのもつ本質的な難しさからくる要件と言ってよい．

さらに，ソフトウェアに課せられた開発すべき規模や時間的制約からすると，ソフトウェア開発は１人ではなく，必ず複数人で構成されるチームで行わなければならない．このため，リーダーシップ，チームワーク，コミュニケーション，さらにこれらから引き起こされるモチベーションといった人間的要素に問題があると，必ず開発されるソフトウェアに影響する[2-15]．例えば，リーダーを信頼できない技術者は，できるだけリーダーと話すことを避けようとする．すると，リーダーに確認すべき項目を確認しないままソフトウェアを作ってしまうという事態が発生する．その結果は明らかである．ハードウェアの設計も，人間の知的作業によって行われるが，決定的な違いは，関係する人員の数である．ハードウェアの設計に比べて，ソフトウェア開発に必要な人員数は桁違いに多いのである．人数が増えると，指数関数的にコミュニケーションは難しくなると言われているが，多数の人員を抱えたプロジェクト体制を組むソフトウェア開発におけるコミュニケーション問題は重要な課題である．

Frederick P. Brooks, Jr.は，人間的要素に関する問題について，「成功のためには，プロジェクトに携わる人々の質，およびその組織形態と管理こそが，使用するツールや採用する技術的アプローチよりもはるかに重要な要因である」と説明した[2-14]．人間的要素がソフトウェア開発に影響を与えることは，ソフトウェアのもつ本質的な特性であるが，解決策はある．人間が誕生して以来，人間的要素はどの領域においても常に問題であり，我々はその問題領域に合わせて課題を解決するという歴史を繰り返してきた．要所毎にそれに適した人材を配置し，常に人間的要素も含めてマネジメントする体制をつくる．互いの開発者が尊敬しあい，よいソフトウェアを開発することに誇りを持つような組織文化をつくる．ソフトウェアのもつ副次的な難しさを徹底的に解決し支援する仕組みをもち，本質的

な難しさの解決へ集中できるソフトウェアプロセスを構築する．このような取り組みにより，人間的要素に絡む問題はかなり解決できるだけでなく，逆に不可能を可能にするようなソフトウェア開発を実現する可能性もあると考える．

2.5.2　ソフトウェア産業の課題

本節では，ソフトウェアの本来もつ特性が引き起こすソフトウェア産業としての課題について議論する．ソフトウェアは変更しやすいといった思い込みのため，ソフトウェアの特性の正しい理解が困難であることが，ソフトウェア産業としての課題の大きな原因となっていると考える．これは，ソフトウェアの歴史の浅さに起因する問題かもしれないが，時間が解決すると楽観的に考えるには根が深いように見える．現実として，高品質ソフトウェア開発の実現に大きな影響を及ぼしている．

ソフトウェアはハードウェアに比べて作ることも修正することも容易である．PC1 台あれば，子供でもソフトウェアを作ることができる．これは一面の事実である．しかし，ビジネスとして提供するソフトウェアがそれと似たようなものと考えるのは誤りである．ソフトウェアがこれだけ社会のあらゆる場面で使われ，しかも社会基盤を構築する重要な要素に成長した現在においても，ソフトウェアの提供側である企業の認識が変わらないとしたら，大きな問題である．本来，知的集約産業であるべきソフトウェア産業が，日本においては労働集約産業になっている現実をみると，企業においてソフトウェアの特性が理解されていないか，理解していてもどうしていいかわからないとしか考えられない．

(1)　多重下請け構造

日本におけるソフトウェア開発は，大手企業が元請となって顧客からソフトウェア開発を受注し，開発そのものは下請け企業に委託するという開発形態をとることが多い．発注形態は，人月に単金を掛け合わせた額で発注されることがほとんどである．開発者の能力差に起因する問題が認識され，研究成果もあるなかで，技術レベルや経験の違いに関わらず，誰でも同じ額で発注される．したがって，ソフトウェア業界では，開発発注時には単金が主な焦点になる．さらに，人・月当りのコード行数で表現する生産性を向上させると，開発に必要な工数が減少するため，人月に単金を掛け合わせた額で発注される現在の発注形態では，逆にソフトウェア開発企業の売り上げ減少を招いてしまう．このため，ソフトウェア開発企業にとって，生産性向上に積極的に取り組むモチベーションが薄い．また，大手企業以外は，ソフトウェア開発企業といっても実体は人材派遣のため，自社のソフトウェアプロセスを構築し高度化させる必要性が低い．人材派遣が中心のため，発注先のソフトウェアプロセスへの適用強制力が強く，自社のソフトウェアプロセスを適用する場面が少ないのである．このようにして，ソフトウェア業界は多重下請け構造による労働集約産業になってしまった．この背景には，顧客からの価格低減への圧力がある．しかし，こ

れはソフトウェア業界自身が努力するとともに，適正なコストを顧客へ説明し，納得してもらうべき課題である．

ここには，「誰でも同じ出来のソフトウェアが作れる」というソフトウェアに対する誤解がある．ソフトウェアエンジニアリングの課題としてここで読み取るべきは，技術レベルによって品質・生産性に大きな差が出ることは認識していても，現時点においてそれを合理的に計測する尺度が見つからないことである．開発者の技術レベルやソフトウェアプロセスの卓越性と，ソフトウェアの出来の関係を示すことが出来れば，この問題は解決するはずである．

(2)　オフショア開発

多重下請け構造のさらなる拡大版が，オフショア開発である．単金がソフトウェア開発における主要課題になってしまったため，グローバル化にともなって，国家レベルで賃金の低いところへ下請け先が流れるようになった．最近まで，日本の主要なオフショア開発相手国は中国だった．近年では，中国の賃金が高騰してきたため，より賃金の安いベトナムやモンゴルなどへオフショア開発相手国が変わりつつある．オフショア開発での問題は，常にソフトウェア開発の品質・生産性である．仕様が伝わらない，文化や言葉の違いのため指示してもオフショア先の技術者が指示通り動いてくれない，テストしてもバグが枯れない，期限までにソフトウェアを完成できないといった問題が多発する．結果として，コスト低減したつもりが逆に日本で開発したほうが安かったということになる．このような失敗事例は数え切れないほどあるにもかかわらず，より安い賃金を求めて，新しい国へのオフショア開発発注が続く．現在，上海では，日本とまったく同じ問題が起こっている．上海企業の単金が高騰しすぎて，上海企業から単金の安い下請け先への発注をするようになった．その結果，彼らもまた日本企業が苦しんでいるのと同じオフショア開発先の品質問題に直面するようになったのである．

オフショア開発では，オフショア開発先へ設計仕様書を渡して，コーディングから単体テストまでの工程を任せることが多い．ここから読み取れるソフトウェアに対する誤解は「仕様書を渡せば正しいプログラムになって返ってくる」ということである．ソフトウェアの特性を理解すれば，これが誤解であることがわかるはずだ．国が異なれば常識は異なる．日本の常識を基盤として日本語で記述されるソフトウェアの設計仕様書を，他国の技術者が説明もなしに正しく理解することはほとんど不可能である．また，設計仕様書にすべてを記述できるわけではないため，日本語で記述された設計仕様の行間を読む必要があるが，それを他国の技術者に求めるのは極めて酷である．オフショア開発で発生する問題のほとんどは，このようなソフトウェア開発における人間的要素への理解不足のために生じている．

(3) まとめ

ソフトウェア産業の課題から，「誰でも同じ出来のソフトウェアが作れる」，「仕様書を渡せば正しいプログラムになって返ってくる」という誤解があることを説明した．

ソフトウェアの特性を念頭に置いたとき，単金だけに注目するのは明らかに誤りである．ソフトウェア産業が多重下請け構造からなる労働集約産業になっている現実が，ソフトウェア開発の品質・生産性問題に与える負の影響は大きい．オフショア開発への流れは，その問題をさらに拡大している．

開発者の技術レベルやソフトウェアプロセスの卓越性によって，ソフトウェア開発の品質・生産性に大きな差が出ることを実証し，認知させる活動が求められていると考える．

2.6 CMMI レベル 5 組織への調査結果からみる課題

本節では，日本の CMMI レベル 5 を達成した日本国内の組織に対する調査結果[2]に基づき，高品質ソフトウェア開発を阻む課題を議論する．CMMI（Capability Maturity Model Integration）とは，米国カーネギーメロン大学 SEI(Software Engineering Institute)が開発したソフトウェア能力成熟度モデル統合である[1-2]．ソフトウェアプロセスの能力成熟度を 5 段階に分け，各段階で達成すべき項目を規定している．レベル 5 の判定を受けるには，有資格者によるアプレイザルが必須であるため，客観的な視点から自社のソフトウェアプロセスが一定レベルの能力を保有していることを証明可能である．CMMI は，ソフトウェアプロセス改善のツールとしては，デファクトスタンダードと言える．

2.6.1 調査の概要

調査の概要は以下の通りである．

- 調査の目的：
  プロセスの能力成熟を実際に達成した組織への調査により，プロセス能力の成熟の意味するところ，プロセス能力成熟の壁と効果のある要因を探り，プロセスの能力成熟とは何かを明らかにする
- 調査内容：レベル 5 挑戦の動機と結果，得られた教訓
- 調査対象：CMMI レベル 5 を達成した日本国内の組織
- 回答結果：12 組織より回答を受領（当時，レベル 5 を達成している日本国内の組織は全体で 20 組織程度である）

[2] 本著者は，調査団体である SQiP ソフトウェア品質委員会の副委員長を務めており，本調査の主責任者である．なお，本著者は，自組織の CMMI レベル 5 達成をリーダーとして推進し，2004 年に達成した経験がある．

・ 調査実施時期：2009 年 7 月
・ 調査団体：日科技連 SQiP ソフトウェア品質委員会

2.6.2 調査結果の概要

調査結果の概要を，図 2-6～図 2-12 および表 2-5～表 2-9 に示す．調査項目のなかには，重複回答を許しているものがあることにご注意願いたい．

CMMI 挑戦のメリットとして，半数以上の 7 社が「定量管理の定着」をあげている（表 2-6 参照）．デメリットとして，半数以上の 8 社が「形式的なプロセスになりがち」をあげている．形式的なプロセスとは，「不要な作業や不要なプロセスエリアの増加」という意味である．他の組織へ CMMI を進めるかとの問いに対して，意外にも 4 割が「薦めない」と回答した（図 2-10 参照）．その理由として，「費用対効果が見込めない」，「レベル達成が目的になりがちなため」があがった（表 2- 7 参照）．一方，薦めると回答した 6 割において，薦める理由としては，「組織的なプロセス改善が可能だから」が 4 社と多かった．ただし，「レベル達成を目的とせず，本質的な改善ができる場合に限る」とコメントした組織が 2 社あり，CMMI は一定の効果があるものの，レベル達成を目的としないような推進上の注意が必要であることがわかる．

プロセスの能力が成熟すると品質は向上するかという問いに対して，1 社を除く 11 社が「品質は向上する」と回答しており，プロセス成熟は品質に対して一定の効果があることがわかる（図 2-11 参照）．なお，残る 1 社は，「プロセスの成熟により，品質のばらつきは少なくなるが，品質向上とは関係が少ない」と回答している．出荷後バグ数の水準に対する問いでは，半数が日本の平均（0.020 件/KLOC）と同等レベル以上と回答し，日本の平均より悪い場合でもインドと同等レベル（0.263 件/KLOC）の範囲に収まっている（図 2-12 参照）．なお，これらの出荷後バグ数の数値は，2003 年に IEEE Software Magazine へ発表された資料[2-16]に基づいている．

品質向上に不可欠な要素は何かという問いに対して，品質を重視するメンバの意識や組織文化といった「人間的要素」を挙げた組織が 5 社とトップだった（表 2-8 参照）．ついで，定量化，トップのリーダーシップ，レビューと続く．品質向上に効果のある技術やノウハウは何かという問いに対しては管理技術，開発技術，レビュー技術，テスト技術など広範囲にわたるさまざまな技術があがった（表 2-9 参照）．複数の組織から支持があった技術は「プロジェクトの見積り手法」「レビュー・インスペクション」であり，どちらも 3 社があげている．ここで注意すべきことは，従来から知られている技術が多いことである．全部で 18 の技術・ノウハウがあがったうち，比較的最近提案された技法は，「派生開発技法（XDDP）」と「テストファースト」の 2 つである．

2.6.3　プロセスの能力成熟とは何か

プロセスの能力成熟のメリットや効果，不可欠な要素で多く登場するキーワードは，「定量化・可視化」と「人間的要素」の 2 つである．一方，デメリットや問題点で多く登場するキーワードは「形式的なプロセス」である．

「定量化・可視化」と「人間的要素」は，ソフトウェアの本質的な難しさに対応する．「定量化・可視化」は，まさに「ソフトウェアは目に見えない」という特性への解である．「人間的要素」は，「ソフトウェア開発は人間の知的作業によって行われる」に対応する．回答組織は，ソフトウェアの難しさに対して，正面から向き合う対応策と成果を最大のメリットと考えていることがわかる．一方，「形式的なプロセス」の意味するところが，不要な作業や不要なプロセスエリアの増加であることを考慮すると，実質的に効果のないことを意図していることがわかる．これらのことから，プロセスの能力成熟の意味するところは，「ソフトウェアの難しさに対して正面から向き合う対応策により，実質的な成果を得ること」と考えることが出来る．

また，プロセスの能力成熟の壁は，「実質的な成果をあげることの難しさ」と考える．「形式的なプロセス」というキーワードからは，しばしば実質的な成果を得られない不要なプロセスに陥ってしまう現場の問題認識を感じる．定量化は，どのソフトウェア開発組織でも実施していることである．レベル 5 達成のメリットの第 1 位として半数以上の組織が「定量管理」をあげたことに，実質的な成果をあげる難しさの壁を理解すべきである．

プロセスの能力成熟に効果のある要因は，従来から知られている技術を的確に適用することであり，特に定量管理とレビューに効果が期待できる．調査結果に見える「改善意識」，「品質を重視するメンバの意識」といったキーワードから，開発者に対する意識変化を促す施策も重要と考える．

2.6.4　CMMI により解決できること・できないこと

調査結果からは，CMMIへの取り組みにより，ある程度の品質向上を見込むことが出来ることがわかる．その出荷後バグ数の水準では，インドと同等か日本の平均より悪い組織が半数を占める．インドと同等の場合は，規模が100KLOC(×10^3LOC)のソフトウェアから出荷後1件間に26件のバグが顧客で発生することになる．本著者の経験では，このレベルは日本市場では不満と評価されるはずである．したがって，CMMIレベル5であっても出荷後バグ数の少なさを強みにできるほどの品質を実現するのは難しいと言える．

CMMI挑戦のメリットとしてあがっている定量管理の実現は，CMMIの重要なプロセス領域であり，CMMIへの取り組みにより成果をあげることが期待できる領域である．また，回答に明示的に現れなかったものの，CMMIの本来の目的である機能的に欠落のないプロセスの構築は，CMMIの大きなメリットである．レベル5達成は，CMMIの定義する22個の

全プロセス領域を実装していることの証明である．開発途中に発生する問題の多くは，プロセスそのものの機能的な不備によるものであり，それは未然防止ができる．一方，形式的なプロセスがデメリットとしてあがっていることから，すべてのプロセス領域が実効あるプロセスになるとは限らないと言える．CMMIは本来，利用者側の目的に合わせて使うツールだが，しばしばレベル達成が目的にすり替わってしまうために，このようなことが発生する．

品質向上に不可欠な要素として，品質を重視するメンバの意識や組織文化といった人間的要素があがった．人間的要素は取り組みの姿勢に関わる問題であり，CMMIでもプロセス領域としてはあがっていない．したがって，人間的要素はCMMIへの取り組みとは直接関連がない．

以上により，CMMI で解決できると考えられるのは，ある程度の出荷後バグ数の低減，機能的に欠落のない網羅的なプロセスの構築および定量管理の実現である．一方，強みに出来るレベルの出荷後バグ数の少なさの実現，実効あるプロセスの構築および人間的要素の改善は，CMMI への取り組みで実現するのは難しい．CMMI で実現するのが難しい項目は，いずれも解決可能な決定的技術が提案されていない．

2.7　おわりに

企業が持続的発展を実現するには，長期的かつ安定的に顧客満足を得ることが求められる．品質の目指すところは顧客満足である．品質は，顧客のニーズという外部価値基準によって決まるという点が重要である．その意味で，品質は，マネジメントの 3 要素と呼ばれる QCD の文脈で意味する狭い範囲の Q ではなく，C や D を包含する幅広い概念でとらえるべきである．ソフトウェアにおいて品質を確保するポイントは，的確な要求事項の把握とプロセスで品質を作り込むことの実践である．

品質を幅広い意味でとらえたとき，ソフトウェア開発における品質に関連する重要な用語であるバグは，単に設計通りに動作しないという狭い範囲を指すだけでなく，本来あるべき特性を実現していない場合も含む範囲とすべきである．

ソフトウェアの難しさには，ソフトウェアの本来もつ特性に起因する本質的な難しさと，目下の生産という局面で発生する副次的な難しさがある．本質的な難しさとは，ソフトウェアのコンセプトを創出してその正しさを実証することであり，副次的な難しさとは，ソフトウェアの実装を中心とした部分に発生する，緻密さと正確さを要求される多量の作業である．本質的な難しさは，「複雑である」，「目に見えない」，「人間の知的作業により作られる」というソフトウェアの本来もつ特性に起因する．これらのソフトウェアの特性は，ソフトウェア産業において正しく理解されているとは言えず，それがソフトウェア産業を労働集約産業にしてしまっているという課題も指摘した．

高品質ソフトウェア開発の実現のためには，ソフトウェアの副次的な難しさを徹底して合理化するとともに，ソフトウェアが本来もつ特性に対して戦略的に対応するという両面からの取り組みが求められる．この取り組みは，形式ではなく実質的な効果を生む取り組みにできるかが大きな鍵である．成果を出すために効果のある技術やノウハウのキーワードは，管理技術（特に定量管理と人間的要素に対するマネジメント）やレビューなどであり，従来からの技術を確実に適用し効果を出すことがポイントである．そのなかで CMMI により解決できることは，機能的に欠落のない網羅的なプロセスの構築と，ある程度の出荷後バグの低減および定量管理の実現である．解決しにくいことは，強みと言えるレベルの出荷後バグ数の低減，実効あるプロセスの構築，および人間的要素の改善である．本論文では，これらの現実と課題を確認した上で，次章以降を論述する．

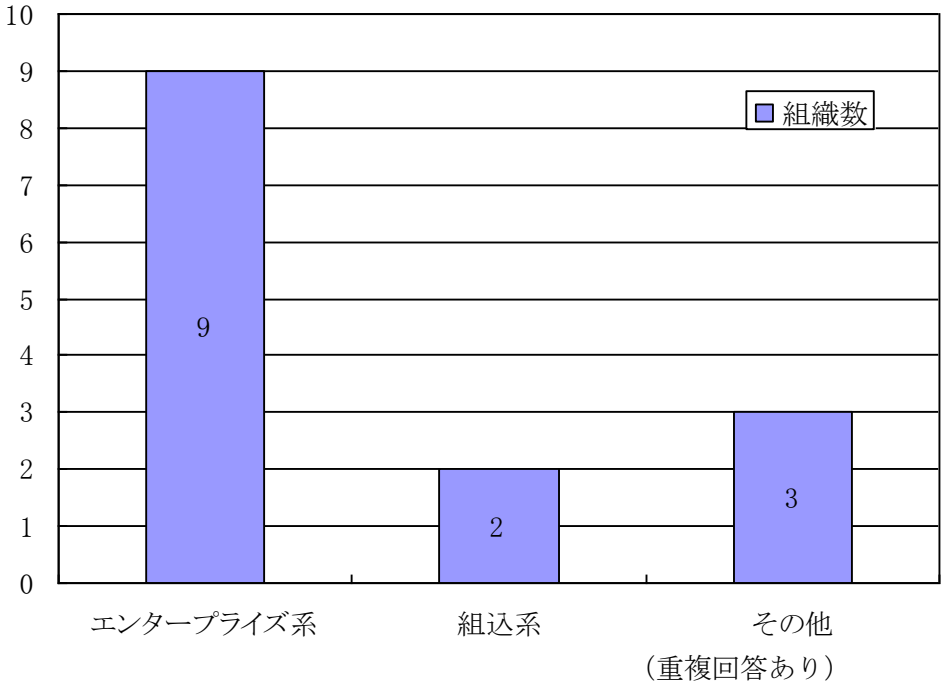


図 2-6　対象組織の業務内容　－CMMI レベル 5 組織に対する調査結果（1）－

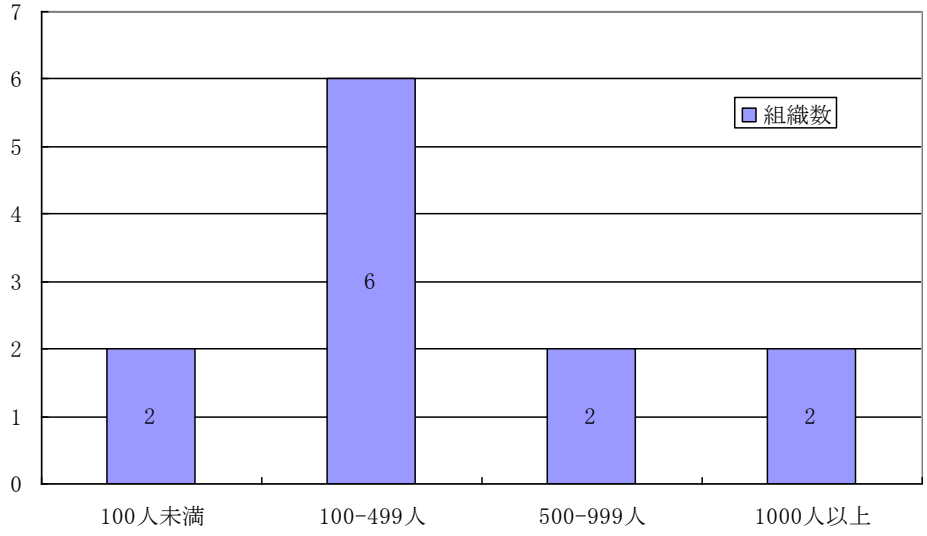


図 2-7　組織の人員数　－CMMI レベル 5 組織に対する調査結果（1）－

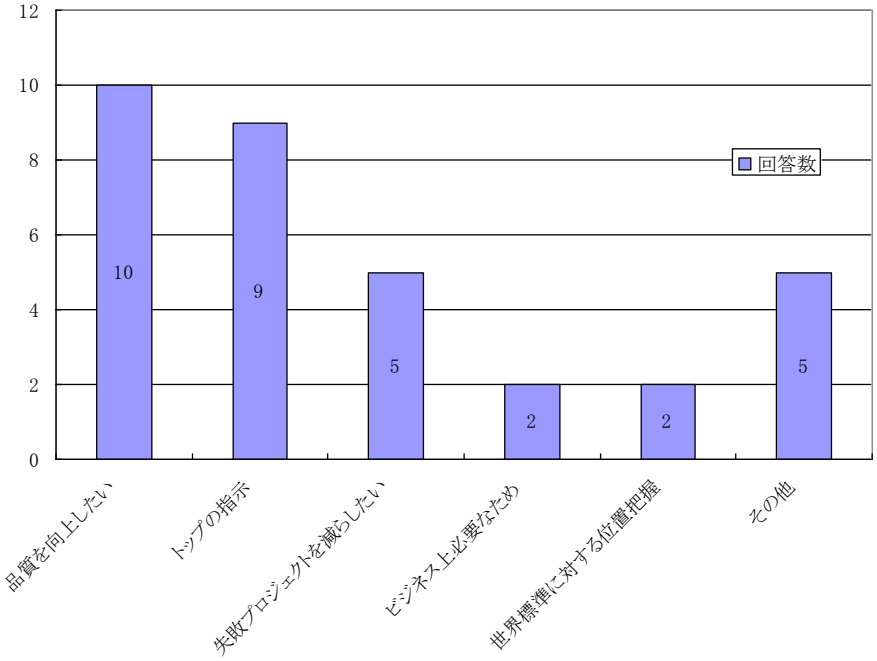
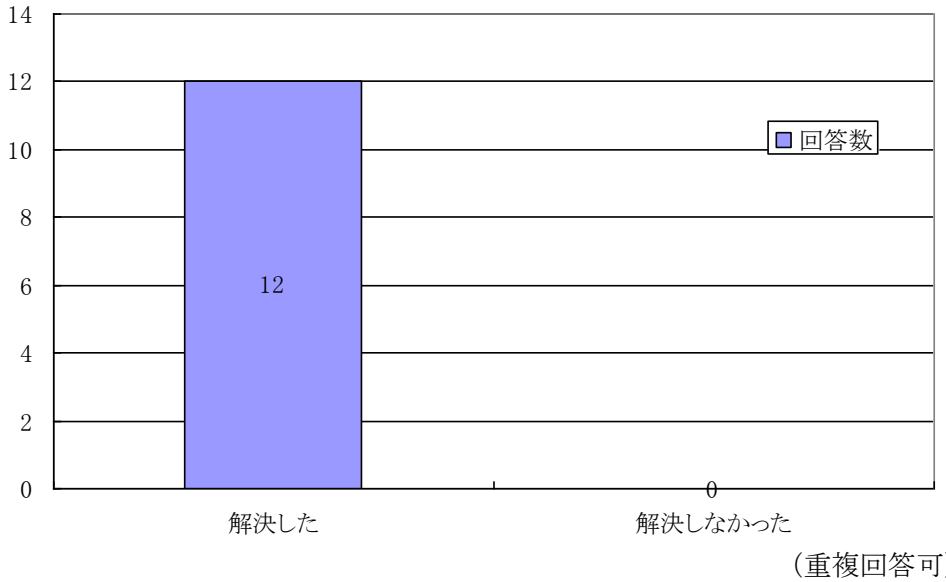


図 2-8　CMMI レベル 5 への挑戦の目的
－CMMI レベル 5 組織に対する調査結果（2）－



（重複回答可）

図 2-9　レベル 5 達成により課題は解決したか
－CMMI レベル 5 組織に対する調査結果（2）－

表 2-5　解決した課題の内容　－CMMI レベル 5 組織に対する調査結果（3）－

No.	具体的な解決内容
1	失敗プロジェクトの大幅削減(3)
2	品質向上(2): ・出荷後欠陥数が3年で77%削減 ・出荷後欠陥数が4年で50%削減
3	生産性向上(2): ・テストコストが17%削減 ・Web開発生産性が4年で44%向上
4	強み/弱みの把握(2)
5	社外へのアピール
6	メンバーの意識向上
7	マネジメントのしくみ構築, 定量管理, 改善のサイクル定着

注意:フォローを十分にしないプロジェクトは未解決

((　)内の数字は同一回答組織数)

表 2-6　CMMI 挑戦のメリット・デメリット
－CMMI レベル 5 組織に対する調査結果（3）－

No.	CMMI挑戦のメリット
1	定量的管理の定着(7)
2	改善意識の定着(3)
3	弱みの把握(2)
4	順を追って改善する重要性の認識
5	レベル5達成による自信
6	品質レベルの底上げ

No.	CMMI挑戦のデメリット
1	形式的なプロセスになりがち(不要な作業やプロセスエリアの増加など)(8)
2	アプレイザルの負担大(2)
3	小規模・短期プロジェクトには不向き
4	トップダウンで強力に進めると「やらされ感」が強くなってしまう

((　) 内の数字は同一回答組織数)

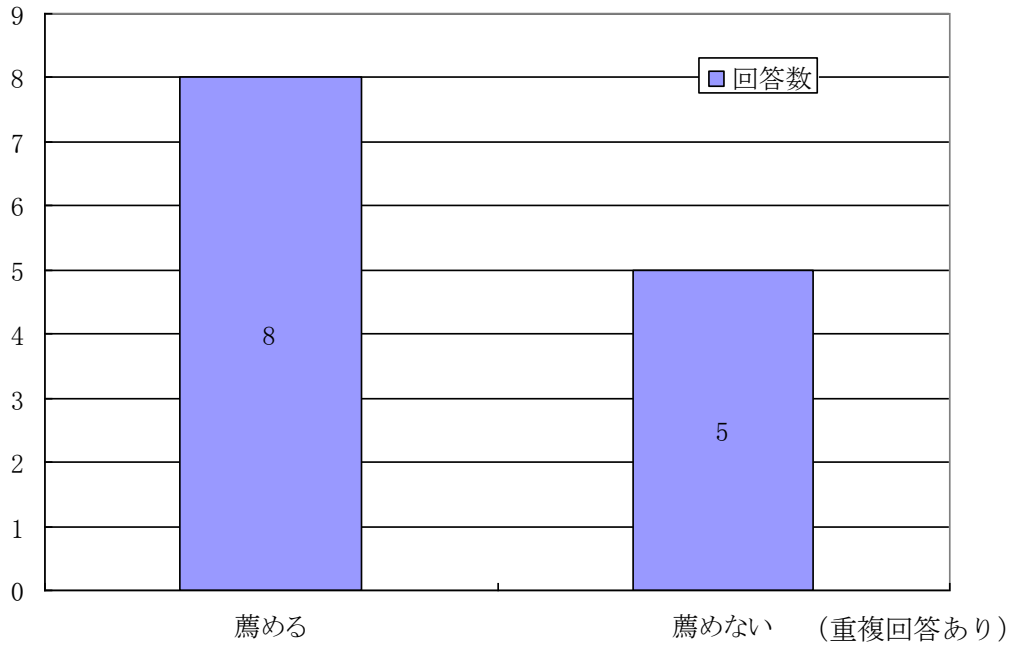


図 2-10　他組織へ CMMI への挑戦を薦めるか
－CMMI レベル 5 組織に対する調査結果（4）－

表 2-7　CMMI を薦める理由・薦めない理由
－CMMI レベル 5 組織に対する調査結果（4）－

No.	CMMIを薦める理由
1	網羅的なプロセス改善が可能だから(3)
2	弱みの把握ができる
3	問題の解決の参考になるため
4	世界標準に対する自社のポジション把握ができる

注意:CMMIを薦めるのは，レベル達成を目的とせず，本質的な改善ができる場合に限る(2)

No.	CMMIを薦めない理由
1	費用対効果が見込めない(4)
2	レベル達成が目的になりがちなため(3)
3	レベル達成のために，必要以上のタスクを定義しがち

（(　) 内の数字は同一回答組織数）

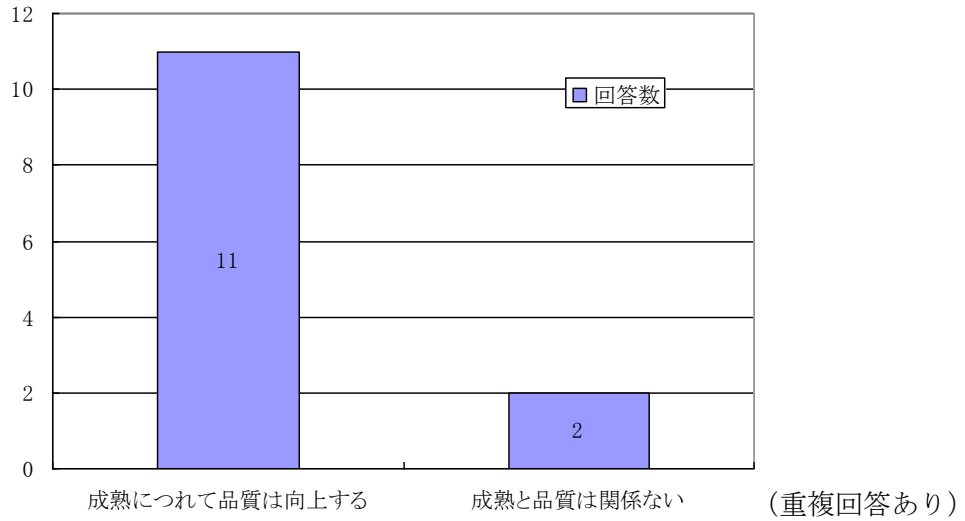
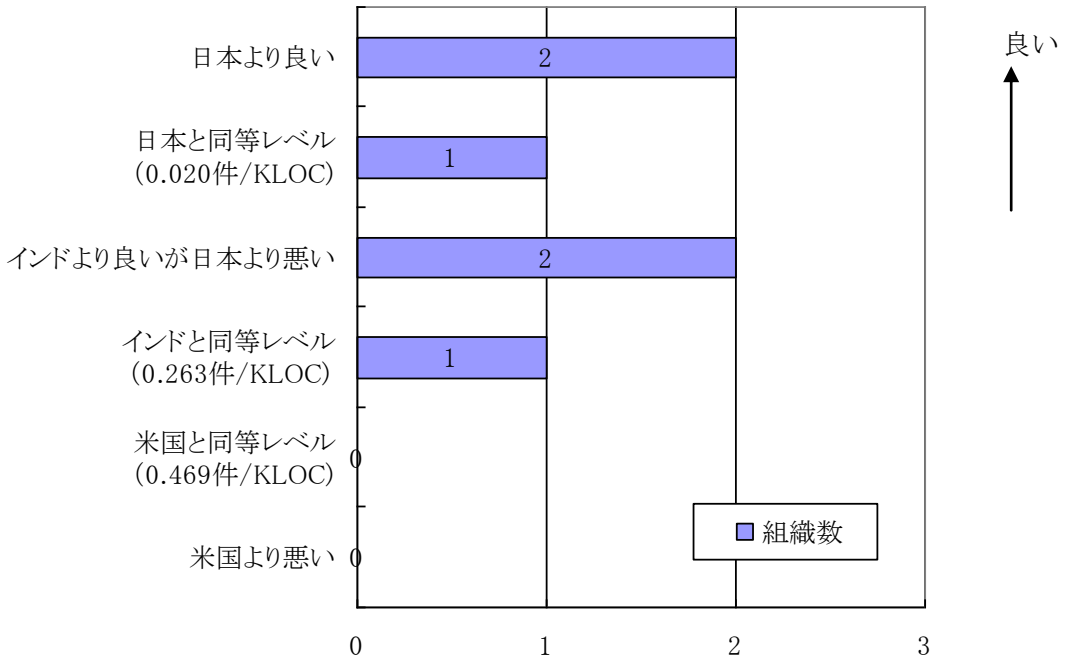


図 2-11　プロセスが成熟すると品質は向上するか
－CMMI レベル 5 組織に対する調査結果（5）－



・半数が回答
・出荷後バグ数：出荷後 12 ヶ月のバグ数（件/KLOC）

図 2-12　出荷後バグの水準　－CMMI レベル 5 組織に対する調査結果（5）－

(M. Cusumano et al., “Software Development Worldwide: The State of the Practice”, IEEE Software, Vol.20, No.6, pp.34-38, 2003)

表 2-8　品質向上に不可欠な要素とは何か

－CMMI レベル 5 組織に対する調査結果（6）－

No.	品質向上に不可欠な要素
1	品質を重視するメンバの意識や組織文化（5）
2	成果の可視化・定量化とフィードバック（4）
3	トップのリーダーシップ（3）
4	レビュー（3）
5	継続的なチェック（2）
6	負荷軽減の姿勢（2）
7	ベストプラクティスの収集と展開
8	ソフトウェア開発の基本技術の習得
9	プロセス実施の徹底
10	開発の各段階での品質作りこみ

(( )内の数字は同一回答組織数)

表 2-9　品質向上に効果のある技術やノウハウ

－CMMI レベル 5 組織に対する調査結果（6）－

No.	品質向上に効果のある技術・ノウハウ	カテゴリ
1	プロジェクト見積もり手法（3）	管理技術
2	過去データの蓄積	
3	見積もりの予実差異管理	
4	信頼性向上の管理指標（ＪＩＳＡ）	
5	要件管理を中心とした開発管理ツール	開発技術
6	ＰＫＧやフレームワークを用いた開発手法の標準化	
7	派生開発手法「XDDP」	
8	レビュー・インスペクション（3）	レビュー関係
9	上工程でのバグ摘出重視	
10	テストファースト	テスト関係
11	テスト時のカバレージ評価	
12	テスト実施状況の可視化（グラフ化）	
13	ソースコードの静的チェック	
14	計画レビュー	第３者によるレビューやチェック
15	ＳＱＡによる進捗レビュー	
16	第三者評価の実施	
17	なぜなぜ分析	なぜなぜ分析と横展開
18	失敗プロジェクトの横展開	

(( )内の数字は同一回答組織数)

第3章　ソフトウェア品質会計

3.1　はじめに

NEC でソフトウェアを専門に開発する組織が発足したのは 1974 年のことである．それまで，ソフトウェア開発は，ハードウェア部門の業務の一部として行われていた．当時は，ソフトウェアの黎明期であり，今後すべての領域にソフトウェアが使われるようになり，そのソフトウェア規模の増大に対して技術者が大幅に不足するであろうという，いわゆる「ソフトウェア危機」に対する危機感があった．さらに，出荷したソフトウェアのバグが多く，その対応に追われて費用がかさみ，ソフトウェア品質問題が経営課題になりつつあった．

幾つかの日本企業は，1970 年代頃からソフトウェア品質問題に対して改善に取り組んできた[3-1][3-2][3-3][3-4]. こうした日本企業のソフトウェア品質確保への取り組みは，1980～1990 年代に欧米に紹介され一定の評価を得た[3-5]. 当時の日本企業の取り組みの特徴は，日本のハードウェアを中心とした製造業において大きな成功を収めた統計的品質管理手法を，ソフトウェアへ取り込もうというものであった．

NEC では，ソフトウェア開発における品質問題を解決するため，20 年以上の間，SWQC 活動などのさまざまな取り組みを展開してきた[3-6] [3-7] [3-8]. その結果構築した技法の 1 つが「ソフトウェア品質会計」[3-9]（以降，「品質会計」と呼ぶ）である．「品質会計」とは，1982 年頃，NEC の IT 製品向けの OS および汎用ミドルソフトウェア製品を開発する部門で発案され，現在まで継続的に改善してきたソフトウェア品質管理手法である．本著者は，品質会計を発案した組織（以降，「品質会計考案組織」と呼ぶ）において，品質会計の構築および適用に中心的な立場で継続的に携わるとともに，現在は当該組織において品質保証の責任を持つ．品質会計考案組織は，第 1 章で説明した NEC 初のソフトウェア専門開発組織である．本章では，ソフトウェア品質会計について論述する．

3.2　技法の概要

3.2.1　品質会計の技術体系

品質会計とは，品質が作り込まれたことを，確かな根拠をもって説明するソフトウェア品質管理手法である．品質会計では，バグ件数を主軸として摘出目標管理することにより，その目的を実現する．したがって，品質会計では，開発中に作り込まれるバグ件数を，常

に精度良い推定値に保持する必要がある．この要求を実現するための技術が，図 3-1 に示す「品質会計の技術体系」である．

品質会計は「バグ目標管理技法」および「バグ予測・見直し技法」から構成される．特に，開発中に発生するさまざまな事象に対応して予測バグ件数を見直すことができるよう，「バグ予測・見直し技法」が多く存在する．

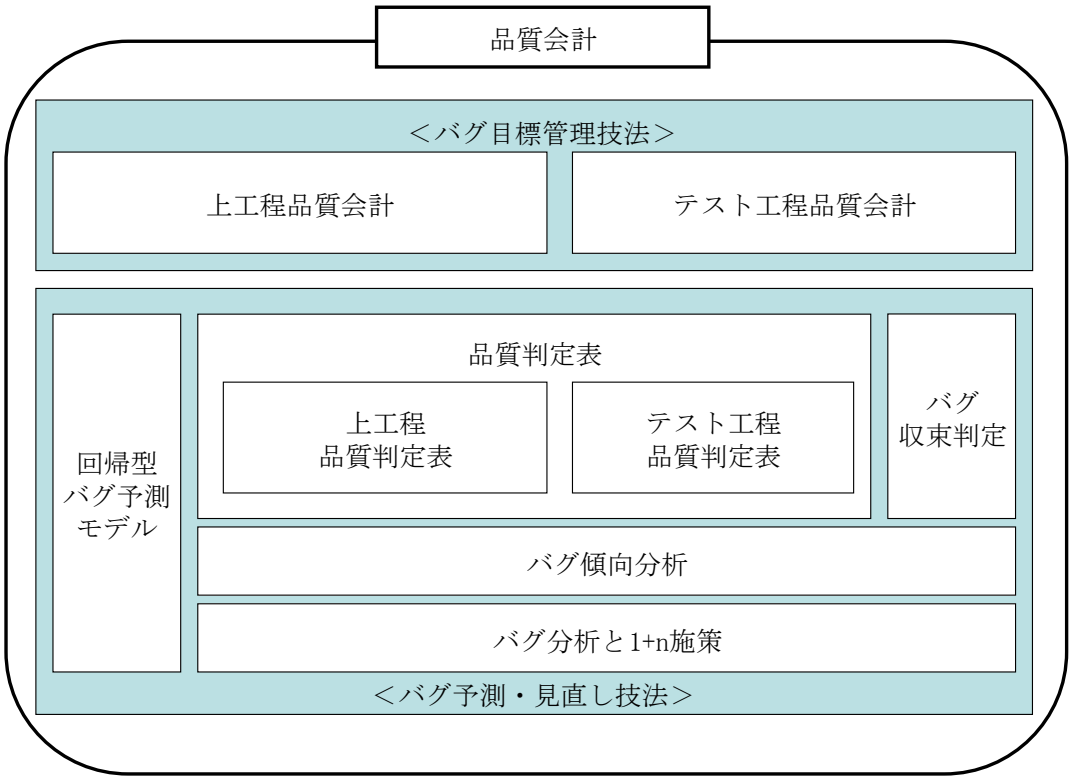


図 3-1　品質会計の技術体系

3.2.2　品質会計で想定する開発プロセス

品質会計で想定する開発プロセスは第 2 章で議論したＶ字モデルである（図 2-2 参照）．基本設計からコーディングまでを上工程とよび，単体テストからシステムテストまでをテスト工程と呼ぶ．上工程の各工程においては，当該工程の「設計」と「レビュー」という 2 つの作業を実施する．設計とレビューの関係を図 3-2 に示す．「設計」は，前工程の出力物である前工程仕様書（最終版）を入力し，当該設計工程の設計仕様書（第 1 版）が完成するまでをいう．「レビュー」はそれ以降のレビューを指し，レビューを繰り返して設計仕様書（最終版）が完成するまでをいう．当該工程で当該工程のバグをバグとして認識し，計測開始するのは，当該工程の「レビュー」以降である．

品質会計は，開発の進行に応じて，図 3-1 に示した品質会計の各技法を組み合わせて適用する（図 3-3 参照）.

