要求定義で困ってませんか？
要求仕様書の品質に関する研究成果報告

要求工学ワーキンググループ

2007年1月24日

目 次

第1章　はじめに

システムの利用者・顧客と開発者が別人であることが多い以上，システムに対する要求を，引き出し，分析し，まとめる技術や技法が必要となる．ソフトウェアが中心となるシステムに関しての，このような技術の集大成が要求工学であると我々は考えている．要求工学に関わる成果物やその作成過程の質を高めれば，最終的に顧客に提供される製品やサービスの質の向上にも貢献する．また，開発プロセスの効率や品質への貢献も期待できる．ところが，具体的に要求工学に関するどの部分の質を向上させると製品の何が向上するのか，そもそもある部分の質が高いか低いかをどう判定すべきかについてのまとまった議論はいままであまり行われてこなかった．本稿は情報処理学会要求工学ワーキンググループ [1, 2] の活動の一部として，上記のような問題意識に基づき 2001 年から 2006 年までの 6 年間，議論をした結果をまとめたものである．

何故要求工学における品質の問題について集中して議論することとなったか，そして品質の問題の中でも特にどんな点に注目してきたかを述べる．要求工学ワーキンググループにおける参加者各自の研究および実践報告，および文献等の調査から，以下の点が大きな問題点であると 2001 年 5 月のソフトウェア工学研究会において結論付けた．

- 要求獲得分野の技術が未成熟.
- 要求変更や管理の観点から，要求記述を比較検討するための指標が必要であるにもかかわらず，標準的，実践的な指標が存在しない.
- 非機能要求の実現可能性の評価法がない.

そして，上記の問題点の内で二番目の項目に関する研究を中期的なグループの研究目標とした．そして，以下のような点を具体的な研究対象とすることとした．

- 要求仕様書自体の品質.
- 要求工学における品質と最終製品 (完成したソフトウェア) の品質の関係.

品質とは一般の辞書 (広辞苑等) では「物質の性質」となっているが，しかし，工学分野では物質だけでなく，開発過程 (プロセス) の性質についても議論するのが通常である．特に，ソフトウェア工学の分野では

1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets customer or user needs or expectations.

と定義されている [3]．これは要求や利用者のニーズの良し悪しには関係なく，要求やニーズに照らし合わせてソフトウェア製品の性質を吟味すると読めなくもない．しかし，現実的には仕様化された要求の品質に問題があった場合，その後全ての成果物の質に問題が出てしまうことを物語っている．

我々は要求工学に関わるプロセスの第一義的なモデルとして図 1.1 を作成した．この図はソフトウェア開発プロセス全体におけるスパイラルな開発モデルから，要求工学に関わる部分のみを射影したものである．要求工学にかかわる成果物は多数存在するが，現時点では要求仕様書のみを議論の対象としている．要求仕様書の品質の典型的な例としては，例えば矛盾がないこと (無矛盾性) 等である．仕様書の中である部分では「3 秒以内」という記述があるにもかかわらず，他の部分では「4 秒以内」と記述されている場合が矛盾の典型例である．矛盾に関する問題点に注目し，形式記述と非形式的記述を比較することで，形式記述の重要性を指摘する議論は古くから行われている [4].

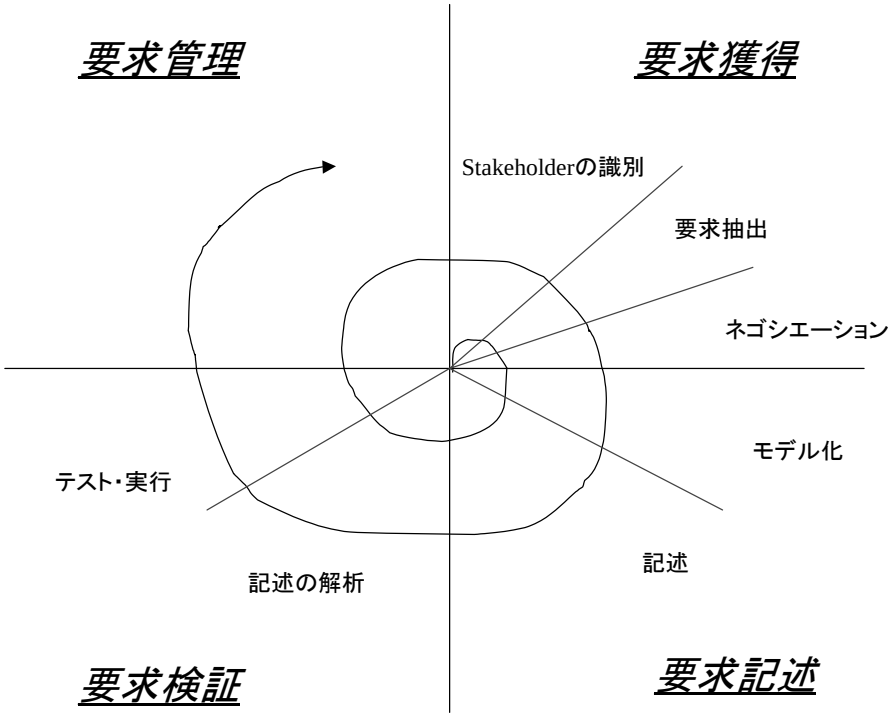

図 1.1: 要求工学プロセス

要求工学に関する品質にかかわる個々の技術については 2 章で議論することとして，ここでは，要求工学，ソフトウェア工学における代表的な教科書，図書における要求工学に関する品質への言及について簡単にまとめる．標準規格 [5] において，良い要求仕様書はどのような性質を持つべきかが言及されている．やはり，標準規格の影響力は大きく，この点は多くの教科書で言及されている [6, 7, 8, 9]．ソフトウェア工学における品質の議論はプロセス改善と結びついている場合がある．文献 [10] でもプロセス改善について言及されている．実務者が執筆した実践向けの図書では品質に関する扱いが大きいようである．例えば，[11] では品質検査について 1 章割いている．[12] でも品質に関わる章がみられる．比較的古い図書では品質に直接言及した記述は少ないようである [7, 13] が，品質測定に関する論文発表は 10 年以上前にも行われている [14].

本稿の構成は以下の通りである．2 章では要求工学の現状および課題について概観する．特

にどのような問題点が認識され，それに対してどのような技術提案されて来たかという観点から考察する．3 章では，要求工学の分野においてどのような計測が必要かつ可能かについて議論する．これは，ソフトウェア工学に限らず品質を議論する際，品質を吟味する対象を測定することが必要条件となるためである．4 章では，ソフトウェア要求に関する規格・基準を概観し，特に品質に深く関わる IEEE Std.830 で述べられた 8 つの品質特性を説明する．5 章では要求仕様書の 8 つの品質特性について，何を計測し，その結果に基づきどのような品質評価を行うかを例示する．6 章では，要求工学におけるどのような品質特性が，製品のどのような品質により強く影響を与えるかについて議論した結果を示す．このような影響関係を明らかにすることで，製品に関する特定の品質を高めることが要求された場合，要求工学プロセスの段階でどのような点に注意すべきかの指針を与えることができる．

第2章　要求分析/定義作業を支援するための技法

要求工学は工学である．工学とは，経験的問題を，科学的知識を使って効果的に解決へ導く技術である [15]．要求工学は，要求者のニーズを識別し，分析から設計へ至る過程の中で，統制化された形式でそれらを文書化することによって，ソフトウェアの目的を明らかにし，その実現を成功に導くための技術である．本章では,「はじめに」でも述べられているように，要求工学が対象とする問題を，要求仕様書を完成させるプロセスの中で，要求獲得（elicitation），要求記述（description），要求検証（verification），要求管理（management）という４つのフェーズ [16] に分けて議論し，最後に，これらのプロセスに入らない最近の話題について言及する．

2.1　要求獲得

要求獲得 (Requirements elicitation) は，ステークホルダを通して，ソフトウェアシステムに対する個別の要求や問題領域の知識を収集するための作業であり，情報収集（Information gathering）などとも呼ばれる．具体的な要求獲得作業に入る前に，システムのゴールやスコープを明確にし，主要なステークホルダを識別しておかなければならない．ゴールは，そのソフトウェアによって達成すべき目標である．システムには，人間が行っている作業の生産性や正確性を高めるために作成されるシステムと，ビジネスプロセスそのものを変革するために導入されるシステムがある．ゴールの違いによって，獲得すべき要求の種類にも違いが生じる．前者では，現状の人間の作業プロセスから要求の抽出が行われるが，後者では，経営的視点に立ったビジョンに焦点が当てられる．新規のビジネスを導入する場合，現行のステークホルダは十分な知識や要求を持っていないので，新たなステークホルダを探さなければならない．要求獲得フェーズを，ステークホルダの識別，要求抽出，ネゴシエーションの３プロセスに分ける．