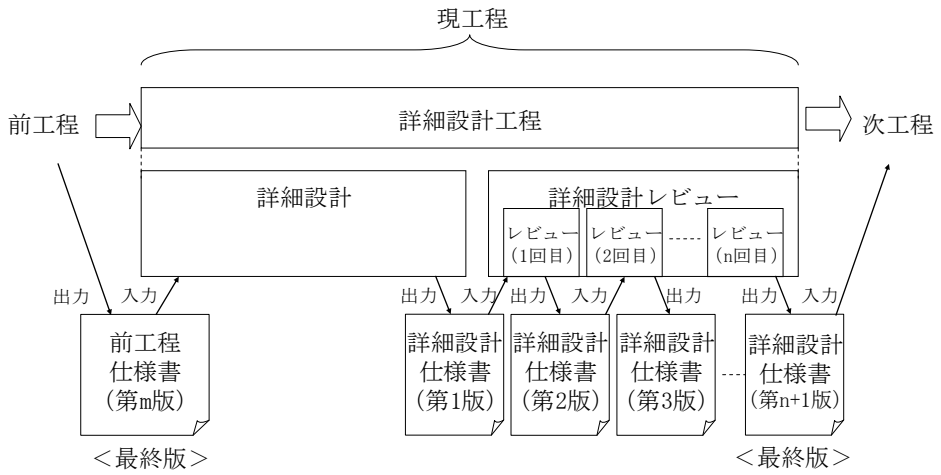


図 3-2　設計とレビューの関係（詳細設計工程の例）

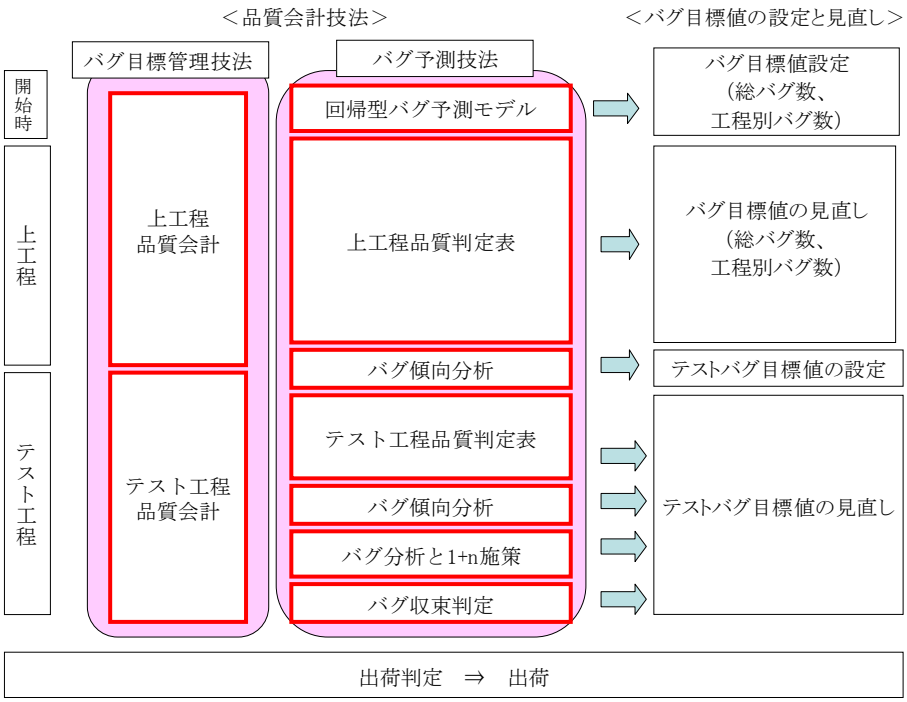


図 3-3　ソフトウェアライフサイクルと適用する品質会計技法

3.2.3　品質会計の適用範囲

品質会計は，IT 製品向けの OS および汎用ミドルソフトウェア製品を開発する部門で構築された技法である．汎用ソフトウェア製品は，特定顧客向けではなく広く汎用的に使用されることを想定して開発するため，開発部門が技術動向や顧客動向を分析して自ら要求仕様を定義するという特性がある．このため，特定顧客向けのソフトウェア開発で重要な顧客との仕様調整は原則として不要である．このように，ソフトウェア領域によってソフトウェア開発に関連する必要作業に違いはあるものの，設計以降の作業は基本的に同じである．こうした背景から，品質会計は，現在，NEC グループ全体に渡って標準的な品質管理手法として適用されている．これまでの適用実績から，品質会計は，以下の範囲で適用可能である．

- ソフトウェアライフサイクルプロセス（ISO/IEC12207/1995(JIS X 0160:1996）)で定義する主ライフサイクルプロセスのうち開発プロセスの範囲で適用し，ウォーターフォールモデルなどの計画駆動型プロセス[3-10]での適用を前提とする．
- エンタープライズ系や組込み系といったソフトウェア領域には依存しない．

3.2.4　品質会計の特徴

品質会計の語源は，バグを負債とみなすことからきている．負債は利子で膨らまないうちに早期に返済するほうが経済的である．ソフトウェアのバグも同様に，1 件のバグが複数のバグを生まないうちに，設計・製造で作り込まれたバグを早期にレビューやテストで摘出するほうが後戻りが少なくて済む．すべての負債を返済したとき，すなわちすべてのバグを摘出したとき，そのソフトウェアを出荷する．

品質会計の基本的な考え方は,「バグは作り込まない．作り込んだバグは素早く摘出する」である．この考え方のもと，上工程では「作り込んだバグは次の工程までに摘出する」ことを，テスト工程では,「作り込んだバグは，すべて摘出してから出荷する」ことを原則とする（図 3-4 参照）．品質会計では，上工程におけるレビューでのバグ摘出を重要視しており，そのための目標が「上工程バグ摘出率 80%」である．上工程バグ摘出率とは，出荷前までの全摘出バグ数に対する上工程での摘出バグ数の比率をいう．

品質会計の特徴は，上記の原則を実現するために考案し改善を重ねた以下の 2 点にある．

- レビューでのバグ摘出による早期品質確保
  - 「上工程品質会計」により，レビューで摘出したバグを作り込み工程と摘出工程の両面から管理
- 的確なテスト完了判断
  - 「バグ傾向分析」,「バグ分析と 1+n 施策」，および「バグ収束判定」の３つの技法の併用による残存課題の把握と解決

3.2.5　品質会計の手順

開発開始後，上工程では上工程品質会計，テスト工程ではテスト工程品質会計を適用してバグ目標管理を実施する．図 3-5 に示す手順に従って，一週間毎などの管理スパンで定期的に品質分析を実施する．管理スパンは各工程終了時では遅いので，工程途中の状況を把握し，問題発生時にリアルタイムで適切な対応策を実施することが重要である．工程終了時では，問題の対応結果の追認にしかならず，途中の対応策が不十分であっても追加対応策を実施するのは難しいことが多い．

品質分析の結果，目標通りの品質が作り込まれているかを判断し，必要であればバグ目標値を見直す．適用する品質分析技法は，上工程とテスト工程で多少異なる．上工程では，ソフトウェアを作り上げる段階であるため，バグの作り込み工程に注目した「作り込み工程別バグ分析」を実施する．バグの作り込み工程とは，当該バグを作り込んだ工程のことであり，図 2-2 に示すＶ字モデルである開発プロセスの上工程のうちいずれかの工程である．また，上工程およびテスト工程を通じて，「品質判定表」により品質を判定する．テスト工程では，計画したテストをほぼ終了した時に，テスト完了判断のため，「バグ傾向分析」，「バグ分析と 1+n 施策」，および「バグ収束判定」の３つの技法を併用して，残存問題の把握と解決を行う．

＜品質会計の原則＞
・ バグは作り込まない．作りこんだバグは素早く摘出する．

＜上工程品質会計の原則＞
・ 作り込んだバグは次工程までに摘出する．
  - 作り込み工程で80%摘出
  - 次工程で残り20%摘出

＜テスト工程品質会計の原則＞
・ 作り込んだバグは，すべて摘出してから出荷する．

＜目標＞
・ 上工程バグ摘出率　80%

図 3-4　品質会計の原則

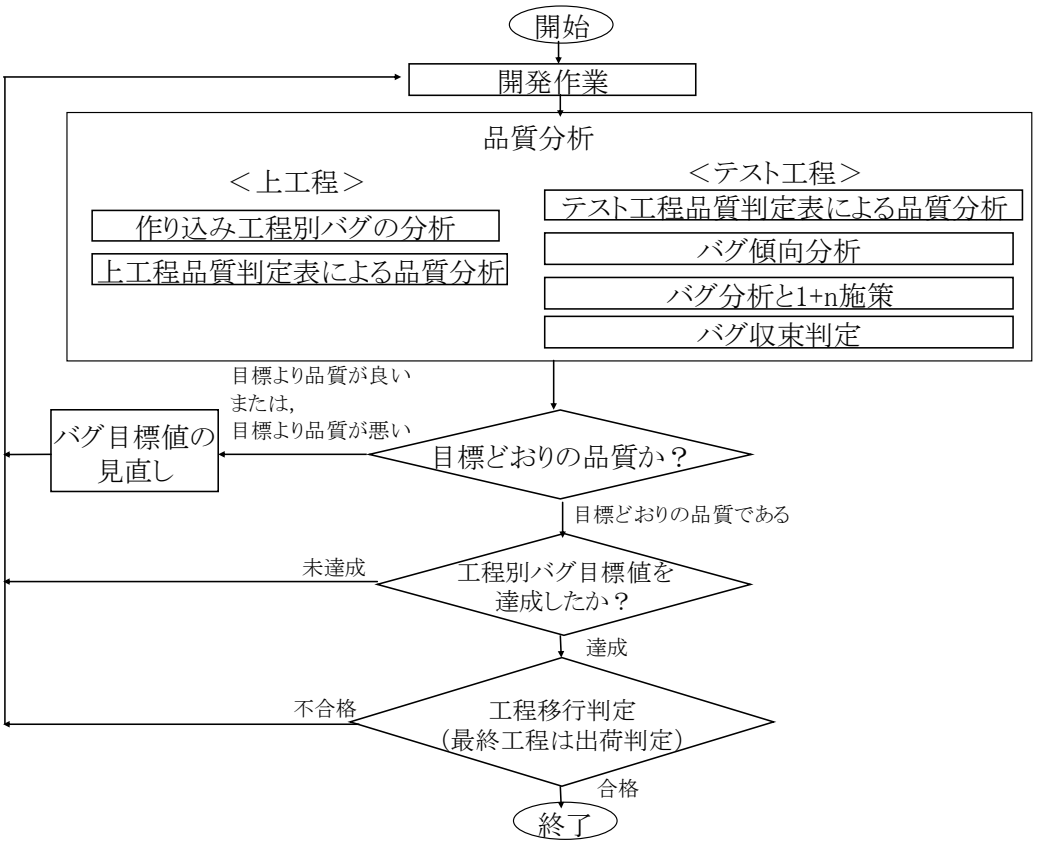


図 3-5　品質会計の手順

3.3　上工程における品質会計の適用方法

開発開始時に，回帰型バグ予測モデルを適用して，当該開発で作り込むと予測されるバグ目標値を設定する．開発開始後の上工程では，図 3-5 に示す品質会計の手順に従って品質会計を適用する．

3.3.1　開発開始時のバグ目標値の設定

バグ件数の予測には，「回帰型バグ予測モデル」を適用する．回帰型バグ予測モデルは，COCOMO モデル[3-11]を参考に，現場での要件を考慮して継続的に改善した結果，現在の形となった．品質会計で使用する回帰型バグ予測モデルを以下に示す．

$$B=C\cdot\prod_{i=1}^{n}\alpha_i\cdot S, \qquad (3\text{-}1)$$

B ：バグ予測値（件数），

C ： 標準バグ件数，

α_i ： バグへの影響要因($i=1,2,\ldots,n$)，

S ： 開発規模（KLOC）．

上記式（3-1）は，バグ予測値が，基本的には開発規模と標準バグ件数（定数）の積により求められ，バグへの影響要因により補正されることを示す．Cは，重回帰分析の結果得られる定数である．バグへの影響要因α_iは，バグ件数への影響度の強い要因 i を選択して使用する($i=1,2,\ldots,n$)．品質会計では，これまでの運用実績および現場への適用のしやすさを考慮した結果，現在は，開発者の「技術力」および開発の「難易度」の 2 種類の影響要因を原則として採用している．これらの影響要因は 5 段階で評価する．5 段階評価の各段階で使用する数値は，統計解析の結果に基づいて設定する．図 3-6 に，回帰型バグ予測モデルによるバグ予測値算出例を示す．

現場への適用のしやすさのため改善した例を示す．バグへの影響要因の 1 つである「技術力」は，当初，5 段階評価ではなく，計測可能なメトリクスとして，技術者の開発経験年数を用いた定量化を試みた．しかし，開発経験年数は技術力とは関連がないことがわかった．一方，開発現場のプロジェクトマネージャからは，担当する開発プロジェクトに対する見解をバグ予測に取り込みたいという要望が強かった．これらの経緯を経て，技術力を 5 段階評価とし，プロジェクトマネージャが評価する現在の方法とした．

回帰型モデルである式（3-1）は，ミドルソフトウェアや OS などのソフトウェア製品毎に，蓄積された過去のデータを使用して求める．ソフトウェア製品毎に作成する理由は，ソフトウェア製品毎にプロジェクト特性やソフトウェア特性などが異なるためである．

さらに，上記の過去データを参考にして，今回の開発での改善目標を考慮し，「摘出工程別バグ目標比率」を設定しておく．摘出工程別バグ目標比率は，上工程バグ摘出率 80%を達成できるように設定することが原則である．式（3-1）により算出したバグ予測値に対して，摘出工程別バグ目標比率を乗じて，摘出工程別の「バグ目標値」を設定する（図 3-7 参照）．

3.3.2 バグの作り込みと摘出

品質会計では，バグを作り込み工程と摘出工程に分けて管理する．上工程品質会計では，特にこの作り込みと摘出の考え方が重要である．図 3-8 は，バグの作り込みと摘出の例である．

バグは，上工程の設計またはコーディングで作り込まれ，レビューまたはテストで摘出される．バグの作り込み工程を的確に判断できるかどうかが品質会計の適用効果を高める 1 つのポイントである．技術レベルの低い開発者の場合，プログラムを修正するという行為に惑わされてほとんどすべてのバグをコーディングバグと判断してしまうことがある．コーディング工程の作り込みバグが多い場合には，作り込み工程の再確認が必要である．一般に上流の設計工程ほど高い技術レベルをもつ開発者が担当することが多いため，バグ

＜回帰型バグ予測モデル＞

$B = C \cdot \alpha_1 \cdot \alpha_2 \cdot S$

- B ：バグ予測値
- C ：標準バグ件数　C=15.0とする
- α_1：技術力(右表参照)
- α_2：難易度(右表参照)
- S ：開発規模(KLOC)

技術力

段階	技術力（$\alpha 1$）	数値
5	技術力がある	0.8
4	やや技術力がある	0.9
3	普通	1.0
2	やや技術力がない	1.1
1	技術力がない	1.2

難易度

段階	難易度（$\alpha 2$）	数値
5	難しい	1.2
4	やや難しい	1.1
3	普通	1.0
2	やや易しい	0.9
1	易しい	0.8

＜対象プロジェクトの情報＞

- 計画開発規模:20.0KLOC
- 技術力(α_1):5(技術力がある)
- 難易度(α_2):4(やや難しい)

＜バグ予測値の算出＞

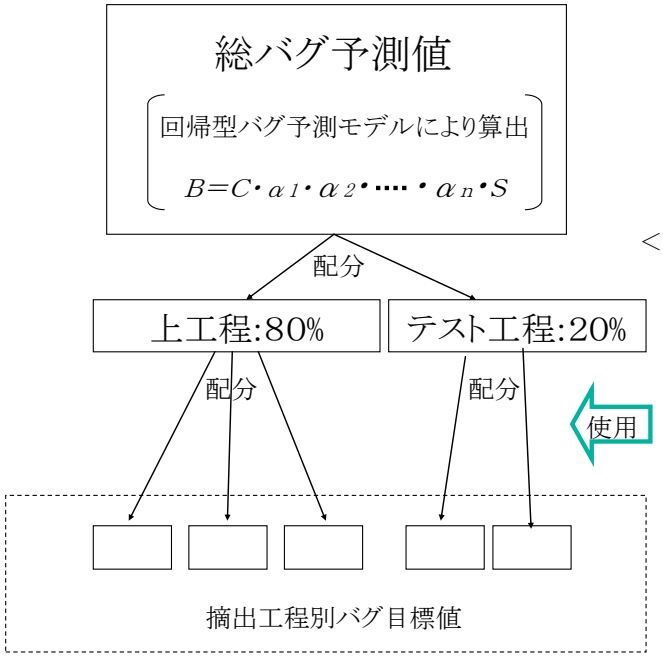
$B = C \times \alpha_1 \times \alpha_2 \times S$

$= 15.0 \times 0.80 \times 1.10 \times 20.0$

$= 264$

∴ バグ予測値＝264件

図 3-6　回帰型バグ予測モデルによるバグ予測値算出例



＜摘出工程別バグ目標比率(例)＞

摘出工程	目標比率(%)
基本設計工程	4
機能設計工程	18
詳細設計工程	20
コーディング工程	40
テスト工程	18
計	100

注意:上工程バグ摘出率80%が達成できるように設定する

図 3-7　工程別バグ目標値の設定方法

作り込み工程の誤判断は少ないが，下流の工程に進むにつれてさまざまなレベルの開発者が関与するため，バグ作り込み工程の判定妥当性を常に注意する必要がある．バグ作り込み工程の誤判断が多いと，誤った品質状況判断をしてしまう危険性がある．

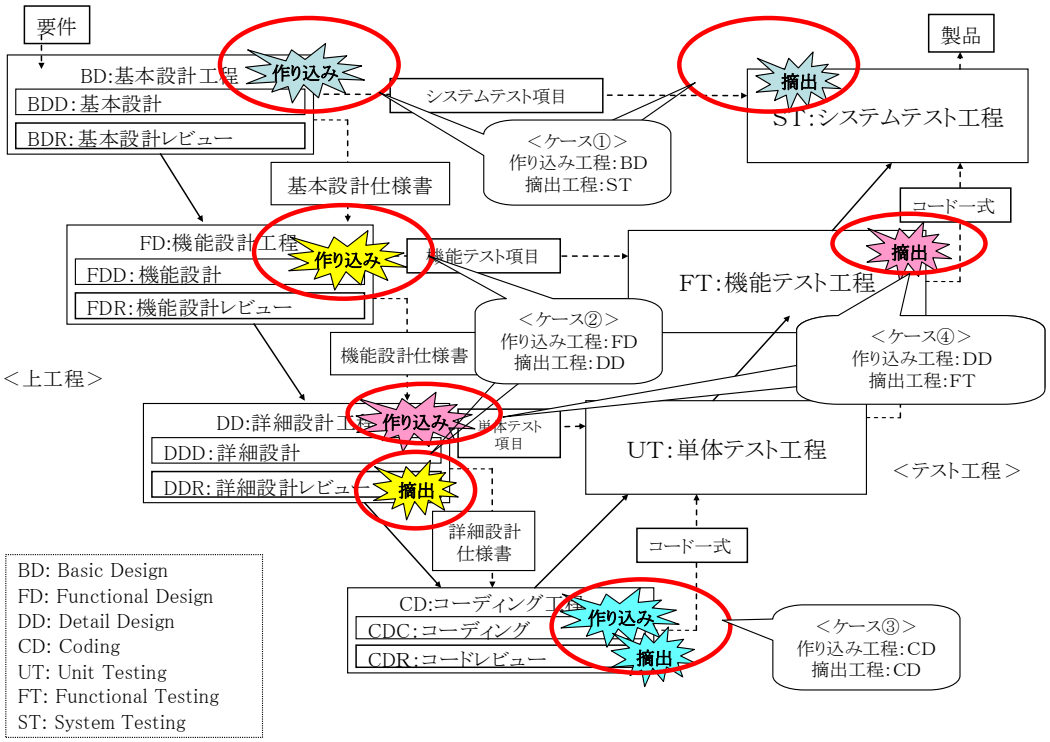


図 3-8　バグの作り込みと摘出（例）

3.3.3　作り込み工程別バグの分析

「作り込み工程別バグの分析」では，レビュー実施毎の形式でデータを整理する（図 3-9 参照）．ここでの分析観点は，以下の 3 点である．

① レビュー回数に対する摘出されるバグ件数の推移

- レビュー回数の増加につれて，バグ件数の合計が増加または横ばいの場合は，そのままレビューを継続しても成果が出ない危険性がある．レビューを一旦打ち切って，その原因を分析する．

② 前工程の作り込みバグの摘出状況

- 品質会計の原則を実現するためのおおまかな目標値では，作り込み工程で 80%，次工程で残り 20%を摘出するとしている．この目標値を超えて前工程の作り込みバグが摘出されている場合は，前工程の成果物の作り込み品質が悪いと判断し，前工程に戻って前工程の成果物を確認する．
- 作り込み工程で 80%，次工程で残り 20%を摘出するという目標値を達成すれば，上

工程終了時に残存するバグは，コーディング工程の作り込みバグ 20%のみとなる．よって，品質会計の目標である「上工程バグ摘出率 80%」を確実に達成することができる．

③ 前工程より上流で作り込まれたバグの摘出状況

- 品質会計の原則に則ると，前工程より上流で作り込まれたバグは，既に摘出されているはずである．それにもかかわらず，前工程より上流で作り込まれたバグが摘出される場合は，その作り込み工程の成果物の品質が悪いと判断し，その作り込み工程の成果物を確認する．

上述の 3 つの観点から分析することにより，当該工程より前の上流工程での成果物の品質問題を早期に検出して対処することが可能となる．特に，②や③で得られる結果は，上工程の設計段階で作り込まれる問題であるため，一般に結合テスト以降でなければ気がつかないことが多い．それを上工程で検出できることが，品質会計の大きな利点である．また，バグが多く作り込まれる工程は当該プロジェクトの弱点であるため，当該工程の問題点を集中的に解決することにより，大きな品質改善効果が期待できる．

	詳細設計	レビュー（1回目）	レビュー（2回目）	…	レビュー（n回目）	累計
基本設計バグ	0	0	2	…	0	2
機能設計バグ	8	2	5	…	0	18
詳細設計バグ	－	13	16	…	5	38
合計	8	15	23	…	5	58
累計	8	23	46	…	58	－

＜分析観点＞

1. レビュー回数を重ねるにつれて，摘出されるバグ数は減少しているか
2. 前工程の作り込みバグが多く摘出されていないか
3. 前工程より上流で作り込まれたバグが摘出されていないか

図 3-9　作り込み工程別バグの分析のためのデータ表（詳細設計工程の例）

3.3.4　品質判定表による品質分析

品質判定表には，上工程品質判定表とテスト工程品質判定表の 2 種類がある．品質判定

表は，レビュー工数またはテスト工数に対する摘出バグ数の状況を分析することにより，当該時点の計画に対する実績の品質を判断する．上工程においては，レビュー工数，テスト工程ではテスト工数を使用する．さらに，バグ目標値の見直しのために，バグ目標値の見直し式を提供している（図 3-10 参照）．

品質判定表		レビュー工数/KLOCまたはテスト工数/KLOC		
		実績＜計画−a％	計画−a％≦実績 ≦計画+a％	計画+a％＜実績
摘出バグ数/KLOC	実績＜計画−a％	品質を判断する時期ではない ⇒レビュー継続	計画より品質が良い ⇒①式で見直し	計画より品質が良い ⇒②式で見直し
	計画−a％≦実績≦計画+a％	計画より品質が悪い ⇒①式で見直し	品質は計画通り	品質は計画通り
	計画+a％＜実績	計画より品質が悪い ⇒①式で見直し	計画より品質が悪い ⇒①式で見直し	計画より品質が悪い ⇒②式で見直し

＜バグ目標値見直し式＞

$$\text{①式:新バグ目標値}^{1} = \text{旧バグ目標値}^{1} \times \frac{\dfrac{\text{レビューまたはテストによる摘出バグ数の当該工程実績値}}{\text{レビューまたはテスト工数の当該工程実績値}}}{\dfrac{\text{レビューまたはテストによる摘出バグ数の当該工程目標値}}{\text{レビューまたはテスト工数の当該工程目標値}}}$$

$$\text{②式:新バグ目標値}^{1} = \text{旧バグ目標値}^{1} \times \frac{\dfrac{\text{レビューまたはテストによる摘出バグ数の当該工程実績値}}{\text{開発規模}^{2}}}{\dfrac{\text{レビューまたはテストによる摘出バグ数の当該工程目標値}}{\text{計画開発規模}}}$$

(注意)・バグ目標値[1]は，上工程では総バグ目標値，テスト工程ではテストバグ目標値を使用する．
・開発規模[2]は，実績開発規模の確定時は実績開発規模，未確定時には計画開発規模を使用する．

図 3-10　品質判定表

レビュー工数およびテスト工数は，別途提供する回帰型の工数予測モデルにより計画値を設定する．品質判定表のしきい値 a%は，20%を基準として，組織の実力に応じて設定する．品質会計の適用実績期間が短いと，計画に対する実績のばらつきが大きくマネジメント精度が低くなる傾向があるため，a を大きい値に設定する．品質判定表による品質分析は，各工程終了時に実施されることが多い．工程途中で実施する場合には，品質判定表の計画値に進捗率を乗じた値を使用する．

レビューおよびテスト方法が従来と変わらないと仮定した場合，レビュー工数およびテスト工数は，バグ摘出努力の程度を表し，摘出バグ数は，レビューまたはテストの結果を表す．品質判定表による品質分析の狙いは，計画時に考慮していなかった，開発途中に発生する品質上の変化の把握と対応である．この「変化」は，レビューまたはテストの状況に最も現れると考えられる．その変化の把握の考え方は以下のようなものである．

・　計画通りの品質であれば，レビューまたはテスト工数とそれに対する摘出バグ数の実績値は，計画値に沿った数値が計測されるはずである．

・ 計画よりも品質が悪い場合は，計画したレビューまたはテスト工数に対して，摘出バグ数の実績値は大きい数値を示すと考えられる．
・ 計画よりも品質が良い場合は，計画したレビューまたはテスト工数に対して，摘出バグ数の実績値は小さい数値を示すと考えられる．

「変化」の把握の結果に基づいて，必要なら図 3-10 に示すバグ目標値の見直し式を使用してバグ目標値を見直す．バグ目標値の見直し式では，過去の現場での適用結果に基づいて，2 種類の式を提供している．①式は，レビューおよびテスト方法が従来と変わらないと仮定した場合に，単位時間当りのレビュー工数またはテスト工数に対する摘出バグ数の，目標値に対する実績値の比が，計画した品質に対する作り込んだ品質の比を表すものと考えて見直す方法である．レビュー工数またはテスト工数の実績が計画よりもかなり上回る場合は，レビューまたはテスト方法に変化があったと考えられるため，①式の考え方は適さない．その場合は，②式で示すように，作り込んだ品質は単位規模当りの摘出バグ数で表されると考え，その目標値に対する実績値の比が，計画した品質に対する作り込んだ品質の比を表すものと考えて見直す．

ソフトウェア開発では，要求仕様の大幅変更，開発規模の増大，開発プロジェクトの構成員の入れ替えなど，開発途中にさまざまな事象が発生する．品質判定表による品質分析の考え方は，品質会計での品質分析の基本的な考え方を示すものである．開発の現場では，この基本的な考え方を念頭におき，個々の開発条件を考慮して品質分析する．

3.4 テスト工程における品質会計の適用方法

テスト工程開始時には，テスト開始時点に残存していると予測されるテストバグ目標値を設定する．テストバグ目標値は，バグ目標値から上工程で摘出したバグ数を除した数値である．テスト工程においても上工程と同様に，図 3-4 に示す品質会計の手順に従って品質会計を適用する．テスト工程途中では，テスト工程品質判定表による品質分析を中心に品質管理を実施する．本章では，品質会計の特徴であるテスト完了判断を中心に論述する．また，テスト工程完了判断で適用する「バグ分析と 1+n 施策」は，なかでも重要な技術であることと，組織的な改善においても効果を発揮するため，本章ではその概要を述べるにとどめ，次章に詳しく論述する．

3.4.1 品質会計におけるテスト完了判断

品質会計では，計画したテストの終了がテスト完了ではない．計画時のテストだけでは，何らかのテスト漏れがあることが経験上非常に多い．計画したテストをほぼ終了した時点

で品質分析し，出荷可能な品質確保へ向けて，残存課題の把握と解決に取り組むことが求められる．そのために，品質会計では，テストがほぼ終了した時点において，「バグ傾向分析」，「バグ分析と 1+n 施策」，および「バグ収束判定」の 3 つの技法を併用して対応する．

「バグ傾向分析」では，テストで摘出したバグ全体をさまざまな角度から分析することにより，設計やレビュー・テストにおける考慮漏れや開発上の弱点を把握し，それに対する追加レビュー・テストを実施する．「バグ分析と 1+n 施策」では，システムテスト以降に摘出した重要なバグに注目し，その作り込み原因と見逃し原因を分析することにより，細かい考慮漏れや作業上の問題を把握する．その分析結果に基づき，追加レビュー・テストを実施する．これらの追加レビュー・テストの結果は，再度品質分析し，新たな残存課題が検出された場合には，さらに追加レビュー・テストを繰り返す．一方，「バグ収束判定」では，テスト工程全体のテスト実施率に対する累積バグ数の推移を分析することにより，収束判定する．これらの 3 つの技法による品質分析の結果，関係者全員が，いずれの結果とも問題なしと合意したとき，残存課題はないと判断し，テストを終了する．これが品質会計におけるテスト完了判断方法である．

3.4.2　バグ傾向分析

バグ傾向分析とは，摘出バグをさまざまな分析観点から区分して観察する分析手法である．計画したテストをほぼ終了した時点に適用するほか，上工程終了時の品質分析にも適用する．分析対象ソフトウェアの特性に合わせた分析観点の設定が，バグ傾向分析の分析レベルを左右する．品質会計では標準的な分析観点を整備している（表 3-1 参照）．

例えば，分析対象ソフトウェアの機能毎にバグ数を開発規模で正規化することにより，機能毎の単位規模当りのバグ数を比較し，バグの多い機能を抽出することができる．さらに，バグを顧客視点での重要度別に分類することにより，顧客視点での機能毎のバグ数を比較できる．その結果に基づいて優先順位を付けて機能毎に品質向上施策を実施することにより，効果的な品質向上を実現できる．表 3-1 に加えて，個々の分析対象ソフトウェアの内容に踏み込んだ分析観点を追加することにより，さらに説得力のある品質分析を実施できる．

図 3-11 に，バグ傾向分析の例を示す．これは，計画したシステムテストを終了した時点における機能テストおよびシステムテストで摘出したバグに対する分析事例である．4 種類のバグ傾向分析を経て，B-2 機能の 3 種類のバグ作り込み原因に対して追加確認が必要との結論を導き出した．本事例はあくまで一例であり，どの分析対象ソフトウェアでも同じ分析をすればよいということではないことに注意してほしい．

表 3-1　バグ傾向分析の分析観点

分析観点	内容
正規化	規模などの単位当りでバグ数を正規化する. バグ数は規模の大小の影響を受けるが, 単位規模当りのバグ数であれば, 数値の比較が可能となる.
構成する機能	バグを, 構成する機能ごとに分類する. どの機能のバグ数が多いかを把握することができる.
バグ作り込み工程	摘出したバグを作り込み工程で分類する. 設計・コーディングのどの工程に問題があったかを知ることができる.
バグ重要度	バグを利用者に与える重要度(致命的, 重要, 軽微など)で分類する. 重要なバグが出ているかを把握することができる.
バグ作り込み原因	バグを作り込んだ原因で分類する. 一般的な作り込み原因には, 設計ミス・考慮漏れ・単純ミス・手順ミス・仕様理解不足・デグレードなどがある. 作り込み原因は, 分析対象ソフトウェアに応じて設定したほうがよい.
発生条件	バグを発生条件(正常系, 異常系, タイミング, 組合せ, 限界値など)で分類する.
発生現象	バグを発生現象(結果異常, システム停止, データ破壊など)で分類する.
摘出者	バグを摘出者で分類する. 意図せずバグが摘出された件数を把握できる.

3.4.3　バグ分析と 1+n 施策

バグ分析と 1+n 施策とは，バグ 1 件に対して，そのバグを作り込んだ根本原因と見逃した根本原因を分析し，その根本原因に対する追加レビュー・テストを実施することにより，同種バグを摘出する技法である（図 3-12 参照）．同種バグとは，そのバグと同じ原因によって作り込まれた，または見逃したバグをいう．1 件のバグに対して同種バグを n 件摘出することから，根本原因に対する追加レビュー・テストを 1+n 施策と呼ぶ.

バグ分析と 1+n 施策は，開発途中においては，システムテスト以降に摘出された重要なバグに対して実施する．また，出荷後の品質向上対策として，全出荷後バグに対してバグ分析と 1+n 施策を実施する．バグ分析と 1+n 施策の詳細については，次章を参照していただきたい.

①作り込み工程別バグ数を機能別に分析

⇒B-2機能は，バグ数が多い．FDバグも摘出されている．

＜詳細機能別作り込みバグ（FT～ST）＞

	B-1機能	B-2機能	B-3機能	B機能全体
BDバグ	0	0	0	0
FDバグ	0	2	0	2
DDバグ	0	3	1	4
CDバグ	1	15	4	20
合計	1	20	5	26

②規模当りのバグ数を機能別に分析

⇒B-2機能は，規模が大きく，規模当りのバグ数が多い。

＜詳細機能別の規模当たりのバグ（FT～ST）＞

B-2機能はB機能の中核部分を実現しているため，規模当りのバグ数が多いと思われる．

	B-1機能	B-2機能	B-3機能	B機能全体
規模（KLOC）	2.9	15.3	7.2	25.4
件/KLOC	0.34	1.31	0.69	1.02

③重要度別バグ数の機能別分析

・　B-2機能は，致命的2件・重要8件のバグが摘出されている．

⇒B-2機能へ絞った品質分析を実施するべきと判断

＜詳細機能別の重要度別バグ（FT～ST）＞

致命的なバグが後工程に摘出されるのは問題である．

重要度	B-1機能	B-2機能	B-3機能	B機能全体
致命的	0	2	0	2
重要	0	8	1	9
軽微	1	10	4	15
合計	1	20	5	26

④問題と思われる機能に絞ったバグの作り込み原因別分析

結論：B-2機能のこれらの観点の追加確認が必要である．

＜B-2機能の作り込み原因別バグ（FT～ST）＞

画面表示系の　テスト強化の結果である．十分性の確認が必要である．

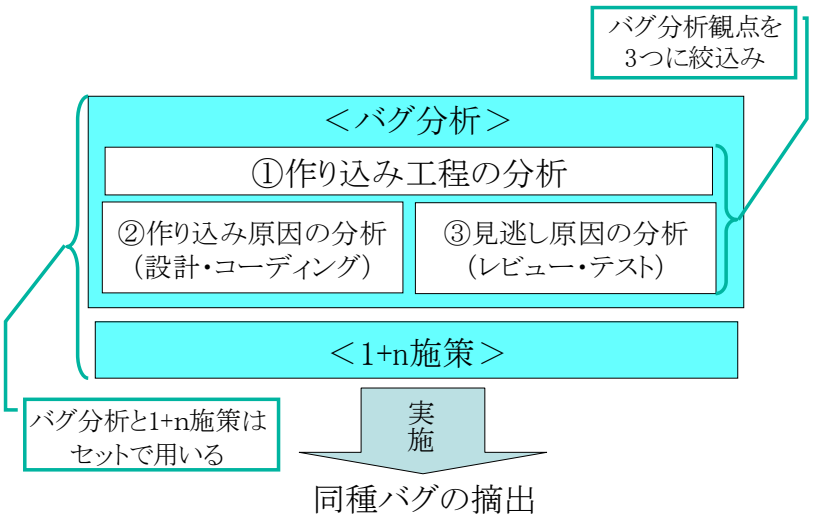
バグの作り込み原因	BDバグ	FDバグ	DDバグ	CDバグ	合計
GUI表示形式の不正			1	8	9
他機能とのインタフェース考慮不足		2	1	3	6
異常系の考慮不足			1	4	5
既存仕様の誤解					0
設計仕様書の記載不足					0
合計	0	2	3	15	20

FDレビュー時にB機能全体をよく知る技術者がたまたま欠席している．他レビューでカバーしたつもりだったが不十分のようである．

うち1件は第3者評価によるバグであり，まだ残存している可能性がある．

図 3-11　システムテスト終了時のバグ傾向分析例

（分析対象バグは，機能テストおよびシステムテストで摘出されたバグである）



■適用場面:テスト終盤の重大バグおよび出荷後バグに対して適用

図 3-12　バグ分析と 1+n 施策

3.4.4　バグ収束判定

バグ収束判定は，ソフトウェア信頼度成長モデル[3-12]を現場で適用するなかで，考案した方法である（図 3-13 参照）．テスト終盤の傾向に注目することによって，現場での一意な収束判定を実現した．テストの傾向とは，テスト項目数に対する累積摘出バグ数を用いた信頼度成長曲線の傾きである．テスト終盤の範囲は，全テスト項目数を 100 としたとき，最後の 80%～100%を基準として，分析対象ソフトウェアにあわせて具体的に設定する．

バグ収束判定では，以下の式（3-2）式により算出されるバグ収束率を使用する．

$\alpha = \Delta 1 / \Delta 0$,　　（3-2）

α　：バグ収束率,

$\Delta 1$：テスト終盤の摘出バグ数／テスト終盤のテスト項目数,

$\Delta 0$：テスト全体の摘出バグ数／テスト全体のテスト項目数.

式（3-2）により算出されるバグ収束率 α は，別途準備する収束基準値と比較し，基準値内であれば収束していると判定する．この収束基準値は，当該組織の過去のプロジェクトにおけるバグ収束率と出荷後品質の関係を分析することにより設定するもので，個々の組織のデータ蓄積により得られる経験値である．収束基準値は，一般的には 0.5 より小さい値となる．

バグ収束判定では，ソフトウェア信頼度成長モデルを用いる分析方法がよく知られている[3-12]．この方法は，適用するモデルやその当てはめ方により収束判定結果が異なる．このため，常に一定レベルの品質確保を要求される開発現場では，適用のための改善が必要

となる．品質会計のバグ収束判定は，その課題を解決するために考案したものであり，誰が判定しても同じ結果となる．

品質会計では，計画したテストがほぼ終了した段階で品質分析し，残存課題に対する追加テストを実施することが求められる．よって，バグ収束判定の判定対象となるテスト終盤は，この追加テストの状況を判定することになる．

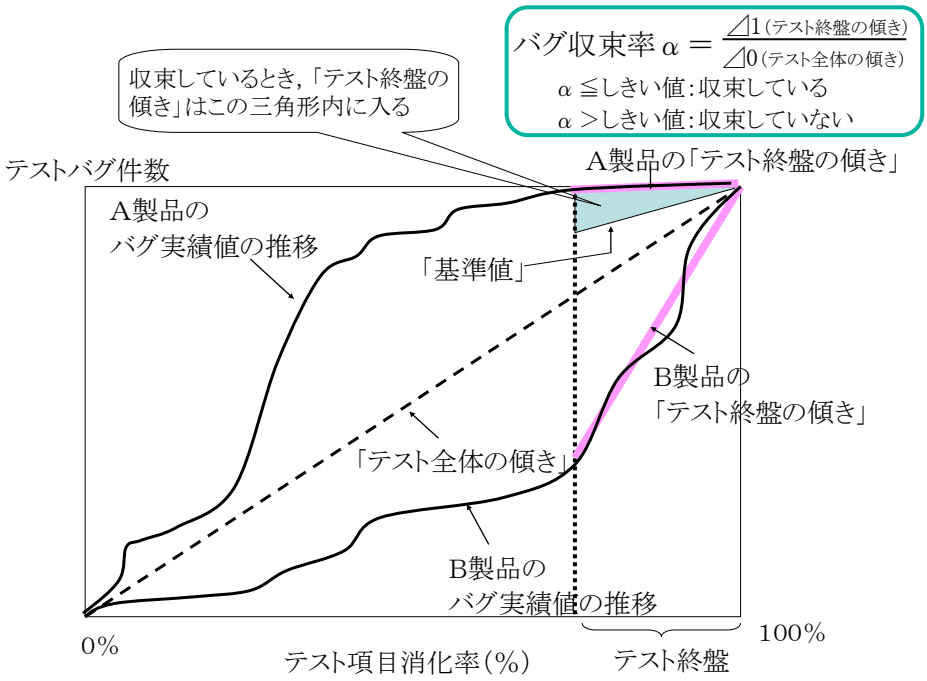


図 3-13　バグ収束判定の考え方

3.5　現場主義の重視

現場主義とは，机上の空論ではなく，「現地で，現物を見て，現実を知る」ことを重視する考え方をいう．品質会計は，現場主義の実践を強く要求する．

品質会計は，データに基づいて品質管理を実施する．しかし，適用場面においては，計測されたデータだけに頼ることなく，現場で実際に起こっている事実の確認を重要視している．その理由は，データだけに頼ると，事実を誤認する恐れがあるからである．上工程においては，設計仕様書やレビュー記録などの実際の成果物を確認し，設計の内容や設計仕様書の記述の詳細度，レビューアの適切性，レビュー内容やレビューにより摘出されたバグの内容などを確認することが重要である．テスト工程においては，テスト仕様書やテ

スト結果，テストプログラムなどを確認するとともに，実際に摘出されたバグの内容を確認することにより，品質確保の状況を判断する．さらに，開発のどの段階にあっても，現場の開発者のモチベーションやチームワークなどの人間的な要素を確認することが非常に重要である．

現場を確認した結果と，品質会計技法を適用して得られるデータによる分析結果を総合して，真の因果関係を分析して品質状況を判断する．現場での状況確認の結果と，データによる品質分析結果が異なる場合は，その食い違いの原因を分析する．ソフトウェア開発では，さまざまな事象や新しい技術要素によって，常に今までにない問題が発生する可能性がある．現場を確認することにより，このような新たな要因によって引き起こされる問題を，早期に発見することができる．必要であれば，新たな要因の計測方法を考案する．これが，品質会計の進化を促してきた．

3.6　おわりに

本章では，品質会計の概要および各技法とその適用方法を論述した．品質会計は，ソフトウェアの品質問題を解決するために，ソフトウェア開発の現場で生まれ，その適用結果による見直しを繰り返しながら，20 年以上かけて構築した技法である．したがって，ソフトウェア開発の現場で実証された技術である．

品質会計は，ウォーターフォールモデルなどの計画駆動型プロセスを前提としており，エンタープライズ系や組込み系といったソフトウェア領域には依存せず適用可能である．品質会計の特徴は，①レビューでのバグ摘出による早期品質確保，②的確なテスト完了判断，という 2 点である．品質会計は，ソフトウェア開発中に作り込まれるバグを摘出目標管理することにより，品質が確かに作り込まれたことを説明する品質管理手法であり，出荷前全摘出バグ数に対するレビュー摘出バグ数の比率である上工程バグ摘出率 80%を目標とする．品質会計の適用による効果は，第 7 章で詳細を述べる．品質会計は，現在，NEC グループのソフトウェア開発技術者 6 万人が適用する標準的な品質管理手法となっている．

第4章　バグ分析と 1+n 施策

4.1　はじめに

ソフトウェア開発の現場では，ソフトウェアの品質確保やプロセス改善のために，発生した問題の根本原因を分析し，その根本原因を解決する取り組みが行われることがある．この根本原因分析は，「なぜなぜ分析」などの名称でよく知られている．なぜなぜ分析は，現場でよく使われる技法の 1 つであるが，実際に適用すると，根本原因にたどりつけずに，現場の不満を列挙するだけになってしまったりするなど，効果的に利用できているとは言えないのが実状である．

本章では，品質会計の技法の 1 つである「バグ分析と 1+n 施策」について，その技法を具体的に述べるとともに，その適用事例に基づき，本技法の特徴と効果を論述する．「バグ分析と 1+n 施策」は，「なぜなぜ分析」の課題を解決し，ソフトウェア開発用に最適化されたバグ原因分析技法である．品質会計では，なぜなぜ分析をバグ分析と呼ぶ．バグ分析はプロセス改善にも効果を発揮する技法であり，高品質ソフトウェア開発の実現には欠かせない技術である．

4.2　関連する技法と「バグ分析と 1+n 施策」の関係

根本原因分析の手法としては，トヨタ式「なぜを 5 回」[4-1]が最も有名である．これは，「なぜ」を 5 回繰り返すことにより，根本原因にたどり着く方法である．事実を重視し，ものごとの因果関係やその裏に潜む本当の原因を突き止める科学的アプローチである．トヨタ式「なぜを 5 回」は，ソフトウェア開発に限定した手法ではなく，広く一般に適用されている．このため，適用対象領域の特性を考慮したさまざまな応用型が見受けられる．

ソフトウェア開発領域に対する根本原因分析としては，PNA(Process Network Analysis)技法が提案されている[4-2]．これは，ソフトウェア開発のプロセスを強く意識した手法であり，マネジメントプロセスを常に意識して分析する点に特徴がある．

また，製品の信頼性確保のために，故障モードに着目しその原因を解析する手法として，FTA(Fault Tree Analysis)や FMEA(Failure Mode and Effects Analysis)などがある[2-12]．これらはソフトウェア開発においても適用されている．根本原因を解析するという意味で，FTA や FMEA も根本分析手法の一種である．

ソフトウェア開発の現場で最も普及しているのは，トヨタ式「なぜを 5 回」およびその応用型であろう．これらは，「なぜ」を繰り返すときの因果関係の注目の仕方に多少の違いは

あるが，「なぜ」を繰り返して根本原因にたどり着くという点では共通である．このため本論文では，以降，トヨタ式「なぜを 5 回」およびその応用型を総称して，「なぜなぜ分析」と呼ぶ．

ソフトウェア開発においては，バグのなぜなぜ分析の分析結果に基づいて，出荷済みのソフトウェアや出荷直前のソフトウェアの品質向上を目的として追加テストを実施することがある．このようなバグの根本原因に基づく追加テストによるバグ摘出を，「バグの横展開」という．バグの横展開の目的は，もとになったバグの根本原因と同じ原因で残存するバグを摘出することである．例えば，技術資料の調査不足がバグの根本原因の場合には，バグの横展開では，同じ技術資料の調査不足を原因として作り込まれたバグの摘出が目的となる．

バグの横展開は，一見，簡便な手法であるため，ソフトウェア開発の現場でよく利用されている．しかし，残念ながらバグの根本原因を分析し，潜在するバグを効率よく摘出するのは難しい．それが難しい理由は，バグの根本原因分析にある．バグの根本原因が 1 つであることはまれであり，ほとんどの場合，バグの原因が複数存在するためである[4-1]．このため，なぜなぜ分析をすればするほど，原因が枝分かれして，発散してしまうように見え，根本原因を特定することが困難になってしまうのである．

バグの根本原因は，固有根本原因と共通根本原因という 2 種類の根本原因に分類することができる．固有根本原因とは，そのバグを作り込み，見逃した直接の原因である．例えば，ある技術情報の欠落による設計ミスや，設計仕様書の記載漏れによるテスト工程項目漏れがこれにあたる．一方，共通根本原因とは，そのバグを作り込み，見逃した間接的な原因である．教育不足，組織標準が不十分といった原因が共通根本原因にあたる．バグの固有根本原因に対する未然防止策を取った場合には，対策実施後にもとのバグと同種類のバグが作り込まれたり見逃されることはなくなるはずである．しかし，バグの共通根本原因に対する未然防止策をとった場合には，間接的な対策になるため，その対策を実施しても，もとのバグと同種類のバグの作り込みや見逃しを確実に防止できるとは限らない．共通根本原因の未然防止策は，もとのバグと同種類のバグを作り込んだり見逃したりする確率を低減できるが，確実に防止できるわけではない．したがって，バグの横展開は，共通根本原因ではなく，固有根本原因に基づいて実施しなければならない．なぜなぜ分析は，固有根本原因と共通根本原因を区別せずに分析する．

「バグ分析と 1+n 施策」の「バグ分析」は，固有根本原因を分析するための技法である．「バグ分析と 1+n 施策」は，なぜなぜ分析をもとに確実に固有根本原因を分析できるよう根本原因分析方法を改良し，効果的にバグの横展開を実施できるように見直した技法である．

4.3 「バグ分析と 1+n 施策」技法の概要

4.3.1 品質会計との関係

第 3 章で述べたように，品質会計は以下の 2 つの特徴をもつ.

- レビューでのバグ摘出による早期品質確保
  - 「上工程品質会計」により，レビューで摘出したバグを作り込み工程と摘出工程の両面から管理
- 的確なテスト完了判断
  - 「バグ傾向分析」,「バグ分析と 1+n 施策」，および「バグ収束判定」の 3 つの技法の併用による残存課題の把握と解決

「バグ分析と 1+n 施策」は，品質会計の 2 つの特徴のうち，後者のテスト完了判断をするために使用される技法である．テスト完了判断は，計画したテストが完了した時点で，以下の 3 つの視点から分析して判断する．3 つの視点とも問題なしと評価したときに，テストを完了する．

- 「バグ傾向分析」技法は，系統的なテスト観点の漏れがないかを分析する．
- 「バグ分析と 1+n 施策」技法は，細かいテスト観点の漏れがないかを分析する．
- 「バグ収束判定」技法は，実際にバグはこれ以上摘出される可能性が低いかを分析する．

「バグ分析と 1+n 施策」は，テスト終盤に摘出された重大バグを対象として実施する．テストの終了段階まで重大なバグが残存しているということは，それまでの開発作業において作業の抜けや漏れの危険性を示唆するものと考える．このような細かいテスト観点の漏れがないことを分析できるようにするために,「バグ分析と 1+n 施策」を開発したのである．

4.3.2 基本的な考え方

バグ分析と 1+n 施策技法は，1 件のバグの固有根本原因を分析し，その固有根本原因に基づき，当該バグとの同種バグを摘出するための技法である．「同種バグ」とは，あるバグと同じ固有根本原因により，まだ当該ソフトウェアに潜在するバグをいう．

本技法では，バグの「作り込み工程」という考え方を用いる．バグの作り込み工程とは，当該バグを作り込んだ工程のことであり，第 2 章の図 2-2 に示すＶ字モデルである開発プロセスの上工程のうちいずれかの工程である．本技法では，固有根本原因を以下の 3 種類に限定する．

- 当該バグの作り込み工程の設計において，当該バグを作り込んだ原因（以降,「作り

込み原因」と呼ぶ）
- 当該バグの作り込み工程のレビューにおいて，当該バグを見逃した原因（以降，「レビューでの見逃し原因」と呼ぶ）
- テスト工程において，当該バグを見逃した原因（以降，「テストでの見逃し原因」と呼ぶ）

本技法において，固有根本原因を上記に限定する理由は，本技法の目的を，ある 1 件のバグに対応する同種バグの摘出としているためである．ある 1 件のバグに対応する同種バグが，当該ソフトウェアに潜在する直接的な原因は，上述する 3 種類の固有根本原因に限定される．

なぜなら，バグが作り込まれるのは，V 字モデルの上工程のうちいずれかの工程だからである．また，そのバグ分析と 1+n 施策が摘出されるのは，当該工程でのレビューおよび当該設計工程と対応するテスト工程におけるテストでしかないからである．V 字モデルは，対応する設計とテストの工程を視覚的に明示しているのが特徴である．V 字モデルを適用することにより，作り込んだ工程を特定できれば，見逃したテスト工程は V 字モデル上で対応するテスト工程と判断できる．本技法が V 字モデルを前提とするのは，このように V 字モデルがバグ作り込み工程とバグ摘出工程の特定に適しているからである．

また，固有根本原因以外に，技術者のスキルやチームのコミュニケーションなど開発プロセスを取り巻くさまざまな共通根本原因が考えられるが，それらはいずれもその 1 件のバグだけでなく他の潜在バグにも共通する原因である．このような共通根本原因は，本技法では根本原因の対象としない．その理由は，共通根本原因を対象とすると，幅広い範囲の根本原因を扱うことになり，逆に同種バグの摘出の焦点を的確に定めることができなくなるためである．

「バグ分析と 1+n 施策」技法のうち，「バグ分析」では，上述した 3 種類の固有根本原因を分析する．「1+n 施策」では，「バグ分析」の結果に基づいて追加レビューやテストの範囲と内容を立案し，これらを実施することにより，同種バグを摘出する．「1+n 施策」の「1」は分析対象バグ 1 件のことであり，「n」は摘出する同種バグの件数を指す（n≧1）．1+n 施策という名称は，1 件のバグが検出された場合には n 件の同種バグが潜在するはずであり，その n 件の潜在する同種バグを必ず摘出するという意味から名付けられた．「バグ分析」と「1+n 施策」は，セットで「バグ分析と 1+n 施策」として適用する．

4.3.3　適用方法

バグ分析と 1+n 施策は以下の特徴をもつ．

- バグの固有根本原因を分析して同種バグを摘出する技法のため，ピンポイントで問題を是正するのに適している．

・　バグ分析には一定の時間がかかる．

このため，本技法は，重大な問題に限定して適用するのが効率的である．こうした事情から，主にテスト終盤の重大な問題や，設計における重大な設計ミスに対して適用される．また，出荷後の客先で摘出されたバグに対しても適用される．

1 件の重大バグに対して，「バグ分析」を実施する．バグ分析では，「作り込み工程」，「作り込み原因」，「見逃し原因」の 3 つの観点から根本原因を分析する．次に，そのバグ分析結果に基づいて，n 件の同種バグを摘出するための 1+n 施策を立案し，同種バグを摘出する．

4.3.4　適用事例

バグ分析と 1+n 施策の効果を理解しやすくするために，本節では，バグ分析と 1+n 施策の適用事例について述べる（図 4－1 参照）．

この事例は，限界値処理のバグである．限界値処理のバグは，ソフトウェアでは発生しがちな問題である．このような場合，追加テストは他の限界値処理部分に対して，その処理の妥当性を確認するために実施されることが多い．このとき，期待する同種バグは，他の限界値処理のバグである．

しかし，図 4-1 の開発上の経緯を見ればわかるとおり，実はこのバグは詳細設計工程終了時の予期せぬ技術者交代により引き起こされた問題である．このバグの固有根本原因は，単なる限界値処理の実装ミスではなく，旧技術者しか知らない書かれざる仕様の存在である．書かれざる仕様とは，技術者の頭の中には存在するが，設計仕様書には記載されていない仕様のことである．図 4-1 の同種バグは，書かれざる仕様を起因としたバグである．書かれざる仕様が他に存在するかどうかは旧技術者しか知らないため，1+n 施策には新技術者がコーディングした範囲を旧技術者がレビューする必要がある．その目的は，設計仕様書に記載していない他の書かれざる仕様の摘出である．このように，表面的な事象では分析できない固有根本原因を分析し，その同種バグを摘出できることが本技法の特徴である．

一方，書かれざる仕様の存在は，どの組織でも発生しうる問題である．設計仕様書の記載詳細度は，組織の技術者の技術レベルや開発対象領域の知識レベルに応じて，組織毎に決定する必要がある．設計仕様書の記載詳細度の問題は，他のプロジェクトでも発生しうる共通根本原因である．なぜなぜ分析では，固有根本原因である予期せぬ技術者の交代と同時に，共通根本原因である設計仕様書の記載詳細度も根本原因の 1 つとして分析される．なぜなぜ分析は，分析を進めるほど，原因が枝分かれして，複数の根本原因にたどり着くのである．本事例のような出荷前の段階では，出荷品質を確保するために，固有根本原因を分析して効率的に同種バグの摘出をすることが求められる．なぜなぜ分析では，固有根本原因と共通根本原因を区別なく分析していくため，効率が悪い．「バグ分析と 1+n 施策」

技法は，このようななぜなぜ分析の問題を解決し，同種バグを効率よく摘出できるように工夫を重ねた技法である．

＜バグ内容＞
- ある入力フィールドへ範囲外の値を入力したところ，プログラムが異常終了

＜このバグの直接的な原因＞
- 範囲外の値かどうかを判定するロジックで，割る数が0（ゼロ）になるケースの処理実装漏れ

＜開発の経緯＞
- 旧担当者は，割る数が0（ゼロ）になるケースがあることを知っていた．しかし，細かい機能のため，そこまで記載しなかった．
- 通常，設計と実装は同一開発者が担当するので，細かい仕様を記載しなくても今まで問題は出なかった．
- たまたま事情により，詳細設計完了後，実装に移行するとき担当者が急に交代した．
- 新担当者は，詳細設計仕様書に基づいて実装した．
- 旧担当者から新開発者への引継ぎの時間はとれなかったが，詳細設計書のレビューアが同じ開発グループにいるのでカバーできると判断した．

<u>バグ分析</u>

＜真の原因＞　書かれざる仕様
- 作り込み工程：詳細設計工程（DD）
- 作り込み原因：詳細設計仕様書に割る数が0（ゼロ）になるケースの記載がなかったこと
- レビューでの見逃し原因：旧開発者がコードレビューに参加しなかったこと
- テストでの見逃し原因
  - 割る数が0（ゼロ）になるケースのテスト項目があがっていなかったこと
  - ただし，割る数が0のケースが存在することは詳細設計仕様書には記載されていない．記載されなければ（設計されなければ）テスト項目にあげることはできず，テストできない．よってこのケースでは，レビューで摘出すべきだった．

<u>1+n施策</u>

＜レビューによる1+n施策＞
- レビュー範囲：新担当者がコーディングした全箇所
- 実施内容：旧担当者によるレビュー
- 確認すべき内容：旧担当者が詳細設計仕様書に仕様を記載しなかったことによる実装漏れは他にないかを確認する．

※実装漏れが検出された場合は，その部分を追加実装する．

＜テストによる1+n施策＞
- テスト範囲：旧担当者が詳細設計仕様書に仕様を記載しなかったことによる全実装漏れ（レビューによる1+n施策で検出された実装もれを含む）
- 実施内容：実装もれ部分のテスト
- 確認すべき内容：実施すべき全テスト項目を実施し，正しく実装されていることを確認する．

※全限界値処理が正しく実装されているかの確認についても，上記1+n施策とは別に実施する．

図 4-1　「バグ分析と 1+n 施策」の適用事例

4.4　「バグ分析」の適用方法

本節では，「バグ分析」の適用方法を述べる．実際の開発現場においては，過去のバグ分析結果を蓄積して，発生頻度の高い根本原因を掲げたバグ分析シート（図 4-2 参照）を使って，根本原因の分析をしやすくする工夫をしている．バグ分析は，原則として分析対象バグを作り込んだ当事者が分析する．それは，当事者でないと知りえない事実があるためである．

<table>
<tr><td colspan="6">バグ分析シート</td></tr>
<tr><td colspan="2">管理番号:</td><td colspan="4">バグ概要:</td></tr>
<tr><td colspan="3">製品名(プロジェクト名):　　発生版数:</td><td colspan="3">発生日:　　登録日:</td></tr>
<tr><td colspan="3">開発グループ名:　　報告名:</td><td colspan="3">責任者:</td></tr>
<tr><td colspan="6">内容:</td></tr>
<tr><td colspan="6">期待結果:</td></tr>
<tr><td colspan="6">摘出者名:</td></tr>
<tr><td colspan="6">重要度:　　現象区分:　　バグ箇所:</td></tr>
<tr><td colspan="3">作り込み工程</td><td colspan="3">□</td></tr>
<tr><td colspan="3">＜原因分析＞</td><td>技術面</td><td colspan="2">進め方</td></tr>
<tr><td rowspan="9">作り込み原因</td><td colspan="2" rowspan="8">概要チェック</td><td>□社外技術調査不足</td><td colspan="2">□必要な開発工程を省略</td></tr>
<tr><td>□関連技術(社内)調査不足</td><td colspan="2">□仕様書に記述するべき事項の記述が不十分</td></tr>
<tr><td>□プログラミング技術不十分</td><td colspan="2">□コーディング規則に則っていない</td></tr>
<tr><td>□コーディングノウハウ不足</td><td colspan="2">□担当間の連携/引継ミス</td></tr>
<tr><td>□設計時記載不十分</td><td colspan="2">□関連部門との調整不十分</td></tr>
<tr><td>□仕様書記載誤り</td><td colspan="2">□開発計画または計画の見直し不十分</td></tr>
<tr><td>□インタフェースルール不足</td><td colspan="2">□その他(　　)</td></tr>
<tr><td>□その他(　　)</td><td colspan="2"></td></tr>
<tr><td colspan="2">詳細</td><td colspan="3"></td></tr>
<tr><td rowspan="11">見逃し原因</td><td rowspan="6">レビュー</td><td rowspan="5">概要チェック</td><td>□レビューをしていない</td><td colspan="2">□レビュー実施方法に問題</td></tr>
<tr><td>□レビュー不足</td><td colspan="2">□レビューが完了していない</td></tr>
<tr><td>□レビューチェック観点漏れ</td><td colspan="2">□指摘事項の修正漏れ･誤り</td></tr>
<tr><td>□レビューアの問題</td><td colspan="2">□レビュ計画が不十分</td></tr>
<tr><td>□その他(　　)</td><td colspan="2">□その他(　　)</td></tr>
<tr><td>詳細</td><td colspan="3"></td></tr>
<tr><td rowspan="5">テスト</td><td rowspan="4">概要チェック</td><td>□テスト項目に挙がっていない</td><td colspan="2">□テスト項目実施方法に問題</td></tr>
<tr><td>□テスト項目の誤り</td><td colspan="2">□テスト項目実施漏れ</td></tr>
<tr><td>□その他(　　)</td><td colspan="2">□担当間の連携/引継ミス</td></tr>
<tr><td></td><td colspan="2">□テスト計画が不十分</td></tr>
<tr><td></td><td colspan="2">□その他(　　)</td></tr>
<tr><td>詳細</td><td></td><td colspan="2"></td></tr>
</table>

図 4-2　バグ分析シート

4.4.1　作り込み工程

バグ分析に先立ち，作り込み工程を分析する．作り込み工程を特定できるようにするには，開発プロセスの各設計工程において設計すべき内容が，明確に定義されている必要がある．

作り込み工程は，バグ内容によって，すぐに特定できる場合と，作り込み原因を分析しながら特定していく場合がある．作り込み工程をすぐに特定できる場合は，当該工程で作成した設計仕様書の記述内容の確認から分析を開始するため，作り込み原因の特定がしやすくなる．

4.4.2　作り込み原因

作り込み原因には，技術面の根本原因と，作業の進め方に関する根本原因が考えられる．根本原因は，技術面と作業の進め方の両方に原因がある場合や，一方だけの場合など，さまざまである．

さらに，作り込み原因を特定しやすくするために，過去に発生したバグの作り込み原因を蓄積した知識ベースを用意して利用している．

4.4.3　見逃し原因

レビューおよびテストの両方で摘出できる可能性がある場合もあれば，バグの内容によっては，レビューまたはテストのいずれかでしか摘出できないだろうと考えられる場合もある．見逃し原因でも技術面と進め方の両面から根本原因を分析する．

4.4.4　バグ分析シートを用いたバグ分析

上述した 4.4.1～4.4.3 の分析をするために，図 4-2 のバグ分析シートを用いる．バグ分析シートは，過去のバグ分析結果を蓄積して，発生頻度の高い根本原因を汎用化して「概要チェック」欄に例示している．バグ分析シートを適用した結果，根本原因の少なくとも 6 割は，概要チェック欄の例示に当てはまることがわかっている．

利用者は，バグ分析シートの「概要チェック」の項目を見ながら，１つ１つ当該バグの根本原因に該当するかを検討する．さらに，その根本原因が発生した理由の影響範囲と内容を具体的に「詳細」欄へ記述する．詳細欄へ記述する理由は，具体的な 1+n 施策へ結びつけるためである．単に概要レベルの根本原因がわかっただけでは，効果的な 1+n 施策を立案するのは困難である．例えば，「社外技術調査不足」が作り込み原因の場合，調査すべき技術，調査方法，調査時期，調査の適任者などを具体的に記述する．これにより，1+n 施策の立案において，調査範囲や調査内容を具体的に特定することができるようになる．

4.4.5　真の原因を判断する基準

バグ分析のゴールは，それをしていたら当該バグを作り込むことはなかった，見逃すことはなかったと判断できる原因に到達したときである．詳細欄に，その影響範囲と内容が具体的に記述されているかどうかが重要である．例えば，テストでの見逃し原因が，「その実行条件がテスト項目に挙がっていなかった」というだけでは具体的とは言えない．「実行条件には A 条件と B 条件がある．A 条件の範囲は洗い出したが，B 条件は気がつかず漏れていた. このため B 条件に関するテスト項目がすべて挙がっていなかった.」というように，

テスト項目にあがらなかった理由の影響範囲と内容を特定することにより，実際のテスト項目を導出できるレベルまで詳細化すべきである．

4.5 「1+n 施策」の適用方法

4.5.1 1+n 施策の立案

1+n 施策は，1+n 施策シートにより施策内容を立案する（図 4-3 参照）．バグ分析で的確に根本原因の影響範囲と内容を特定できていれば，その根本原因をそのまま実行することが 1+n 施策となる．例えば，レビューでの見逃し原因が，「C 領域の専門家である D さんが不在のまま，E 設計仕様書のレビューを実施した」ための場合は，「C 領域の専門家である D さんによる E 設計仕様書のレビューを実施する」が 1+n 施策となる．

次に，当該 1+n 施策の実施者，実施期間，実施項目数，および当該 1+n 施策による摘出バグ目標値を具体化する．摘出バグ目標値の予測は，バグ分析の実施者に任される．その理由は，同種バグが存在する可能性は，個々の根本原因およびそのソフトウェアの特性に大きく左右されるため，統計的な予測が適さないからである．例えば，上述の「D さんによる E 設計仕様書のレビュー」が 1+n 施策の場合，「D さん」がレビューすることにより期待される E 設計仕様書内の C 領域に関連する箇所と設計内容，それに合致するソースコードの箇所や難易度によって，同種バグの潜在する可能性が大きく異なるのである．

1+n 施策の対象範囲も同様に，根本原因の影響範囲をそのまま対象範囲とする．根本原因によっては，広範囲が対象となる場合がある．そのような場合には， 1+n 施策を長期的な実施計画とする．根拠がないまま，現実的に実施できる範囲に 1+n 施策を絞り込むのは「バグ分析と 1+n 施策」の目的に反する．実際に，1+n 施策で摘出されるはずだった同種バグが，妥当な理由がないまま対象範囲を絞り込んだために，顧客にて発生してしまった失敗事例が複数存在するからである．

4.5.2 1+n 施策実施結果の確認

1+n 施策の実施後には，関係者が集まって，実施結果に応じて以下を確認する．

- 同種バグが摘出された場合は，その同種バグの内容を確認し，1+n 施策の範囲と内容の妥当性を検証し，残存する課題がなければ終了とする
- バグを摘出したものの，そのバグが同種バグでない場合は，たまたまそのバグが摘出された可能性が高い．そのバグを分析対象として，新たにバグ分析と 1+n 施策を実施する．

・ バグが摘出されなかった場合は，その原因を分析する．バグ分析内容や 1+n 施策の範囲や内容が不十分であったと考えられる場合は，再度バグ分析と 1+n 施策を実施する．残存する課題がないと判断する場合は，ここで終了とする．

バグ分析と 1+n 施策は，残存する課題がなくなるまで実施する．摘出したバグ件数の数にかかわらず，摘出したバグの内容を確認し，さらに実施すべき課題が残っていないかを関係者全員が集まって検討する．これは，関係者全員の経験やノウハウを結集して，残存する課題の有無を確認することを意味する．残存する課題が見つかった場合は，バグ分析と 1+n 施策を再実施する．これを繰り返して，残存する課題がなくなるまで実施する．

1+n施策シート		
レビュー施策	計画	対象範囲: レビュー施策内容:
	目標値	レビュー項目数:　摘出バグ数:　工数: 期間:　担当者:
	実施結果	
	実績値	レビュー項目数:　摘出バグ数:　工数: 期間:　担当者:
テスト施策	計画	対象範囲: テスト施策内容:
	目標値	テスト項目数:　摘出バグ数:　工数: 期間:　担当者:
	実施結果	
	実績値	テスト項目数:　摘出バグ数:　工数: 期間:　担当者:

図 4-3　1+n 施策シート

4.6　考察

本章では，上述した成果に基づき，「バグ分析と 1+n 施策」の特徴を考察する．「なぜなぜ分析」と「バグ分析」との違いについても言及する．

なぜなぜ分析は，広く一般的に適用可能な根本原因を分析するための技法である．よって，なぜなぜ分析を実施する目的は，適用場面によりさまざまである．上述したように，分析対象の根本原因は１つではなく，複数存在することがほとんどである．このため，特定の目的を狙って根本原因を分析しなければ，欲しい根本原因へたどり着くことが出来ず，複数の根本原因が分析されてしまう．単に「なぜ」を繰り返すのでなく，特定の目的を意識して，その目的へ向かって根本原因を分析しなければ，所定の成果は得られない．

これに対して，「バグ分析と 1+n 施策」の「バグ分析」は，同種バグの摘出という目的を達成するために最適化した固有根本原因分析技法である．その目的を達成するため，分析対象バグの作り込み工程を意識しながら，作り込み原因，レビューでの見逃し原因，およびテストでの見逃し原因の 3 点に絞って固有根本原因を分析する．このような分析の枠組みを提供できるのは，「バグ分析」技法が，同種バグの摘出という目的に対して，V字モデルを前提として，バグの根本原因の構造を整理しているためである．さらに，過去の固有根本原因を蓄積したバグ分析シートを準備し，分析しやすくする工夫をしている．広範囲に適用可能な「なぜなぜ分析」に比べて，きわめて目的指向である．

「1+n 施策」では，目的指向の「バグ分析」結果を受けて，固有根本原因の影響範囲と内容をそのままにして施策を実行する．根拠なく 1+n 施策実施範囲を絞り込まないことや，摘出バグ目標値の設定は当事者に任されている点が特徴である．

根本原因分析の能力向上は，組織のプロセス改善に寄与すると言われている[4-3]．根本原因を的確に分析する能力は，組織における課題に対する的確な対応に通じるからである．このような面からも，「バグ分析と 1+n 施策」を適用し，「バグ分析と 1+n 施策」成功率を向上させることは，出荷するソフトウェアの品質向上だけでなく，組織のプロセス改善に寄与すると考える．

4.7 おわりに

本論文では，「バグ分析と 1+n 施策」の具体的な技法の内容，技法としての特徴および適用方法について論述した．ソフトウェア開発の現場では，なぜなぜ分析を使いこなすことができないため，効果的な根本原因分析ができずに悩む開発現場が数多く存在する．「バグ分析と 1+n 施策」は，それに対する 1 つの回答を提示するものである．

「バグ分析と 1+n 施策」は，根本原因を固有根本原因と共通根本原因の 2 つに分類し，作り込み原因，レビューでの見逃し原因およびテストでの見逃し原因の 3 つの原因に絞って，各々の固有根本原因を分析する．V 字モデルを前提としているため，バグ作り込み工程が判明すれば根本原因を特定しやすい．また，過去のバグ分析結果を蓄積して，発生頻度の高い根本原因を掲げたバグ分析シートや，施策立案のための 1+n 施策シートを準備することにより，効率的な「バグ分析と 1+n 施策」の実施を支援している．これにより，バグの根本原因に基づく，同種バグの効果的な摘出を実現した．

なぜなぜ分析は，その長所と短所を理解して，目的に応じてなぜなぜ分析の適用方法を工夫することが重要である．「バグ分析と 1+n 施策」は，同種バグの摘出を目的として最適化した技法である．

第5章　ソフトウェア品質会計を支える技術

5.1　はじめに

本章では，品質会計を支える技術について論述する．

初めに，レビュー技術を取り上げる．品質会計は，レビューによる品質向上が大きな特徴の１つである．品質会計を適用してレビュー実施状況をマネジメントすることにより，レビューによる効果を最大限引き出すよう工夫している．本章では，そのレビュー技術そのものについて述べる．

次に，幾つかの日本企業が 1970 年代頃から取り組んできたソフトウェア品質問題に対する解決策に基づき，その共通的な仕組みを解説する．それらは，以下の３点である．

- データに基づく短サイクルのマネジメント
- 独立した品質保証部門によるプロセスとプロダクトの両面からの品質確認
- 複数人による出荷判定

上記は，異なる企業での品質向上への取り組みであるにもかかわらず，結果としてほとんど同じ実装方法となったプラクティスである．NEC の品質会計を考案した組織でも上記プラクティスを実装している．これらの意味するところについても考察する．

5.2　レビュー技術

5.2.1　ソフトウェア開発におけるレビュー技術の位置付け

ソフトウェア開発において，レビューは非常に重要である．図 5-1 に示すように，ソフトウェア開発では，ソフトウェアのバグ修正が後工程になればなるほど，そのバグ修正にかかるコストは指数関数的に膨れあがる[3-11]．要求定義段階におけるバグ修正コストを 1 としたとき，設計時のバグ修正コストはその 3～6 倍であり，開発テスト時には 15～40 倍，運用時には 40～1000 倍にまで増大する[5-1]．言うまでもなく，本番稼動直前のテストで懸命にバグ摘出するよりも，設計時のレビューで早期にバグを摘出したほうが，バグ修正コストは桁違いに小さく，コストメリットが非常に大きいのである．

レビューによる効果がこれほど大きいにもかかわらず，実際にはレビューを効果的に実施している組織がそれほど多くないのも事実である．テストなしでソフトウェアを出荷する事例はないだろうが，レビューなしでソフトウェアを出荷する事例は，おそらく存在する．その理由で考えられるのは，レビューしたくてもレビューの仕方を知らなかったり，頭の

中で考えるレビューよりも，ソフトウェアを実際に動かして確認したほうが早いという思い込みのためであろう．レビューは，テストに比べて技術的に確立されておらず，専門書も少ない．レビュー技術を学ぶ機会が少ないために，効果的なレビューができずにレビューの導入をあきらめてしまうケースが多いと考える．また，実際に動くところをできるだけ早期に見たいという要求に対して，近年の技術進歩が応えられるようになったことも逆に災いしているかもしれない．

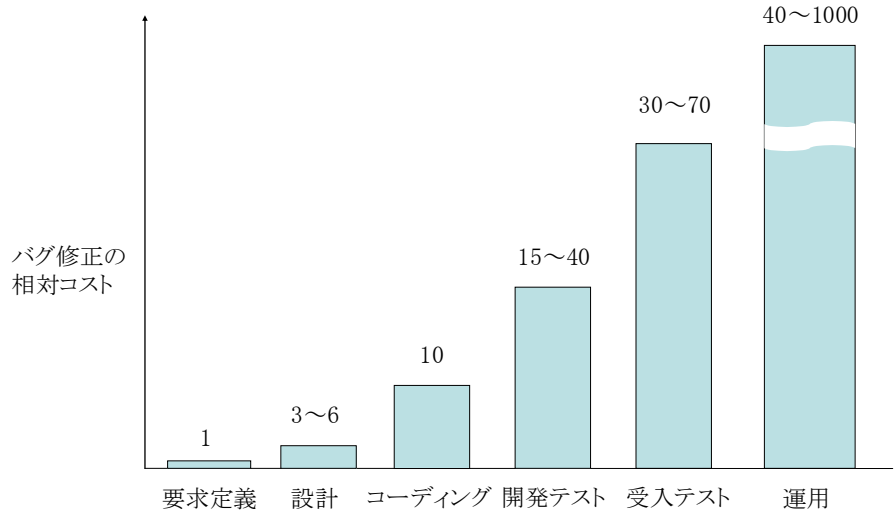


図 5-1　バグ修正コスト[3-11]

5.2.2　ソフトウェアのレビューとは

ソフトウェア開発におけるレビュー対象物は，設計書やプログラムだけでなく，プロジェクト計画書，要件定義書，テスト仕様書，テストプログラムなど多岐に渡る．さらに，開発途中の成果物に対してだけでなく，テストの一部としてレビューを実施する場合もある．実際に動作させて発生するにはタイミングが難しい事象や，すべてのソフトウェア・ハードウェア設備を揃えることが困難なケースなどにおいて，机上でレビューすることにより確認するようなケースである．これは，テスト技術のなかの静的技法の 1 技法としてのレビューである．また，レビュー参加による教育効果を期待することもある．OJT の一貫として，若い技術者をレビューに参加させるようなケースである．このように，レビューは，ソフトウェア開発のさまざまな局面で多様な目的のため，実施される．本節では，このう

ち，設計書やプログラムを対象にしたレビューに絞って解説する．

ソフトウェア開発においては，V&V と呼ばれる重要な品質確保の考え方がある．V&V（Verification & Validation）とは，検証と妥当性確認を意味する．検証（Verification）は，各開発工程の成果物が前工程の要求事項を満足しているかを確認することであり，妥当性確認（Validation）は，ユーザニーズを満足しているかを確認することである．V&V の確実な実施は，ソフトウェア開発の基本中の基本である．V&V では，ユーザニーズを満足していることの確認は，どの工程においても必要と言われている．上工程でのレビューは，V&V を意識して実施すべきである．すなわち，V&V という意味での上工程でのレビューは，当該工程の成果物が前工程の要求事項を満足していること，およびその段階までに具体化されたユーザニーズを満足していることを確認することが目的である．

ソフトウェア開発では，さまざまなレビュー方法が提案されている（表 5-1 参照）．レビュー手順が決められ，正式な記録を要求する「インスペクション」から，「アドホックレビュー」と呼ばれる非公式で即席で実施される方法まで多数の種類がある．品質会計で想定しているレビューは，表 5-1 のうち「チームレビュー」が最も近い．ただし，開発内容や必要性に応じて，ウォークスルー，パスアラウンド，ピアデスクチェック，ペアプログラミング，アドホックレビュー等を組み合わせて柔軟に実施している．現場では，表 5-1 に示すようなレビュー方法から，自然にその場で必要なレビュー方法を選択して実施しているというのが実情である．

また，レビュー対象成果物の確認方法（リーディング方法）についても，幾つかの方法が提案されている（表 5-2 参照）．当該組織においては，チェックリストを用いたレビュー（チェックリスト･リーディング）を必須としているが，開発内容や必要性に応じて他のリーディング方法も適用している．

レビューの大原則は，「人を憎まず，バグを憎む」である．バグを憎む余り，そのバグを作り込んだ人を責めてはならない．レビューは，レビュー対象物作成者の能力を評価する場ではない．レビューは，建設的に品質を確保する場である．バグ（失敗）を共有し，同じ失敗を繰り返さないように工夫するきっかけを作る場でもある．このような姿勢をレビュー参加者全員が共有していることが，レビュー成功の第一歩である．

5.2.3　レビューの手順

レビューは，主に V&V 観点からのバグの早期発見，およびより良い設計方法の検討を狙って実施する．その狙い達成のために，場面に応じて，レビューの参加者・レビュー方法・リーディング方法を適切に選択して実行することが重要である．

図 5-2 は，当該組織のレビューの流れを説明したものである．上工程では，ある工程の成果物のドラフト完成時以降，レビューの段階に入る．まず初めに，開発責任者がレビュー対象物の内容をチェックし，レビューに足る内容を確保できているか確認する．不十分な

表 5-1　ソフトウェア開発におけるさまざまなレビュー方法

([5-1][5-2][5-3]を参考に本著者が作成)