(a) ステークホルダの抽出

開発しようとするシステムに対する要求を持っている人をユーザと呼んだ時代もあったが，その利用範囲が拡大するにつれて，システムに何らかの利害関係を持っている人，すなわちステークホルダを要求者と認識するようになった．すべてのステークホルダが開発プロジェクトのメンバーとなるわけではない．しかし，真の要求を知っているステークホルダがプロジェクトに参加しているかどうかが要求の品質を大きく左右する．Robertson らは，ステークホルダとして，そのシステムの依頼者（Client），ユーザ（User），購入者（Customer），システムの操作員（Operator），管理者（Administrator），プロジェクトの責任者（Owner），システムの開発者（Engineer）を挙げている [17]．同じステークホルダでも，その背景や立場によって異なった要求を持っていたり，熟練した専門家，未熟な担当者，非常勤の作業者といったようなレベルの違いによって持っている知識や要求が異

なる．ゲームソフトやワードプロセッサーのソフトなどの大衆商品に組み込まれたソフトウェアの場合には，ユーザを特定することが出来ない．そのような場合は，製品のビジネスプランなどを参考に，仮想のユーザグループを想定することになる．他のシステムと連携する場合は，情報システムを仮想ステークホルダと見做すことが必要となる．Sharp らは，ユーザ，開発者，立法者，意思決定者という基盤ステークホルダを定義し，よりシステマチックなステークホルダの識別を提案している [18]．はじめに，基盤ステークホルダのすべての役割を洗い出し，それぞれの役割に対する，サプライヤ，クライアントおよびサテライトという 3 種類の具体的なステークホルダを識別してゆく．ここで，サプライヤとは，基盤ステークホルダに情報を提供したり，その作業を支援するステークホルダであり，クライアントとは，基盤ステークホルダの製品を操作したり検査するステークホルダのことである．また，サテライトは基盤ステークホルダと何らかの形で接触を持つステークホルダである．

(b) 要求抽出

要求抽出（extraction）では，ステークホルダから問題領域の知識や要望を収集する．インタビューは，情報収集のための最も一般的で強力な技法である [19]．一見簡単そうなこの技法も，正しい要求を効果的に収集するための適当な手順やツールをもたないため，収集した情報を整理したり，情報の真偽を判定しなければならない [20]．調査によって問題領域の知識や個別の要求を収集する手段としては，インタビューの他にも，アンケートによる調査，ドキュメントの調査，作業現場の観察，分析者が実際に作業体験をする徒弟制などといった方法もある．何れの方法も，収集された情報の整理が課題として残る．グループ技法は，チーム・アプローチなどとも呼ばれ，複数のステークホルダや分析者が一堂に会して，情報や意見を交換し，知識の共有化を図るための技法である．作業の効率や，情報の品質などは，会議をリードするファシリテータの能力に依存するところが大である．グループ技法は，新しいアイデアを創出するのに有効である．ブレーンストーミングは，アイデア発想法の 1 つで，グループメンバーの発想の違いから，さらに多くのアイデアを生み出そうという手法である．KJ 法は，人間の直観を用いて問題の意味や構造を整理し，新たな解や発想を導出する [21]．フォーカスグループは，マーケティングでよく使われる技法で，特定のテーマに関し，無作為に選んだメンバーによる話し合いによって，様々な意見を収集する [22]．特定の個人を対象にしたインタビューに比べ，多様な意見の収集が可能である．ステークホルダの声を直接反映し，コンセンサスを得るためにもグループ技法が用いられる．Joint Application Develop（JAD）は，1970 年代に IBM によって提唱された技法 [23] で，ステークホルダが問題を定義したり，解を発見したりするのを分析者が支援する．認知心理学的技法は，特定の個人の知識を入手するための手法で，知識システム構築の中で専門家の知識を獲得するための知識獲得法の流れを引いている [24]．プロトコル分析では，指名されたステークホルダは，その認知プロセスを観察者に伝えるために，自らの行動を声に出しながら作業する．ラダーリングは，人工知能の知識獲得で使用された方法で，調査用の定型的な質問によって知識の構造を順次解明してゆく．カードソーティングでは，ステークホルダに，キーワードの書かれたカードを問題領域別に分類してもらう．レパートリ・グリッドは，実体ごとに，その属性と値をステークホルダに訪ねることによって，属性マトリックスを作成する．文脈的技法は，古典的技法や認知技法の代替として 1990 年代に登場した [25]．他の技法が対象世界を抽象化しようとするのに対し，文脈的技法では組織的，社会的な理解が重要であるという考えのもとに，観察された現象の普遍的モデルを作ろうとする [26]．ここでは，対象の観察といった民俗

学的な手法が使われ，会話分析では，会話のパターンを見つけるために詳細な分析が行われる [27].

(c) ネゴシエーション

獲得されたばかりの要求には，矛盾や曖昧性が含まれている．こうした問題をステークホルダと共に妥協点を発見し，優先順位を決定するのがネゴシエーションである．利害関係の異なったステークホルダ同士が異なった要求を持つことは避けられないし，問題領域に対する共通の理解だけでそれが解消されるとは限らない．個々のステークホルダの貢献度をモデル化し [28]，妥協点を発見しなければならない [29]．WinWin アプローチは，ステークホルダ間での要求の違いを解消するのではなく，互いの妥協点を求めるネゴシエーションのためのモデルとして提案された [30]．このモデルでは，ステークホルダの要求は，そのステークホルダの condition として表現され，対立点は issue として記録される．issue を解消するための option が WinWin プロセスのなかで提案され，ステークホルダのネゴシエーションによって，1 つの option が agreement として選ばれる [31]．ソフトウェア開発プロジェクトは，限られたコストと期間内で要求を実現しなければならないので，一度にすべての要求を実現できるとは限らない．個々の要求に優先度を決める必要がある [5]．優先順位の決定は，解決すべき問題の重要度や緊急性，ステークホルダの好みなどを考慮して決定されるため，相反する複数の意見に対する意思決定機構が必要となる．階層化意思決定法（AHP： Analytic Hierarchy Process）は，複数の代替案に対する意志決定を行うための技法である．問題を階層構造あるいはネットワーク構造を使って表現し，個々の案を一対比較によって評価し，その構造内での相対的な関係をつくり上げる [32].

2.2 要求記述

要求抽出によって獲得された問題領域の知識や要望は，その内容を設計者に正確に伝えるために，厳密性を高めて記述する必要がある．要求を記述するのに，問題領域に関する知識をモデルとして表現する場合と，要求そのものを仕様として記述する場合がある．モデルは，図式を使って記述されることが多いが，形式仕様記述言語のようなテキストによって記述されることもある．要求記述フェーズを，モデル化と記述の 2 つのプロセスに分ける．

(a) モデル化

モデルは要求のもう 1 つの表現である．モデルによって要求への理解が深まる．モデルとは物事の本質的な部分だけをある観点に従って抜き出し，整理したもので，実際にそれを書き下すための「記述」と密接につながっている．モデル化手法は，どのような事項が本質的か，それをどう整理するかの情報を提供してくれ，モデルを構築する人間の思考過程をガイドする役割を持っている．問題領域やシステムの稼動環境をモデル化することによって，システムの目的や設計の正しさを容易に判定することができるようになり，ステークホルダと分析者のコミュニケーションの基盤を提供してくれる．抽象化された問題領域をグラフィカルに表現したモデルは，テキストによる表現に比べ，直感的な理解に優れている反面，論理的な説明能力には乏しい．モデルの記述能力は，モデル記述言語（ビジュアル言語とも呼ばれる）の記述能力に依存する．それぞれのモデル化手法によって独自の図式が用いられるが，IDEF[33] や UML のような記法の標準化も進んでいる．UML は，様々なオブジェクト指向方法論のモデルの記法を統一化するために制定