	No.	名称	内容
公式	1	インスペクション	標準や仕様から外れた例外を発見するレビュー方法．チェックリストを用いることが多い．最も公式なレビューであり，正式な記録が必要である．
↑	2	チームレビュー	チームにより実施されるレビュー．インスペクションほど公式でなく，「軽インスペクション」に分類される．
	3	ウォークスルー	レビュー対象成果物の内容を確認するために，複数人のレビューアが質問やコメントする形で実行されるレビュー．インスペクションが定められた公式な手順に基づいて実施されるのに対し，ウォークスルーは手順化されていないため再現性に乏しいと言われている．
	4	ラウンドロビンレビュー	レビュー参加者全員が順に司会者とレビューアの役を持ち回りで勤める形式のレビュー．回転式レビューとも呼ばれる．
	5	ピアレビュー	同僚(ピア：peer)によるレビュー全般の呼称．
	6	ペアプログラミング	2人が1つのマシンを共有して協同でプログラミングを行う方法．レビュー方法の1つとしても位置づけられる．eXtreme Programing(XP)の代表的なプラクティスである．
	7	パスアラウンド	レビュー対象成果物を複数のレビューアへ配布(または回覧)し，レビューする方法．電子レビュー(電子ファイルを回覧して，レビューアがコメントを書き込んだり結果をメールしたりするレビュー方法)はパスアラウンドの1種である．
	8	ピアデスクチェック	レビューアが個人でレビューし，その記録に基づいてレビュー対象成果物の作成者とフォローアップセッションを実施する方法．
↓			
非公式	9	アドホックレビュー	目前の問題を解決するために，近くの同僚に声をかけて知恵を借りるという程度の非公式なレビュー．ピアレビューの中でも最も非公式であり，即席レビューとも呼ばれる．

表 5-2　レビュー対象成果物の確認方法 ([5-4]を参考に本著者が作成)

No.	名称	内容
1	チェックリスト・リーディング	チェックリストを用いて成果物を確認する方法．チェックリストは，過去のバグ等に基づき組織的な経験知となるよう準備されたものを使用することが多い．チェック項目が多いとすべてを確認するのが難しくなる．また，生きたチェックリスト(常に見直されているか，チェック項目が抽象的または具象的すぎてレビューアが理解できないことはないか等)となっていることがポイントである．
2	パースペクティブ・ベースド・リーディング	レビューアが様々な利害関係者になりきって，それぞれの立場・観点(パースペクティブ)から成果物を確認する方法．パースペクティブの例として，ユーザ，開発者，テスト担当者等が考えられる．
3	ユースケース・ベースド・リーディング	ユーザーの利用視点で作成したシナリオ(ユースケース)を用いて成果物を確認する方法．妥当性確認に適している．
4	テストケース・ベースド・リーディング	ユースケース・ベースド・リーディングの派生方法で，テストケースを用いて成果物を確認する．

場合は差し戻しをする．レビュー可の場合は，レビューアの人選をする．レビューアの人選は非常に重要である．プロジェクトメンバだけでなく，開発するソフトウェアと同時に動作する可能性のあるソフトウェア製品の開発者，当該ソフトウェア領域について見識の高い技術者，関係するハードウェア技術者，マーケティング担当などに参加してもらう．ここで，レビューアに抜け漏れがあると，重要なバグを摘出できないままに後工程へ進んでしまい，大きな問題に発展してしまう．実際に，必要なレビューアがレビューに参加しないまま開発を進めたために，後から悔やむ結果になった経験は一度ならずある．このため，特に多くのソフトウェアと関連して動作するようなソフトウェア製品開発では，インタフェースを確認すべき相手先一覧などを準備し，レビューによる確認の抜け漏れのないようにチェックしている．レビューアの人数が多い場合は，効率を考慮して，テーマを分けてレビューを開催する．例えば，「商品としての価値を検討する外部仕様のレビュー」，「関係するソフトウェアとのインタフェースを中心としたレビュー」というようにレビュー目的を分けて設定する．

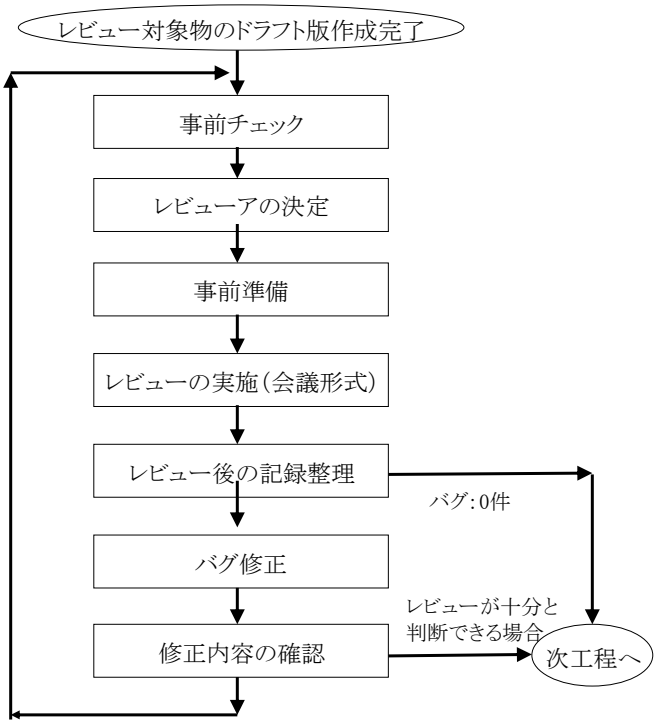
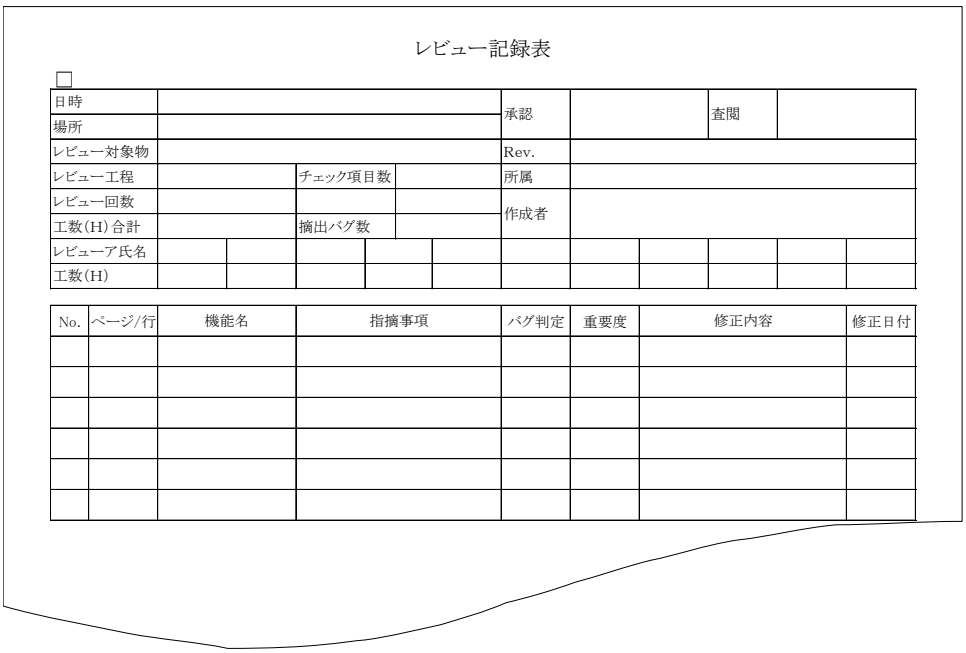


図 5-2　レビューの流れ

実際にレビューを実施する前に，レビュー対象物を配布し，レビューアにはできるだけ事前チェックしてもらう．レビューは，会議形式で開催することを原則とする．オフショア開発の場合は，レビューのためにレビューアが出張することも当り前のように行われている．電話会議の場合もあるが，これはお互いをよく知っている場合に限られる．優秀な

レビューアは忙しく時間が合わないので，レビュー会議が夜遅くに開催されることもある．しかし，顔を合わせて議論する重要性には変えられない．

品質会計においては，組織的なレビューチェックリストを準備し，それを適用してレビューすることを原則とする．製品によっては，レビューチェックリストのレビュー項目を，その製品向けにレビューしやすく表現を変えたり，追加したりして，その製品専用のレビューチェックリストを作成している．レビューアがプロジェクト内のメンバの場合は，レビューチェックリストを適用したレビューが多い．一方，プロジェクト外のレビューアが中心となるレビューの場合は，各レビューアが各自の立場で気がついたことを自由に指摘していくレビュー方法をとることが多い．



レビュー記録表

□

日時						承認		査閲			
場所											
レビュー対象物						Rev.					
レビュー工程			チェック項目数			所属					
レビュー回数						作成者					
工数(H)合計			摘出バグ数								
レビューア氏名											
工数(H)											

No.	ページ/行	機能名	指摘事項	バグ判定	重要度	修正内容	修正日付

図 5-3　レビュー記録表（例）

レビューでの指摘項目は，予め決められた記録者がレビュー記録表（図 5-3 参照）へ記録する．その場で回答できない指摘については，レビュー対象物作成者が会議後に調査して報告する．レビュー終了後，レビュー記録表を整理し，バグ判定をする．レビュー対象物作成者は，レビュー記録表に従ってレビュー対象物を修正し，その修正内容および日付をレビュー記録表に記録する．すべてのバグを修正後，版数を上げてレビュー対象物を発行する．開発責任者は，レビューの十分性を吟味し，再レビューすべきか，次工程へ進んでよいかを判断する．原則として，バグが出続けている場合は，再レビューを実施する．バグ修正をした結果，新たなバグが作り込まれていないか，前回のレビューでは気づかなかったバグがバグ修正によって顕在化していないか等を確認することが目的である．レビュ

一状況の品質分析は上工程品質会計により実施する（上工程における品質会計適用方法については 3.3 節を参照）.

設計書のレビューにおいては，バグ判定が難しいことがある．例えば，誤字脱字の間違いをバグと判定するかどうかは人によって判断が異なるであろう．品質会計では，設計書のバグを定義している（表 2-4 を参照）．この定義によると，単なる誤字脱字はバグとしないが，その誤字脱字により仕様を誤解する可能性が高い場合はバグとする．このように，設計書のバグについては，組織として統一した判定基準によってバグ判定できるように，具体的な定義が必要である.

5.2.4　レビューの見える化

2.5.1 節で述べたように，ソフトウェアは目に見えないという特性をもっているため，レビューの見える化においては特に工夫する必要がある.

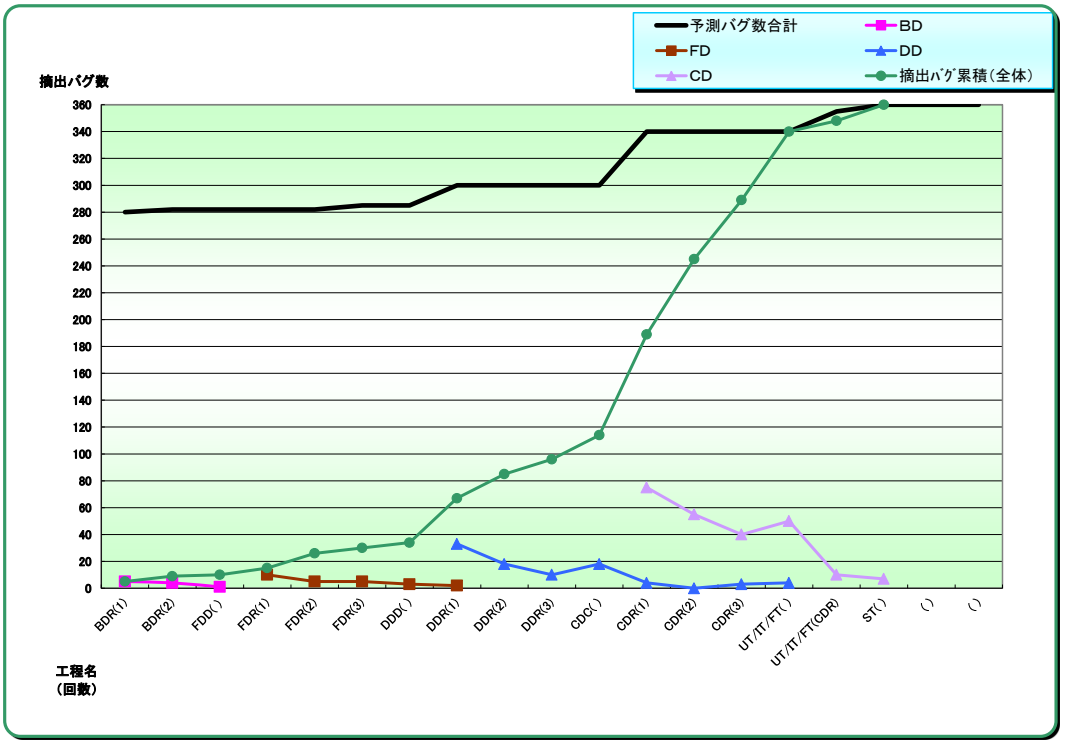


図 5-4　上工程品質会計票グラフ

品質会計では，バグの作り込みと摘出という考え方によって，見える化をしている（バグの作り込みと摘出については 3.3.2 節を参照）．上工程品質会計では，摘出したバグをバグの作り込み工程に分類して，摘出推移を時系列にグラフ化する（図 5-4 参照）．レビュー

の分析は, 品質会計技法のうち, 作り込み工程別バグの分析によって行う(3.4.3 節を参照). 少なくとも当該工程でのレビュー毎の摘出推移が減少傾向でなければ終了と判断しない. レビューをすればするほどバグが増加傾向を示すのであれば，それはまだバグが潜在しているとみなしてレビューを継続する．その場合には，レビュー不十分と考えられる観点を分析し，その観点について見識をもつ技術者をレビューアとするなどの工夫をする．また，グラフ化しているため，後工程になってから基本設計など上流のバグが摘出された場合などの異常ケースが一目でわかるようになっている．上流工程のバグ摘出が続く場合には，原因の源を断つために，その工程まで後戻りして再レビューすることを検討する．

5.2.5　レビューによる効果

上工程品質会計では，作り込んだバグはできるだけすばやく摘出するという方針のもと，当該工程で作り込んだバグを次工程までに摘出することを目標とする．また，品質会計の目標は，上工程バグ摘出率 80%である．したがって，テスト工程開始時に残存するバグは，作り込んだバグの 20%となる．品質会計を適切に適用している組織では，上工程バグ摘出率 80%を達成している．

レビューは，ソフトウェア品質の向上に大きな成果をもたらす．レビューに時間をかけるより，テストしたほうが早いというのは大きな間違いである．レビューとテストは，各々長所と短所をもっている（表 5-3 参照）．テストは，実際に動作させるため，多くの時間と人員，設備等のコストが必要である．これに対してレビューは，適切な能力をもつ技術者を集めれば，わずか数時間内にテストで同じ時間内に摘出できる件数よりも何倍ものバグを摘出することができる．レビューの実施時期は早ければ早いほどコストメリットが高い．しかも，レビューでは，実際に動作させるのが難しいバグまで摘出可能である．しかし，レビューは，あくまで机上の確認であるため，最終的な確認とすることはできない．一方，テストは，実際に動作させるために，技術者の考え違いがあっても見落とすことはない．また，ソフトウェアと同時に動作するハードウェアとの関係を確認できる．

レビューの長所と短所を理解して適用することによって，レビューはソフトウェア品質の向上に大きく寄与するはずである．

5.2.6　まとめ

レビューは，ソフトウェアの品質を向上させるための大きな武器である．しかし，残念ながらその武器がうまく使いこなせていないことも事実である．本節で言及したように，テストに比べてレビューに関する日本語の技術書籍は少なく，今後積極的に取り組むべき技術領域である．欧米と比較すると，日本のソフトウェア開発の方がレビューの重要性に注目しており，成果を出したレビュー実施事例を複数見かける．実際の開発の現場で行わ

れていることを持ち寄って，さらなる具体的かつ実践的なレビュー技法の確立を目指したい．

表 5-3　レビューとテストの特徴

	レ ビ ュ ー	テ ス ト
長所	・低コストで実施できる ・プログラム構造の良し悪しがチェックできる ・冗長，無駄なロジックを検出できる ・規約どおりに作っているかを確認できる	・実際の処理結果が確認できる ・関係するハードウェア／ソフトウェアとの確認ができる ・実機による確認のため，考え違いによる見落としがない ・設計の考慮漏れを検出できる
短所	・意識していない機能漏れは検出しにくい ・見落としの危険があり最終的な検証手段とはならない	・テスト条件や環境を作るのに労力がかかる ・間違いがあっても処理結果が正しく出る場合がある（領域の初期設定漏れなどの場合）

5.3　品質確保のための仕組み

本節では，幾つかの日本企業が 1970 年代頃から取り組んできたソフトウェア品質問題に対する解決策[3-1][3-2][3-3][3-4]に基づき，その共通的な仕組みを取りあげて論述する．本節で取り上げる仕組みは，異なる企業での品質向上への取り組みであるにもかかわらず，結果としてほとんど同じ実装方法となった．NEC の品質会計を考案した組織でも同じ仕組みを実装している．

5.3.1　データに基づく短サイクルのマネジメント

データに基づくマネジメントは，どの日本企業でも共通して取り組んできた仕組みである．これは CMMI でも基盤となる考え方であり，幾つかのプロセス領域に渡って解説されている．

ここでは，データに基づくマネジメントを一歩進めて，短サイクル（少なくとも週次）でのデータに基づくマネジメントとする．品質改善に成功する組織は，いずれも短サイク

ルでのデータに基づくマネジメントを実施することによって，問題発生の早期把握をしている．問題解決後に報告を受けるスタイルのマネジメント方法では，問題の対処結果の追認にしかならない．問題発生時にそれをより早く把握し，関係者全員が合意しながら問題の真の原因を的確に究明して対策を実施し，現場の状況をデータで確認しながら終結することによって，関係者が各々保有する経験やノウハウを結集して利用できるようになる．

さらに，マネジメントの実施方法がフェイスツーフェイスの場合は，さらにコミュニケーションが改善されて，形式的ではなく実質的に効果のある行動がとりやすくなる．

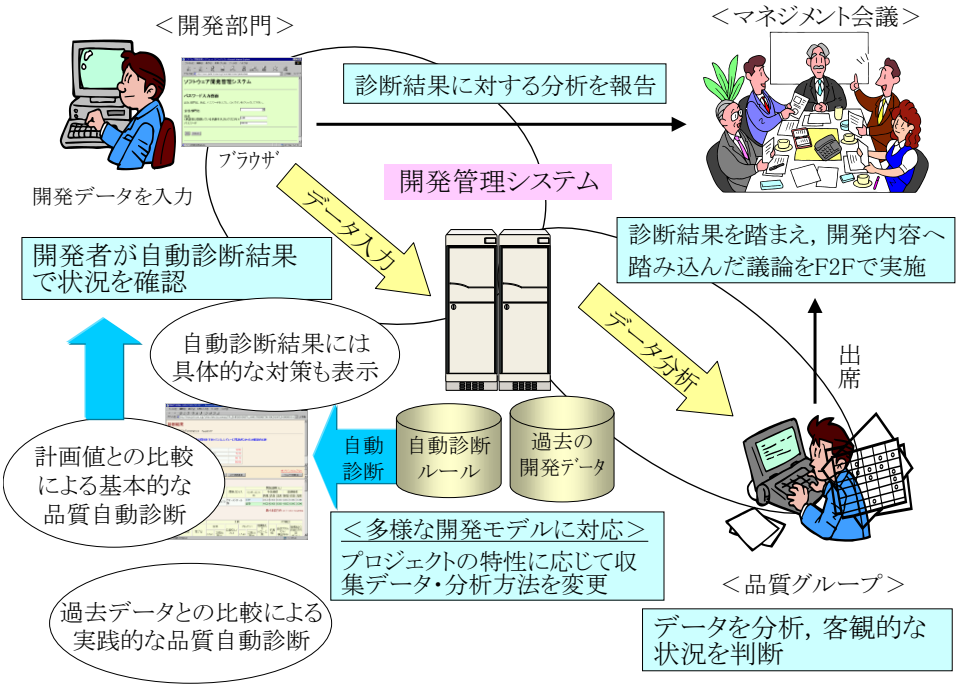


図 5-5　開発管理システムによる効率的な開発管理

収集と分析利用に関する品質会計考案組織の実例を紹介する（図 5-5 参照）．品質会計考案組織では，データ収集・分析のために自製の開発管理システムを開発し，運用している．マネジメント会議は週次で決まった曜日の決まった時間帯に開催されるため，データ入力の締め切り日時も曜日で決まっている．開発者は当該週の開発結果を開発管理システムに入力し，自動診断機能により自動診断を実施する．自動診断とは，データの入力ミスなどのチェックに加えて，計画値との比較や過去に蓄積したデータから設定した品質会計に基づく自動診断ルールに従って，基本的な品質状況を診断する機能である．これにより，開発者は入力したデータを使って基本事項のチェックをすることができる．この基本事項では，レビューやテストの実施状況とバグの摘出データなどから，レビュー不足やテスト不足，計画に対する実際の品質状況などの診断が含まれる．開発者は診断結果を検討し，必要ならその対策を立案した上でマネジメント会議へ出席する．一方，同じデータを使用し

て，品質保証部門でも品質分析を実施する．品質保証部門は毎週のマネジメント会議に出席しているため，タイムリーに開発状況を把握している．前回までのマネジメント会議での情報を加味しながら品質分析を実施し，その結果を次のマネジメント会議へ持参する．開発部門と品質保証部門は，各々の分析結果や施策をつき合わせて，最適な対策を検討し実施する．

品質会計考案組織では，原則としてフェイスツーフェイスによりマネジメント会議を開催している．それは過去の経験から，フェイスツーフェイスのほうがより的確に現場の問題を把握できることがわかっているためである．

5.3.2　独立した品質保証部門によるプロセスとプロダクトの両面からの品質確認

幾つかの日本企業が同じ実装方法に至った代表例の 1 つに，開発部門と独立した品質保証部門の存在がある[5-5]．品質保証部門の果たす機能に多少の差異はあるものの，客観的な立場で開発途中のデータ分析によりプロセス遂行状況を把握するとともに，各工程での成果物の出来を確認し，最終成果物は実際にテストして評価するという方法は共通である．どんなに優秀な開発者であっても，開発途中に客観的な視点を持ち続けることは難しい．品質保証の仕組みには，開発部門とは独立した部門による品質確認の機能が必須である．また，開発には開発技術が必要であるのと同様に，品質確認には品質を確認するための技術が必要である．そのために品質の専門家が必要なのである．

グローバルで見たとき，開発部門から独立した品質保証部門は，テスト専門チームであることがほとんどである．開発の一環として実施するテストの一部を請け負うようなタイプである．例えば，システムテストを実施するテストチームなどはその典型的な事例である．日本型の品質保証部門の立場に最も近いのは，IV&V(Independent Verification & Validation) [2-12]である．その名称の通り，開発部門から独立して V&V を実施する組織である．ただし，IV&V の定義では，開発活動の発注を受けた供給者とは別の供給者による V&V の実施など，開発事例に応じて IV&V の方法を決める場合が多く，日本企業のように固定的な組織として品質保証部門を保有するタイプとは異なる．また，日本型の品質保証部門は，開発部門が自ら V&V を実施し，そのうえで品質保証部門が品質保証活動を実施するため，IV&V とは本質的に異なる．また，IV&V の場合は，IV&V の結果に基づいてプロセスの見直しにフィードバックすることが難しい．固定的な組織として品質保証部門を保有するメリットは，ノウハウの蓄積とそれによる組織的な改善を可能とする点にある．組織として成功事例と失敗事例を蓄積しやすく，それらの情報が一箇所に集約されているためプロセスへのフィードバックも実行しやすい．

また，品質確認は，必ずプロセス品質とプロダクト品質の両面からでなければならない．これは，品質マネジメントの原則でもある（第 2 章の 2.4.2 節参照）．プロセス品質とプロダクト品質の各々から把握できることは限られている．プロセス品質の確認により，各工

程で実行するべきことが確実に実行されていることを確認し，プロダクト品質の確認により，出来あがった成果物が所定の要件を確保していることを確認する．両者をあわせて初めて確実に全体状況を把握できるのである（図 5-6 参照）．

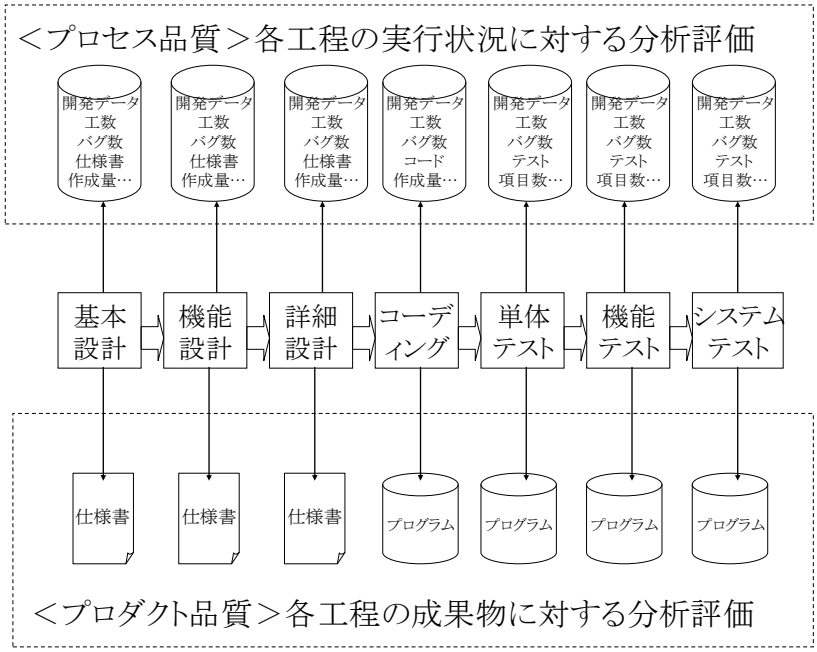


図 5-6　品質保証部門によるプロセス品質とプロダクト品質の両面からの品質保証

5.3.3　複数人による出荷判定

複数人による出荷判定も，日本企業の共通した実装方法である[5-5]．開発プロジェクトの責任者単独での出荷判定では，プロジェクトに責任をもっているだけに開発者に都合のいい判断になってしまいがちである．プロジェクト外の指示系統の異なる判定者が出荷判定に関与することによって，顧客視点での出荷判定が可能になる．指示系統の異なる判定者の代表例が，品質保証部門の責任者である．品質保証部門の責任者は，自組織が実施したプロセス品質とプロダクト品質の確認結果を，開発の経緯とともに理解している．また，他の開発プロジェクトのデータを出荷後データを含めて保有しているため，出荷前後の因果関係を考慮しながら，より客観的に判断可能である．

出荷判定基準は，あらかじめ明確に設定し，できればすべて数値で判断できる基準にすることが望ましい．さらに，出荷判定基準には，的確にテスト完了を判断できる基準が含まれる必要がある．品質保証部門によるプロダクト評価もその重要な判断材料の１つである．品質会計の場合は，テスト完了判断は，バグ傾向分析，バグ分析と 1+n 施策，およびバグ収束判定の３つの技法による判断結果を採用する．

複数人による出荷判定のため，判定結果が分かれたときの判定方法を明確にしておく必要がある．この場合は，品質保証部門の責任者の判定結果を採用するようにするべきである．開発部門の責任者による開発者に都合のいい判定を抑制するためである．また，不合格となった場合に備えて，ビジネスとして対処できる方法を確立しておく必要がある．品質会計考案組織では，ビジネス責任者の承認により出荷範囲を限定したり条件をつけて特別出荷する，いわゆる「特採」の方法をもっている．この場合でも，出荷判定は合格するまで実施し，合格したとき，はじめて正式な出荷製品として認められる仕組みである．

複数人による出荷判定を実施することで，常に同じ出荷判定基準による公正な判定が実現する．これは，技術者へ大きな影響を与える．開発途中および最終成果物であるソフトウェアが，定められた基準を満足しなければ出荷できないことを意味するからである．基準を満足しない場合は，出荷判定が不合格となる．それを身をもって体験することで，技術者自ら率先して品質確保を考えるようになり，自主的なプロセス改善のサイクルが回るようになるのである．

5.4　おわりに

本章では，品質会計を支える技術として，レビュー技術および幾つかの日本企業によるプラクティスをあげて説明した．

品質会計考案組織では，品質会計の構築と同時に，品質会計の効果を引き出すための品質保証の仕組みを継続的に整備してきた．その仕組みは，結果として幾つかの日本企業が各々独自で考案し構築してきた品質確保の仕組みと共通であった．この事実から，品質改善を実現するためには，正攻法ともいえるプラクティスがあることがわかる．それらは，データに基づく短サイクルのマネジメント，独立した品質保証部門によるプロセスとプロダクトの両面からの品質確認，および複数人による出荷判定という３つの要素である．

品質会計考案組織では，品質会計を適用する中で得られた課題を解決するために，設計技法やテスト技法などの新たな技法を導入し継続的に改善してきた．このうち重要な技術としてレビュー技術がある．また，品質確保のためには，品質を作り込むソフトウェアプロセス全体に渡る品質保証の仕組みの構築が不可欠である．その重要な要素が上記の３点と考えられる．

品質会計技法は品質管理手法であり，品質会計が単独で効果を出す範囲は限られる．品質会計は，品質向上を先導する役割を果たす．品質会計を適用することにより，ソフトウェア開発上の問題点を明らかにし，その原因を分析する糸口が得られる．その結果構築される品質会計の効果を引き出すための技術や仕組みが，品質向上に欠かせないのである．

第6章　ソフトウェアファクトリ

6.1　はじめに

効率的なソフトウェア開発のためには，開発技術，管理技術，開発環境などの整備が欠かせない．品質会計考案組織では，1970 年代からソフトウェア開発環境や技術の整備に取り組んできた．近年は，性能が良く使いやすい開発ツールが普及してきたこともあり，このような開発ツールや技術を組み合わせて，最適な開発環境を構築することが重要になってきた．本章では，その取り組みについて論述する．

6.2　ソフトウェアファクトリの変遷

1970 年代は，ソフトウェア規模が急激に増大したため，それにともなってさまざまな問題が発生するようになった時代である．特に，ソフトウェアのバグ対応費用の増大は深刻であり，ソフトウェア品質問題は経営課題となった．いわゆるソフトウェア危機と呼ばれる状況である．このソフトウェア危機に対応するために，NEC は 1981 年に SWQC を開始した[3-6]．SWQC（SoftWare Quality Control）とは，ソフトウェアの総合的品質管理活動であり，ソフトウェアに対する品質管理活動を全社的に展開する実践としては世界初の試みである．「ソフトウェアは，手工業や家内工業ではなくエンジニアリングや製造業に近いやり方で作られるべきだ」（元日本電気副社長・水野幸男博士）との考え方が基盤となっている．SWQC の理念は，「品質を追求しよう．生産性は後からついてくる」であり，その目的は「ユーザに喜んで買っていただき，しかも社会の発展に貢献するソフトウェアの実現」である．SWQC は，経営トップからのトップダウン的アプローチと，小集団活動を中心とするボトムアップ的アプローチの両面による活動である．現在では，SWQC は SWQC コミュニティという形式で，社内組織を超えたソフトウェア開発者交流の場として継続している．

SWQC 活動を開始するとともに，ソフトウェア開発に関するさまざま研究にも取り組んだ．ソフトウェアファクトリもその１つである．有識者を招いての定期的な勉強会のなかで，「ソフトウェアの『生産工場』と称するものが，単に同じ建物の中で『職人』達が机を並べて仕事をしているだけという状況を脱して，品質の良い製品を作り出す『ノウハウ』を蓄積し，したがって生産性も高い，近代的な工場に発展してゆくための起爆剤になってほしい」（東京大学名誉教授・森口繁一博士）との発言が記録されている．これが，ソフトウェアファクトリへ取り組むきっかけである．

図 6-1 は，品質会計考案組織におけるソフトウェアファクトリの進化の経緯である．全社的な SWQC 活動の高まりとともに，1980 年代には，設計技術・レビュー技術・テスト技術といった開発技術の整備と，品質会計をはじめとする管理技術の整備が行われ，それらのツール開発によって整備が進んだ．これらは，すべてメインフレームコンピュータが主流だった時代に実施された．

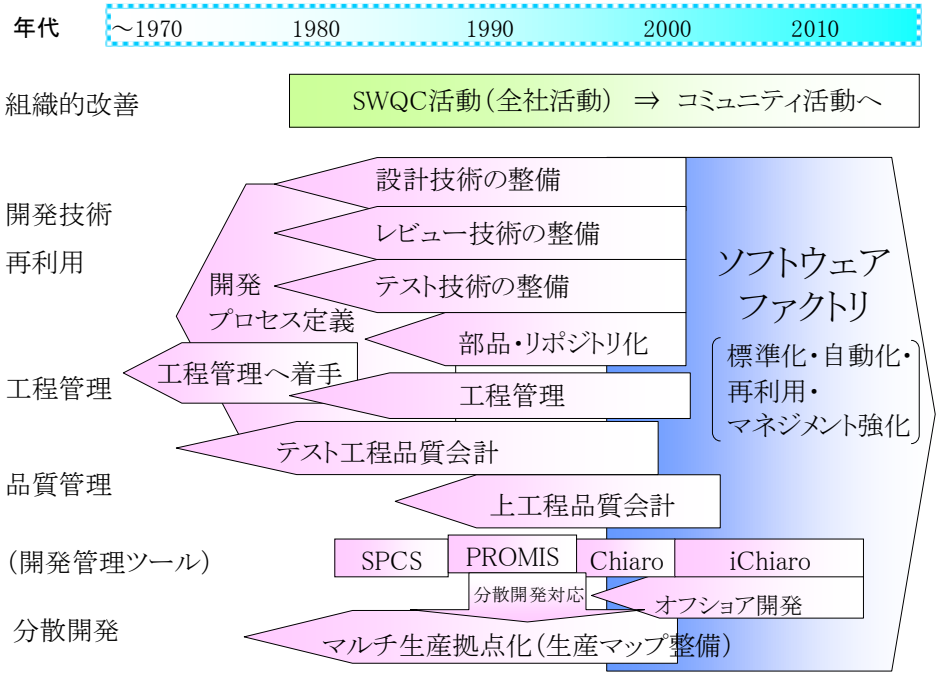


図 6-1　ソフトウェアファクトリ進化の経緯

1990 年代に入ると，ネオダマと呼ばれるパラダイムシフトが起きた．ネットワーク・オープン・ダウンサイジング・マルチメディアの頭文字をとって表現したのがネオダマである．性能の良い PC 上で動作するツールが次々と生み出され，ソフトウェア開発は良い意味でも悪い意味でも大きな影響を受けた．作成したソフトウェアはすぐに実行可能になったため，「詳細設計は不要」，「単体テストは不要」といった誤った考え方が広まった．メインフレーム時代は，高価なメインフレームを使う時間の制約が厳しかったために，できるだけ机上での確認が必要だったが，オープン化の時代に入って安価な実行環境が整備されると，むしろその実行環境を利用することに重点が置かれるようになったのである．

2000 年代に入り，オープン化の影響による弊害が出てきた．製品毎に最適化したソフトウェア開発環境をもつようになったため，組織的なエンジニアリング施策の実行が難しくなった．例えば，統一的なツール導入を図るには，製品毎のソフトウェア開発環境へ，各々開発ツールのインストール作業が必要になり，大幅な労力と時間をかけなければならない．2004 年に，品質会計考案組織が CMMI レベル 5 の達成を確認した際の「エンジニアリン

グ領域での組織的な検討が弱点」との指摘が契機となり，当該組織ではソフトウェアファクトリの構築検討を開始した．本著者は，検討のリーダとしてソフトウェアファクトリのコンセプトおよび基本設計の確立に寄与した．ソフトウェアファクトリの構築開始以降は，サブリーダの立場で関与している．

6.3　構築の考え方

ソフトウェアファクトリの構築にあたって，世界の有名ソフトウェア企業のソフトウェア開発環境をベンチマークした．その結果に基づき，ソフトウェアファクトリをモデル分類したのが表 6-1 である．ソフトウェアファクトリのもつべき機能の配置状況に基づき，[A]完全中央集約型，[B]中央＋一部分散型，[C]中央＋ローカル分担型，および[D]分散型の 4 種類に分類した．このうち最も進んでいる形態は[A]完全中央集約型である．この形態をもつソフトウェア企業では，世界中に分散する技術者が物理的に 1 つのソフトウェアファクトリにログインして開発する．昨日は米国で開発し，翌日は欧州でその開発の続きをするといったことが，現実として行われていた．開発資産はすべて 1 箇所のソフトウェアファクトリに集約されているため，開発状況の把握や開発資産の管理，ツールの入れ替えなどが非常に効率的に実施できる．

表 6-1　ソフトウェアファクトリのモデル分類

	[A]完全中央集約型	[B]中央+一部分散型	[C]中央+ローカル分担型	[D]分散型
特徴	・全ての機能はセンター集約 ・全ての開発者はセンターファクトリ上で作業	・基本的な機能はセンターに集約 ・テストの一部は各開発拠点で作業可能	・管理機能はセンターファクトリに集約 ・その他の機能はセンターとローカルで分担	・機能・資産・環境とも分散配置
開発資産	・センターファクトリー括保有	・センターファクトリー括保有	・センターファクトリとローカルファクトリで分担保有 ・最終資産はセンターファクトリに集約	・センターファクトリとローカルファクトリのどちらかに保有 ・最終資産はセンターファクトリとローカルファクトリのどちらかに保有
開発環境	・センターファクトリー括保有	・基本的な機能はセンターファクトリに集約 ・テストの一部は各開発拠点で作業可能	・センターファクトリとローカルファクトリで分担保有	・センターファクトリとローカルファクトリのどちらかに保有 ・適用ツールはプロジェクトにより異なる
管理データ	・センターファクトリー括保有	・センターファクトリー括保有	・センターファクトリー括保有	・センターファクトリとローカルファクトリのどちらかに保有 ・適用ツールはプロジェクトにより異なる

当時の品質会計考案組織は，[D]分散型であった．製品毎にソフトウェア開発環境を運用し，開発資産はセンターファクトリまたはローカルファクトリのどちらか，または両方で管理していた．開発ツールは製品の特性に合わせて選択していたため，組織的に見るとば

らばらだった．実は，1980 年代のメインフレーム時代には，品質会計考案組織のソフトウェアファクトリは[A]完全中央集約型だった．高価なメインフレームを製品毎に保有するのは事実上不可能だったため，中央集約にならざるを得なかった．それが，オープン化により，製品毎に少しずつ開発環境，開発言語，実行環境，サポートするプラットフォームなどが異なるようになり，さらにオフショア開発など分散開発の進展につれて個別最適化が進み，[D]分散型に変化したのである．

この調査結果に基づき，品質会計考案組織のソフトウェアファクトリは[B]中央＋一部分散型を選択することにした．[B]中央＋一部分散型とは，基本的な機能はセンターファクトリに集約し，テストの一部のみ分散開発拠点で実施可能とするモデルである．IT 製品向けのミドルソフトウェアという開発製品の特性を考慮すると，多種多様のストレージやネットワーク機器に対する実機テストが必要であり，それらをすべてセンターに集約して実施することは，オフショア開発を含む分散開発体制をとっていることから，事実上不可能なためである．

6.4　ソフトウェアファクトリの狙い

ソフトウェアファクトリとは，ソフトウェア開発方法論，ツール，環境を統合したシステムをいう．ソフトウェアファクトリの目的は，ソフトウェア開発の品質・生産性の大幅な向上である．ソフトウェアファクトリの目的を達成するための狙いを表 6-2 に示す．狙いは 4 つあり，標準化，自動化，再利用，およびマネジメント力向上である．標準化，自動化，再利用は，この順番で取り組んでいる．

表 6-2　ソフトウェアファクトリの狙い

狙い	内容	主な実施項目
標準化	開発のあらゆる場面の標準化を進めることにより，究極の標準化を実現する．	・プロセス，環境，ツールの更なる標準化 ・開発付帯作業の切り出し，集約化
自動化	自動化が可能な範囲において，適切な自動化を推進する．	・プロセス，環境，ツールの更なる標準化 ・ビルドサイクルのデイリー化 ・オートメーション化推進
再利用	アーキテクチャ，ソースコード，ドキュメントを部品化し，再利用を進める．	・アーキテクチャ，設計，ソースコードの部品化，再利用 ・設計仕様書，マニュアルの構造化，再利用
マネジメント力向上	ソフトウェアファクトリの効果を最大化するマネジメントを実現する．	・指標管理の高度化 ・情報漏えい防止の強化 ・知財マネジメント強化

ソフトウェアファクトリの狙いを以下に示す.

（1）標準化

開発のあらゆる場面の標準化を進めることにより，究極の標準化を目指す．この標準化の過程において，どのプロジェクトでも共通に発生する作業の集約を図っている．例えば，プロセスやツールなどの標準化やエンジニアリング技術の更新を専門のエンジニアリンググループに集約して実行したり，成果物バックアップなどの開発付帯業務を運用グループに集約している．これにより，最小限のコストで最新のソフトウェア開発環境を維持可能となる．ソフトウェアファクトリの利用者は，本来業務であるソフトウェア開発に集中できる.

開発付帯作業を専門グループが担当することにより，開発付帯作業そのものの効率化と高度化が進むことも期待している．専門グループ化すると，自らの地位保全と作業内容の改善のため，専門グループ内で自然と作業の効率化や高度化が進むからである.

（2）自動化

ソフトウェア開発のあらゆる場面において，自動化を推進する．自動ビルドや自動テストはもちろん，最新のツールによる自動チェックなどを進めることにより，品質・生産性の飛躍的な向上を実現する．新しいツールの検討や導入は，エンジニアリングを専門に検討するグループでなければ難しい．開発者が自ら手がけると，開発業務繁忙時にはどうしても作業が止まってしまうからである.

（3）再利用

再利用は，これから重点領域として取り組む予定である．ソフトウェアの複雑性と目に見えないという特性（第 2 章の 2.5 節参照）は，再利用を極めて困難にする.

現在の取り組みは，ソフトウェア開発の各段階での現実的な再利用の推進である．現段階ではソースコードの再利用が中心だが，将来的にはアーキテクチャや設計レベルでの再利用を目指す．当該組織では UML を標準的に適用しており，UML による再利用の推進を検討している.

（4）マネジメント力向上

ソフトウェアファクトリを適用することにより収集可能なデータを最大限利用し，マネジメントに役立てる．例えば，各日にチェックインされる成果物の増分による進捗管理や，深夜実行する静的検証ツールなどによる成果物の内容の確認により，開発者の報告だけに頼らない実物による管理を実現できる．情報漏えい防止や知財マネジメントも重要である．ソフトウェアファクトリは，アクセス制御や知財侵害チェックなどにより，問題を未然防止することもできる.

ソフトウェア開発の成功のためには，人材の質やマネジメントのほうがツールや技術よりはるかに重要な要因である[2-14]．したがって，マネジメント力の向上は非常に重要である．

6.5　ソフトウェアファクトリの概要と効果

ソフトウェアファクトリの概要を図 6-2 に示す．ソフトウェアファクトリは，ソフトウェア開発者へ開発環境をクラウド形態で提供する．センターファクトリにすべての機能を集約し，開発資産を一元的に管理する．センターファクトリはデータセンターで構築されているため，運用は 24 時間体制である．毎晩，自動的に開発資産のバックアップが実行され，遠隔地のデータセンターに保管される．ソフトウェアファクトリはイントラネット内に構築されており，アクセス制御されている．各プロジェクトには 1 つの VM(Virtual Machine)が与えられる．当該 VM にアクセス可能な技術者は，登録された当該プロジェクトメンバのみである．ソフトウェアファクトリの運用は，専門の運用チームが担当する．ソフトウェアファクトリで使用するすべてのソフトウェアの修正物件適用もこの運用チームが担当し，常に最新状態を維持する．利用者側からすると，ソフトウェアファクトリは常に最適な状態が保証されている開発環境である．

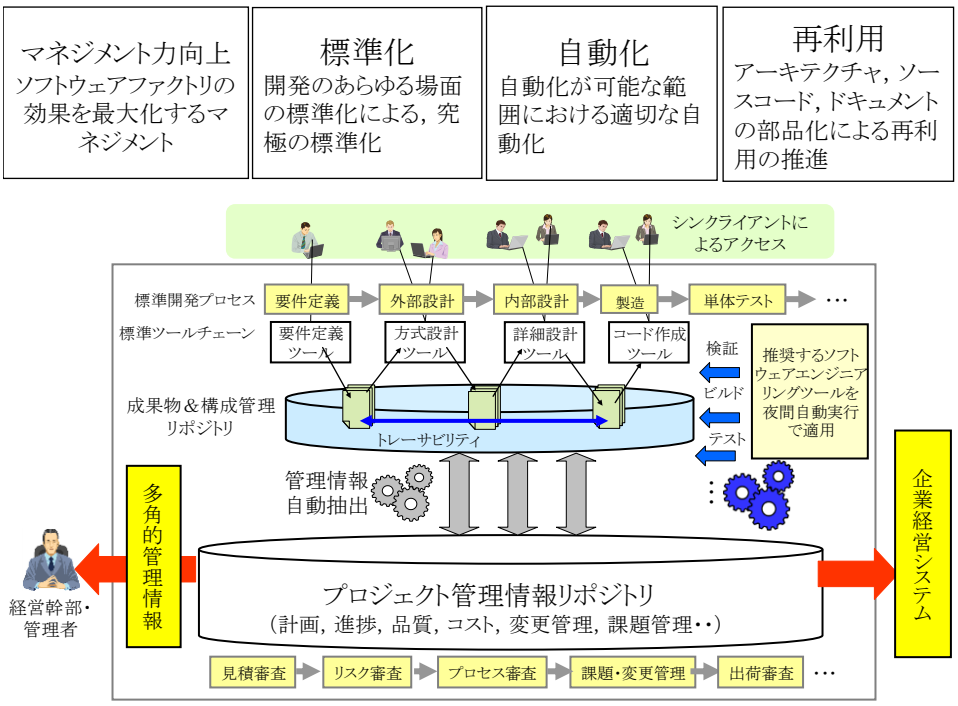


図 6-2　ソフトウェアファクトリの概要

ソフトウェアファクトリの効果を表 6-3 に示す．ソフトウェアファクトリは，現場だけで

なく経営視点でもメリットが大きい．開発環境に対する投資を最適化できるだけでなく，経営レベルの施策を迅速に実施できる．海外拠点からも日本のソフトウェアファクトリの利用が可能なため，グローバルレベルの分散開発にも容易に適用可能である．

表 6-3　ソフトウェアファクトリの効果

	現場目線	経営視点
収益力	●標準の開発管理技法・開発環境による開発コスト低減 ●PJ毎の開発プラットフォーム準備作業の不要化	●開発環境の効率的一元化による開発投資全体の最適化 ●PJマネジメントの質の向上
ガバナンス	●PJ内の情報共有，現場の見える化による現場マネジメント高度化 ●標準メトリクス情報による組織間分析，開発力底上げの実現	●経営レベルの指示・施策の迅速な展開，徹底 ●正確なマネジメント情報の把握によるタイムリーな指示・対策の実施
仕事の進め方の変革	●本来業務に集中，顧客対応や設計・改善などへ注力 ●グローバルレベルの開発手法・プロセスによる技術者のスキルアップ	●属人性を低減，組織力・ノウハウ蓄積を継続的に強化 ●グローバル開発のリスク軽減，グローバル事業の加速 ●人材流動化を加速

6.6　ソフトウェアファクトリによる品質・生産性の向上

本節では，現時点において既に実現しているソフトウェアファクトリの機能について述べる．現時点で実現しているのは，ソースプログラムを中心とした効率化とマネジメントの強化が中心である．

プロジェクトを効率的に推進するための機能を上げて，その効果を述べる．

① 新規プロジェクトを 2 日で開始

新規プロジェクト開始時には，プロジェクト名およびプロジェクト利用者名簿などの情報を提出するだけで，当該プロジェクト用の開発環境が準備される．プロジェクト情報提出の 2 日後からプロジェクト開始可能である．ソフトウェアファクトリがない時代には，当該プロジェクトの開発用サーバを準備し，構成管理ソフトウェアなどのツール類をインストールし，アクセス可能なプロジェクトメンバを登録するといった作業が必要であり，最低でもプロジェクト開始までに 2 週間はかかった．

② 一元的アクセス管理

開発途中の開発資産を保全するには，アクセス管理は非常に重要である．アクセス可能な人だけが当該プロジェクトの開発資産をアクセスできるようにしなければならない．また，プロジェクトメンバはその役割に応じて，細かくアクセス可能範囲を定義できる．例えば，プロジェクトリーダーは全プロジェクトデータをアクセスできるが，オフショア開発メンバはオフショア開発部分に限定してアクセスするように設定可能である．

③ 定期的な成果物の静的検証

ソフトウェアファクトリでは，毎晩，ソースプログラムの静的検証が実行され，その結果は翌朝参照することができる．これにより，日々のソースプログラムの増分だけでなく，以下のような観点から毎日チェック可能である．

- 各ソースファイルの規模およびコメント行，実行行，空白行の規模と比率の適切性
- 複雑度やネストの深さなどの数値による適切性
- バグ被疑箇所の有無
- コードクローンの有無

上記の観点は，バグ以外は，ソースプログラムの保守性に関わる特性である．例えば，ネストの深いプログラムであっても，実行上は問題ないものもある．ただし，長期的な視点でプログラムの保守を考えると，ネストの深いプログラムに対する改造はバグを作り込む危険性が高くなる．これらは，テスト完了後に判明しても修正できない．コーディング途中の指摘だからこそ修正可能な指摘である．その意味で，これらの静的検証を毎晩実行できる環境は，保守性の向上に寄与すると考える．

また，各々のツールの実行結果をソフトウェア開発の現場ですぐに使えるように調整していることも重要である．例えば，コードチェックツールを単純に実行すると，バグとほぼ確定できるレベルの指摘から単なる注意事項の指摘まで，人間が処理できないような何千という指摘結果が返ってくる．技術者は，重要な指摘も調査が不要な指摘も同じ 1 件として調査しなければならないため，非常に時間がかかってしまい効率が悪い．ソフトウェアファクトリ上で実装するツールは，修正が必要と考えられる指摘に絞って実行結果を返すように調整しているため，技術者はその厳選された指摘だけに対応すればよい．また，ツール間での重複結果がないように調整しているため，その意味でも技術者は効率的に作業をすることができる．

さらに，定期的な静的検証はマネジメント面でも効果がある．成果物の静的検証により，実際の成果物の出来をリアルタイムで把握できるようになった．このため，開発中に保守性に問題のある成果物を特定して対策を打ち，出荷後の保守や改造時の問題発生を未然防止できるようになった．ソフトウェアファクトリのない時代は，成果物の出来は，出荷する最終成果物でなければ確認できなかったため，問題点が判明しても対策を打つのは難しかった．

④ 自動ビルドとテスト

ソフトウェアファクトリには，自動ビルドとテストの機能を備えている．このため，ビルド条件とテスト条件を登録しておけば，デイリービルドとテストを実施できる．ソフトウェアファクトリがない時代には，自前で自動ビルドとテストの機能を準備しない限り，ビルドは大変な作業だった．最新のファイルを集め，ビルドするとほぼ半日必要であった．手動のため間違いも発生しやすいが，間違いに気がつくのはたいていテスト実行後である．このため，時間的な後戻りが大きい．

⑤ マネジメントの高度化・詳細化

プロジェクト実施中に発生するさまざまな問題は，チケットを発行して管理する．テスト中の故障（Failure）だけでなく，仕様変更案，プロジェクト遂行上の調整事項などあらゆるものがチケットとして発行される．そのうちバグは，バグ管理情報として使用される．品質会計の品質分析もこれらの情報を使用する．今までにないそのメリットは，チケットが構成管理情報とリンクしている点である．各バグの修正前後の内容が，ソースプログラムを参照しながら確認できる．このため，テスト中にバグが急増したり発生が収まらないような場合，具体的にバグの内容を 1 件ずつ確認することによって発生しているバグの傾向を把握できる．これらは，ソフトウェアファクトリがない時代には，時間をかけて調査しなければわからなかった．

アクションアイテムもチケットとして管理されるため，現時点での残存アクションアイテム，担当者，進捗状況をリアルタイムで把握できる．ボトルネックになっている技術者などがいれば，チケットを通じて担当者の変更も可能である．これらは，ソフトウェアファクトリがない時代には，週１回のプロジェクトマネジメント会議でなければ把握できなかった．

6.7 おわりに

本章では，ソフトウェアファクトリの位置づけ，概要および効果について述べた．ソフトウェアファクトリは，既に世の中に出回っているさまざまなツールや技術を組み合わせることによって，全体として大きな効率化を図ることを狙っている．個々のツールの成果を組み合わせ，全体としてうまく機能するように整備する取り組みは，ソフトウェア開発プロジェクトにとって非常に大きな効果をもたらす．何より，技術者をソフトウェア開発に関わるさまざまな煩雑な作業から解放するという心理的な効果がある．これにより，本来のソフトウェア開発という作業に集中できるようにすることが，ソフトウェアファクトリの狙う最も大きな効果である．

ソフトウェアファクトリは，ソフトウェア開発に要求される開発作業や成果物の緻密さや正確さといった実装面を中心とした難しさを解決する重要な施策である．この種の緻密

さや正確さは，きめ細かく確認したつもりでも漏れが出ることのある，人間が苦手とする領域である．それを徹底的に仕組みとして解決することは，高品質ソフトウェア開発を実現するうえで重要である．

ソフトウェアファクトリの経営面での効果も見逃せない．ソフトウェアファクトリには，まだ今後実装予定の機能が多数あるが，その途中経過時点においても，費用的な削減効果が出ている．オフショア開発でも既に適用されている．リスク管理面での効果も大きい．災害時などの対応策として，バックアップサイトのソフトウェアファクトリが準備されているほか，データセンター運用のため，もともと災害に対して堅牢である．ソフトウェアファクトリは，現在 NEC 全体の施策として全社展開している．

第7章　ソフトウェア品質会計の適用による品質向上の実例

7.1　はじめに

本章では，ソフトウェア品質会計の適用により，実際に品質向上を達成した 3 つの事例を紹介する．1 つ目の事例は，品質会計考案組織（以降，A 組織と呼ぶ）の事例である．A 組織では，品質会計技法を考案して現在の技法を構築するとともにそれを適用し，品質向上を実現した. 2 つ目の事例は, 品質会計を適用していたものの品質問題に悩んでいた組織（以降，B 組織と呼ぶ）が，従来からの品質会計の適用方法を見直し，ソフトウェア開発全体の仕組みを見直すことによって適用効果を挙げた事例である．3 つ目の事例は，中国のオフショア開発の品質向上事例である．オフショア開発とは，海外でのソフトウェア開発である．オフショア開発拠点（以降，C 組織と呼ぶ）における品質会計の適用と，ソフトウェア開発全体の仕組みの見直しにより，品質・生産性が向上した事例である．これらの改善事例に基づき，品質会計の厳格な適用と品質会計を取り巻くソフトウェアプロセス全般に対する総合的な取り組みが，高品質ソフトウェア開発の実現に効果を発揮することを示す.

なお，A 組織および B 組織の品質向上事例では，出荷後バグ数の絶対値を品質を判断する指標として使用する．出荷後バグ数を開発規模等により正規化しない理由は，①顧客視点では，顧客から見える出荷後バグ数そのものが品質を表す重要な側面である，②汎用ソフトウェア製品の出荷後バグ数は，顧客数や顧客の使用環境の厳しさに大きく影響を受け，開発規模とは比例関係にない，③出荷時には，開発規模の大小に関わらず，顧客視点から同じ条件で出荷しているため，正規化は不要である，という考え方に基づく．A 組織および B 組織では，この考え方に基づき，出荷後バグ数の絶対値を組織の品質向上目標と設定し，目標管理している.

7.2　A 組織の品質向上事例

A 組織は，ソフトウェア品質会計技法を考案した組織である．A 組織では，品質会計を考案・適用することによって，20 年以上前に 1 年間に発生する出荷後バグ数を 1／20 に削減するという大きな成果を得た．以降，A 組織のソフトウェアビジネスを取り巻く環境は，メインフレームからオープン化へのパラダイムシフト，OSS（オープンソースソフトウェア）の登場，オフショア開発の進展など，大きな変化があった．そのようななか，A 組織は品質向上活動を継続することによってこれらの変化に対応し，その出荷後バグ数のレベルを維持している.

本節では，A 組織がソフトウェア品質会計を考案する過程を含めて，1980 年代以降から現在までの経緯を述べる．本著者は，A 組織の品質保証部門に配属されて以降，ソフトウェア品質会計の構築と適用に携わり，中心的な立場でA組織の品質向上活動に寄与してきた．

7.2.1　A 組織の品質向上活動

図 7-1 品質会計構築の経緯に示すように，品質会計は 3 段階の発展段階を経て現在に至っている．品質会計の 3 段階の発展段階と A 組織の取り組みの経緯を以下に示す．

（1）第 1 期　テスト中心のバグ叩き出しによる品質向上

A 組織がソフトウェア品質会計を考案するきっかけは，A 組織自身の品質問題である．当時，A 組織は，出荷後バグ数が多いため，その対応に追われて開発が滞る負のスパイラルに悩んでいた．現場の開発者もトラブル対応に追われて疲弊しており，品質問題の解決は必須の状況だった．A 組織が最初に取り組んだのが，第 1 期の「テスト中心のバグ叩き出しによる品質向上」である（図 7- 1 を参照）．テストを強化し，テストでできるだけバグを摘出して出荷後のバグ数を減少させようというものである．第 1 期には，品質会計技法のうち「テスト工程品質会計」を考案し，テスト工程でのバグ摘出を目標管理した．バグ目標値を設定するための回帰型バグ予測モデル，およびテスト終了を判断するためのバグ収束判定の技術確立に取り組んだ．組織の標準化にも着手し，マネジメント技術，設計技術，テスト工程技術，レビュー技術などの検討グループに分かれて，技術の標準化に対して積極的に取り組んだ．標準類の初版の発行は，第 1 期終盤から第 2 期初年度の 1986 年から 1989 年に集中している（図 7-1 参照）．

図 7-2 は，A 組織の出荷後バグ数の推移を表す．図 7-2 は，1985 年の出荷後バグ数を 100 としたときの相対値で示している．グラフの下位部分に示すのは，品質会計の 3 段階の発展段階である．1985 年～1988 年が第 1 期に該当する．この第 1 期の活動により，1988 年には 1985 年の約 1/2 に出荷後バグ数が減少した．

（2）上工程での素早いバグ摘出による品質向上

第 1 期でのテストでのバグ摘出は品質向上に成果をあげたものの，最終工程であるテストの強化は予定した出荷時期の延期を招いてしまう．このため，次第に上流工程から品質を良くすることを考えるようになった．それが，第 2 期の「上工程での素早いバグ摘出による品質向上」である（図 7-1 参照）．第 2 期では，レビューによるバグ摘出の強化に取り組み，レビューでのバグ摘出を目標管理する「上工程品質会計」を考案した．また，レビューとバグ摘出の関係から品質状況を判断する品質判定表，バグ摘出の傾向を分析するバグ傾向分析の確立に取り組んだ．レビューでのバグ摘出の主要指標が，上工程バグ摘出率である．上工程バグ摘出率とは，出荷前に摘出する総バグ数のうち，レビューで摘出した

バグ数の比率をいう．第 2 期は 1989 年～1995 年であり，この時期に出荷後バグ数は一段と減少し，A 組織の出荷バグ数は 1985 年度比 1/20 まで低減した（図 7-2 参照）．また，上工程バグ摘出率は，第 2 期の間に向上し，80%を超えた（図 7-3 参照）．A 組織は，以来，その出荷後バグ数の水準と，上工程バグ摘出率 80%超を維持している．

（3）第 3 期　バグ分析に基づく開発プロセスへのフィードバック

現在は，第 3 期の「バグ分析に基づく開発プロセスへのフィードバック」にある．これは，「良いソフトウェアは良いプロセスから作られる」というプロセス指向の考え方に基づく．バグ分析により根本原因を分析し，開発プロセスへフィードバックすることによって常にプロセスを見直し，良いプロセスを維持する取り組みである．これらの取り組みの結果，2004 年には，A 組織は CMMI レベル 5 の達成を確認した．

図 7-4 は，CMMI のレベルと上工程バグ摘出率を示したものである．レベル 5 の上工程バグ摘出率は 65%である．これに対して A 組織の上工程バグ摘出率は 80%超である．これはソフトウェア品質会計，特に上工程品質会計による効果を示すものである．

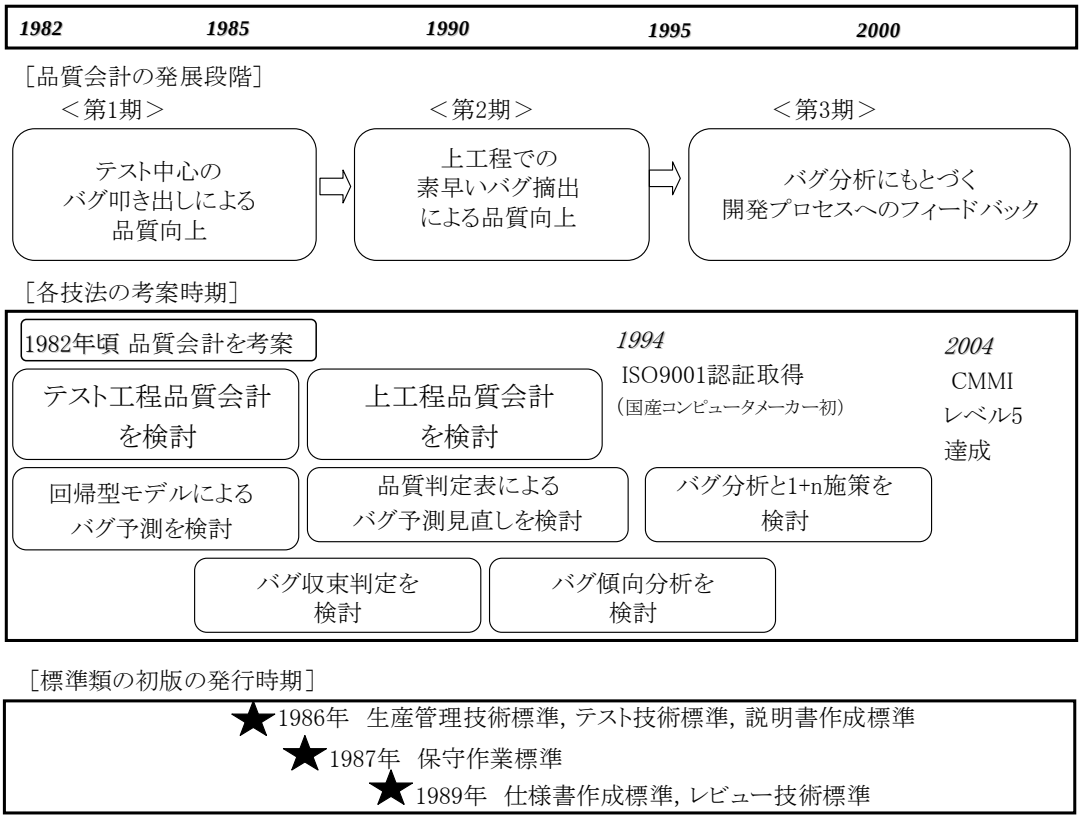


図 7-1　品質会計構築の経緯

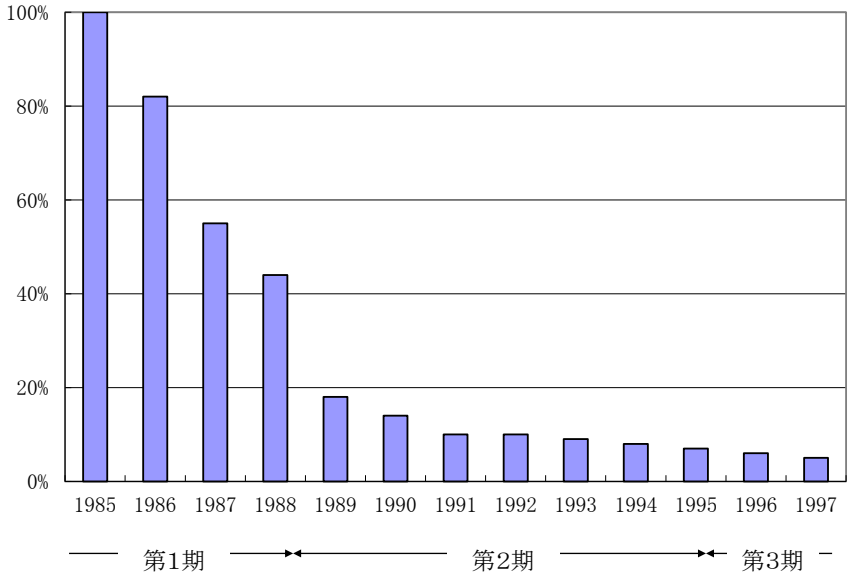


図 7-2　出荷後バグ数の推移
（1985 年の出荷後バグ件数を 100%とした相対比を表示）

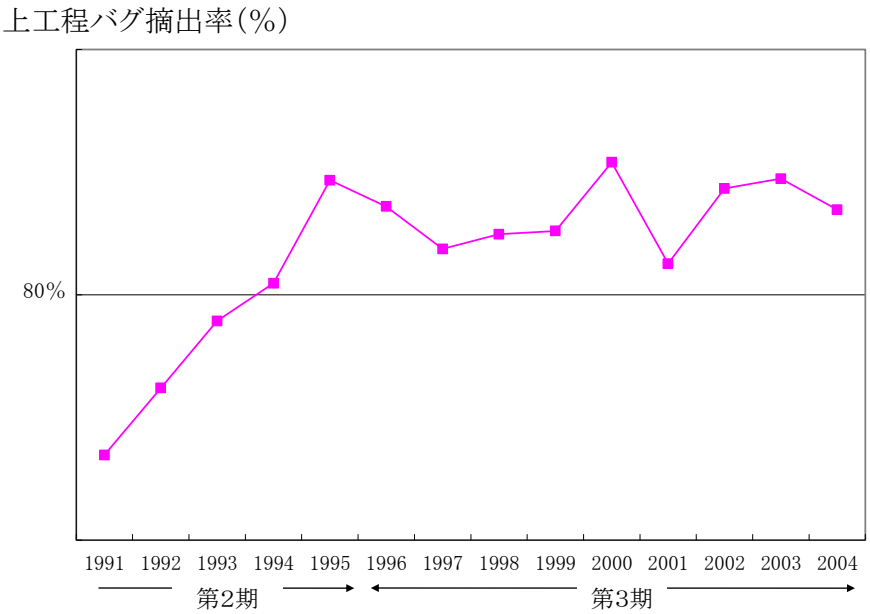


図 7-3　上工程バグ摘出率の推移

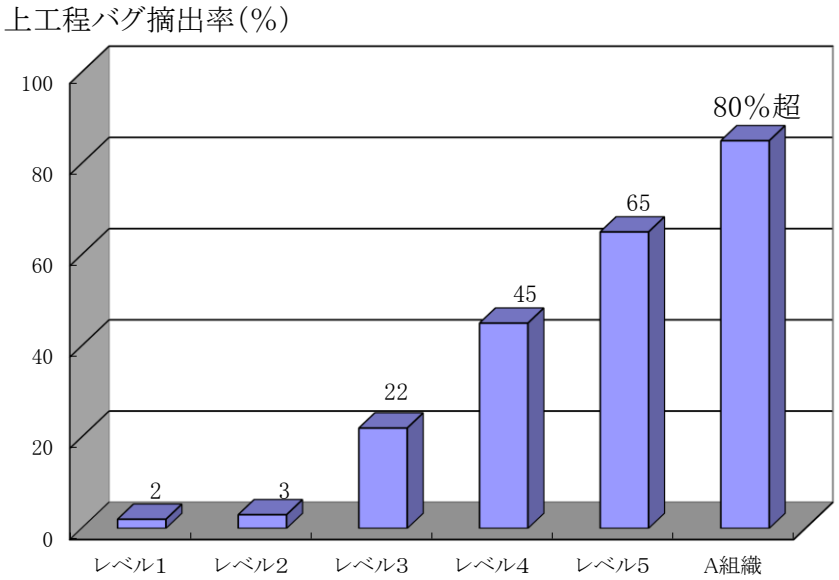


図 7-4　CMMI のレベルと上工程バグ摘出率
（レベル 1～5 のデータ出典：日経コンピュータ（2001.7.30.号））

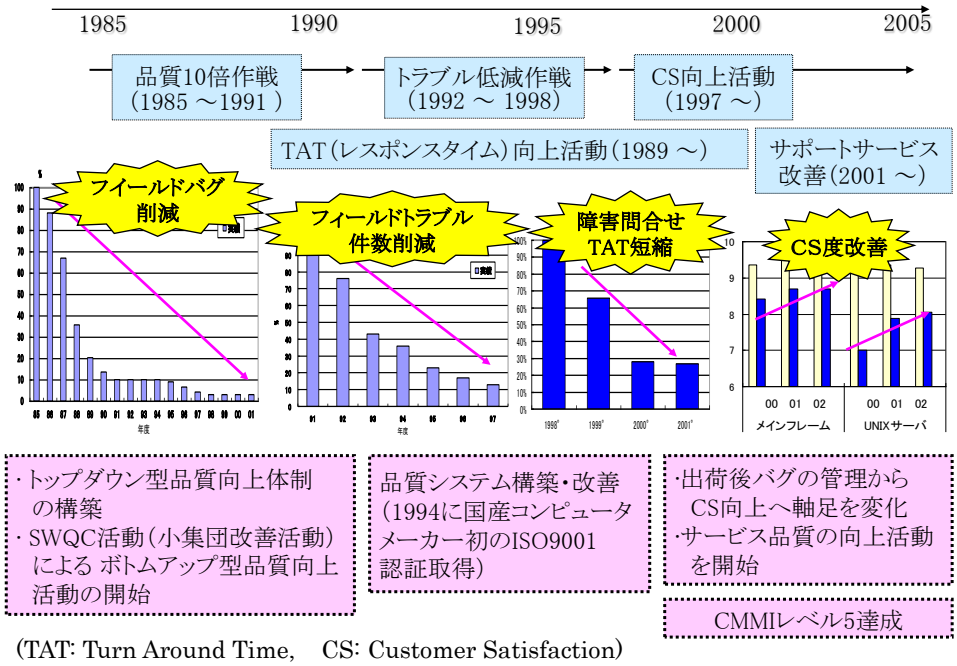


図 7-5　A 組織のソフトウェア品質向上への取り組み
（20 年以上に渡って品質・生産性向上活動を継続）

A 組織は，現在に至るまで，1985 年度比 1/20 の出荷後バグ数の水準を維持している．出荷後バグ数の低減だけでなく，20 年以上に渡って，トラブル低減や顧客満足向上などさまざまな改善活動を実施している（図 7-5 参照）．A 組織は，もともとメインフレームの OS を開発する組織だった．オープン化に伴う事業環境の変化により，主要開発製品は IT 系ミドルソフトウェアへ変化し，OSS の台頭への対応，サービス事業への対応，オフショア開発などの開発環境の変化などに対応しながら，出荷後バグ数の水準を継続して維持している．出荷後バグ数の少なさを強みとしており，品質重視の組織文化をもつ．それは，トップから現場の開発者までの一致した認識である．

7.2.2　考察

A 組織において，上述のような効果を上げることができた理由を考察する．

出荷後バグ数を 1／20 に削減し，上工程バグ摘出率 80%を達成した背景には，ソフトウェア品質会計の考案および適用による効果だけでなく，開発技術の標準化などによるソフトウェアプロセス全体の整備や，組織をあげての品質向上推進による効果が影響していると考えられる．そのなかで，品質会計は改善を実現するための強力な推進役を果たしたと言えるとともに，品質会計による直接的な効果は少なくとも 5 割程度あると考えている．それは以下の理由による．

- 品質会計の強みの 1 つは上工程品質会計によるレビューでのバグ摘出目標管理である．上工程品質会計により，レビュー毎の作り込み工程別のバグ件数を記録し，グラフ化する．これにより，レビューの推移に対するバグ件数の推移を視覚化して把握できるようになる．
- レビューの推移に対する作り込み工程別のバグ件数が増加あるいは減少しない場合は，当該作り込み工程の品質に問題があると判断できる．このため，当該工程だけでなく，より上流工程の品質問題にも素早く対処できる．上工程品質会計を実施しなければ，より上流工程の品質問題を上工程で検出するのは難しく，テスト工程までずれ込んだ結果，大きな後戻りを招いてしまう．
- 上工程品質会計のバグの作り込み工程情報から，組織として作り込む品質が悪いことの多い問題工程を特定可能である．このため，その問題工程の成果物の品質問題を解決するのに必要な開発技術の改善を促す効果がある．
- 品質会計のもう 1 つの強みは，確実にテスト完了判断ができる点にある．この技法は第 1 期には確立していなかったものの，第 3 期以降に出荷後バグ数を低水準のまま維持するのに効果を発揮した．確実にテスト完了判断ができるため，出荷後に顧客でトラブルが多発するような品質問題を未然防止することができ，出荷後バグ数の維持を実現した．
- 品質会計による定量化の推進は，開発対象ソフトウェアの品質向上を促すだけでなく，

組織全体の年毎の推移などを把握可能とし，組織全体の長期的な改善を促す効果がある．A 組織では，年次での開発データ分析を 20 年以上継続的しており，組織としての強み・弱みを分析して，年次改善計画に盛り込んできた．それが，図 7-5 に示したように，継続的な品質向上活動につながった．

7.3 B 組織の品質向上事例

本節では，出荷後バグ数の多さに悩む組織（以降，B 組織と呼ぶ）が，品質会計の本来の狙いに沿って品質会計の適用方法を見直しすることにより，品質向上に成功した事例を述べる．B 組織は，A 組織とビジネス環境が似ていることもあり，A 組織をベンチマークして改善施策を立案した．本事例は，2008 年～2012 年にかけて改善活動を実施している社内事例であり，改善開始 4 年後の 2011 年までの結果を述べる．B 組織の改善活動は本著者が改善のリーダーを務めている．

7.3.1 A 組織と B 組織の概要

A 組織及び B 組織は，どちらも NEC に属し，IT 製品向け汎用ソフトウェア製品を事業領域を分けて開発する組織である．A 組織と B 組織の顧客層はほぼ同じエンタープライズ領域であり，A 組織と B 組織の出荷高，開発量などはほぼ同規模である．どちらも約 2000 人のソフトウェア技術者を保有し，オフショア開発を含む分散開発体制をとっている．2 つの組織は，もともと 1 つの組織だったものが分割された経緯があるため，ほぼ同じソフトウェアプロセスを適用している．A 組織および B 組織ともに，2000 年代初期に CMMI レベル 5 を達成した．A 組織および B 組織は，主に V 字モデルを適用し，V&V を実施している．設計やテストなどの開発技法はほぼ同一であり，どちらも品質管理手法として NEC が考案したソフトウェア品質会計技法[3-9]を採用している．

7.3.2 改善前の出荷後バグ数の状況

A 組織と B 組織（改善前）の出荷後バグ数（相対値）を図 7-6 に示す．出荷後バグ数（相対値）とは，A 組織の 1 年間の出荷後バグ数を 100 としたときの B 組織の相対的な出荷後バグ数を表す．図 7-6 に示す A 組織および B 組織（改善前）は，同じ年の 1 年間の出荷後バグ数のデータを使用している．

平均値で比較すると，A 組織の出荷後バグ数が 100 に対して，B 組織（改善前）の出荷後バグ数は 236.78 と，B 組織は A 組織の 2 倍以上の出荷後バグ数が発生している．B 組織は，出荷後バグ数の多いことが課題であり，実際にそれがビジネス上の課題になっていた．

これが B 組織の品質向上活動のきっかけである．B 組織は，A 組織との比較分析から，5 年間で A 組織と同等レベルの出荷後バグ数へ削減することを目標と設定した．

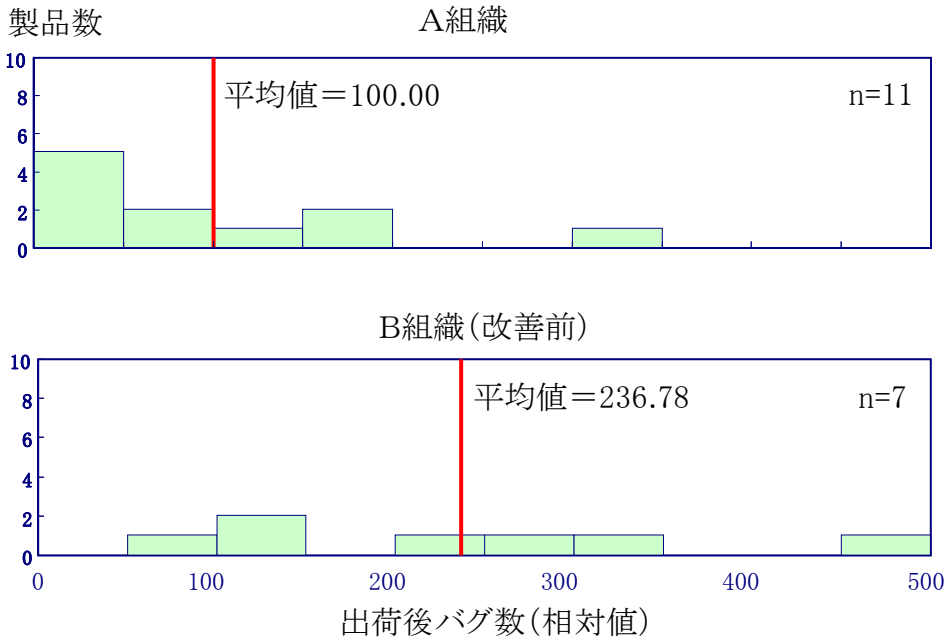


図 7-6　A 組織と B 組織（改善前）の出荷後バグ数の比較
（出荷後バグ数（相対値）とは，1 年間に客先で発生したバグ数について，
A組織の平均値を 100 としたときの相対値）

7.3.3　改善結果

B 組織が改善活動を開始し，4 年間が経過した．その改善結果を図 7-7 に示す．図 7-6 と同じ条件で，A 組織の 1 年間の出荷後バグ数を 100 とした相対値で表す．改善推移を示すために，B 組織の改善前，改善施策を実施開始してから 3 年後と 4 年後のデータを示す．使用したデータは同じ 7 製品のデータである．改善開始後の 1 年後および 2 年後を割愛した理由は，データのばらつきが大きく傾向をつかむのが難しいためである．B 組織は，海外も含めたソフトウェア技術者 2000 人規模の大規模な開発組織であるため，改善施策が浸透して成果を出すまでにそれなりの時間がかかったことがその原因と考えられる．

図 7-7 によると，出荷後バグ数の平均値は，改善前が 236.78 から，3 年後には 199.62 へ減少し，4 年後には 133.78 となった．B 組織（4 年後）では，出荷後バグ数（相対値）が 450～500 の製品がなくなり，100 以下の製品が改善前の 1 製品から 3 製品に増えた．

B 組織の 5 年間で A 組織と同等レベルの出荷後バグ数にする目標は，現時点において

順調に推移しており，目標を達成できる見込みである．

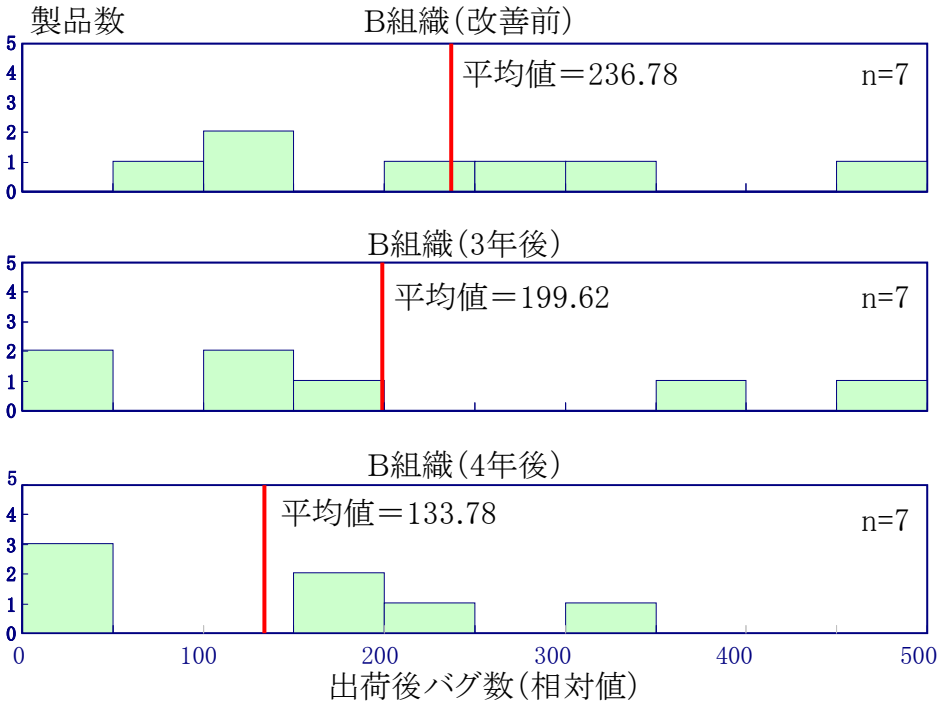


図 7-7　B 組織の改善推移

（出荷後バグ数（相対値）とは，１年間に客先で発生したバグ数について，
A組織の平均値を 100 としたときの相対値）

7.3.4　改善施策と実施結果の分析

B 組織は，A 組織をベンチマークすることにより，A 組織との顕著な相違点を分析した．その結果に基づき，改善施策を立案し実施した．主要な改善施策とその結果を述べる．
B組織の改善計画の立案および結果の分析に使用したデータ項目の定義を表7-1に示す．表7-2にA組織およびB組織の改善前，改善活動開始から3年後，および4年後のデータを示す．表7-2のデータ項目は，上工程バグ分析摘出率（No.9）および1+n施策成功率（No.10）を除き規模（KLOC）で正規化した上で，No.1～No.10の数値をA組織の平均値を100とした相対値で表している．B組織の改善前後の各データ項目の平均値に対して，データに対応がある場合の母平均の差に関する検定を，信頼率95%で分析した結果を表7 - 3に示す．