されたもので，複数の図式から構成されている．要求をモデル化する最初の試みは，ステークホルダとシステムの動的な変化あるいは機能的な振る舞いをモデル化することであった．構造化手法では実世界（問題領域）の業務から情報（データ）の流れを抽出し，データを処理するためのシステムの機能を，データフロー図を使って表現した [34]．JSD 法は，実世界を実体とその行動という 2 つの視点で捉え [35]，現在のオブジェクト指向によるモデル [36] の先駆的役割を果たしていると言える．要求そのものをより直接的に表現するユースケースモデルは，単に要求を記述するだけでなく，ソフトウェア開発工数の見積もり [37] や，プログラムのテスト [38] などにも使用されている．

ドメインモデル

ドメインモデルは，システムへの要求を直接的に表現するというより，問題領域の構造を正しく理解し，実世界とソフトウェアの構造的な同一性を実現するために用いられることが多い [39]．問題領域を抽象化することによって，問題の本質を明らかにしたり，問題領域に関する知識の曖昧さや矛盾を解消したり，異なった考え方を持ったメンバー間の理解と意見の統一を図ったりするのに有効である [40]．実装言語の進化と共に，ドメインモデルの構成要素も異なってきた．Jackson は，手続き型言語を前提に，実体とその行動ならなる実体構造図によって問題領域をモデル化した [35]．オブジェクト指向言語を前提とした方法論では，実体の静的側面と動的側面を別々にモデル化しようとした．実体の概念はクラス図によって，その行動はシーケンス図などの補助的図式を使って表現される [36].

ゴール指向モデル

ステークホルダの意図を基に，要求を「達成すべきゴール」という観点で整理していったものがゴール指向モデルである．ゴール指向モデルでは，ゴール分解によって，階層化されたサブゴール木が展開される．KAOS や i*といったゴール指向のモデル化手法では，組織の構造や目標，構成メンバーの役割や責務，タスクなどがモデル化される．要求は，ゴールの操作化 [41] や Means-end 分解 [42] などを通して導出される．また，複数の上位ゴールに対する寄与度を計算することによって，採用すべきサブゴールを選択するための方式も提案されている [43].

ユースケースモデル

要求そのものを，より直接的に表現しようというのがユースケースモデルである [44]．Jacobson の方法論 Objectory の中で使用されていたモデルが UML の中に取り込まれた．ユースケースはユーザの視点から捉えたシステムへの機能要求であり，その詳細はシナリオによって定義される．ユースケースモデルが表現できるのは機能モデルだけであるという批判に対し，Jacobson は，非機能要求は機能要求の属性であり，機能要求が定義できれば，FURPS モデル [45] を使って非機能要求の導出は可能だとしている [46]．また，事前条件や事後条件などを明確にすることによって，ユースケースモデルの厳密性を高めるためのユースケース記述が提案されている [47, 48].

(b) 記述

ステークホルダの要求を，ソフトウェアの開発者に伝えるために，厳密に記述したものが，要求仕様である．仕様の記述にはテキスト表現が用いられることが多い [49]．自然言語で要求仕様を記述する際の文書構造の標準，例えば IEEE Std.830[5] など，が存在している．要求と仕様の関係については，いくつかの説明が行われている．Jackson は，環境と機械によって仕様を説明している [50]．要求はシステムの外にある環境について述べ

たものであり，システム（機械）は要求を実現する手段である．仕様は，環境とシステムの境界上で，システムの振る舞いを，現象として記述したものである．Parnas は，利用環境とソフトウェアの間の４変数モデルによって要求と仕様との関係を説明している [51]. 環境とソフトウェアの間には，センサーなどの入力デバイスと，アクチュエータなどの出力デバイスがあると考える．環境は，入力デバイスへの観測変数によって認識され，出力デバイスからの制御変数によって操作される．一方，ソフトウェアは，入力データから出力データを生成する機構である．観測変数と制御変数の関係が要求であり，入力データと出力データの関係が仕様である．厳密な仕様を記述したい場合は，VDM [52] や Z [53] のような形式仕様言語を使用する．形式仕様言語は，数学や論理学を使って，問題領域の実体や事象の生起を正確かつ厳密に記述することができ，仕様の検証も可能である [54].

2.3 要求検証

検証と妥当性確認は Verification & Validation の訳である．Verification はソフトウェアの開発工程の品質を検査することであり，Validation は製品そのものの品質を検査することであるといえる．本章で扱う要求検証は，最終製品の品質検証ではなく，要求仕様書の品質検証についてである．要求検証フェーズを，記述の解析とテスト・実行の 2 つのプロセスに分ける．

(a) 記述の解析

記述の解析では，記述の形式的な正しさの解析と，内容の見直しという 2 種類の解析が行われる．仕様の形式的な正しさを検証する場合，仕様が形式的言語で記述されていれば，自動推論 [55] やモデル検査によって，仕様の一貫性と完全性を自動的にチェックすることが可能である [56]. モデル検査は，モデルがシステムに要求される性質を満たすかどうかを自動的に検査する手段で，どのような経路を通ってもデッドロックのような望ましく無い事象が発生しないという安全性（safety）と，望ましいことはいつか必ず起きるという活性（liveness）を検査する．ラベル付きの遷移システムとして記述したモデルの性質を，時相論理によるモデル検査手法でチェックする手法が多く提案されており [54]，SPIN のようなモデル検査系も提供されている [57]. 手動で検証を行う場合は，インスペクション [58] やウォークスルーなどの手法が用いられる．記述内容は，インパクト分析，リスク分析，技術要素分析という 3 つの視点から見直しが行われる．インパクト分析では，ステークホルダ，ビジネスプロセス，社会環境などへのインパクトを分析する．ステークホルダが受けるインパクトは，システムの導入によって生じる環境変化への対応力によって異なってくる．ビジネスプロセスや製品製造プロセスへのインパクトとは，処理手順の変更や処理速度の変更によってプロセスに発生する遅延や障害である．個人情報の流出の可能性などは，社会環境へのインパクトである．リスク分析では，人的リスク，コスト的リスク，技術的リスクなどを分析する．いずれも，要求を実現する上で欠かせない条件である．人的リスクには，技術者だけでなく管理者の能力も含まれるし，コスト的リスクには，予算と期間が含まれる．技術的リスクは要求を実現するための技術を導入できるかどうかという問題で，人的リスクやコスト的リスクと密接に関係している．リスクを回避できない場合は，要求を変更しなければならない．環境の変化によってリスクも変化するので，常に注意が必要である [59]. 技術要素分析は，適用可能な技術のなかから，適切な技術を選択するために行われるが，他の技術との組み合わせや，その技術の安定性などが重要な判断基準となる．

(b) テスト・実行

作成されたモデルや仕様を実際に動かして要求を検証することもできる．プロトタイピングは，その品質を定量化できない場合や，ステークホルダの意見や反応が必要な場合に用いられる手法であり [60]，感覚的な判断が必要とされるユーザインタフェース要求などに良く用いられる．プロトタイピングは，曖昧な要求を確定するためだけではなく，複雑なアルゴリズムの検証や，開発の可否を判定するためのデモンストレーションとしても用いられる．アニメーションは，システムの振る舞いを視覚的に表現することができるため，対象の動的な変化を時間の経過にしたがって確認することができるが，正確な順序制御や矛盾のチェックを行うには向いていない [61, 62].

2.4　要求管理

要求の変更は避けられない現象である．要求変更の発生には，要求獲得作業の失敗と，外部環境の変化という 2 つの原因がある．前者は修正作業に無駄な追加コストがかかるので好ましく無い変更，後者は製品の使用期間の長期化や販売数量の増加につながる好ましい変更であると言われている [17]．要求変更が発生すると，その要求変更がリーズナブルなものどうかを判定し，関連する文書の該当箇所を修正しなければならない．要求仕様書への主たる変更は，要求の追加，削除，修正である．追加は，ステークホルダの要求の変更や分析ミスなどによって生じる．削除は，開発コストや期間の超過によって発生し，一旦定義された要求の取り消しが行われる [63]．モデルの部分的な変更は，同じモデルの他の部分との間に論理的な矛盾や衝突を引き起こすため，変更の影響を分析し，モデルの他の部分にも，必要な修正を施さなければならない [64].