図 7-8 には，A 組織および B 組織の，出荷後バグ数に対する設計・製造工数，レビュー工数，およびテスト工数の散布図を示す．図 7-8 の上段は改善施策立案時に使用した A 組織と B 組織（改善前）の散布図，下段は改善結果を分析するために使用した B 組織の改善前，3 年後，および 4 年後の散布図を示す．表 7-4 には，図 7-8 内で示した回帰直線に関する情報を示す．

表7-1 データ項目

No.	データ項目	単位	定義
1	総工数	人H/KLOC	開発に費やした総工数(人H)/開発規模(KLOC). 総工数= 設計・製造工数 (No. 2) + レビュー工数 (No. 3) + テスト工数 (No. 4)
2	設計・製造工数	人H/KLOC	設計およびコーディングに費やした工数(人H)/開発規模(KLOC)
3	レビュー工数	人H/KLOC	設計およびコードに対するレビューに費やした工数(人H)/開発規模(KLOC)
4	テスト工数	人H/KLOC	テストに費やした工数(人H)/開発規模(KLOC)
5	総摘出バグ数	件/KLOC	出荷前に摘出した総摘出バグ数(件)/開発規模(KLOC). 総摘出バグ数 = レビュー摘出バグ数 (No. 6) + テスト摘出バグ数(No. 7)
6	レビュー摘出バグ数	件/KLOC	総摘出バグ数 (No. 5)のうち, テスト開始前の上工程(設計, 製造およびレビュー)で摘出されたバグ数/KLOC. 上工程で摘出されるバグは, 主に設計レビューまたはコードレビューにより摘出されるため, レビュー摘出バグと呼ぶ.
7	テスト摘出バグ数	件/KLOC	総摘出バグ数(No. 5)のうち, テスト工程で摘出されたバグ数/KLOC. テスト工程で摘出されるバグは, 主にテストにより摘出されるため テスト摘出バグと呼ぶ.
8	テスト項目数	項目/KLOC	テスト項目数(項目)/開発規模(KLOC)
9	上工程バグ摘出率	%	上工程バグ摘出率 =(レビュー摘出バグ数(No.6)/総摘出バグ数 (No. 5)) ×100
10	1+n施策成功率	%	ソフトウェア品質会計の技法のうち,「バグ分析と1+n施策」による同種バグの摘出状況に関するメトリクスで, 以下の式で算出する. 1+n施策成功率=(1件以上の同種バグを摘出した1+n施策実施ケース数/総1+n施策実施ケース数) ×100

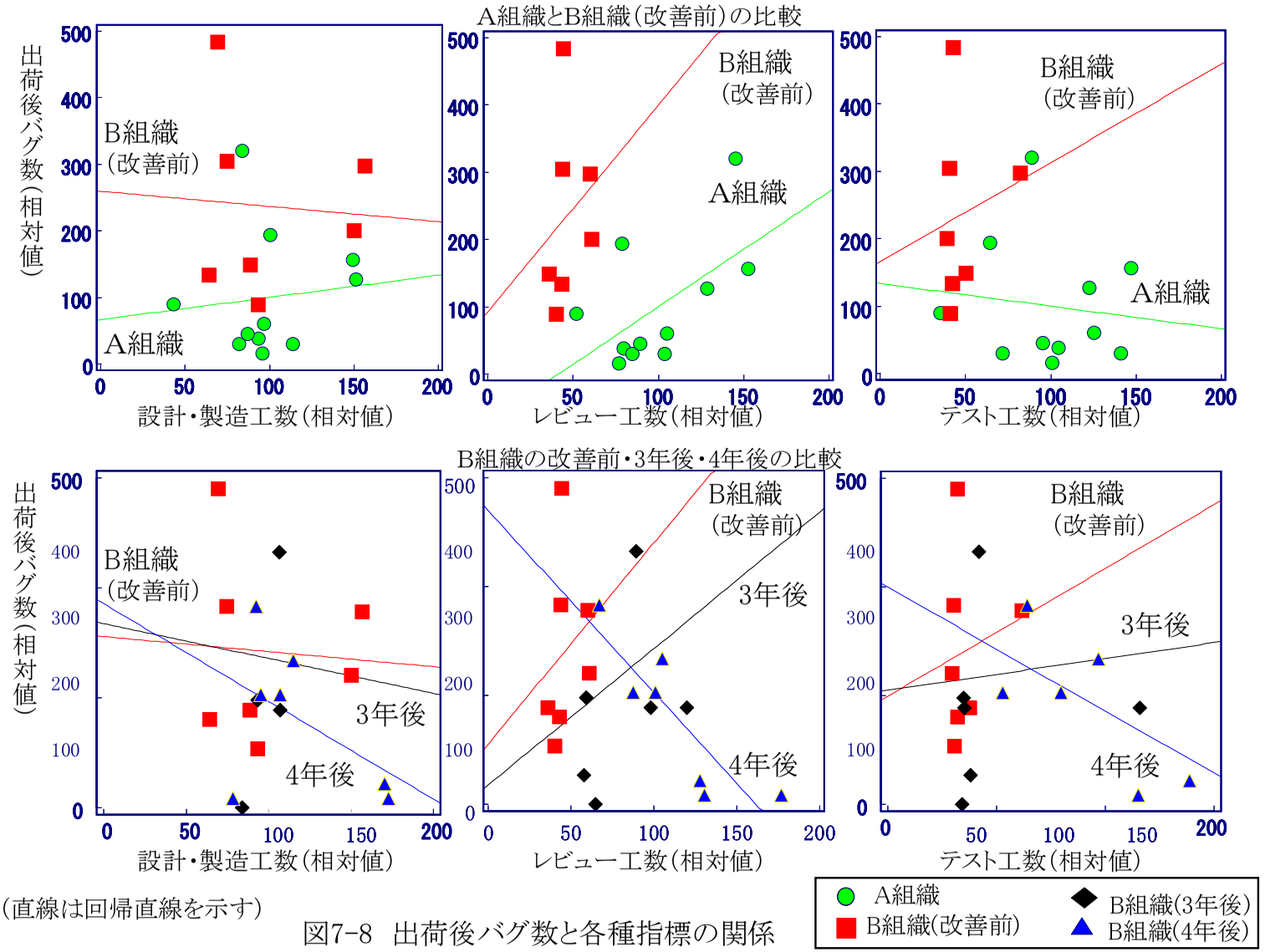
表7-2 データ一覧

（すべての数値は、A組織の平均値を100とした相対値）

No.	データ項目	A組織					B組織														
							改善前					3年後					4年後				
		n	平均	最小	最大	標準偏差	n	平均	最小	最大	標準偏差	n	平均	最小	最大	標準偏差	n	平均	最小	最大	標準偏差
1	総工数	11	100.00	37.78	149.51	30.17	7	67.63	50.97	107.27	20.75	7	100.86	62.76	216.42	54.33	7	128.59	82.37	204.48	45.03
2	設計・製造工数	11	100.00	43.38	151.59	30.57	7	99.60	64.28	156.65	99.60	7	150.70	84.03	330.92	94.37	7	118.63	78.33	172.44	37.79
3	レビュー工数	11	100.00	52.12	152.90	31.22	7	47.00	35.91	61.12	9.76	7	81.48	57.66	119.74	22.92	7	113.62	67.34	177.06	35.52
4	テスト工数	11	100.00	35.81	147.47	33.79	7	48.46	39.53	82.10	15.24	7	67.35	45.46	154.45	39.45	7	139.86	70.64	249.32	62.16
5	総摘出バグ数	11	100.00	73.83	131.25	17.61	7	80.84	64.01	115.01	15.94	7	87.43	78.45	104.73	9.62	7	117.70	82.61	161.52	24.85
6	レビュー摘出バグ数	11	100.00	73.31	131.20	18.81	7	63.07	46.04	101.57	17.95	7	80.25	72.62	96.82	7.97	7	110.53	77.52	145.37	22.45
7	テスト摘出バグ数	11	100.00	74.44	131.58	18.31	7	187.77	137.74	284.89	46.48	7	130.68	75.56	181.07	34.30	7	160.91	113.26	258.80	49.74
8	テスト項目数	11	100.00	56.11	172.78	34.56	7	57.56	18.64	84.14	23.80	7	129.75	35.97	237.66	69.64	7	187.56	115.00	264.22	52.47
9	上工程バグ摘出率	11	100.00	94.56	105.91	2.60	7	77.61	57.60	88.47	9.55	7	92.15	85.72	100.79	5.02	7	94.21	90.15	100.01	3.50
10	1+n施策成功率	11	100.00	0.00	214.59	66.83	7	46.50	0.00	128.76	54.99	7	83.20	0.00	160.94	52.87	7	123.56	71.53	214.59	44.01

表7-3. B組織(改善前)に対するB組織(3年後)およびB組織(4年後)の母平均の差の検定結果

No.	データ項目	改善前			3年後					4年後				
		平均値	分散	標準偏差	平均値	分散	標準偏差	t値	P値(両側)	平均値	分散	標準偏差	t値	P値(両側)
1	総工数/KLOC	67.63	430.71	20.75	100.86	2951.37	54.33	-2.496	0.047	128.58	2027.93	45.03	-3.785	0.009
2	設計製造工数/KLOC	99.60	1452.24	38.11	150.70	8905.22	94.37	-2.193	0.071	118.63	1427.72	37.79	-1.408	0.209
3	レビュー工数/KLOC	47.00	95.23	9.76	81.48	525.08	22.91	-3.950	0.008	113.62	1261.97	35.52	-5.109	0.002
4	テスト工数/KLOC	48.46	232.19	15.24	67.35	1556.57	39.45	-2.055	0.086	139.86	3863.95	62.16	-3.611	0.011
5	総摘出バグ数/KLOC	80.84	253.97	15.94	87.43	92.46	9.62	-0.763	0.474	117.70	617.65	24.85	-6.225	0.001
6	レビュー摘出バグ数/KLOC	63.07	322.30	17.95	80.25	63.48	7.97	-2.115	0.079	110.53	504.13	22.45	-7.462	0.000
7	テスト摘出バグ数/KLOC	187.77	2160.70	46.48	130.68	1176.68	34.30	3.301	0.016	160.91	2474.06	49.74	1.341	0.229
8	テスト項目数/KLOC	57.56	566.56	23.80	129.75	4850.07	69.64	-3.255	0.017	187.56	2753.56	52.47	-7.063	0.000
9	上工程バグ摘出率	77.61	91.23	9.55	92.15	25.19	5.02	-6.092	0.001	94.21	12.22	3.50	-4.608	0.004
10	1+n施策成功率	46.50	2519.94	50.20	83.20	2794.97	52.87	-1.085	0.319	120.50	1965.69	44.34	-4.101	0.006



(直線は回帰直線を示す)

図7-8　出荷後バグ数と各種指標の関係

表7-4　図7-8の散布図中の回帰直線の分析結果

No.	項目	設計・製造工数と出荷後バグ				レビュー工数と出荷後バグ				テスト工数と出荷後バグ			
		A組織	B組織			A組織	B組織			A組織	B組織		
			改善前	3年後	4年後		改善前	3年後	4年後		改善前	3年後	4年後
1	相関係数	0.109	-0.064	-0.273	-0.493	0.573	0.221	0.262	-0.867	-0.122	0.165	0.079	-0.779
2	回帰定数項	66.693	259.546	279.094	309.644	-70.878	92.193	29.724	449.232	133.716	165.494	175.012	332.993
3	回帰係数	0.333	-0.229	-0.527	-1.482	1.709	3.076	2.085	-2.776	-0.337	1.471	0.365	-1.424
4	t値	0.330	-0.144	-0.636	-1.266	2.098	0.507	0.608	-3.899	-0.370	0.375	0.178	-2.776
5	P値（両側）	0.749	0.891	0.553	0.261	0.065	0.633	0.570	0.011	0.720	0.723	0.866	0.039

(1)　レビューとテストの強化

表 7-2 により，A 組織と B 組織（改善前）を比較する．設計・製造工数の平均値は，A 組織 100 に対して B 組織（改善前）99.60 とほぼ同程度である（表 7-2・No.2 参照）．これに対して，B 組織（改善前）のレビュー工数は 47.00，テスト工数は 48.46，テスト項目数は 57.56 と，A 組織 100 の約 1/2 である（表 7-2・No.3，4，8 参照）．これに対応して，レビュー摘出バグ数は 63.07 と少ないものの，テスト摘出バグ数は 187.77 と A 組織の 2 倍近く多い（表 7-2・No.6，7 参照）．これらの結果から，B 組織（改善前）は，設計・製造には十分な工数をかけているものの，レビューでのバグ摘出が不十分であったために，テスト開始時点で残存するバグが多く，少ないテストでも多くのバグがテストで摘出される状況であったことが推察される．

次に，図 7-8 上段のグラフにより，A 組織と B 組織（改善前）の出荷後バグ数と各データ項目の関係を分析する．図 7-8 上段左の設計・製造工数の散布図では，A 組織と B 組織（改善前）のデータの多くが重なって分布している．これに対して，レビュー工数とテスト工数では，A 組織と B 組織（改善前）のデータは，ほとんど重ならずに分布していることが観察される（図 7-8 上段中央と右を参照）．回帰直線の傾きでもその傾向の違いは明確である．特にテスト工数と出荷後バグ数の散布図では，A 組織の傾きは右下がりの傾向を示しており，テストをすればするほど出荷後バグ数が少なくなる傾向にあるのに対して，B 組織（改善前）では逆の傾向を示している（図 7-8 上段右および表 7-4 を参照）．

7.2 節で述べたとおり，A 組織は過去に，ソフトウェア品質会計を考案し適用して，レビューによるバグ摘出を強化することにより，1 年間に発生する出荷後バグ件数を 1／20 へ削減した経験をもつ．レビューによるバグ摘出はソフトウェア品質会計技法の特徴であり，そのレビューによるバグ摘出の効果を表す上工程バグ摘出率は，A 組織の改善期間中に 80%を超えた．

A 組織と B 組織（改善前）との分析結果，および A 組織の経験に基づき，B 組織は，レビューとテストの強化を改善施策と決定し実施した．その結果得られた，B 組織の改善開始から 3 年後および 4 年後のデータが表 7-2 である．

B組織の改善前後の数値の変化の有意差を統計的に確認するために，各データ項目の平均値に対して，データに対応がある場合の母平均の差に関する検定を行った結果が，表7-3である．信頼率95%で分析した結果，有意水準5%としたとき，B組織（改善前）に対してB組織（3年後）の平均値に有意差があると認められたものは，総工数，レビュー工数，テスト摘出バグ数，テスト項目数，および上工程バグ摘出率の5 つのデータ項目である．また，B組織（改善前）に対してB組織（4年後）の平均値に有意差があると認められたものは，設計・製造工数およびテスト摘出バグ数を除く8つのデータ項目である．この結果から，改善活動３年後の改善成果は半数のデータ項目において観察されており，4年後にはその成果が拡大していることがわかる．数値の変化に有意差が認められなかったのは，設計・製造工数およびテスト摘出バグ数の2つのデータ項目である．このうち設計・製造工数は，設計・

製造に対する直接的な改善活動を実施していないことが数値の変化に有意差が認められなかった理由である．テスト摘出バグ数については，以降で考察する．

B 組織がレビューおよびテストの強化を実施した結果を，平均値に有意差があると認められたデータ項目により分析する．改善開始 3 年後には，レビュー工数およびテスト項目数の増加が認められた．その結果としてテスト摘出バグ数が減少し，相対的に上工程バグ摘出率が増加した．これは，レビューによるバグ摘出が増加した結果，テスト開始時点に残存するバグ数が減少し，テスト項目を増加してもテストで摘出されるバグ数が減少したことを示す．改善開始 4 年後には，レビュー工数およびテスト項目数に加えてテスト工数が増加した．その結果として，レビュー摘出バグ数が増加し，総摘出バグ数も増加した．この結果から，B 組織のレビューおよびテスト強化の改善活動は成果をあげたと言える．

一方，課題も見られる．改善前の数値に対して統計的な有意差が認められなかった B 組織（4 年後）のテスト摘出バグ数は 160.91 と多いままである．B 組織（3 年後）には 130.68 と減少したものの，4 年後に再度増加した．これは，A 組織と比較してテスト開始時点で残存するバグが多いためと考える（表 7-2・No.7 参照）．B 組織（4 年後）の総摘出バグ数が 117.70 と多いことから，設計そのものの質を向上させることが必要である（表 7-2・No.5）．今回の改善活動では，設計・製造に対する改善を実施していないが，その必要性を示していると考える．

また，レビューとテストを強化する施策の成果はでているものの，B 組織（4 年後）のレビュー工数 113.62，テスト工数 139.86 はどちらも A 組織より多いことから，工数増加によるバグ摘出の増加だけでなく，その効率化に取り組む必要がある（表 7-2・No.3，4）．

B 組織（4 年後）の総工数が，128.59 と増加している点について考察する（表 7-2・No.1）．総工数の増加は，単位規模を開発するための工数を意味する生産性の低下を表す．生産性の低下は，今後，ビジネス上の大きな課題となりうる．B 組織の改善前は，①適正な工数をかけて開発されていなかった，このため，②出荷後のバグが多くその対応工数がかかっていた，という 2 点が課題だったと考える．①の「適正な工数」を判断するのは難しい．組織の保有する技術力と，出荷後バグ数に基づいて判断する必要がある．B 組織の場合は出荷後バグ数を課題としており，A 組織との比較分析でレビューとテストが不足と判断した．これは，組織の保有する技術力と要求される出荷後バグ数に対して，適正な工数がかけられていなかったとみるべきである．一方，②については，B 組織では出荷後バグ数が減少しているので解決の方向にある．一般に，出荷バグ数が多いと，開発開始時から出荷時までの範囲で単位規模を開発する工数が増加したとしても，出荷後の保守段階までの範囲で比較した場合では，総工数は逆に減少すると考えられる．本著者が所属する組織での計測結果によると，出荷後にバグ 1 件を修正するための工数は，設計・製造段階で修正する工数の少なくとも 100 倍以上必要である．出荷後に出荷後バグとして顕在化するまでの期間が長期化すればするほど，当該バグを含む機能を直接開発した開発者が減り，開発情報が不足するため，修正工数は増加する．したがって，B 組織のようにレビュー摘出バグ数を増加す

ることにより，出荷後バグ数を低減する改善活動によって総工数が増加した場合は，出荷後の保守工数まで考慮すると総合的なコスト増にはつながらないと考える．しかしながら，レビューやテストの効率化は B 組織の今後の課題の 1 つと考えており，効率化は常に念頭におくべき項目と考える．

次に，図 7-8 下段の散布図により，B 組織の出荷後バグ数に対する各データ項目の改善前，改善開始 3 年後，および 4 年後の推移を観察する．図 7-8 下段左の設計・製造工数の散布図では，B 組織（改善前），B 組織（3 年後），B 組織（4 年後）のデータはほぼ重なって分布している．これに対して，レビュー工数とテスト工数では，B 組織（改善前）と B 組織（3 年後）の分布に大きな差異は見られないが，B 組織（4 年後）は明らかに傾向が変化していることが観察できる．B 組織（4 年後）の回帰直線の傾向を見ると，レビュー工数と出荷後バグ数では，右下がりの傾向が観察される（図 7-8 下段中央および表 7-4 参照）．これは，レビューをすればするほど出荷後バグ数が減少することを示している．テスト工数でも同様の傾向が観察される（図 7-8 下段右および表 7-4 参照）．これらの結果からも，B 組織のレビューおよびテスト強化の改善活動は成果をあげたことがわかる．

(2)　バグ分析と 1+n 施策の実施強化

ソフトウェア品質会計の技法の 1 つである「バグ分析と 1+n 施策」は，改善前から B 組織においても実施していた．「バグ分析と 1+n 施策」の成果は，1+n 施策成功率という以下の算出式により算出される尺度で計測する．

1+n 施策成功率（%）＝（1 件以上の同種バグを摘出した 1+n 施策実施ケース数／総 1+n 施策実施ケース数）×100.

バグ分析と 1+n 施策の目的は，同種バグを摘出することである．このため，同種バグを摘出できた「バグ分析と 1+n 施策」の実施ケースを成功とし，摘出できなかったケースを失敗とみなしてその成功率を算出するのが上記算出式の意味である．

表 7-2 を参照すると，改善前の B 組織の 1+n 施策成功率の平均値は 46.50 であり，A 組織の約 1/2 であった．B 組織は A 組織と同様，出荷後に顧客で発生したバグ分析に対してバグ分析と 1+n 施策の実施を義務付けていたものの，同種バグが摘出できず成果が出ていなかった．それは図 7-9 でも観察できる．A 組織の回帰直線は右下がりの傾向を示しており，同種バグの成功率が高いものほど出荷後バグ数が少ない．これに対して，B 組織は出荷後バグ数と 1+n 施策成功率は逆の右上がりの傾向を示している（図 7-9 および表 7-5 参照）．このように成果がでていなかった理由は，B 組織のバグ分析方法がソフトウェア品質会計の「バグ分析」ではなく独自のなぜなぜ分析であったことと，1+n 施策がバグ分析の結果と結びついていないケースがあったためである．

B 組織の改善活動においては，「バグ分析と 1+n 施策」を，そのテスト終盤に摘出された

重大バグおよび出荷後顧客で検出されたバグに対して実施するよう改めて義務付けた．本事例では，これらの条件に該当する全「バグ分析と 1+n 施策」ケースに対して改善に取り組んだ．

B 組織の改善開始後，1+n 施策成功率の平均値は 3 年後に 83.20，4 年後に 123.56 と，改善前から比べて 2.6 倍となった．改善前に対して，4 年後の数値は統計的にも有意差がある（表 7-3 を参照）．図 7-9 を参照すると，出荷後バグ数と 1+n 施策成功率は，改善前の傾向に比べて 3 年後はまだ変化に乏しいものの，4 年後は明らかに傾向の変化が観察される．

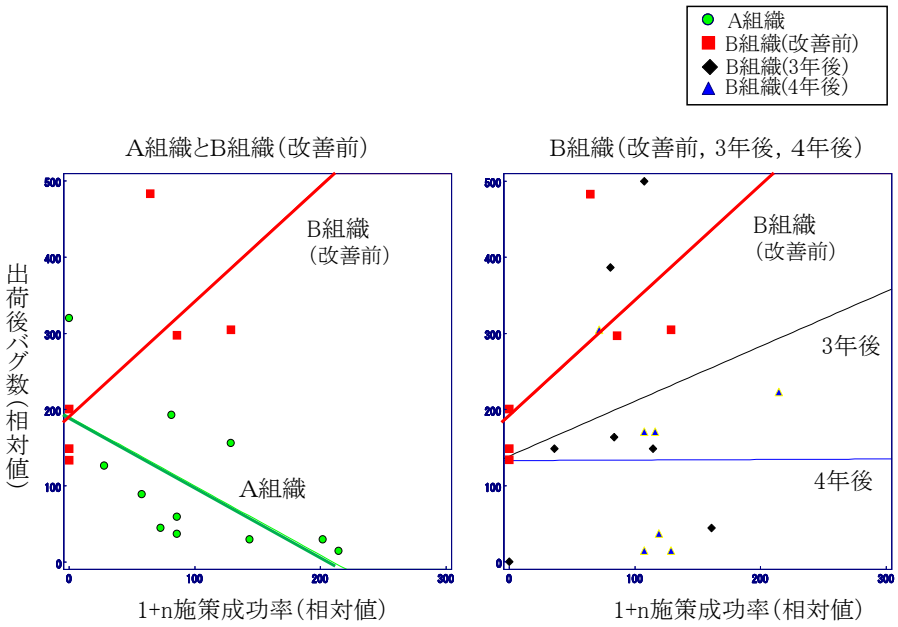


図 7-9　出荷後バグと 1+n 施策成功率との関係

表 7-5　図 7-9 の散布図中の回帰直線の分析結果

	A組織	B組織		
		改善前	3年後	4年後
相関係数	−0.651	0.64	0.212	0.003
回帰定数項	190.821	190.829	138.94	132.889
回帰係数	−0.908	1.517	0.729	0.007
t値	−2.576	1.665	0.485	0.006
P値(両側)	0.03	0.171	0.648	0.995

改善前と比較し，B 組織における「バグ分析と 1+n 施策」の適用上の変更点を以下に述べる．

- 品質会計のバグ分析手法（具体的には第 4 章の図 4-2 に示すバグ分析シートを使用）を全面的に適用するよう変更する．
- 品質会計のバグ分析手法を適用するにあたって，バグ分析の教育を実施する（開発拠点は中国オフショア開発を含め，日本全国に散在しているため，全拠点を訪問して教育）．
- 開発部門によるバグ分析結果に対して，開発部門とは独立した組織である品質保証部門が，その分析結果の妥当性について 1 件毎にレビューする．的確に分析できていない場合は，具体的に指導をする．
- 同時に 1+n 施策立案内容についても，品質保証部門がその妥当性をレビューする．従来は，バグ分析結果と 1+n 施策が結びついていない事例や，1+n 施策対象範囲を根拠なく絞り込む事例が散見されたが，それを未然に防止することを狙った．
- 1+n 施策実施結果に対して，品質保証部門がレビューし，実施結果の妥当性をレビューした．同種バグの摘出を継続すべきと判断した場合は，再度，バグ分析と 1+n 施策の実施を指示した．
- バグ分析と 1+n 施策が終了した時点で，バグ分析の対象バグの重大度に応じて，各種マネジメント会議での報告を義務付けた．これは，トップによるバグ分析と 1+n 施策実施結果のレビューとともに，組織内での情報共有の意味をもつ．

B 組織が独自の「なぜなぜ分析」ではなく，ソフトウェア品質会計の「バグ分析」を適用することによって成果を出した理由は，4.6 節でも考察したとおり，バグ分析の目的設定に関係する．繰り返しになるが，なぜなぜ分析は，根本原因を分析するための，広く一般的に適用可能な技法である．分析対象の根本原因は 1 つではなく，複数存在することがほとんどである．このため，特定の目的を狙って根本原因を分析しなければ，複数の根本原因が分析されてしまい，欲しい根本原因へたどり着くことができない．単純に「なぜ」を繰り返すのでなく，特定の目的を意識して，その目的へ向かって根本原因を分析する必要がある．これに対して，「バグ分析と 1+n 施策」の「バグ分析」は，同種バグの摘出という目的を達成するために最適化した根本原因分析技法である．その目的を達成するため，「バグ分析」の根本原因の分析観点を，分析対象バグの作り込み工程，作り込み原因，および見逃し原因の 3 点に絞っている．さらに，過去の根本原因を蓄積したバグ分析シートを準備し，分析しやすくする工夫をしている．分析対象を限定せず，広範囲に適用可能ななぜなぜ分析に比べて，きわめて目的指向である．

「バグ分析と 1+n 施策」の実施のために，教育や品質保証部門による分析支援，組織トップによる実施結果のレビューと組織内の情報共有の仕組みも，1+n 施策成功率をあげるために欠かせない重要な仕組みであり，B 組織ではこれらの基盤となる仕組みを同時に整備している．これが，B 組織で成果を出した理由である．

本節において，「バグ分析と 1+n 施策」の実施結果の改善を個別に取り上げて説明しているには理由がある．根本原因分析の能力向上は，組織のプロセス改善に寄与すると言われ

ている[7-1]．それは，根本原因を的確に分析する能力が，組織における課題に対する的確な解決能力に通じるからである．したがって，B 組織の 1+n 施策成功率の向上は，課題への的確な解決能力を向上することと同等であり，この能力の向上が，B 組織の品質向上施策全体の成功の原動力となったと考える．

(3)　品質保証部門による顧客視点の評価の実施

B 組織の品質保証部門は，開発部門とは独立した組織である．品質保証部門は，改善前には，開発途中に計測した開発データを分析することにより，開発状況を把握していた．しかし，実際の開発成果物を動作させて確認する評価は実施していなかった．

今回の改善施策により，開発部門によるテストとは別に，品質保証部門が開発の最終成果物に対する顧客視点の評価を実施することにした．この施策により，開発途中のデータによる品質管理だけでは把握できないソフトウェアのバグが実際に検出できるようになった．顧客視点の評価とは，顧客での利用形態を想定した利用シナリオに沿った評価である．

品質保証部門による顧客視点の評価の成果は，非常に大きかった．一連の改善施策のなかでも，最初に効果を出した施策である．その理由は，開発データにより問題点を指摘するだけでなく，最終成果物のバグを摘出することにより，問題の存在を実証したからである．この施策は品質保証部門の地位を向上させ，以降の改善施策を実行しやすくした．なお，品質保証部門の顧客視点の評価の実施により，テスト摘出バグ数（表 7-2・No.7）の 4%にあたるバグが摘出されるようになった．

さらに，品質保証部門による顧客視点の評価結果を出荷判定基準に取り込み，基準値以上のバグが摘出された場合は，出荷停止とするなど，出荷判定基準を改訂した．これにより，品質に問題のある製品は実際に出荷停止となり，出荷判定基準を満足するまで改善施策を実施することが義務付けられるようになった．

(4)　プロジェクトマネジメント会議の工夫

開発に関係する全開発部門の責任者と品質保証部門の責任者が出席するプロジェクトマネジメント会議（以降，「日程会議」と呼ぶ）の開催方法を 3 つの面から改善した．フェイスツーフェイスにて週次開催するようにしたこと，当該週の開発データに基づいて開発状況を議論するようにしたこと，および品質保証部門が参加するようにしたことである．改善前は，週次で開催していたもののフェイスツーフェイスではなく，品質保証部門は参加していなかった．また，開発データをまとめて報告するのは工程終了時であり，その際に品質保証部門による工程移行判定を書面で実施していた．改善後は，品質保証部門が毎週の開発データに基づく分析結果を報告して議論するようにしたことと，フェイスツーフェイスであるために，特に分散開発拠点の責任者とのコミュニケーション向上により，開発途中の問題点が早期に検出できるようになったと考える．開発途中の問題を発生時に把握できるようになると，その解決策の立案から解決まで日程会議の参加者全員が合意して進

めるようになる．このため，形式的な書面による工程移行判定が不要になり，より実質的なフォローができるようになった．また，週次できめ細かくフォローできる体制が整ったため，レビューやテストの強化のフォローに効果をあげた．

7.3.5　考察

B 組織が出荷後バグ数の低減に成功した大きな要因として考えられるものを，以下にあげる．ここであげる取り組みはいずれも，品質会計技法の厳格な適用および品質会計に関連した仕組みの整備により実現した内容である．

① レビューによるバグ摘出 80%の推進

B 組織は，従来から品質会計を適用していたため，レビューでのバグ摘出そのものは定着していた．今回の改善により，レビューによるバグ摘出がほぼ 80%を達成するまでに向上し，図 7-7 に述べるような成果を得ることができた．品質会計考案組織のこれまでの事例でも，レビューによるバグ摘出は品質改善に効果があった．さらに，それが出荷前摘出バグ数の 80%以上をレビューで摘出するようになると，出荷後バグ数の低減に大きな効果をもたらすものと考えられる．

② 的確なテスト完了判断

出荷時点における品質レベルの判断精度の向上は，出荷後バグ数の低減に直接影響を及ぼす重要な要因であるが，出荷時点において，所定の品質を確保しているかどうかの判断は非常に難しい．B 組織では，計画したテストをほぼ終了した時点で適用する品質会計の 3 つの技法（バグ傾向分析，バグ分析と 1+n 施策，およびバグ収束判定）の厳密な適用と，品質保証部門の顧客視点での評価を組み合わせることによって，それを実現した．B 組織において新たに開始した品質保証部門の顧客視点の評価において基準値以上のバグが摘出された場合は，出荷不可とするよう改訂したため，品質に問題のある製品は必ず出荷延期となった．さらに，その検出された品質問題を解決するための追加テストにおいては，品質会計による分析結果により，具体的なテスト内容を提供した．特に，バグ分析と 1+n 施策では，改善を推進した 3 年間の間に，2.6 倍の同種バグ件数を摘出することができた（表 7-2，No.10　1+n 施策成功率，改善前：46.50，4 年後：123.56）．

③ データに基づく短サイクルのフェイスツーフェイスマネジメント

データに基づく品質確認は，従来は工程移行時に実施（以降，工程移行判定と呼ぶ）していた．この工程移行判定という手法は，ソフトウェア開発現場で広く普及している方法である．しかしながら，工程移行判定は品質向上にはあまり効果がないことが多いと考えられる．その理由は，工程移行判定での議論内容は，開発途中に発生した問題の終結と工程を終了した結果を確認するものであり，結果の追認にしかならないことが多いためである．実施すべきことは，問題発生時にそれをより早く把握し，問題の真の原因を的確に究明して対策を実施し，現場の状況を見ながら終結させることである．

少なくとも週次の短サイクルでデータに基づくマネジメントを実施することによって，問題発生に早期に気がつくことができる．さらに，フェイスツーフェイスの会議は，特に意識しなければ顔を合わせることがない分散開発拠点にとって，コミュニケーションを向上し，現場の問題を早期に報告し解決する行動をとりやすくするという利点がある．

本改善事例において，日程会議運営方法の見直しは品質向上に重要な役割を果たした．

④ 品質保証の仕組みの整備

B 組織では，最終的に A 組織と同じ品質保証体系（図 7-10 参照）を適用した．週次のデータ収集，日程会議でのマネジメント，品質保証部門による顧客視点の評価の追加など，開発工程全体に渡るソフトウェアプロセスの総合的な整備が，品質向上に寄与したものと考える．品質保証体系の各要素は，どれも B 組織にとって欠かすことのできない要素である．品質保証の仕組みは，開発するソフトウェアや顧客の特性によって要件が異なる．適切な品質保証の仕組みを整備するには，出荷後バグの分析が必要である．当該組織が開発するソフトウェアにとって重大問題と考えられるバグが顧客で多数発生している場合は，品質保証の仕組みが充分であるとは言えない．例えば，高信頼性が求められるソフトウェアであるにもかかわらず，動作停止を招くバグが顧客で多数発生しているような場合である．この場合は，品質会計のバグ分析により，その出荷後バグを作り込んだ原因と見逃した原因を分析し，それらの原因を未然防止するように設計技法を見直す等の仕組みの見直しを実施する．その繰り返しによって，組織の特性に合わせたソフトウェア全体に渡る品質保証の仕組みを構築することができる．

⑤ 現場主義の実践

品質会計は，数値データだけに頼ることなく，現場へ行って事実を把握することを強く推奨している．その理由は，数値データだけでは，現場のさまざまな問題の予兆を見逃す危険性が高いからである．上述のフェイスツーフェイスマネジメントの推奨も現場主義の一環である．今回の B 組織での改善経緯を考慮すると，短サイクルでのフェイスツーフェイスマネジメントや組織的な取り組みは，関係者間のコミュニケーションの向上を促し，品質問題に関する活発な意見交換や，自らの改善への取り組みにつながったと考える．現場主義の実践による直接的な成果を提示するのは難しいが，現場に根ざした事実を重視する活動は，ソフトウェアの品質向上に貢献すると考えられる．

7.4 オフショア開発 C 組織における品質向上事例

本事例は，NEC 中国子会社（以下「C 組織」と呼ぶ）のオフショア開発の品質向上事例である．オフショア開発とは，ソフトウェア開発を海外に所在する企業にアウトソーシン

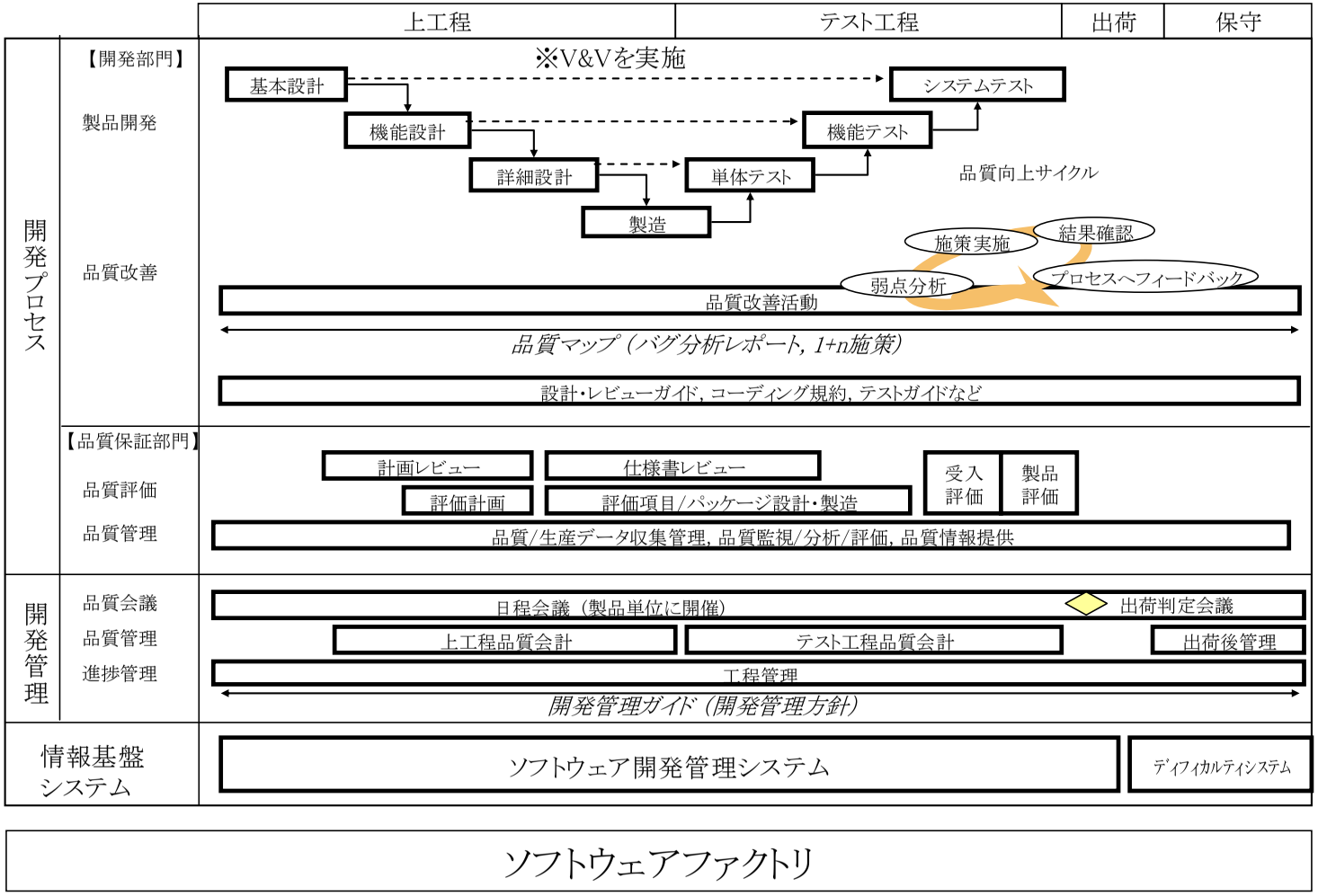


図7-10　A組織およびB組織の品質保証体系

グすることをいう．オフショア開発は，海外企業とのソフトウェア開発技術の差異に加えて，コミュニケーション言語や文化の違いなどに起因した問題発生が多く，それらに対する取り組み事例が報告されている[7-2][7-3][7-4][7-5].