2.5　その他

インターネットをはじめとする技術革新によって，ソフトウェアを取り巻く世界は，急速な発展と多様化の度合いを増しつつある．進化と多様化は，要求の不安定性となって現われる [65]. 要求進化の原因は，ニーズの絶え間ない変化であり [66]，多様化の原因は，異なったステークホルダの異なった視点と興味である [67]．こうした要求の進化と多様化に対応するための技術についてまとめて概括する．

(a) 初期要求プロセス

要求は本来的に不安定であり，その不安定性を極力縮小するには，変化の原因を理解する必要がある，という考え方に基づいて，システム化対象領域だけでなく，要求そのものが発生するビジネス環境全体をモデル化しようという機運が高まった．所謂，ビジネスモデリングである．ビジネスモデリングでは，単に現状のビジネスをモデル化するだけではなく，ビジネスの将来像をモデル化することによって，要求の変更を先取りしようとしている．Eriksson らは，プロセスよりも詳細なビジネス活動に注目し，UML のアクティビティ図を発展させたモデルでビジネス活動を描き [68]，Marshall らは，ビジネス活動を構成している組織や実体に注目し，それをクラス図によって描いている [69]．こうした手法は，要求獲得作業に入る前段階に実施することによって，その後の要求変化への対応を容易にしようとするねらいがある．ビジネス上の意図を，明示的にモデルに組み込むため

に，ビジネスモデルにゴール指向を組み入れたのが i*法である [42] ．i*法が扱うビジネス上の意図は，企業経営レベルの意図ではなく，アクター同士の依存関係というレベルに限定されており，ソフトウェア技術者にも扱えるものとなっている．この手法によって，従来の要求工学が対象としていた「初期要求をもとに高品質な要求仕様書を作り上げる」フェーズよりも前のフェーズの存在とその重要性が認識されてきた．i*法で扱うゴールには，通常のビジネスゴールと共に，達成されたかどうかの判断が難しいソフトゴールも明示的にモデル化され，ゴール分解によって必要なタスクが導出されるようになっている．

(b) メソッド工学

進化し多様化する要求に対応するためのもう 1 つの技術は，既成の方法論をそのまま適用するのではなく，それぞれのプロジェクトの状況に合わせて個別の手法を適用することである．そうした問題を解決するための技術にメソッド工学がある．メソッド工学は，方法論を工学的に捉え，方法論自体の開発，評価，使いこなすための教育などの技術である．要求プロジェクトが置かれた状況は多様で，使用する技術を間違えば，プロジェクトそのものが破綻してしまいかねない．プロジェクトの目標や，遭遇した問題ごとにそれを解決するための手法を選択し，作業の進展に合わせて，それらを組み合わせて使用することが求められる [70]．しかし，これまでに提案されてきた技法や手法の種類は膨大な数に上り，多様なメンバーから構成されるプロジェクトの中で，目標とするシステムに合わせて，適切な手法を選択し，それらを組み合わせるということは容易なことではない．こうした問題を解決するためには，手法の選択と統合と言う問題を解決しなければならない．ステークホルダの視点の違いは，所属する組織，組織内での責務，個人的な世界観や知識など，個人のバックグランドの違いによって生じる．こうした視点の違いに着目したモデル化や要求獲得法が提案されている．Finkelstein らは，システム開発チームのメンバーのスキルや知識や経験などのバックグラウンドと開発チームの中で果たすべき役割を観点（perspective）として捉え，異なったスタイルでそれを表現するための Viewpoints というフレームワークを提案している [71]．Viewpoint アプローチは，ステークホルダの観点ごとに異なった表現形式で記述された部分要求を，視点統合規則によって連携させ，表現形式の意味論的一貫性を保障するためのフレームワークを提案している [72]．視点統合規則は，それぞれの視点ごとに準備された Viewpoint テンプレートの中に，実行されるべきアクションとして記述されている．一方，Situational Methods Engineering は，既存の手法から断片を切り出し，それらを組み立てて，プロジェクト固有の手法を構築しようとする技術である．Brinkkemper らは，ソフトウェア成果物である製品断片と共に，手法の断片も工程断片に分割し，それぞれの断片を組み合わせて新たな手法を構築するために，メタモデルのレベルでの統合ガイドラインを示している [73].

第3章　要求工学における計測とは？

プロジェクトにおける計測の目的は理解／制御／向上といわれる [74]．理解とはプロジェクトの現在の状態を知り，今後の展開に対する予測の基準にすることをさす．制御とは計測の結果と予測との違いを把握して活動を修正することであり，たとえば作業量を調整するために労働時間をモニターすることがそれにあたる．また，向上とは結果を改善のための目安として利用することである．たとえばレビューによってソースコードの複雑さを把握し，改善のための行動をおこなうことをさす．逆に，計測をおこなわずに制御や向上をおこなうことは困難である [75].

要求定義工程もプロジェクトの工程の一部である．そして要求定義の失敗が後工程に与える影響は設計段階や実装段階での失敗に比べて大きい [76]．実際，要求定義の失敗はプロジェクトの主要な失敗原因のひとつである [77][78]．したがって要求仕様書の品質はプロジェクトを遂行する上で重要な関心事であり，要求仕様書の品質を計測して制御・向上をはかることが必要である．

この章では計測について一般的な議論をおこない，後の章での尺度設定に対する指針を与える．

3.1　計測と尺度

計測とは，対象の属性を区分や数値に写影することをいう [74]．たとえば，人の身長を何らかの基準で高い・低いに分類することや，数値化して 170cm などとすることが計測である．区分に写影するための分類基準や数値に写影するための物差しを尺度 (scale) と言う．ただし，計測する属性を持つすべての対象を写影することができ，かつ，ひとつの対象を複数の区分や数値に写影することがあってはならない．通常，ひとつの属性に対して目的に応じた様々な尺度を考えることができる．

直接計測できない／しにくい属性を計測するには，他の計測可能な属性から間接的に求めることが出来るとするモデルを用意すればよい．たとえば密度の場合，重さを体積で割ると密度になるというモデルが知られている．このように複数の属性に対する計測を組み合わせた尺度のことを，複合尺度と呼ぶ．

また人間の属性に賢さがあるとすると，この属性は直接計測することはできない．そのため賢さを評価するためにはモデルが必要である．たとえば知能テストの得点や額の広さに比例する，というモデルを考えることができる (ここではモデルの正否を問題にしない)．このようにモデルは複数あっても良い．

なお，測定したい属性によっては状況にあわせてモデルを修正し，より計測の目的に即した結果を得ることができるようにする場合もある．

3.2　尺度の分類

尺度は，その性質によって以下の4種類に分類することができる．

1. 名義尺度 (nominal scale)
2. 順序尺度 (ordinal scale)
3. 差尺度 (interval scale)
4. 比率尺度 (ratio scale)

3.2.1　名義尺度

属性をいくつかの区分に分類する尺度をいう．ある写像が名義尺度であるためには，次の二つの条件を満たす必要がある．

1. 区分が全体を網羅していること (区分できない対象がないこと).
2. 二つ以上の区分に属する対象がないこと．

名義尺度の区分の間は比較できる必要がない (比較できる場合は順序尺度となる)．たとえば人間の性を男性・女性・その他・不明に分類する場合，これは名義尺度になる．

3.2.2　順序尺度

区分間で比較できる名義尺度を順序尺度と呼ぶ．たとえばソフトウェアモジュールの複雑さを「明解」「単純」「適切」「複雑」「理解不能」に分類した場合,「より複雑」あるいは「より単純」といった比較ができるため，順序尺度になる．

比較は推移律を満たす必要がある．たとえばAがBより「より複雑」であり，BがCより「より複雑」であるならば，AはCより「より複雑」でなければならない．しかし，各区分間の差については何も言うことができない.「複雑」と「適切」の差と「適切」と「単純」の間の差を比較し，どちらの方が複雑さの差が大きいか，などと議論することは不可能である (比較できる場合は差尺度になる)．また「明解」なモジュールと「適切」なモジュールが半数づつ含まれている上位モジュールに対して，平均して「単純」であると言うこともできない．

3.2.3　差尺度

区分間の差の大きさが定義されて順序尺度を差尺度と呼ぶ．これによって区分同士の加算や減算が可能になる．たとえば，ある出来事の発生日を西暦で表示する場合を考える．西暦2006年8月15日におきた出来事は，2006年8月12日におきた出来事より3日間遅いといえる．もし区分間を比率で評価できる場合，それは比率尺度となる．

3.2.4 比率尺度

比率による比較が可能で 0 が存在する差尺度を比率尺度と呼ぶ．たとえば身長を長さに写影する尺度は比率尺度であり，170cm の長さは 17cm の 10 倍長いと言うことができる．

一般に同じ属性に対する異なる比率尺度同士では計測結果の値が異なる．たとえば同じ対象について身長を計測したとしても，尺度によって 170(cm) や 1.7(m) という数値になりうる．しかし「身長と手の長さの比」のような量を考えると，どのような比率尺度を用いて計測しても同じ値が得られる．このように値そのものに意味があるような尺度を，絶対尺度 (absolute scale) と呼んで区別する場合がある．

3.3 尺度の品質

Woodings は尺度に対する品質特性として 13 の項目をあげている [79] が，ここでは以下の 3 つの特性について議論する．計測の目的に対して利用する尺度および計測方法に意味があるかどうかは以下の要因から評価しなければならない．

1. 必要な操作をおこなうことができるか
2. 正確性／信頼性
3. 計測にかかるコスト

たとえば遊園地のジェットコースターでは，基準線よりも上か下かのふたつの区分にしか身長をわけないし，計測方法も板に書かれた基準線の前に (靴も脱がずに) 立つだけであるが，利用目的には合致している．またアンケートの満足度評価などでは手間がかかる計測をおこなわず，自己申告による順序尺度を用いることが多い．どのような尺度を用いるかは計測の手間と尺度の利用目的のトレードオフで判断するべきである．

3.3.1 必要な操作をおこなうことができるか

得られたデータに対して適用可能な統計的な手法は，尺度の種類によって異なる．表 3.1 に示したように比率尺度を利用すると，もっとも豊富な手法を利用することができる．

3.3.2 正確性／信頼性

計測の際の技術的な問題から，本来属性から得られるべき区分や数値から一定の傾向のずれが生じてしまい，正確な計測ができない場合がある [80]．必要な正確さで結果を得ることが出来ない場合，正確性が足りない計測ということになる．正確性をはかる目安を系統誤差と呼ぶ．

一方，同じものを繰替えし計測しても読み間違いなど偶然の要素によって異なる結果を得る場合がある．必要なだけの精度が得られない場合，信頼性が足りない計測という．信頼性をはかる目安を偶然誤差と呼ぶ．

文書の一部から品質を計測する場合，計測対象をどのように選択するかに依存して誤差が発生する．これは偶然誤差である．一方品質評価のために計測をおこなう時に，たとえば何を曖昧と見なすかなどはアルゴリズムによって結果に幅が生じる．これは系統誤差である．

偶然誤差は計測を繰り返すことで統計的に評価・改善することができる．しかし系統誤差は計測の過程に含まれる様々な操作や道具に含まれる誤差をひとつづつ検討し，排除したり上限を評価したりしなければならない．

複合尺度などで複数の尺度から値を導出する場合，誤差の伝播という形で正確性／信頼性を評価することができる．表 3.2 に，$x \pm e_x, y \pm e_y$ という二つの計測結果があった場合の誤差の大きさを示す．なお，正規分布など確率分布が仮定できる場合は，より正確に推定することができる．

3.4　ソフトウェア工学における計測

本節では，要求工学を含むソフトウェア工学に着目して，どのような属性が計測対象になりうるか，またそれらをどのように導出するかを説明する．これらの知識は要求定義工程における計測対象を検討する際に利用することができる．

一般にソフトウェア工学における計測対象は，プロセス／プロダクト／リソースに分類することができる [74].

プロセス　ソフトウェアの作成に関係する一連の作業をプロセスと言う．いつ作業が終了するか，どのくらいのコストがかかるか，制御のしやすさや見える化の度合いなどに興味がある．また他のプロジェクトとくらべてどうなのか，という点を比較できる必要がある．

プロダクト　最終的にリリースする製品だけでなく，途中で作成する要求仕様書などの文書やプロトタイプなども計測の対象になる．品質の制御・保証が計測の目的になる．

リソース　リソースにはソフトウェアの作成にかかわる全ての入力が含まれる．我々はその大きさ，コスト，品質について関心があり，それらが計測対象になる．

3.4.1　代表的な属性

ソースコードの理解しやすさはメンテナンスする側の人とソースコードとの相性によって大きく異なる．そのためソースコード自体に内在する品質というよりは対象と環境の間の関係が持つ属性と考えて良い．このような属性を外部属性と呼ぶ．一方行数のように対象そのものが持つ属性を内部属性と呼ぶ．ソフトウェア工学で扱われる代表的な属性を表 3.3 に示した．

3.4.2　属性の導出

ここでは広く知られている GQM 法と特性分析による方法を紹介する．

GQM 法では達成したい目的に則した具体的な計測対象を得ることができる．そのため実プロジェクトやプロセス標準を元に計測を設計するには適しているが，目的に依存してしまうため一般性が損なわれてしまう．一方特性分析による方法は品質モデルが必要になるが，一般性を損なうことがない．

GQM 法 GQM(Goal-Question-Metrics) 法 [81] は，高い抽象度のゴール (目的) を分解してサブゴールを導き，段階を踏んで具体化していくことで計測可能な属性を用いた質問を得る方法である (表 3.4).

特性分析 一方，品質モデルを利用した方法 [83] では，抽象的な品質特性から始めて順次分析をおこない．最終的に計測可能な属性を導く．表 3.5 によく知られたソフトウェアの品質モデル ISO9126[84] を採用した例を示す.

3.5 要求工学における計測

要求工学もソフトウェア工学の一部であり，前節の議論はそのまま成立する．どのような属性を計測対象とするか，またどのような尺度を採用するかについては第 5 章で議論をおこなうため本節では触れず，尺度設計の指針のみを示す.

3.5.1 尺度の設計指針

手法に対する中立性を保つため，特定の手法に依存することがないようにするべきである．

たとえば要求獲得時の手法としてインタビューによる方法を採用した場合「立場の異なる全てのステークホルダーにインタビューしたか」という尺度を考えることができる．一方ゴール指向アプローチを用いた場合は「システム化が可能なサブゴールまで分解したか」という尺度がありうる．しかしこれらの尺度はそれぞれの手法に依存しているため，尺度としては採用するべきではない．一方「矛盾する要求に対する解消方法が用意されているか」という尺度は手法に依存しないため採用して良い.

また，プロジェクトでの尺度の利用方法を明確にするために，以下の項目について示すことが望ましいだろう.

- 計測の目的
- 計測対象
- 誰 (役割) が，どのタイミングで計測するのか
- 計測方法
- 計測結果の目安

表 3.1: 4 種類の尺度およびその例と，可能な統計的操作 [74].

Scale type	**Examples**	**Defining relations**	**Examples of appropriate statistics**	**Appropriate statistical tests**
Nominal	Labeling, classifying entities	Equivalence	Mode Frequency	Non-parametric
Ordinal	Preference, hardness, air quality, intelligence tests(raw scores)	Equivalence, Greater than	Median, Percentile, Spearman r, Kendall τ, Kendall W	Non-parametric
Interval	Relative time, temperature(Fahrenheit, Celsius), intelligence tests(standardized scores)	Equivalence, Greater than, Known any intervals	Mean, Standard deviation, Pearson product-moment correlation, Multiple product-moment correlation	Non-parametric
Ratio	Time interval, length, temperture(Kelvin), counting entities	Equivalence, Greater than, Known ratio of any intervals, Known ratio of any two scale values	Geometric mean, Coefficient of variation	Non-parametric and parametric

表 3.2: 誤差の伝播 [80]. ただし，$x \gg e_x, y \gg e_y$ とした.

演算	誤差
和／差	$e_x + e_y$
積	$x \cdot e_x + y \cdot e_y$
商	$(x/y) \cdot (e_x/x + e_y/y)$

表 3.3: プロダクト／プロセス／リソースに属する対象とその属性の例 [74].