NEC から中国へのオフショア開発は，1980 年代後半から始まった．C 組織は，2000 年代初めの品質向上活動開始前には，バグが多い，納期が遅延する，コミュニケーションが悪い，といったオフショア開発でよく見かける問題を抱えていた．C 組織の親組織にあたる A 組織は，改善の兆しが見えない状況に危機感をもち，このままでは A 組織のビジネスに影響を及ぼしかねないとの判断から，C 組織の品質向上プロジェクトを発足した．この品質向上プロジェクトは，2002 年～2004 年にかけて活動し，1 年 10 ヶ月で品質および生産性の目標を達成し，ソフトウェア開発における日本国内並みの品質・生産性を達成することに成功した．本著者は，この品質向上プロジェクトのリーダを務めた．

7.4.1　C 組織の課題

C組織の開発内容は，ソフトウェア汎用製品の開発であり，A組織の開発を一手に引き受けている．ソフトウェア汎用製品であるため，同一製品を継続して開発していくタイプの開発である．このため，要員を固定化して技術向上を図りやすいという特徴がある．

C 組織は当時設立から約 10 年経過していたものの，A 組織から C 組織への技術移転と品質・生産性向上がなかなか進まず，このままでは，同じグループ会社として双方にとって大きな損失となることが予想された．具体的な問題点は以下の通りである．

（1）人員数の急拡大によるマネジメント力不足

C組織は，A組織の要請を受けて，100人規模から600人規模へ2000年以降技術者数を急拡大させている．しかし，これに対するマネジメント層が相対的に薄く，マネジメント力が不十分であった．このため，せっかく増加させた人員をうまく使えない状況にあった．

（2）業務内容の高度化による開発技術力不足

C組織は，人員数を急拡大させた影響で，若年層の技術者が多く製品開発経験が少ない上に，OJT（On the Job Training）型で身に付けるような開発ノウハウを学ぶ機会がほとんどなかった．また，ソフトウェア開発に欠かせない組織的な開発力が不足していた．一方，A組織側はオフショア開発経験が少なく，海外へのスムーズな技術移転方法をもっていなかった．

（3）日本語力不足によるコミュニケーション不足

日本向け製品を開発している事情から，技術者の日本語力は必須だが，仕様を議論できるような高度の日本語能力をもつ人材は少なかった．これにより，仕様を説明してもうま

く伝わらなかったり，開発スケジュールの変更があっても伝達漏れがあり，対応が遅れる等の問題が発生した.

7.4.2　改善施策

上述の課題は，A組織とC組織のどちらか一方の課題でなく，両方の課題であるとの認識をもった．また，改善のために早期に投資することにより，後のメリットも大きいとの見通しをもった．そこで，A組織とC組織のトップ指示のもと，両者が協力してC組織のソフトウェア開発の品質・生産性を向上させる活動を開始し，オフショア開発の改善に取り組んだ．主な改善施策は以下の通りである.

(1)ステップアップモデルによる改善フレームワークの定義

改善の進め方として，改善対象組織の改善意識の向上を促すことによって，改善対象組織が自ら技術や仕組みの改善に取り組み，技術や仕組みの確立を待つという方法と，既に確立した技術や仕組みを改善対象組織へ導入して定着させる方法の2つが考えられる．C組織のケースでは，後者の方法を選択した．その理由は，ビジネス要求として短期間でC組織の品質・生産性を向上させる必要があったためである．C組織自ら工夫を重ねて技術や仕組みを改善していく前者の方法では，長期の改善期間が必要となる．これに対して，既に効果があるとわかっているA組織の，品質会計を軸とした技術や仕組みをそのまま導入するほうが，短期間での改善を達成しやすいと判断したのである.

改善の全体のフレームワークを定義するために，段階を追って確実に技術や仕組みを移転し，同時に品質・生産性を向上させるステップアップモデルを考案した（図7-11参照）．ステップアップモデルとは，既に日本で確立された技術や仕組みを，短期間でオフショア開発先の組織へ導入・定着させるためのモデルである．このモデルに従うと，最短で1.5年で確実に技術を移転し，C組織は確実に品質･生産性を向上できる．ステップアップモデルは，以下の3段階で構成されており，各ステップの終了基準として，品質，生産性，日本語力に対して具体的な数値目標を設定した.

①ステップ1：「共同開発」

OJTで修得するような基礎的な技術と基本的な製品知識の移転を狙いとする．開発チームをA組織とC組織の共同チーム編成にし，可能な限りともに開発を実施する．基礎的な技術修得により，大きく品質・生産性がばらつくことはなくなり，ある範囲内で品質・生産性を確保できるようになる．ソフトウェア品質会計により，A組織が組織的な定着に成功したレビューによる早期品質確保も基礎的な技術に含まれる.

②ステップ2：「準自立開発」

ソフトウェア品質会計をはじめとしたマネジメント技術，保守技術，および自主開発管理ができるレベルの製品開発知識の移転を狙いとする．A組織の作成した要求仕様書に基づ

き，機能設計から機能テストまでをC組織が実施するとともに，C組織は自主管理による開発の進め方を修得する．品質会計による品質分析の結果に基づく品質向上施策の実施や予防策の実施など，組織による開発方法を修得する．これにより，組織として確実に所定の品質・生産性を確保できるようになる．

③ステップ3：「自主開発」

品質・生産性がA組織と同等レベルになるように，ステップ2までに修得した開発技術・管理技術・保守技術・製品開発知識に対して，C組織が自主的に改善を繰り返す．ステップ3を達成すると，C組織は日本国内ソフトウェア組織と同等の開発力をもつ自主開発組織となる．

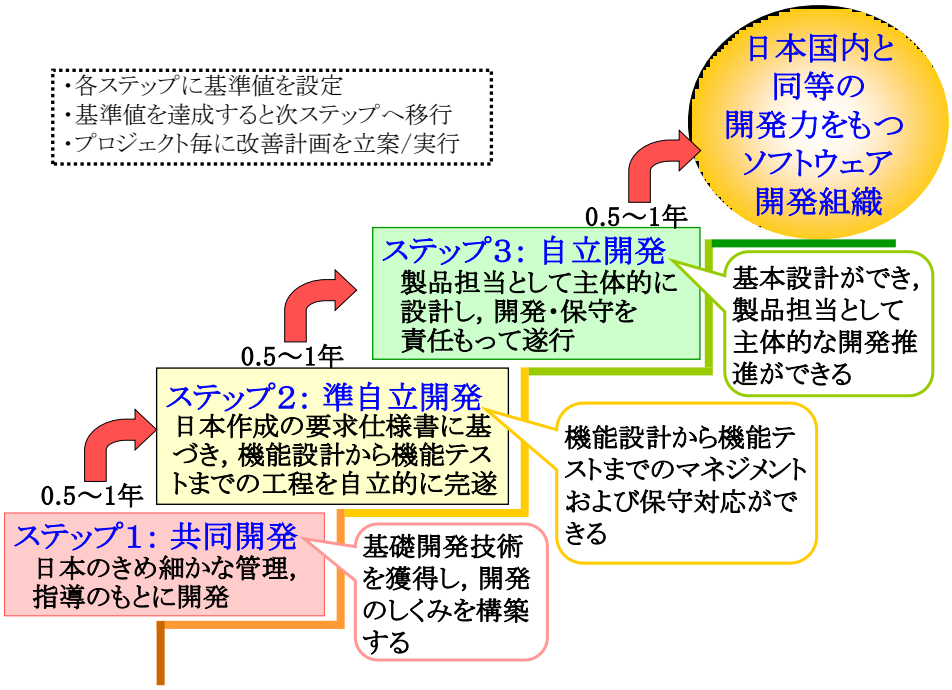


図7-11　ステップアップモデル

(2)アセスメントによる開発弱点の検出と強化

ステップアップモデルを補足するツールとして，開発の強み・弱みを検出するアセスメントシートを用意し，開発の弱点を効果的に改善した．このアセスメントシートは，開発作業・開発管理・スキル向上・設備/体制面の4カテゴリ15項目の基本項目から構成されている（図7-12参照）．アセスメントシートに沿って各カテゴリを評価することにより，評価点が算出され，強み・弱みが自動的に判明する．アセスメントシートは，ステップアップモデルのステップ毎に3段階準備しており，A組織（発注元）とC組織（発注先）の両面から評価できる．このため，弱点は，A組織とC組織の双方から検出し改善できる．

(3)標準化の推進

標準化は，C組織とA組織の両方から実施し，その結果を情報共有Webサイトに掲載している．C組織では，開発ライフサイクル全般に渡って，設計仕様書，言語別のコーディング規則，テスト仕様書，レビュー技術，テスト技術等の標準を整備した．A組織では，オフショア開発方法の仕様書記載方法，進捗管理方法などを整備した．

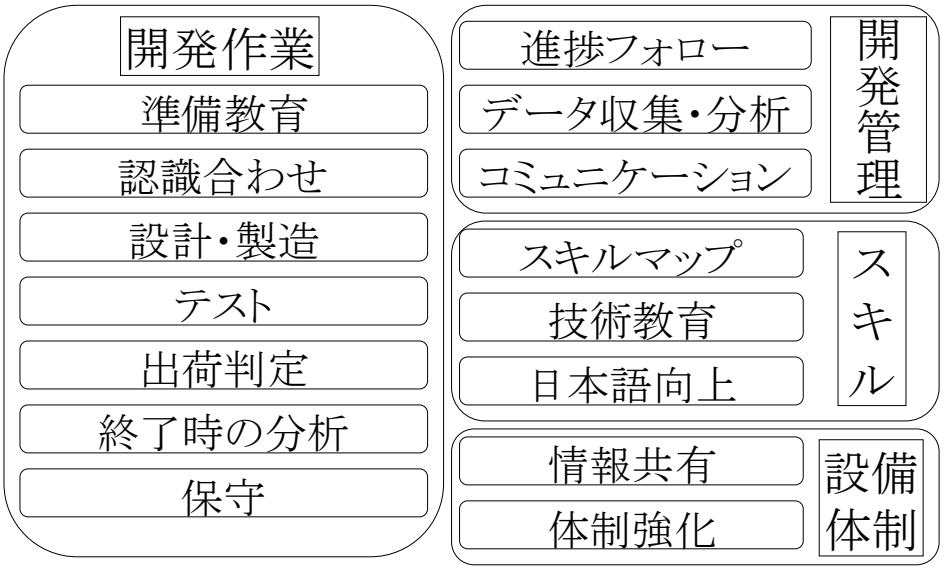


図7-12　アセスメントモデルの基本項目

(4)インフラの整備

オフショア開発を円滑に進めるためには，それをサポートするインフラの整備が欠かせない．インターネットによる遠隔会議設備，情報共有Web，FTPサーバ，開発管理ツール等を整備することにより，改善を側面からサポートした．

(5)日本語能力の強化

C組織の全社的に技術者の日本語能力を強化するという方針のもと，日本語能力向上に取り組んだ．日本語講座カリキュラムの充実と出席チェック，計画的な日本への長期出張や業務実習，日本語を使う職場環境の整備などを実施した．

(6)定期的なトップマネジメントレビューの開催

推進計画に基づき改善活動を進めるとともに，半年に１回の頻度でA組織のトップがC組織を訪問し，C組織トップとともにC組織技術者と顔を合わせたトップマネジメントレビューを開催して，改善状況をフォローした．この効果は大きく，C組織技術者と発注元であるA組織技術者双方にとって刺激となり，改善サイクルがうまく回るようになった．

7.4.3　改善結果

以上の取り組みを開始して1年10ヶ月経過し，C組織の品質・生産性は大きく改善した．以下にその内容を示す(図7-13参照)．

（１）品質

開発物件納入後のA組織で摘出されるC組織のバグ数は，改善前に比べて85%削減し，開発物件納入時品質が6倍向上した．また，開発条件によって，納入後のバグ数が大きくばらつくケースが減り，安定して開発できる実力がついてきた．すべてのプロジェクトがステップ2以上の実力を備え，機能設計から担当できるようになった．

（２）生産性

単位時間当りの開発量は，改善前に比べて3倍向上した．標準化が進み，品質が改善されるのに伴って，効率的な開発ができるようになるとともに，後戻り作業が激減した．

（３）日本語力

日本語検定の各級保有比率は，改善前に比べて3.4倍と大幅に向上した．日常の開発では通訳不要となり，開発者による日本語での仕様書作成が可能となった．開発を進める上でも，スムーズにコミュニケーションできるようになったと実感している．

（４）開発コストの削減

開発コストは，改善前に比べて大きく削減し，活用効果率は改善前より14%向上した．活用効果率とは，日本国内で開発するコストを100%としたとき，オフショア開発によりかかったコスト（オフショア開発費および日本国内の管理コストを含む）を差し引いた比率をいう．活用効果率を金額に換算した額は，オフショア開発を利用することによる開発削減額である．この開発削減額は本改善活動を実施しなかった場合と比較して，年間4億円を超える．

7.4.4　考察

C組織の品質向上事例は，オフショア開発においても品質向上が可能であることを示すものである．また，納品後バグ数の削減と同時に生産性（Line/人・時間（H））が向上している点が重要である．C組織はステップ１の段階から，レビューの強化を実施した．レビューの強化を初めて実施する場合は，開発を回り道するように感じられるため現場から反発が起こることが多い．技術者からすると，コーディングできる状態にあるにもかかわらず，レビューばかりしていて，コーディングへの着手を妨げているように感じられるからだ．また，従来よりもレビュー時間を増やすにもかかわらず，納期は変わらず，生産性目標値を向上（＝単位規模当りの開発にかかる工数を低減）することを求められている点も矛盾しているように感じられるのである．C組織でも，品質向上活動の初期段階では，反発があった．これに対して，技術力がありレビューでの効果を出せそうなプロジェクトを選んで，

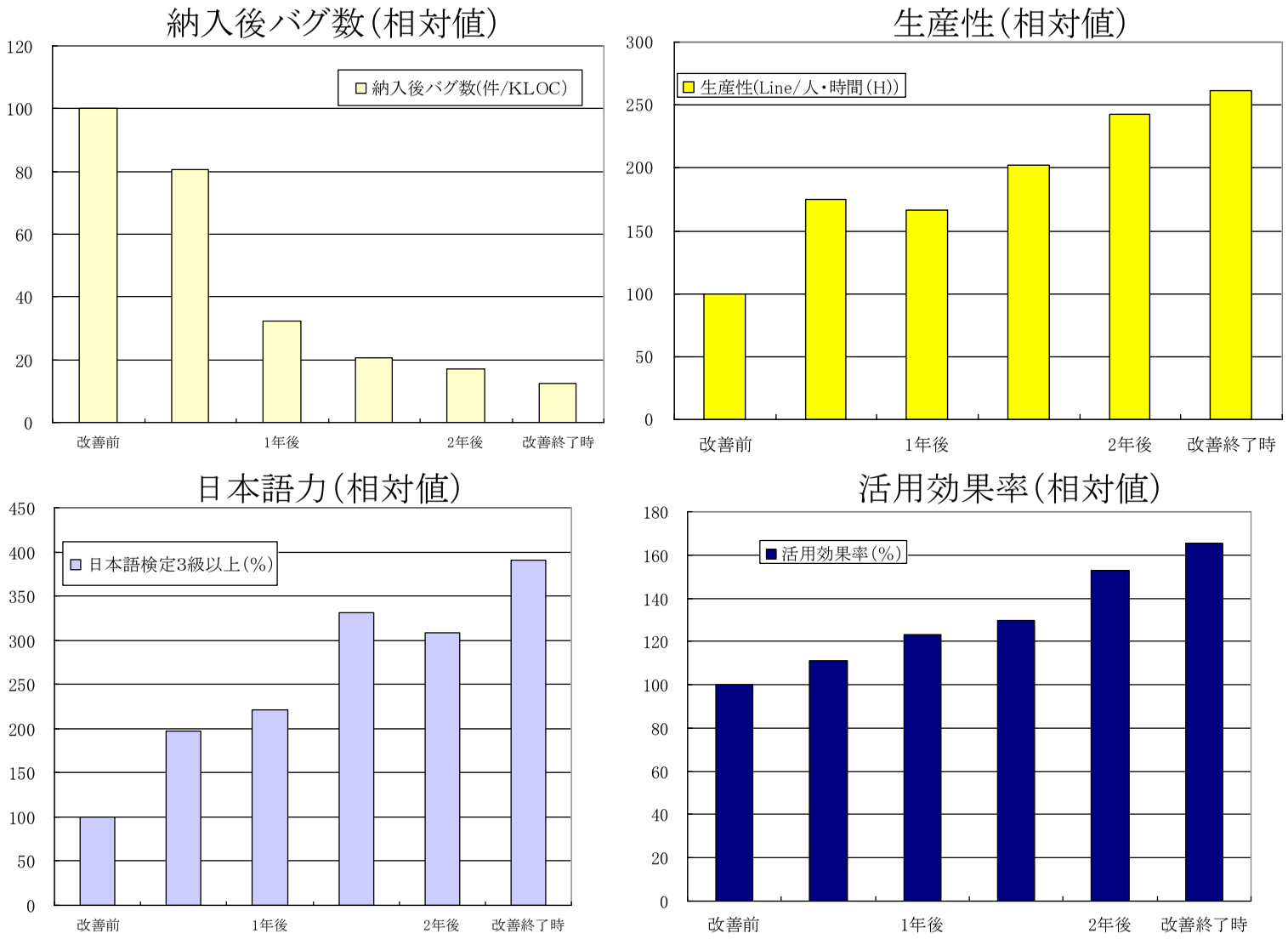


図7-13　C組織の改善結果

(すべての数値は改善前を100としたときの相対値)

集中的に支援することにより，先行して成功する事例を作った．成功事例ができると，あとは競争のように成果が上がるようになった．現場は納得し，積極的に改善活動に参加するようになったのである．レビュー増加は，全体の開発工数の増加に直結するわけではない．むしろ，レビューを確実に実施することで，後戻り工数が減り，全体の工数を低減することが出来るということを，C組織は実体験したのである．NECには，「品質を追求しよう，生産性は後からついてくる」という教えがある．まさに，レビュー強化によって早期品質確保を図ったことにより，結果として生産性向上を達成することができた．活用効果率の向上から，実際に開発コストを削減できている点も重要である．

C 組織は，本品質向上活動終了後，自ら改善活動を継続し，その半年後に CMMI レベル 5 を達成した．ステップアップモデルでは，自立的な開発を実現するソフトウェア開発組織を目指したが，まさにそれを達成した証である．

7.5　おわりに

本章では，A 組織，B 組織，および C 組織の 3 つの品質向上事例を紹介した．3 つの事例に共通することは，ソフトウェア品質会計を軸とした改善活動であることである．特に，レビューによる早期品質確保を実行していること，同時に品質を保証する組織的な仕組みの構築によりレビューによる効果を支援していることが重要である．さらに，C 組織においては，レビュー強化による品質確保が結果として生産性向上をもたらすことを示した．ソフトウェア開発で，予定より工数が増大してしまう大きな理由のひとつは後戻り作業である．後戻り作業の代表的なものは，上流工程で見落とした仕様漏れや設計ミスなどが最終のテスト段階で検出されるために，再度設計・コーディング・テストをやり直すことである．レビューの強化による早期品質確保は，これらの後戻り作業を削減する効果がある．これにより，上工程ではレビュー強化のため一見工数が増加するようにみえるが，出荷までの全体工数は減少するのである．これが C 組織において生産性向上，およびコスト削減（活用効果率向上）をもたらした理由である．

3 つの事例から，組織的な仕組みの構築の重要性を注目すべきである．ソフトウェア開発においては，レビュー強化のような単一の改善施策だけでの品質向上は難しい．同時にソフトウェア開発全体の仕組みを整備していくことにより，改善施策が効果的に実施されるようになり，結果として品質向上が実現するのである．さらに，組織的な仕組みの構築が，人間的要因へ好影響を及ぼすことも注目すべきである．B 組織では，改善期間中において，技術者の行動に変化が見られた．組織的な仕組みを整備し，フェイスツーフェイスのマネジメントを推奨することにより，関係者間のコミュニケーションが向上し，品質問題に対して活発に意見交換するようになったのである．人間的要因の改善は，直接的ではないものの，品質向上の好循環を生み出すことに寄与したと考える．

第8章　ソフトウェア品質会計の工学的価値

8.1　はじめに

本章では，ソフトウェア品質会計の工学的価値を論ずる．ソフトウェア品質会計は，バグ数の摘出目標管理を軸とした品質管理技法である．品質会計の第一の価値は，このバグ摘出目標管理というわかりやすい方法を採用している点である．これが，品質会計の導入を容易にしている．当然のことながら，品質会計の価値は，そのような表面的な価値にとどまらない．

第 3 章の図 3-4 に示した品質会計の原則を下記に再掲する．これらの原則が，品質会計技法においてどのように実現されたのかを述べながら，その工学的価値を論述していく．

＜品質会計の原則＞
・ バグは作り込まない．作りこんだバグは素早く摘出する．

＜上工程品質会計の原則＞
・ 作り込んだバグは次工程までに摘出する．
  - 作り込み工程で80%摘出
  - 次工程で残り20%摘出

＜テスト工程品質会計の原則＞
・ 作り込んだバグは，すべて摘出してから出荷する．

＜目標＞
・ 上工程バグ摘出率　80%

図 8-1　品質会計の原則（再掲）

8.2　レビューによる早期品質確保

レビューによるバグ摘出を推進することによって，ソフトウェアライフサイクルの早期

に品質を確保しようという考え方は，ソフトウェア品質会計の核となる考え方の 1 つである．負債は利子で膨らまないうちに早期に返済するほうが経済的であるのと同様に，1 件のバグが複数のバグを生まないうちに，設計・製造で作り込まれたバグを早期にレビューで摘出するほうが後戻りが少なくて済む．品質会計は，上工程品質会計を考案することによって，それを実現した．

上工程品質会計の大きな特徴は，バグ 1 件毎に作り込み工程と摘出工程という 2 つの情報を明らかにすることにより，作り込み工程と摘出工程の 2 つの面からマネジメントする点である．バグの作り込み工程からのマネジメントは，品質管理領域における源流管理と同じ発想である．当該バグを作り込んだ工程を分析することによって，原因工程を明らかにする．バグ作り込み件数の多い設計工程は，そのソフトウェアの問題工程であるから，当該設計工程の成果物を再度見直す必要がある．それを上工程途中の早い段階で検出できる点が，品質会計の大きな強みである．さらに，上工程品質会計の原則で述べるように，作り込んだバグは次工程までに摘出するという考え方は，作り込んだバグによる影響を次工程までにとどめ，早期に品質確保することを意味する．それを，単なる考え方にとどまらず，バグ数を数値として表し，作り込み工程と摘出工程の 2 つの面からマネジメントすることによって，実際に分析可能としている点に価値がある．

源流管理の視点でマネジメントしていないプロジェクトでは，上流の設計工程の原因であるほどテストの後半にならなければ問題の認識が困難である．残念なことに実際の開発現場では，問題を認識する場面の多くが，最終工程であるシステムテストである．その結果，システムテストで摘出された設計問題を修正するため，多くの後戻り作業が発生する．再度，設計・コーディングを実施し，関連する周りの部分を含めたテストを実施しなければならないからである．それまで順調に進んでいたはずのプロジェクトが，システムテストに入って，突然，数多くのバグが摘出され，止まらなくなる事態の多くは，上流の設計で作り込んでしまったバグに気が付かずに開発を進めてしまったことに起因する．ソフトウェア品質会計は，そのような設計問題に起因する失敗プロジェクトを防止し，ソフトウェア開発の早い段階からの品質確保を実現する．

上工程品質会計では，レビュー工数（人・時間（H）/KLOC）とレビューによる摘出バグ数（件数／KLOC）という 2 つのメトリクスを使用して，品質確保状況を分析可能とする．品質を判断するための技術が，作り込み工程別バグ分析と品質判定表である．品質確保状況の視覚化のために，上工程品質会計票グラフを提供している．

品質会計が目標として掲げる上工程バグ摘出率 80%も，重要な目標値である．第 7 章の A 組織と B 組織の事例を参照すればわかるように，両組織とも，上工程バグ摘出率が 80%を超えた時期に出荷後バグ数が大きく低減した．これらの経験から，品質会計の掲げる上工程バグ摘出率 80%は，出荷後バグ数の低減に対して意味のある基準値と考える．

ソフトウェア開発において，レビューによる品質確保の重要性は古くから提言されてきたが，ソフトウェア開発の現場に適用可能な具体的なレビューのマネジメント方法の提案

は乏しい．そのようななかで，ソフトウェア品質会計技法は，レビューによるバグ摘出を的確にマネジメントできる手法および基準値を提供している．ソフトウェア品質会計は，レビューによる早期品質確保を実現する技法であり，その目標である上工程バグ摘出率80%は，出荷後バグ数の低減に対する１つの具体的な基準値を提供していると言える．

8.3　的確なテスト完了判断

品質会計の原則のうち，テスト工程品質会計会計の原則では，「作り込んだバグは，すべて摘出してから出荷する」としている．これは，品質会計の考案当時からある考え方である．もともと品質会計は，「バグを負債とみなしてその負債をすべて返してから出荷する」という考え方から発展してきた．この「負債をすべて返したか」を判断するためのテスト完了判断技法を提供している点が，品質会計の大きな特徴の１つである．

品質会計では，計画したテストをすべて実施した後にテスト完了判断をするために，「バグ傾向分析」，「バグ分析と 1+n 施策」，および「バグ収束判定」という３つの技法を使用して，テスト結果を分析する．３つの技法には，各々目的がある．「バグ傾向分析」は，摘出した全バグをさまざまな観点から層別して分析することで，テスト観点の大きな抜け漏れがないかを確認する．「バグ分析と 1+n 施策」は，テスト終盤に摘出した重要なバグに対して実施するものであり，重要なバグがテスト終盤まで残存した理由を明らかにし，その原因に対して対策を打つ．バグ分析と 1+n 施策は，バグ１件毎に注目することにより，細かい視点での重要なテストの抜け漏れを防止するための技術である．さらに「バグ収束判定」により，実際にバグが収束傾向にあることを確認する．この３つの技法の分析結果がいずれも問題なしとなったとき，「負債はすべて返した」と判断し，テストを完了する．負債とは残存課題，すなわちその残存課題により残存するバグを意味する．

ソフトウェア開発では，緻密さと正確さを要求される．１箇所の見落としが致命的な問題を引き起こす．顧客での致命的な問題の発生は，それまでの品質確保のための積み重ねをすべて無に返してしまう．それを未然防止するための技法が，品質会計が提供する 3 つの技法の組み合わせによるテスト完了判断である．テスト完了判断は，ソフトウェア開発組織にとって，難易度の高い問題である．B 組織の事例（第 7 章の 7.3 節参照）が示すように，品質会計の適用を徹底することにより，精度の高いテスト完了判断が実施できるようになる．品質会計の重要な特徴は，3 つの視点から判断し，いずれも問題なしと判断したときにテスト完了と判断する点にある．ソフトウェアは複雑であり，目に見えず，人間的要素の影響を大きく受けるという特性をもつ．このような特性をもつソフトウェアのテスト完了を，一面的な結果だけで判断するのは困難である．その意味で，総合的かつ具体的なテスト完了判断方法を提示している技法はほとんど見当たらない．そのようななかで，品質会計がテスト完了判断方法を提供しているのは価値がある．

8.4　バグ分析による課題解決能力の向上

第 4 章において，バグ分析と 1+n 施策の技法としての特徴を述べた．品質会計の技法の１つを取り上げて詳細に説明したのは，特にバグ分析の能力が，プロセス改善に与える影響の大きさを考慮したからである．

的確にバグ分析ができる技術者は，事実に基づいて原因と結果の因果関係を整理して考える能力があると言える．この原因と結果の因果関係を分析できる能力は，プロセス改善においても，的確に問題点とその原因を分析し，原因に対して対策を打つことができるという課題解決能力につながる．この因果関係を分析できる能力が乏しい場合は，思い込みによる根本原因分析になってしまう場合が多い．B 組織の事例（第 7 章の 7.3 節参照）においても，改善前には思い込みによる勘違いの改善活動が見られた．例えば，データを見て考える習慣のない技術者が，パレート図で比率の多い順に原因を表示しているにもかかわらず，それを使わずに自分の思いこんだ原因に対して改善活動を実施したために，成果があがらないケースが実際に散見された．しかし，B 組織の品質向上活動を開始し，1+n 施策成功率が向上するにつれて，データや事実を見て因果関係を考える習慣ができ，課題解決能力が増したものと推察される．1+n 施策成功率の数値は，ある意味で問題点と原因の因果関係の能力を表すものと考えてよい．

品質会計におけるバグ分析は，同種バグを摘出するという目的のために，根本原因の構造を整理して，バグの作り込み原因と見逃し原因の２つの視点から分けて分析する方法を提供している．さらに根本原因のうち，固有根本原因を意識して分析し，共通根本原因は同種バグ摘出には効果が低いため，根本原因の対象には含めないとしている．固有根本原因とは，技術情報の欠落による設計ミスなど，そのバグを作り込み見逃した直接の原因である．共通根本原因とは，教育不足など，そのバグを作り込み見逃した間接的な原因である．共通根本原因は間接的な原因であるため，共通根本原因に対して施策を実施しても，確実に同種バグを摘出できるとは限らないのである．このような，バグ分析の目的を意識して根本原因の構造を整理する考え方は，プロセス改善の場面においても応用可能である．バグ分析は，ソフトウェア開発の現場でよく用いられる技法の１つである．しかしながら，現場で最も適用されているのは，どのような場面でも使える汎用的な技法であるため，時間をかけているわりに効率的に根本原因を分析できない．汎用的な技法であるトヨタ式「なぜを５回」をソフトウェア領域へ応用する方法の提案は多いが，なぜの導き方のコツを示しているものがほとんどで，根本原因の構造を整理する考え方を提示しているものは見当たらない．そのようななかで，ソフトウェア品質会計は，根本原因の構造を整理する考え方を示したうえで，同種バグの摘出という目的に絞ったバグ分析技法を提供している点に大きな価値がある．さらに，バグ分析と 1+n 施策の適用により培ったバグ分析の能力は，プロセス改善などの汎用的な課題解決能力の向上にも寄与する．

8.5　品質改善ドライバとしての価値

ソフトウェア品質会計は，品質管理技法として直接的な効果を発揮するだけでなく，品質改善ドライバとして価値がある．品質改善ドライバとは，ソフトウェアプロセス上の弱点を特定し，その弱点を解決する技術や仕組みの導入を促す先導役をいう．

品質会計の適用により，バグの作り込み件数と摘出件数を初めとしたさまざまな開発途中のデータを得て開発状況を分析するようになる．その結果，自らの適用技術や仕組みの強みや弱みを得ることができる．その弱みを強化する技術や仕組みを導入することによって，品質会計を中心とした全体的な品質保証の仕組みを構築することができる．これが，品質改善ドライバとしての価値である．事業環境の変化によって変わる組織の弱みを的確に把握して継続的に改善することは，長期的に安定した高品質ソフトウェア開発の基盤となる．

例えば，バグの作り込み件数の多い設計工程がある場合は，その工程で適用する設計技術に弱点があることを示している．さらに，具体的な作り込みバグの内容を分析することにより，設計技術の弱点部分を特定することができる．これを繰り返すことにより，設計技術を向上することができる．

また，出荷後バグ数を分析することにより，同様にソフトウェアプロセス上の弱点を特定することができる．この場合には，その弱点を解決する技術や仕組みを導入するとともに，その効果を開発途中に測定できるように，品質会計の適用や品質保証の仕組みを工夫する．これによって，出荷後まで待たなくても改善の効果を測定することができる．

第 7 章で紹介した A 組織は，20 年以上前に出荷後バグ数を 1/20 に低減後，そのレベルを維持している．その 20 年の間に起きた，メインフレームからオープンへのパラダイムシフト，OSS の出現，オフショア開発の台頭などのソフトウェア産業の大きな変化を考慮すると，そのような大変化のなかで出荷後バグ数の少なさを維持できた背景には，品質会計の品質改善ドライバとしての役割の価値が大きいと考える．例えば，メインフレームからオープンへのパラダイムシフトの時期には，コーディングしたプログラムをすぐに実行できる環境が低コストで手に入るようになった．ところが，その便利さが災いして，実装設計や単体テストを不用意に省略するケースが増加し，結果としてテスト最終段階での問題多発や出荷後バグの増加が発生した．これらは，品質会計の厳格な適用とそれによる全体的な品質保証の仕組みにより，大事に至る前にその兆候を把握し早期に対策を打つことができた．

このように，ソフトウェア品質会計は，品質改善ドライバとして価値がある．品質会計技法を，品質改善ドライバとして利用することにより，組織全体のソフトウェア品質保証の仕組みを整備することができる．品質会計の直接的な適用成果も重要であるものの，品質改善ドライバとしての価値は，長期的な視点でメリットが大きい．

8.6　現場主義の重視

品質会計の適用精度の向上には，ソフトウェア開発途中にきめ細かくバグ予測値を見直すことが欠かせない．ソフトウェア開発途中には，計画時には想定していなかったさまざまな変化が発生する．仕様変更はもちろん，予定した技術者の確保ができない，予定していたプログラムが流用できない，予定よりも開発規模が増加した，といったさまざまな問題が発生する．これらの問題は，リスク管理することはできても事前の確実な発生予測はできない．これらの変化を反映して，実態に沿ったバグ予測値の見直しをすることが，品質会計の成功の鍵である．

そのきめ細かなバグ予測値の見直しのためには，ソフトウェア開発の現場の把握が必須である．単なる数値上の変化だけを追っていては，現実の変化をバグ予測値へ反映できない．数値の変化や予定と実績の乖離には，必ず理由がある．その理由を現実を踏まえて理解するからこそ，実態に沿ったバグ予測値の見直しが実現する．

品質会計が現場主義を重視するのは，現場主義が品質会計そのものの適用精度に直接影響するためである．言い換えれば，現場主義を重視した姿勢がなければ，品質会計の適用は形式的になり，成功しない．品質会計の適用において，バグ予測値を実態に則して見直ししようとすれば，現場主義にならざるを得ないはずである．

現場主義を重視する姿勢は，いうまでもなく高品質ソフトウェア開発の実現に大きく影響する．すなわち，形式でなく実質として効果があがっているのか，効果があがった理由は何かなどを事実によって判断することに直結するからである．

ソフトウェア品質会計は，現場主義であることを要求する．品質会計の適用を成功させるためには，現場主義の姿勢が必須なのである．

8.7　おわりに

本章では，ソフトウェア品質会計技法の工学的価値について論述した．

ソフトウェア品質会計は，レビューによる早期品質確保，的確なテスト完了判断，バグ分析による課題解決能力の向上，品質改善ドライバとしての価値，現場主義の重視，という 5 つの価値を提供するものであり，いずれの項目も同様の価値を提供する技術はあまり見当たらない．重要な点は，ソフトウェア品質会計が，実際の開発現場で構築され，複数の組織で実績をあげている技術ということである．

一方，ソフトウェア品質会計を構成する個々の技法は，ソフトウェア開発の現場でよく使われているものが多い．バグ分析，バグ傾向分析，バグ収束判定などが好例である．ソフトウェア品質会計の価値は，個々の技法の適用結果を細かくフィードバックして現場で

使えるように仕上げていることと，それらを組み合わせて価値を創出している点である．上記の価値は，いずれもその結果として生み出されたものである．

本章で述べた内容は，20 年を超える年月をかけて，数千人を擁する大規模開発組織で実証済みである．何よりもそこに大きな価値があると言える．

第9章　高品質ソフトウェア開発の成功要因

9.1　はじめに

本章では，第 8 章までの議論に基づき，高品質ソフトウェア開発の成功要因を論述する．

高品質ソフトウェア開発は，組織レベルとプロジェクトレベルの 2 つのマネジメント局面から考える必要がある．1 回の高品質ソフトウェア開発を実現するだけなら，プロジェクトレベルのマネジメントを実施すればよいが，組織で実施するプロジェクトが高確率で高品質ソフトウェア開発を実現するためには，組織としての取り組みが必須である．

本章の提案を適用する組織の前提として，ソフトウェア開発に必要な一定レベルの開発スキルと規律を身に付けた技術者により構成された組織を想定する．ソフトウェア開発に限らず，開発に携わる技術者に対して一定レベルの開発スキルと規律の保有を要求するのは当り前のことである．ここでそれを明記するのは，現在のソフトウェア産業が労働集約産業になっており，ともすると単金の低さを重視とした発注先の選定が行われる傾向があるためである．

9.2　高品質ソフトウェア開発の成功要因と全体像

ソフトウェア開発組織では，組織内に常に複数のソフトウェア開発プロジェクトが進行している．各プロジェクトは，所定の QCD 目標を満足しながら独自の成果物を開発することに集中する．そのためのマネジメントがプロジェクトレベルのマネジメントである．一方，組織は，組織で実施するプロジェクトが高確率で高品質ソフトウェア開発を実現することに集中する．そのためのマネジメントが組織レベルのマネジメントである．

図 9-1 に高品質ソフトウェア開発の成功要因の全体像を示す. 高品質ソフトウェア開発を実現する成功要因は，8 つある．

組織レベルのマネジメントは，①外部指標による品質向上目標の設定，②品質向上目標のフォローに基づくプロセス改善，および③ソフトウェア開発方法論・ツール・開発環境を統合したシステムの提供という 3 つの成功要因から構成される．プロジェクトレベルのマネジメントは，ソフトウェア開発プロセスである設計・製造工程およびテスト工程に対して，①レビューによる早期品質確保，②的確なテスト完了判断を伴うテストの実施，③データに基づく短サイクルのマネジメント，④独立した品質保証部門によるプロセスおよびプロダクト両面からの品質確認，および⑤複数人による出荷判定という 5 つの成功要因から構成される．

各プロジェクトでは，プロジェクトレベルのマネジメントに示す成功要因を実行する．

組織レベルでは，組織レベルのマネジメントに示す成功要因を実行する．組織で実施するプロジェクトが高確率で高品質ソフトウェア開発を実現するには，実行の程度に差異はあっても，これら 8 つの成功要因はすべて必要である．

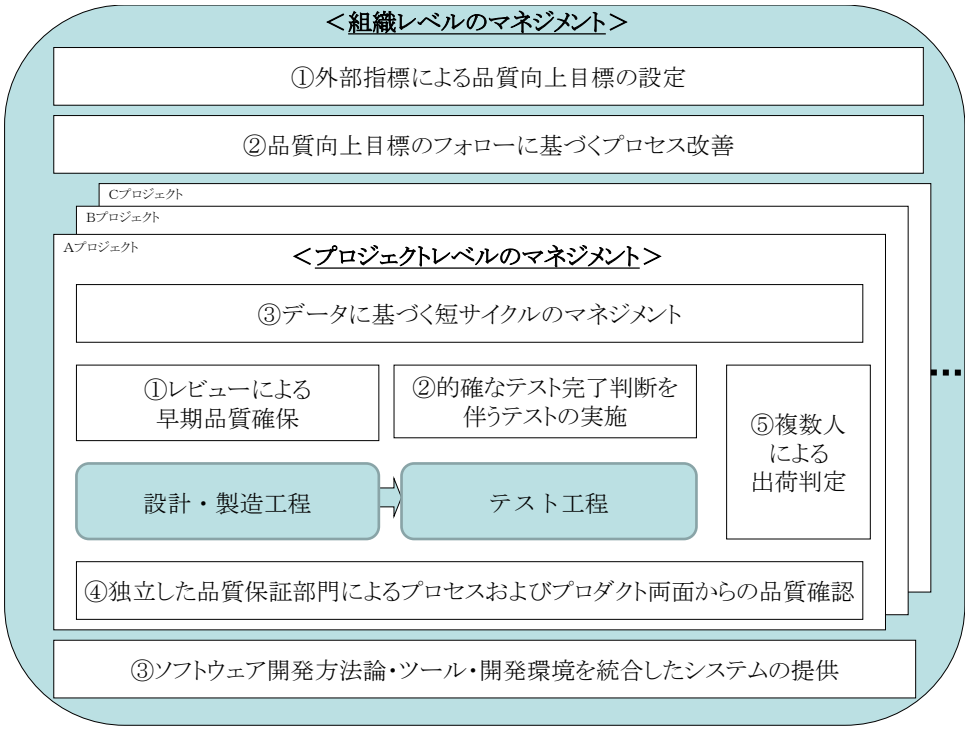


図 9-1　高品質ソフトウェア開発の成功要因の全体像

9.3　組織レベルのマネジメントにおける成功要因

組織レベルのマネジメントは，当該組織で実行するプロジェクトの成功確率を向上することが目的である（第 2 章の 2.4.3 節参照）．本節で説明する各項目は，この組織レベルのマネジメントの目的を念頭において実行する．

①外部指標による品質向上目標の設定

組織の品質向上目標設定は，品質向上活動の方向性を決めるために重要である．品質向上活動は顧客ニーズという外的基準へ適合する目的のために計画されるべきである．したがって，品質向上目標は，顧客の意見を反映した外部指標を採用するべきであり，その意味から売上高とクレーム数をセットで目標設定することが最適である．提供する製品やサービスが顧客から支持されれば売り上げは増加し，クレーム数は減少あるいは低い値を維持するはずだからである．また，指標の性格上，少なくとも数年に渡る目標設定になるもの

と考える．

品質の側面の評価になるものの，出荷後バグ数も品質向上目標の代表候補の 1 つである．重要なのは，開発組織から結果を制御できない外部指標を目標として設定することである．品質向上目標として不適切なのは，顧客の意見が反映されない指標の採用である．レビュー工数やテスト項目数の増加などの開発組織が制御可能な開発途中の指標は，品質向上のためのプロセス改善の目標には適するが，品質向上目標そのものには適さない．

また，品質向上目標は，組織トップの合意のもとで設定するべきである．その上で，組織トップが品質向上目標を達成するという強い意思を，組織の構成員に対して継続的に示すことが，高品質ソフトウェア開発の実現の第一歩である．

②品質向上目標のフォローによるプロセス改善

品質向上目標と実績値とのギャップを分析することにより，プロセスに機能的な欠落や不十分な項目が見つかった場合は，そのプロセスの構築や見直しを実施する．その際に，ソフトウェアプロセス改善のデファクトスタンダードである CMMI が参考となる．CMMI は，22 個のプロセス領域を定義し，各プロセス領域には詳細な特徴，ゴール，およびプラクティスが提示されている．CMMI を参考にして，欠落または不十分と考えられるプロセス領域の仕組みを実装することによって，必要なプロセスを構築できる．

問題点の根本原因を分析するためには，根本原因分析を適用する．ソフトウェア品質会計の「バグ分析と 1+n 施策」のように，根本原因を分析する場合には，根本原因の構造の整理を念頭において分析するべきである．この場合の分析の目的は，プロセス改善に結びつけることなので，プロセスの欠落や不十分さに関連する原因を分析する．根本原因の分析の的確性は，プロセス改善の効果に直結する．

プロセスを見直すだけでなく，構築したプロセスが有効に働いているかを確認することが重要である．その際には，①で設定した品質向上目標を達成できているかで判断する．解決すべき問題が解決できていることを数年に渡って監視し，解決の程度が不十分であれば再度プロセスを見直す．この繰り返しが，形式的でなく実効のあるプロセス構築につながる．

プロセス改善においては，日本的な「カイゼン」の姿勢が重要である．「カイゼン」は，単なる 1 回限りのプロセス見直しではなく，組織のトップからボトムまで全員が参加し，常に今より良い方法はないかを考えながら日々プロセスの見直しに取り組む活動である．この姿勢が高品質ソフトウェア開発を実現する基盤になる．

③ソフトウェア開発方法論・ツール・開発環境を統合したシステムの提供

ソフトウェア開発方法論・ツール・開発環境を統合したシステムとは，第 6 章で述べたソフトウェアファクトリを想定している．「②品質向上目標のフォローによるプロセス改善」の結果見直されたプロセスは，すべて統合化されたシステムへ反映して開発者へ提供する

というのが，この成功要因の意図である．

ソフトウェア開発の方法論，ツール，開発環境は，ソフトウェア開発の品質向上に大きく影響する．ソフトウェア開発に要求される 1 文字の間違いも許さない緻密さや正確さを，人間が完璧に実行することは困難である．細心の注意を払って作業をしたつもりでも，人間の作業では緻密さや正確さの完全性を保証することは難しい．それは，人間ではなく統合化したシステムで保証するべきである．一方，技術者の能力はむしろ，ソフトウェアのコンセプトを作り上げその正しさを実証することに発揮できるようにする．このシステムの狙いは，そこにある．

統合化したシステムの提供は，プロジェクト単独の活動でも可能である．しかし，継続的にシステムを改善し提供するには，組織的な取り組みでなければ困難である．あるプロジェクト単独で先進的なシステムを構築したものの，継続的なシステムの改善ができずに，後から組織的に構築したシステムよりいつのまにかだいぶ遅れたものになってしまったという実例を本著者は経験している．組織的に取り組むべき施策は，組織で取り組まなければ所定の成果をあげることはできない．

9.4　プロジェクトレベルのマネジメントにおける成功要因

プロジェクトレベルのマネジメントは，当該プロジェクトに求められる QCD を達成することが目的である．本節で説明する各項目は，このプロジェクトレベルのマネジメントの目的を念頭において実行する．

①レビュー重視による早期品質確保

設計・製造工程での品質確保を重視し，レビューによるバグ摘出を推進する．第 7 章で示した 3 事例はいずれも，レビューによる早期のバグ摘出により出荷後バグ数を低減した．C 組織の事例では，レビューを強化することによって，後戻り作業を減らし，生産性（Line/人・時間（H））向上と開発コストの低減を実現した．A 組織と B 組織では，上工程バグ摘出率が 80%に達するようになると，出荷後バグ数が一段と低減した．レビューによるバグ摘出は早期品質確保に効果を発揮するとともに，出荷後バグ数の低減に効果がある．特に，その比率が 80%を超えると，出荷後バグ数は大きく低減するものと考える．レビュー重視の姿勢は，日本企業の特徴の 1 つでもある[5-5].

レビューが不慣れな組織では，レビューを強化する決断は非常に難しい．その理由は，レビューの強化が設計・製造工程の工数増加を意味し，直接的に生産性（Line/人・時間（H））の低下と納期の遅延を想起させるためである．レビュー強化の決断を促すのがオフショア開発 C 組織の事例（第 7 章の 7.4 節参照）である．C 組織は，レビューを強化することによって後戻り作業を減少させ，結果として開発コストを低減することに成功した．レビュ

一強化の決断は，組織トップにしかできない．

②的確なテスト完了判断を伴うテストの実施

テストは，計画したテストを実施すれば終了するのではなく，必ず実施したテストの十分性を確認してテスト完了判断をするべきである．テスト完了判断では，多角的な視点からの十分性の確認が必須である．一面的な視点からでは判断ミスをする危険性がある．ソフトウェア品質会計では，テストで摘出したバグを分析することにより，系統的なテスト観点の抜け漏れ，細かいテスト観点の抜け漏れ，およびバグ収束の 3 点からテスト完了判断をしている．これにより，テストの抜け漏れを防止している．テスト完了判断のための視点には，品質会計の考え方以外にもさまざまなものが考えられる．重要なことは，多角的な視点からテスト完了判断をするべきという点である．それが的確なテスト完了判断につながる．多角的なテスト完了判断基準を保有したうえで，テスト完了判断を実施することが重要である．

③データに基づく短サイクルのマネジメントチェック

少なくとも週次単位の短サイクルでデータに基づくマネジメントを実施することによって，問題発生を早期に把握できる．さらに，フェイスツーフェイスのマネジメントの場合は，コミュニケーションが改善されて，形式的ではなく実質的に効果のある行動がとりやすくなる．CMMI レベル 5 組織の調査結果（第 2 章の 2.6 節参照）では，実質的な成果をあげることの難しさを指摘した．マネジメントチェックにおいて，単なる数値上の変化や形式的な報告にとらわれずに，現場で起こっている事実を的確に把握できるかが，実質的な成果につながる．

B 組織の事例（第 7 章の 7.3 節参照）において，プロジェクトマネジメント会議運営方法の見直しは品質向上に重要な役割を果たした．問題解決後に報告を受けるスタイルのマネジメント方法では，問題の対処結果の追認にしかならない．問題発生時にそれをより早く把握し，関係者全員が合意しながら問題の真の原因を的確に究明して対策を実施し，現場の状況をデータで確認しながら終結することによって，関係者が各々保有する経験やノウハウを結集して利用できるようになる．これが，個人の経験やノウハウを組織的に共有し活用することにつながる．

④独立した品質保証部門によるプロセスとプロダクト両面からの品質確認

幾つかの日本企業が同じ実装方法に至った代表例の 1 つに，開発部門と独立した品質保証部門の存在がある．品質保証部門の果たす機能は企業によって多少の差異はあるものの，客観的な立場で開発途中にデータ分析を行うことによってプロセス遂行状況を把握するとともに，各工程での成果物の出来を確認し，最終成果物は実際にテストして評価するという方法は共通である．このような品質保証部門のあり方は，グローバルレベルでは他に例

を見ない日本企業特有のものである．固定的な組織として品質保証部門を設置することにより，品質保証のノウハウや事例が一箇所に集まり，組織共通のプロセスへフィードバックできるようになるというメリットがある．独立した品質保証部門の設置そのものは，組織レベルの判断が必要である．

開発部門が優秀であっても，開発途中に常に客観的な視点をもち続けることは難しい．どうしても自分に都合のよい開発者視点の考え方になってしまう．顧客視点の判断を実現するには，開発部門の影響を受けずに，独立した立場で品質を判断できる機能が必要である．さらに，それは必ずプロセス品質とプロダクト品質の両面から判断することが重要である．プロセス品質は，各工程で実行すべきことを確実に実行することによって確保され，プロダクト品質は出来上がった成果物が所定の要件を満足していることによって確保される．プロセス品質とプロダクト品質は，互いに補完関係にある．一方だけで正しく品質を把握することはできない．B 組織は，欠落していたプロダクト面からの品質保証を追加することにより，改善施策全体を軌道にのせた．プロセス面とプロダクト面の両面からの品質確認は，品質マネジメントの基本原則でもある．

⑤複数人による出荷判定

複数人による出荷判定も，複数の日本企業が同じ実装方法に至った仕組みである．開発プロジェクトの責任者単独での出荷判定では，プロジェクトに責任をもっているだけに開発者に都合のいい判断になりがちである．出荷判定者としてプロジェクト外の指示系統の異なる判定者が出荷判定に関与することによって，顧客視点での出荷判定が可能になる．都合の悪い事柄も判断の材料としてあがり，公正な出荷判定が行われるようになる．指示系統の異なる判定者の最適例が，品質保証部門の責任者である．

複数人による出荷判定では，判断が分かれた場合の判定方法を予め決めておく必要がある．プロジェクトの責任者に最終判定権を持たせるのは不適切である．なぜなら，複数人による出荷判定の意図は，開発者に都合のいい判断ではなく，あくまで顧客視点での出荷判定ができるようにするためだからである．最適な最終判定者は，品質保証部長である．これが，日本企業の共通した実装方式である．品質保証部長に最終判定権を持たせることにより，品質保証部門に顧客視点の品質保証に対する責任と権限を与えるのである．これにより，品質保証部門は高い専門性を求められるようになり，「④独立した品質保証部門によるプロセスとプロダクト両面からの品質確認」の活動へ好影響を及ぼす．品質保証部門の専門家としての意識が高まり，保有する品質技術が向上し，的確な品質保証が実行されるようになるという好循環を生む．

さらに，出荷判定基準には，「②的確なテスト完了判断を伴うテストの実施」結果だけでなく，「④独立した品質保証部門によるプロセスとプロダクト両面からの品質確認」結果を含めるべきである．出荷判定は，顧客へ問題ソフトウェアを出荷しないようにする最終関門である．出荷判定基準は，過去の出荷判定失敗事例を確実にフィードバックして何重に

も多角的な判断要素を持たせることにより，判定精度を向上することが重要である．

9.5　人間的要素の改善について

CMMI レベル 5 組織の調査（第 2 章の 2.6 節参照）で明らかにしたように，品質を重視するメンバの意識や組織文化は，品質向上に不可欠である．人間的要素の改善を解決可能な決定的技術はなく，具体的な施策を実行する中で品質を重視するメンバの意識や組織文化を育むしかない．人間的要素の改善に対して重要なのは，組織トップの品質向上への強い意志である．組織トップが品質向上の重要性を継続的に示すことは，組織の品質を重視する文化の醸成への第一歩となる．複数人による公正な出荷判定は，品質を重視する組織文化の醸成に良い影響を与える．顧客視点の評価基準による判断を確実にすることにより，組織内に，常に変わらない判断基準が育まれる．組織の構成員は，各自が顧客視点で考え，判断するようになる．逆に，常に変わらぬ判断が保証されない組織では，立場や状況によって判断基準が変わり，トップがそのときどきの自分の考えで個々の事項を判断するため，組織の構成員はトップの決定を待ち，従うしかない．常に変わらぬ評価基準で判断しているかどうかは，組織構成員の自主性尊重へ決定的な影響を与える．

8 つの成功要因を実行するなかで，組織の技術者に行動の変化が見られるようになるはずである．B 組織の改善（第 7 章の 7.3 節参照）においても，2 年程度のうちに品質の重要性に対する発言が増加するなど技術者の行動の変化が見られた．C 組織（第 7 章の 7.4 節参照）も同様である．人間的要素を直接改善できる技術は存在しない．8 つの成功要因を継続的に実行することが，最も確実に人間的要素を改善できる方法である．

9.6　考察

はじめに，高品質ソフトウェア開発の成功要因の全体像と，ソフトウェア品質会計との関係を述べる．

高品質ソフトウェア開発の成功要因の全体像のうち，プロジェクトレベルのマネジメントの成功要因である「①レビューによる早期品質確保」と「②的確なテスト完了判断を伴うテストの実施」は，ソフトウェア品質会計により実現可能である．③～⑤は，ソフトウェア品質会計の効果を引き出すために考案した仕組みであり，品質会計と親和性が高い．したがって，プロジェクトレベルのマネジメントの成功要因は，ソフトウェア品質会計を適用することによって，実現可能である．もちろん，他の技法の適用の可能性を否定するものではなく，ソフトウェアの特性によって適切な技法を検討するべきである．

組織レベルのマネジメントの成功要因である「①外部指標による品質向上目標の設定」

および「②品質向上目標のフォローに基づくプロセス改善」は，ISO9001 などに基づく品質マネジメントシステムを運用していれば当り前に実現することである．重要な点は，ソフトウェアにおける品質向上目標に対して顧客の意見を反映した外部指標を採用することである．また，「③ソフトウェア開発方法論・ツール・開発環境を統合したシステムの提供」は，ソフトウェア開発組織として積極的に取り組むべき項目である．③のシステムは，ソフトウェアの副次的な難しさを徹底的に軽減するためのものである．現場のノウハウや知見の集積であり，効果のあるシステムを構築するには一定の時間をかけた試行錯誤が必要と考える．

次に，プロジェクトレベルのマネジメントと組織レベルのマネジメントの関係について，考察する．プロジェクトを成功させるには，一定のプロジェクト条件を整備すればプロジェクトレベルのマネジメントを実行することで成功可能である．しかし，ソフトウェア開発組織で実施されるプロジェクトの成功確率を継続的に高めるためには，組織レベルのマネジメントの実行が必須である．特に，「②品質向上目標のフォローに基づくプロセス改善」に求められる「カイゼン」の姿勢が重要である．「カイゼン」とは，単なる 1 回限りのプロセス見直しではなく，組織のトップからボトムまで全員が参加し，常に今より良い方法はないかを考えながら日々プロセスの見直しに取り組む活動である．「カイゼン」の考え方が基盤にあるからこそ，常にプロセス改善がまわるようになり，「③ソフトウェア開発方法論・ツール・開発環境を統合したシステムの提供」のシステムが，常に見直されて最新の使いやすいものとなる．それがプロジェクトに好影響を与え，成功確率を高めるのである．

最後に，本章で提示する高品質ソフトウェアの成功要因の全体像と，CMMI との関係を考察し，本章の提案の意味を考える．

CMMI は，22 個のプロセス領域を定義し，各プロセス領域のすべてのプラクティスをひとまとまりとして実装するとそのプロセス領域のゴールを達成すると考える．その実装方法は，利用者に任される．実装方法が利用者に任されていて自由度が高いものの，レベル 5 であっても必ずしも出荷後バグ数の高水準を保証できないことは，2.6 節で示した通りである．B 組織の改善前の出荷後バグ数でも実証している．ソフトウェアプロセスの機能的欠落や不十分性が，出荷後バグ数の増加を招くことは間違いないが，22 個のプロセス領域すべてが必須というわけではない．品質会計を考案した A 組織は，CMMI が考案されるより前に出荷後バグ数を 1/20 に低減することに成功しており，その時点で 22 個のプロセス領域すべてを網羅的に構築していたわけではないからである．

これらのことから，高品質ソフトウェア開発のためには，必ずしも CMMI が提案する 22 個のプロセス領域すべてを網羅して構築する必要はないと考える．むしろ，顧客の意見を反映した品質向上目標を設定し，目標と実績のギャップから，その組織にとって必要なプロセス上の欠落や不十分性を把握して解決する方法のほうが，効率よく必要なプロセスを構築可能である．それを実現するのが，組織視点のマネジメントで提案した①～③の成功要因である．「①外部指標による品質向上目標の設定」により顧客の意見を取り込み，「②

品質向上目標のフォローに基づくプロセス改善」で顧客志向を反映できるようプロセスを改善する．その結果は，「③ソフトウェア開発方法論・ツール・開発環境を統合したシステムの提供」に反映されてシステムとして技術者へ提供する．

さらに，NEC を含む幾つかの日本企業が，1970 年代からソフトウェアの品質向上に取り組んだ結果，結果的に同じ実装方法にたどりついた点を重視し，その共通するプラクティスは積極的に取り込むべきと考える．それらのプラクティスは，複数の企業でその効果を確認済みである．プロジェクトレベルのマネジメントに含まれる①～⑤がその共通するプラクティスである．各プラクティスの目的を理解して実行すれば，必ず効果が出るはずである．

9.7　おわりに

本章では，高品質ソフトウェア開発の成功要因とその全体像を論述した．

本著者の NEC における 20 年以上にわたる経験と研究に加えて，幾つかの日本企業によって実証されたプラクティスを重ね合わせることによって，高品質ソフトウェア開発の成功要因を導いた．顧客視点による評価を品質向上目標として取り込むとともに，既に複数の企業によって実証されたプラクティスを組み合わせて構築している点が特長である．

8 つの成功要因を並べて考察すると，そこに一貫するのは顧客志向だということに気がつく．ソフトウェア開発中において，いかに顧客視点を持ちながらソフトウェア開発を成功させるかを考えていくことが，高品質ソフトウェア開発につながると考える．

第10章 結論

序論において，第 2 のソフトウェア危機の問題を提起した．本論文は，その第 2 のソフトウェア危機を乗り越える 1 つの方策を提案するものである．

本論文を貫くのは，実証された技術を重視するという姿勢である．この論文で提示した技術は，すべて複数の組織でその実効性を確認した技術である．実証された技術を組み合わせることにより，高品質ソフトウェア開発の実現は可能である．以下に，本著者の主張を整理する．

第 2 章は,「品質」および「バグ」という用語の意味するところを定義した．本論文では，品質を「本来備わっている特性の集まりが要求事項を満たす程度」と定義し，要求事項が受け取り手である顧客の基準によって決まることを示した．バグは通常想定するバグの対象範囲よりやや広く，本来あるべき特性を含めた範囲を対象範囲と定義した．すなわち，本論文におけるバグとは，狭い意味での信頼性という意味だけではなく，顧客が求める本来あるべき特性の全体を意味する．そのうえで，品質のマネジメントはプロセス面とプロダクト面の両面から品質保証することが重要であるとともに，品質のマネジメントには組織視点とプロジェクト視点の 2 つの局面があることを示した. ソフトウェアの特性を述べ，ソフトウェアの難しさには，コンセプトを作り上げその正しさを実証する本質的な難しさと，ソフトウェアに求められる緻密さや正確さに関わるソフトウェアの実装過程を中心とした副次的な難しさがあることを論じた．副次的な難しさは解決可能であり，副次的な難しさを徹底的に解決することにより，本質的な難しさも幾分緩和される可能性があることを論じた. そのうえで, ソフトウェアプロセス改善のデファクトスタンダードである CMMI のレベル 5 組織への調査結果に基づき，品質向上に影響を及ぼす要因と CMMI による効果について議論した． CMMI レベル 5 組織の調査結果を用いたのは，プロセスに機能的な欠陥がないという条件のもとで品質向上に影響を及ぼす要因を分析するためである．その結果, CMMI では, 出荷後バグ数の少なさを強みにできるほどの品質を実現するのは難しく，高品質ソフトウェア開発の実現のためには，実質的な効果をあげるプロセスの構築と人間的要素の改善が必要であることを示した．

第 3 章は，NEC のソフトウェア開発の現場で考案し，20 年以上かけて構築・適用してきたソフトウェア品質会計について，その概要と適用方法を説明した．ソフトウェア品質会計の大きな特徴は，レビューでのバグ摘出による早期品質確保，および的確なテスト完了判断という 2 点を実現している点にある．この 2 つの特徴が，高品質ソフトウェア開発の実現に大きく寄与する．第 4 章は，ソフトウェア品質会計技法のうち,「バグ分析と 1+n 施策」技法について，いわゆる「なぜなぜ分析」との差異を明確にしながら，その技法の詳

細を説明した．バグ分析は，ソフトウェア開発の現場でしばしば使用される技法であるものの，効果をあげにくい技法でもある．その理由が，根本原因の構造を意識した分析をしていないためであり，その解決は汎用的ななぜなぜ分析では難しいことを論述した．そのうえで，バグ分析と 1+n 施策技法が，根本原因の構造を意識して根本原因を分析する技法であり，同種バグ摘出を目的として最適化した技法であることを示した．

第 5 章では，ソフトウェア品質会計を支える技術として，レビュー技術および品質会計を取り囲む品質確保のための仕組みを論じた．品質確保のための仕組みとは，データに基づく短サイクルのマネジメント，独立した品質保証部門によるプロセスとプロダクトの両面からの品質確認，および複数人による出荷判定の 3 つである．この 3 つの仕組みは，他の幾つかの日本企業が 1970 年代からソフトウェア品質向上への取り組みによって実装した仕組みと，結果として共通であった．1 つの企業だけでなく，複数の企業により実証された仕組みである点が重要である．

第 6 章では，ソフトウェア開発方法論・ツール・開発環境を統合したシステムである「ソフトウェアファクトリ」を紹介した．ソフトウェアファクトリは，ソフトウェア開発に要求される開発作業や成果物の緻密さや正確さといった実装過程を中心とした難しさを解決する重要な施策である．ソフトウェアファクトリという開発基盤が提供されるからこそ，技術者は，1 文字の間違いも許さない正確さや緻密さの要求というソフトウェアの特性から開放され，ソフトウェアの本質的な難しさであるコンセプトの考案とその正しさの検証に注力することができる．

第 7 章では，ソフトウェア品質会計の適用による A 組織，B 組織および C 組織の品質向上の実例を論述した．A 組織は，ソフトウェア品質会計を考案した組織であり，品質会計技法を構築しながら，出荷後バグ数を 1/20 へ低減し，以来 20 年以上に渡ってそのレベルを維持している．B 組織は，CMMI レベル 5 を達成しているものの出荷後バグ数の多さに悩む組織である．ソフトウェア品質会計技法を厳格に適用するとともに，品質会計を取り囲む仕組みを整備することにより，出荷後バグ数の低減に成功した．B 組織は，5 年間で A 組織と同レベルの出荷後バグ数の水準になることを目標とし，4 年経過した現在，順調に成果をあげており，残り 1 年で A 組織の出荷後バグ数の水準を達成できる見込みである．C 組織の事例は，中国オフショア開発組織の品質向上事例である．オフショア開発組織向けに考案したステップアップモデルに基づき，品質会計を厳格に適用するとともに，品質会計を取り囲む仕組みを整備した．これにより，C 組織から発注元組織への納品時の開発物件の品質の向上，生産性(Line/人・時間（H)）の向上，日本語力の向上，および開発コストの削減を実現した．レビューを強化することによって，後戻り作業を削減して全体の開発コストが低減することを実証したことには大きな意味がある．本章では，ソフトウェア品質会計の適用と品質会計を取り囲む仕組みを整備することにより，出荷後バグ数を低減し維持することが可能であることを実証するとともに，この方法が海外オフショア開発組織においても有効であることを示した．特に強調したいのは，ソフトウェア品質会計を適用す

ると同時に，品質会計を取り囲む仕組みの整備が必要であるという点である．

第 8 章では，第 7 章までを踏まえて，ソフトウェア品質会計技法の工学的価値を論述した．品質会計の工学的価値は，レビューによる早期品質確保，的確なテスト完了判断，バグ分析によるプロセス改善の的確性の向上，品質改善ドライバとしての価値，現場主義の重視，という 5 点にある．このいずれの項目も同等の価値を提供する技術はあまり見当らない点が，品質会計技法の価値を高めるものである．

第 9 章では，第 8 章までに基づき，高品質ソフトウェア開発の成功要因の全体像を論述した．高品質ソフトウェア開発の成功要因とは，①外部指標による品質向上目標の設定，②品質向上目標のフォローに基づくプロセス改善，および③ソフトウェア開発方法論・ツール・開発環境を統合したシステムの提供，という組織レベルの 3 つのマネジメント要因と，①レビューによる早期品質確保，②的確なテスト完了判断を伴うテストの実施，③データに基づく短サイクルのマネジメント，④独立した品質保証部門によるプロセスおよびプロダクト両面からの品質確認，および⑤複数人による出荷判定，という 5 つのプロジェクトレベルのマネジメント要因である．プロジェクトレベルのマネジメント要因のうち①と②はソフトウェア品質会計により実現可能である．残りの③～⑤は，品質会計を取り囲む仕組みに該当する．プロジェクトレベルの 5 つのマネジメント要因を実施することにより，実施対象プロジェクトを成功可能である．組織レベルの 3 つのマネジメント要因は，当該組織で実施するプロジェクトの成功確率を高める．また，実効あるプロセスの構築や人間的要素の改善は，この 8 つの成功要因を実装していく過程において，常に変わらない顧客視点の公正な判断を繰り返していくことで実現できる．

次に，今後の課題と展望を述べる．今後の課題としては 2 点ある．

課題の 1 番目は，高品質ソフトウェア開発の成功要因の全体像を実用可能とするようモデル化することである．本論文では，高品質ソフトウェア開発の成功要因とその全体像を示した．これを実際のソフトウェア開発の現場で適用可能にするには，各成功要因を構成する要素に分解し，優先順位を付けることによって，現場での適用順序を明らかにするとともに，各要素の詳細内容を説明したモデル化が必要である．例えば，プロジェクトレベルのマネジメント要因の③データに基づく短サイクルのマネジメントを実現するには，収集すべきデータ項目の明確化，データ分析の方法，マネジメントの場面で議論すべき項目の具体化などの整理や適用の順序化が必要である．成功要因間の関係も同様である．各成功要因のゴールの明確化およびプロセスの実効レベルの判定を可能とする基準の明確化も必要である．それにより，どのソフトウェア開発組織においても本モデルが適用可能となる．本モデルを実際に試行し，その試行結果によるフィードバックも必要である．これにより，高品質ソフトウェア開発を望む企業が，高品質ソフトウェア開発を実現するまでの道のりが効率化されるはずである．

課題の 2 番目は，実用可能となった高品質ソフトウェア開発の成功要因の全体像のモデ

ルを実際のソフトウェア開発の現場に適用し，その結果を分析することである．開発されるソフトウェアの品質や生産性が，開発者の技術レベルやソフトウェアプロセスの卓越性に大きく影響を受けることは古くから指摘されている．これを実際の開発データで実証し，高品質ソフトウェア開発の成功要因の全体像のモデルの正当性を裏付けるとともに，広く社会へ啓蒙するための材料とする．第 2 のソフトウェア危機を乗り越えるためには，ソフトウェアの特性およびその難しさと，高品質ソフトウェア開発の条件を社会が認知する必要がある．それなしに，ソフトウェア産業の労働集約産業から知的集約産業への転換は実現しない．

本論文は，ソフトウェアという，人類が今まで遭遇したことのない特性に対して，現場での長年の取り組みを基盤として構築した技術と知見を組み合わせ，実証することによって，そのソフトウェアの特性に立ち向かうだけの道筋を示した．本論文が，日本のソフトウェア産業の発展に貢献できれば幸いである．

参考文献

[1-1] 玉井哲雄：「ソフトウェア社会のゆくえ」，岩波書店，東京，2012.

[1-2] CMU Software Engineering Institute: *CMMI for Development, Version 1.3(CMMI-DEV, V1.3)*, CMU/SEI-2010-TR-033, 2010.

[2-1] ISO9000：*2005, Quality Management Systems –Fundamentals and Vocabulary* (JIS Q 9000:2006, 品質マネジメントシステム-基本及び用語)．

[2-2] R. S. Pressman（著)，西康晴，榊原彰（監訳)：「実践ソフトウェアエンジニアリング -ソフトウェアプロフェッショナルのための基本知識」, 日科技連出版社, 東京, 2005.

[2-3] G. M. Weinberg（著)，大野徇郎（監訳)：「ワインバーグのシステム思考法　－ソフトウェア文化を創るⅠ」，共立出版，東京，1994.

[2-4] J. Martin（著)，芹沢真佐子ほか（訳)：「ラピッド・アプリケーション・デベロプメント 1」，リックテレコム，東京，1994.

[2-5] 石川馨：「日本的品質管理　～TQC とは何か」，日科技連出版社，東京，1981.

[2-6] 狩野紀昭，瀬楽信彦，高橋文夫，辻新一：“魅力的品質と当り前品質”，日本品質管理学会誌（「品質」)，Vol.14,No.2, pp.39-48,1984.

[2-7] 飯塚悦功：「現代品質管理総論」，朝倉書店，東京，2009.

[2-8] ISO/IEC 9126-1:*2001, Information Technology – Software Product Quality – Part 1：Quality Model*（JIS X 0129-1：2003 ソフトウェア製品の品質　-第一部：品質モデル)．

[2-9] JIS X 0014:1999 情報処理用語　–信頼性，保守性及び可用性.

[2-10] M. V. Mäntylä and C. Lassenius: “What Types of Defects Are Really Discovered in Code Reviews?”, *IEEE Transactions on Software Engineering,* Vol.35, No.3, pp.430-448,2009.

[2-11] W. S. Humphrey（著)，ソフトウェア品質経営研究会（訳)：「パーソナルソフトウェアプロセス技法」，共立出版，東京，1999.

[2-12] SQuBOK 策定部会：「ソフトウェア品質技術体系 -SQuBOK ガイド-」，オーム社，東京，2006.

[2-13] Project Management Institute（著)，PMI 東京支部（監訳)：「プロジェクトマネジメント知識体系ガイド（PMBOK ガイド）第 3 版」，Project Management Institute, 東京，2004.

[2-14] F. P. Brooks Jr.（著)，滝沢徹，牧野祐子，冨澤昇（訳)：「人月の神話　–狼人間を撃つ銀の弾はない-（原著発行 20 周年記念増訂版)」，アジソン・ウェスレイ・パブリッシャーズ・ジャパン，東京，1996.

[2-15] 誉田直美，西康晴，片山豊，八木広行，高橋宗雄，中将貴幸：“ソフトウェアプロジェクト管理における技術者のモチベーションに関する研究”，プロジェクトマネジメント学会誌，Vol.3，No.3，pp. 33-39，2001.
[2-16] M. Cusumano et al., “Software Development Worldwide: The State of the Practice”, *IEEE Software*, Vol.20, No.6, pp.34-38, 2003.

[3-1] 菅野文友，吉澤正（監修），日科技連ソフトウェア品質管理研究会（編）：「21 世紀へのソフトウェア品質保証技術　-日科技連ソフトウェア品質管理研究会 10 年の成果-」，日科技連出版社，東京，1994.
[3-2] 久保宏志（監修）：「富士通におけるソフトウェア品質保証の実際」，日科技連出版社，東京，1989.
[3-3] 保田勝通：「ソフトウェア品質保証の考え方と実際」，日科技連出版社，東京，1995.
[3-4] 岡崎毅久（編著）：「ソフトウェアテストと品質保証の実際」，日本テクノセンター，東京，1999.
[3-5] M. A. Cusumano, *Japan's Software Factories*, Oxford University Press, New York, 1991.
[3-6] 水野幸男（監修）：「ソフトウェアの総合的品質管理」，日科技連出版社，東京，1990.
[3-7] 誉田直美，真野俊樹：「見積りの方法」，日科技連出版社，東京，1993.
[3-8] N. Honda, R. Kurashita, and N. Tsuboi: “A Software Development Management System with a Template”, *Proceedings of the India - Japan Conference on Quality in Software Systems Engineering,* pp.2-14, 1997.
[3-9] 誉田直美：「ソフトウェア品質会計　－NEC の高品質ソフトウェア開発を支える品質保証技術―」，日科技連出版社，東京，2010.
[3-10] B. W. Boehm and R. Turner (著)，河津正幸，原幹（監訳）：「アジャイルと規律」，日経ＢＰ社，東京，2004.
[3-11] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[3-12] 山田茂：「ソフトウェア信頼性の基礎　－モデリングアプローチー」，共立出版，東京，2010.

[4-1] 大野耐一：「トヨタ生産方式」，ダイヤモンド社，東京，2003.
[4-2] 飯塚悦功，金子龍三：「原因分析-構造モデルベース分析術-」，日科技連出版社，東京，2012.

[5-1] T. Gilb and D. Graham（著），伊土誠一，富野壽（監訳）：「ソフトウェア　インスペクション」，共立出版，東京，1999.

[5-2] K. E. Wiegers（著），大久保雅一（監訳）:「ピアレビュー　高品質ソフトウェア開発のために」，日経ＢＰソフトプレス，東京，2004.
[5-3] 大西健児ほか（著）:「ＪＳＴＱＢ認定テスト技術者」，翔泳社，東京，2007.
[5-4] 野中誠 : “ソフトウェアインスペクションの効果と効率”，情報処理，Vol.50， No.5，pp.385-390，2009.
[5-5] SPC 国際検討タスクフォース : “日本のソフトウェア品質管理の特徴とその背景”，第 22 回ソフトウェア生産における品質管理シンポジウム報文集，pp.427-450, 2003.

[7-1] S. Sasabe and R. Kaneko: “Integrated Software Quality Management Based on Multiple Process Improvement Models and Organization’s Own TQM Method – A new Process Network Oriented Method”, *Proceedings of the 3rd World Congress for Software Quality,* Vol.1, pp. 279-289, 2005.
[7-2] オフショア開発研究会 :「ソフトウェア開発　オフショアリング完全ガイド」，日経 BP 社，東京，2004.
[7-3] 誉田直美，野田正道，倉田克徳，飯泉紀子，大西健児，桜木霞，堀田文明 : “オフショア開発におけるプロジェクトマネジメントの現状調査と分析”，プロジェクトマネジメント学会 2004 年度春季研究発表大会予稿集,pp. 377-382，2004.
[7-4] 誉田直美，野田正道，倉田克徳，飯泉紀子，大西健児，桜木霞，堀田文明 : “オフショア開発におけるプロジェクトマネジメント改善事例と考察”，プロジェクトマネジメント学会 2004 年度春季研究発表大会予稿集,pp. 371-376，2004.
[7-5] 誉田直美，松本正雄 : “ソフトウェア開発の品質/生産性向上モデル”，電子情報通信学会技術研究報告，Vol.105，No.255，pp. 7-12，2005.

謝辞

本研究を遂行するのにあたり，多大なご指導・ご鞭撻を頂きました鳥取大学大学院工学研究科社会経営工学講座山田茂教授，得能貢一教授，小柳淳二准教授ならびに山口大学大学院理工学研究科田村慶信准教授に深く感謝いたします．特に主指導教員の山田教授には，鳥取大学大学院工学研究科博士後期課程に在学中，懇切なご指導と格別なご配慮，暖かい励ましを賜り，心から感謝しております．

また，本研究を進めるにあたり，さまざまな面でご支援を頂きました，井上真二助教ならびに山田研究室の方々に深く感謝申し上げます．

本研究は，日本電気株式会社入社以来，基本ソフトウェア開発本部から IT ソフトウェア事業本部に至る品質保証部門における業務が基礎になっています．ソフトウェア開発の現場で生まれた品質会計技法の構築と適用を，本著者が中心的な立場で推進する幸運に恵まれたことに深く感謝するとともに，その活動をさまざまな形で支援してくださった日本電気株式会社山元正人執行役員常務に厚く御礼申し上げます．

本著者とともに品質会計の構築と適用に取り組んだ仲間でもある佐藤孝司氏，纐纈信子氏，森岳士氏ならびに倉下亮氏に深く感謝の意を表します．職場の先輩，同僚，関係者各位にも改めて感謝申し上げます．

業務に加えて，日本科学技術連盟の SQiP ソフトウェア品質委員会での活動を通じて，ソフトウェア品質に関する深い洞察を得ることができました．SQiP 前委員長の元東京大学教授飯塚悦功先生，現委員長の東洋大学野中誠准教授をはじめ委員の皆様方に深く御礼申し上げます．本著者が山田茂先生にご指導をお願いするきっかけを作ってくださった SPC 委員会（SQiP ソフトウェア品質委員会の前身）元委員長の菅野文友先生には特別の御礼を申し上げます．

最後に，本研究を進めるにあたり，日頃から心の支えとなり励ましてくれた，夫井出村重夫，息子重仁，母誉田公子，義母井出村綴，天から見守ってくれた父誉田邦夫ならびに義父井出村英夫に感謝します．

研究業績一覧

主論文

1. N. Honda：“Beyond CMMI Level 5 - Comparative Analysis of Two CMMI Level 5 Organizations –”, *Software Quality Professional,* Vol. 11, No.4, pp. 4-12, September 2009.
2. N. Honda and S. Yamada：“Characteristic Analysis of Software Development Organizations Having Great Success”, *Proceedings of the 17th International Conference on Reliability and Quality in Design,* pp.379-383, August 2011.
3. 倉下　亮，吉村　博昭，野中　誠，誉田　直美：“ＣＫメトリクスの分布に基づくソフトウェア設計の質の定量評価”，第 31 回 SQiP ソフトウェア品質シンポジウム 2011 発表報文集 (SQiP2011)，CD-R，8 pp.，2011 年　9 月.
4. N. Koketsu, N. Honda, S. Kawamura, J. Nomura, and M. Nonaka：“Improvement of the Fault-prone Class Prediction by the Process Metrics Use”, *Proceedings of the 5th World Congress for Software Quality (5WCSQ),* CD-R, 9 pp., November 2011.
5. T. Mori, R. Kurashita, and N. Honda：“Proposal of “Ask Why” Framework to Analyze Defect Root Causes”, *Proceedings of the 5th World Congress for Software Quality (5WCSQ),* CD-R, 8 pp., November 2011.
6. N. Honda and S. Yamada：“Empirical Analysis for High Quality Software Development”, *American Journal of Operations Research,* Vol. 2, No.1, pp. 34-42, March 2012.
7. N. Honda and S. Yamada：“Defect Root-Cause Analysis and 1+n Procedure Technique to Improve Software Quality”, *International Journal of Systems Assurance Engineering and Management,* Vol. 3, No.2, pp.111-121, August 2012.
8. N. Honda and S. Yamada：“Software Quality and Productivity Improvement Activities at NEC”, *Proceedings of the 11th International Conference of Industrial Management (ICIM2012),* pp. 34-42, August 2012.
9. N. Honda and S. Yamada：“Success Factors to Achieve Excellent Quality - CMMI Level 5 Organizations Research Report –”, *Software Quality Professional*, Vol.14, No.4, pp.21-32, September 2012.
10. 誉田直美, 山田茂:“ソフトウェア品質会計による高品質ソフトウェア開発の実現”, 日本ソフトウェア科学会誌（「コンピュータソフトウェア」）（採録決定済），Vol. 30，2013.

参考論文（研究報告，発表，解説）

1. 誉田直美：“ソフトウェア開発におけるデザインレビュー”，クオリティマネジメント，Vol.60, No.8, pp. 34-41，2009 年 8 月.
2. 誉田直美，山田茂：“レビューでの品質向上を実現する品質会計技法　－ソフトウェア品質会計に関する研究（1）－”，プロジェクトマネジメント学会 2011 年度春季研究発表大会予稿集， pp. 213-218，2011.
3. 誉田直美，山田茂：“同種バグの摘出を可能とするバグ分析と 1+n 施策　－ソフトウェア品質会計に関する研究（2）－”，プロジェクトマネジメント学会 2011 年度秋季研究発表大会予稿集, pp. 157-162，2011.
4. 誉田直美，山田茂：“高品質ソフトウェア開発を実現する品質会計技法”，プロジェクトマネジメント学会誌, Vol.13，No. 5，pp. 9-14，2011 年 10 月.
5. N. Honda: “Success Factors to Achieve Excellent Quality - CMMI Level 5 Organizations Research Report -“, *Proceedings of the 4th Japan-Korea Software Management Symposium –Recent Development and Future Trend* -, pp. 11-23, November 2011.
6. 誉田直美：“CMMI レベル 5 を超える品質の実現”，日本品質管理学会誌（「品質」），Vol.42, No.4, pp.54-62, 2012 年 10 月.