ENTITIES	**ATTRIBUTES**	
Products	*Internal*	*External*
Specification	size, reuse, modularity, redundancy, functionality, syntactic correctness, ...	comprehensibility, maintainability, ...
Designs	size, reuse, modularity, coupling, cohesiveness, functionality, ...	quality, complexity, maintainability, ...
Code	size, reuse, modularity, coupling, functionality, algorithmic complexity, control-flow structuredness, ...	reliability, usability, maintainability, ...
Test data	size, coverage level, ...	quality, ...
...	...	...
Processes		
Constructing specification	time, effort, number of requirements changes, ...	quality, cost, stability, ...
Detailed design	time, effort, number of specification faults found, ...	cost, cost-effectiveness, ...
Testing	time, effort, number of coding faults found, ...	cost, cost-effectiveness, stability, ...
...	...	...
Resources		
Personnel	age, price, ...	productivity, experience, intelligence,...
Teams	size, communication level, structuredness, ...	productivity, quality, ...
Software	price, size, ...	usability, reliability, ...
Hardware	price, speed, memory size, ...	reliability, ...
Offices	size, temperature, light, ...	comfort, quality, ...
...	...	...

表 3.4: GQM 法の適用例 [82].

Goal	Questions	Metrics
Plan	How much does the inspection process cost?	Average effort per KLOC, Percentage of reinspections
	How much calendar time does the inspection process take?	Average effort per KLOC, Total KLOC inspected
Monitor and control	What is the quality of the inspected software?	Average faults detected per KLOC, Average inspection rate, Average preparation rate
	To what degree did the staff conform to the procedures?	Average inspection rate, Average preparation rate, Average lines of code inspected, Percentage of reinspections
	What is the status of the inspection process?	Total KLOC inspected
Improve	How effective is the inspection process?	Defect removal efficiency, Average faults detected per KLOC, Average inspection rate, Average preparation rate, Average lines of code inspected
	What is the productivity of the inspection process?	Average effort per fault detected, Average inspection rate, Average preparation rate, Average lines of code inspected

表 3.5: ISO9126 にもとづいた特性分解の例 (一部).

Factor	Criteria	Metrics(属性)
保守性	解析性	…
	変更性	変更に失敗した回数，変更に要した工数，変更の回数
	安定性	…
	試験性	テストの精度，テストに要した工数

第4章　ソフトウェア要求仕様に対する規格・標準の例

この章では，ソフトウェア要求仕様に関する規格・標準として IEEE Std. 830-1998 (Revision of IEEE Std.830-1993) Recommended Practice for Software Requirements Specifications（意訳：ソフトウェア要求仕様書のための推奨実践）について解説する．本規格では，何を記述すべきかを推薦しているのは勿論，何を書くべきでないかも随所に言及している．例えば，設計や実装の詳細に立ち入ることや，品質保証計画については要求仕様書ではなく，他の文書に記述すべきであるとしている

4.1　構成

本文書は，本文 5 節，付録 2 節から成っており，それぞれの概要は以下のとおりである．
1 節 概要
2 節 参考文献
3 節 定義: 用語の定義
4 節 良い要求仕様書が備えるべき特性
5 節 要求仕様書に含まれる部品
付録 A. 要求仕様書のテンプレート
付録 B. IEEE/EIA 12207.1-1997（ソフトウェアライフサイクルに関するデータの標準規格）との関連についての説明

1 節ではこの標準が対象としている範囲について述べている．この標準ではソフトウェア要求仕様書に含むべき内容とその品質についての説明と，いくつかの例が示されている．2 節では，この標準を作成するにあたって参考とした文献の出典一覧が記されている．3 節の「定義」では，用語についての簡単な定義が示されている．要求仕様書を作成する上で必要な用語である “Contract”, “Customer”, “Supplier”, “User” の 4 つについて，IEEE Std.610.12-1990 （ソフトウェア工学における用語集）から定義を引用している．契約（Contract）: 顧客（Customer）とシステム供給者（Supplier）の間で同意される文書で，技術的要求，組織的要求，費用，製品（Product）のためのスケジュールを含む．契約には，合意事項や関係する第三者の期待などの非公式だが意味のある有益な情報を含むこともある．利用者（User）は，実際にシステムを操作する人であり，オペレータや最終的なサービスの受け取り手であるエンドユーザが含まれる．4 節「良い要求仕様書が備えるべき特性」については 4.2 で，5 節「要求仕様書に含まれる部品」については 4.3 で説明する．

4.2　4節「良い要求仕様書が備えるべき特性」

4節ではいわゆる良い要求仕様書の品質特性について述べられており，最も重要な部分である．ここでは要求仕様書を記述する際，考慮しなければならない背景を以下の8項目に分けて説明している．

1. 要求仕様書の本質
2. 要求仕様書の環境
3. 良い要求仕様書がもつべき特性
4. 要求仕様書の合同準備
5. 要求仕様書の進化
6. プロトタイピング
7. 要求仕様書内の設計事項
8. 要求仕様書内のプロジェクト要求

「1 要求仕様書の本質」には，そもそも要求仕様書とは何かが説明されている．要求仕様書とは，特定の環境下で，特定の機能を発揮する特定のソフトウェア製品やプログラム (群) の仕様書であるとしている．よって，要求仕様書は以下の点を扱わなければならないとしている．

- 機能性 (Functionality): どのような仕事を支援してくれるのかの説明.
- 外部インタフェース (External Interface): 人間，ハードウェア，他のソフトウェアとのインタフェース.
- 効率 (Performance): スピード，可用性，レスポンス時間，障害復帰時間等.
- 属性 (Attribute): ポータビリティ，プログラムの妥当性，保守性，セキュリティ.
- 実装を制約する設計制約: 開発言語やOSの指定など.

以降で，IEEE Std.830での要求仕様書の品質特性について解説する．

「4節 良い要求仕様書がもつべき特性」の中に，要求仕様書の持つべき特性として以下の8つが直接的に言及されている．

- 妥当である (Correct): 仕様書にある全ての要求が，開発されるソフトウェアが満たすべき事柄と一致している場合，仕様書が妥当であるとする．これは，顧客や利用者の真のニーズに合致しているか否かを示す性質であり，これを自動的，系統的に確かめる術はない．当節で後述する追跡可能性は，この性質を確認する助けになる．なお，他の仕様と要求仕様書が一致していない場合には，どの仕様が有効 (Valid) であるかを確認する必要がある．また，妥当であるか否かは，要求仕様書より上位の仕様書 (例えば，システム要求仕様書 IEEE-Std-1233-1998 など) や，プロジェクトに関する文書，適用すべき標準規格などとの整合性も含んだ判断が必要である．なお，妥当性については本報告「5.2 妥当性」にても解説されているので参照されたい．

- 非あいまいである (Unambiguous): 仕様書内の全ての要求が唯一の解釈を持つ場合，その仕様書は非あいまいであるとする．逆に同じ仕様書内に書かれている同じ記述の要求が異なって解釈されるような場合にはあいまいであることになる．例えば「大きい，小さい」などの表現ではなく，具体的な数値による指定などにより，この性質は充足される．非あいまいであることについては本報告「5.3 非あいまい性」にても解説されているので参照されたい．

- 完全である (Complete): ここでは考えうるケースについて十分な検討がされ，それについて要求仕様書として扱うべき性質について記述されているレベルかどうかを問題にしている．具体的に満たすべき条件は 3 つ挙げられている．これらが満たされた場合，仕様書は完全であるとする．
  a) すべての重要な要求は，要求仕様書が扱わなくてはならない性質である機能性・外部インタフェース・効率・属性・実装を制約する設計制約条件について関連づけられていなくてはならない．特にシステム仕様にある外部インタフェースは述べられていなくてはならない．
  b) すべての実際に起こりうる状況における入力データに対するソフトウェアの対応についての定義がされていること．重要な点は，有効値・無効値両方を網羅していることである．
  c) SRS にある図や表やその単位についてのタイトルや参照元がすべて存在すること
  完全であることについては本報告「5.4 完全性」にても解説されているので参照されたい．

- 無矛盾である (Consistent): ここでは，矛盾とは要求仕様書内部の有無に着目した基準とされている．要求仕様書内に矛盾がない場合には，要求仕様書は無矛盾であるという．しかし，外部，例えば上位のシステム仕様書との矛盾がある場合には，その仕様書は前述した性質の「妥当である」を満たしていないものとする．なお，仕様書内部の矛盾として以下が挙げられている．
  a) 複数箇所に仕様が記述され，その内容が異なっている．例えば，ある仕様について，処理結果が表形式で出力されることになっている箇所と，テキスト形式で出力する記述がされている箇所があるといった場合には，矛盾している．
  b) 1 つの要求について，複数の記述がされていて内容が異なっている．例えば A は B の次に動作するという記述と，A と B は同時に動作する，という両方がある場合には矛盾している．
  c) 仕様に使われる用語が標準化されていない場合にも，一つの動作を複数の名称で表していたり，別の意味に解釈されることがあり，矛盾している．
  無矛盾であることについては本報告「5.5 無矛盾性」にても解説されているので参照されたい．

- 重要度と安定性がランク付けられている (Ranked for importance and/or stability): 各要求について，その重要度あるいは安定性について明確に指標が存在し記述されている場合に，その要求仕様書は重要度と安定性がランクづけられているとする．それぞれの要求の重要度・安定性の差異を明らかにするために，それぞれ目安となる記述をする必要がある．安定性については，今までの経験等に基づきその変更が関連部門や関係者，システムに与える影響を考慮した上で，将来の変更予測などを基に回数などで表現する．また，要求そのものに「必須の要求」「条件付要求」「あってもなくても良い要求」とランクをつける

こともできる．これら重要度と安定性のランクづけについては本報告「5.6.1 重要度と安定性のランク付け」にても解説されているので参照されたい．

- 検証可能である (Verifiable): 開発されるソフトウェアが要求仕様書の記述内容を満たすか否かをチェックするための方法があり，チェック作業が妥当なコスト内，妥当な時間内に行える場合，その要求仕様書は検証可能であるとする．例えば,「良い」,「快適」などの曖昧な記述を含む場合には検証可能とは言えず，数値などで指定された記述は検証しやすい．これら検証可能性については本報告「5.6.2 検証可能性」にても解説されているので参照されたい．

- 変更可能である (Modifiable): 仕様書が容易に完全性と無矛盾性を損なわず変更できる場合，変更可能であるとする．主に要求仕様書に求められる変更可能性は以下の 3 点である。a) 目次や内容の構成，クロスリファレンスなどが整備され使いやすい構成になっている b) 重複・冗長性がない c) 各要求が他の要求と混同されておらず，独立・分離して記述されている変更可能性については，本報告「5.6.3 変更可能性」にても解説されているので参照されたい．

- 追跡可能である (Traceable): 将来の開発で改良された要求仕様書との対応や，他の開発段階における文書との対応がとれている場合，要求仕様書は追跡可能であるとする．具体的には 2 種類の追跡可能性が期待される．a) 後方追跡可能性 (Backward traceability): 要求仕様書以前の文書の中に，その要求の起源が明示的に存在すること b) 前方追跡可能性 (Forward traceability): 要求仕様書にある各要求が固有な名前か今後参照可能な番号を持っていること追跡可能性については，本報告「5.6.4 追跡可能性」にても解説されているので参照されたい．

以上が，IEEE Std830-1998 に記述されている良い要求仕様書が持っていなければならない性質である．

これらの良い要求仕様書が持っているべき性質については，その判断が “備えている”か “備えていない” の二値に限定されてしまいそうに思われる．しかし，要求工学プロセスの途中で，これら性質が段階的に充足されていくことを鑑みれば，それぞれがどの程度の達成度であるかを判定することは，十分に意味がある．

4.3　5 節「要求仕様書の項目」について

5 節では，SRS を構成する内容について，項目を挙げてそれぞれに何を記述するか説明している．この項目は模範的な SRS の章立てと考えてよい構成になっている．1 章が概要について，2 章が開発するソフトウェア全般について，3 章がすべての要求についての具体的な記述，4 章が補足である．

以下に，それぞれの項目と解説を記す．また，SRS のイメージを持ってもらうために，記述例として国際会議プログラム委員長のシステムを題材として簡易な記述を併記した箇所があるので，参考になれば幸いである．

国際会議のプログラム委員長の業務（要求工学ワーキンググループ作成）
http://www.selab.is.ritsumei.ac.jp/~ohnishi/RE/problem.html

1 はじめに (Introduction)

1.1 目的（Purpose）：
a) SRS の目的を描く
b) 意図する読者を挙げる

＜記述例＞
1.1 目的
国際会議のプログラム委員長 (PC) の業務を成功裡に進めることを補助する目的で本情報システムを構築する．この文書には，要求仕様書として本システムが満たすべき要件やその理由・目的などを記述する．よって，システムの設計者を含む開発グループおよびプログラム委員長自身が本文書の主たる読者と設定する．

1.2 対象範囲（Scope）：
a) 作成するソフトウェア・プロダクト（成果物）を名称によって定義する
b) そのソフトウェア・プロダクトがどんなもの（何をするか）であるかを説明する．もし必要なら，何をしないかについても説明する．
c) 定義されているソフトウェアのアプリケーションについて，生じる利益や目的（目標），ゴールについても含めて説明する
d) 上位レベルの定義書（システム要求定義書など）があるなら，そこの類似文章と矛盾していないこと．

＜記述例＞
1.2 対象範囲
本システムは「PC(プログラム委員長) 支援システム」と命名される．プログラム委員長が行う業務のうち，自動化可能なものは自動化し，委員長の意思決定の助けとなる情報を自動作成することを目的とする．これによって，PC 自身の仕事量の削減，および仕事の遂行時間を短縮し，参加者への応答時間も短縮することにより，効率的かつ迅速な会議運営を実現することがシステム化の最終的な目標である．対象とする範囲は以下に挙げるとおりである，

- 期間： 国際会議開催決定から，会議終了報告をホームページに掲載するまで （必要に応じてその後も利用できる）
- 対象者： 更新可能者はプログラム委員長，プログラム・コミッティー，国際会議事務局，他当システム管理者が必要と認めた人．閲覧可能者は，当システム管理者が許可した人．
- システム範囲： 他インターネットサイトとの関連はホームページ上のリンクによる。また参加費のクレジットカード決済などについては相互に定める方式に則って行われるものとし，他システムとのデータベースの共有などは行わない方針とする。個人情報の取り扱いは別に定める．

1.3 定義・用語・略語など（Definitions, acronyms, and abbreviations）:
SRS が正しく説明されるように，すべての用語，頭字語（例：SOA ;Service Oriented Architecture など），略語の定義が示される．SRS の追記 (Appendix) や，他の文書への参照でといった形式で示してもよい.

＜記述例＞
1.3 定義・用語・略語
本システムで使用する用語を以下に定義する。

- PC 支援システム: 本プロジェクトの開発対象であり，当文書で定義している情報システムの名称
- 本システム: PC 支援システムをさす.
- プログラム委員長: 国際会議についての責任者。本システムの主たる利用者
- PC: プログラム委員長（Program Chair）
- PC メンバー: プログラム委員会のメンバー
- 投稿者: 論文投稿者
- MUA: メールユーザーエージェント Outlook 等が代表的.
- MTA: メールトランスファーエージェント sendmail 等.

1.4 参照 （References）:
a) SRS で参照されるすべての文書や書籍については，完全な参照リストを提供する
b) 各ドキュメントに，タイトル,（可能なら）報告番号，日付，発行部門をつける
c) どこからその参照対象を入手できるのかという情報源を明記する

＜記述例＞
1.4 参照
本システムの支援対象となる業務の内容は以下の二つの文書をもとに遂行されている．要求仕様書は本節に記載される文書を基礎として作成される.

- 本システムに関する提案依頼書（RFP: Request for Proposal）200x 年 y 月発行 第ｎ版のもの
- IEEE による国際会議開催の手引き
  http://www.computer.org/portal/site/ieeecs/index.jsp より,
  Conferences → Conference Organizer Resources のページ.
- 要求工学ワーキンググループによる業務の手引き
  http://www.selab.is.ritsumei.ac.jp/~ohnishi/RE/problem.html

1.5 概要（Overview）

a) SRSが含む項目・事柄について記述する
b) どのようにSRSが構成されているかを説明する

2 全体像についての記述（Overall description）：
ここではプロダクトとその要求が影響を与える一般的な要素について説明しなくてはならない．このセクションでは特定の要求について述べないが，その代わりに要求の背景を説明し，SRSのセクション3において詳細に要求を定義して，理解しやすくしている．
2.1 プロダクト概観（Product perspective）：
この節には以下の制約事項を含むべきである．
2.1.1 システム・インタフェース（System interfaces）：各システムのインタフェースが挙げられ，システム要求を実現するためのソフトウェアの機能性と，システムに対応するためのインタフェース記述が定められている．
2.1.2 ユーザ・インタフェース（User interfaces）：ソフトウェア・プロダクトとユーザ間の各インタフェースの論理的な特徴．ソフトウェア要求を満たすための構成の特徴を含む（スクリーンフォーマット，ページ/ウィンドウレイアウト，ファンクションキー設定など）
a) システムのユーザに提供されるインタフェース要求のすべて．ユーザに対してシステムがどのように見えるかという観点でかかれるので，してはいけないことや，すべきことのリストになることもある．
2.1.3 ハードウェア・インタフェース（Hardware interfaces）：ソフトウェア・プロダクトとシステムのハードウェア・コンポーネントの間のインタフェースの論理的な特徴が定義される．構成の特徴（ポートの数，命令セットなど）が含まれる．また，何のデバイスがサポートされるかなども含まれる．
2.1.4 ソフトウェア・インタフェース（Software interfaces）：他のソフトウェア・プロダクトの利用や他アプリケーションシステムとのインタフェースについて定義する．他の必要なソフトウェア・プロダクトについては，以下の情報が提供されなくてはならない [名前，ニーモニック，仕様番号，バージョン番号，情報源]．各インタフェースは，このソフトウェア・プロダクト（成果物）と他のソフトウェアとの関連の必要性，メッセージ内容，およびフォーマットに関するインタフェース定義を含む．詳細な文書定義は必要ないが，インタフェース定義の文書への参照は必要である．
2.1.5 通信インタフェース（Communications interfaces）：通信インタフェース，ローカルネットワークのプロトコルなどを定義する．
2.1.6 メモリ制約（Memory constraints）：主メモリ（プライマリー）と外部記憶（セカンダリ・メモリ）の上限と適用可能な特徴を定義する．
2.1.7 ユーザ操作（Operations）：ユーザによる通常および特別なオペレーションについて定義する．
a) オペレーションモードにはどんなものがあるか
b) 直接人が対応するオペレーションの時間と，直接人が介在しないオペレーションの時間
c) データ処理サポート機能
d) バックアップ処理とリカバリーオペレーション
2.1.8 サイト適合要求（Site adaptation requirements）：
a) 定められた場所，ミッション，オペレーションモードに特有なデータや設定順序を定義する
b) ある特別な導入の際に，ソフトウェアで適用変更させなくてはならない場所やミッションに関連する特徴について，定義する．

＜記述例＞
2.1 システム概観
プラットフォームおよびオペレーティングシステム：本システムはパーソナルコンピュータ上で動作させるアプリケーションソフトウェアである．オペレーティングシステムはWindowsXP（SP 4）を基本とする。メールシステム：本システムはMUAの機能も有しており，これらを使って，PCメンバーや投稿者との情報交換を行うことができる．なおMUAは暗号化しないSMTPを使用する．
（以降 略）

2.2 プロダクトの機能（Product functions）：
SRSはソフトウェアが行う主な機能についてのサマリを提供しなくてはならない．
a) 機能のリストは，この文書を初めて読む人にとっても分かりやすい方法で構成されなくてはならない．
b) 各機能との関連を示すために文章やさまざまな図表が使われる．ただし，そのような図表はプロダクトの設計を示すものではなく，変数間の論理的な関係を示さなくてはならない．

＜記述例＞
2.2 機能概要
PC支援システムの主な機能を以下に挙げる．

- 論文募集の告知を行う
- メールで投稿された論文を収集し，投稿者に対して受諾の返事を行う．
- 事前に登録されたPCメンバーがどの論文を査読すべきかの情報を保持する．
- 査読結果を自動的に収集し，全論文の仮の順位付けを行う．
（以降 略）

2.3 ユーザの特徴（User characteristics）：
プロダクトの対象ユーザについて，教育レベルや経験や技術的な専門知識などを含んだ全般的な特徴を説明するものである．

2.4 制限事項（Constraints）：
開発者の選択を制限する事項について全般的な説明をする．
a) 規定の方針（Regulatory policies）
b) ハードウェア制限（シグナルタイミングなど）
c) 他のアプリケーションへのインタフェース
d) 並列オペレーション（Parallel operation）
e) 監査機能 (Audit functions)
f) 管理機能（Control functions）
g) 使用する開発言語
h) シグナル・ハンドシェイクプロトコル
i) 信頼性要求

j) アプリケーションの重大性（Criticality of the application）
k) 安全とセキュリティ考慮

＜記述例＞
2.4 制限事項
本システムで保有する個人情報については，該当国際会議に関する業務のみで使用し，情報へのアクセス権は必要のあるメンバーに限定される。具体的にはシステム管理者とプログラム委員長にてアクセス許可を行う．収集対象個人情報については，別途詳細に定義し保管期間も定める．使用する開発言語は，Java および C++を原則とするが，必要に応じてシステム管理者が決定する．データ・バックアップは毎日実施される．
（以降 略）

2.5 前提および依存要因（Assumptions and dependencies）：
SRS で述べられた要求に影響のある要因をリストアップしなくてはならない．それら要因はソフトウェアの設計制限ではなく，SRS の要求に変更を加える可能性のあるものである．

2.6 要求のプライオリティ（Apportioning of requirements）：
要求の順位付けを行い，当バージョンで実現する要求か，将来のバージョンで実現する要求かを明確にしなくてはならない．

3 要求仕様（Specific requirements）：
この章ではすべてのソフトウェア要求について，設計者がそれらの要求を満たすシステム設計ができるように，あるいはテスタがそれらの要求を満たしているかテストができるように，十分に詳細化した要求を含んでいなくてはならない．この章を通じて，すべての記述された要求が，ユーザ，オペレータ，あるは他の外部システムによって理解されやすいものでなくてはならない．要求は，システムへの全入力・全出力，関連する全機能を含まなくてはならない．
a) “良い要求定義書が備えるべき品質”に記述されているすべての特徴とあるそれぞれの要求は矛盾無く記述される
b) 要求は，関連する以前の文書と相互参照されている
c) すべての要求はそれぞれ固有（ユニーク）に定義できる
d) 読みやすくするために要求を再構成することには，最新の注意が必要である．

3.1 外部インタフェース（External interfaces）：
2 章で説明されているインタフェースを完全にするものであり，おなじ情報を繰り返してはならない．以下の内容とフォーマットを含む
a) アイテムの名前
b) 目的記述
c) 入力の源と，出力先
d) 有効な範囲，精度，許容範囲
e) 測定単位
f) タイミング
g) 他の入力／出力との関連
h) 画面フォーマット，構成

i) ウィンドウのフォーマット，構成
j) データフォーマット
k) コマンドのフォーマット
l) 終了メッセージ

＜記述例＞
3.1 外部インタフェース要求
3.1.1 ユーザ・インタフェース
・マウスおよびキーボードの入力によって機能を遂行する GUI を提供すること．
3.1.2 ハードウェア・インタフェース
・同システムを稼動させるコンピュータは 2 つ以上のボタンを持つマウスと，キーボードを有すること．
・同システムを稼動させるコンピュータは MTA と通信するための装置を持つこと
（以降 略）

3.2 機能要求（Functions）：
ソフトウェアが入力を処理しているときや出力を作成しているときに起きなくてはならない基本的なアクションを定義している．一般的に，“そのシステムは … すること”と表現される．
a) 入力値の妥当性チェック
b) オペレーションの正確な順序
c) 異常な状況への応答：１）オーバーフロー，２）通信手段，３）エラーハンドリングと回復
d) パラメータの影響
e) 出力から入力への関連：1）入力/出力順序，２）入力から出力への変換式

＜記述例＞
3.2 機能要求
3.2.1 モード 1: PC メンバー登録
3.2.1.1 PC メンバーの情報を登録できること．以下の 6 項目は入力必須:
名前（文字・全角・字数は 25 文字まで）
所属（文字，全角・字数は 100 文字まで，1 人 2 件まで登録）
メールアドレス（文字・半角・字数は 40 文字まで。1 人 2 件まで登録可能）
電話番号（数字，半角，30 文字まで，1 人 2 件まで登録可能）
住所（文字，全角・字数は 100 文字まで，1 人 2 件まで登録）
得意分野（文字，全角／半角，文字数無制限）のリスト
3.2.1.2 システム管理者は登録内容の更新を禁止できる．
a （以降 略）

3.3 パフォーマンス要求（Performance requirements）：
ここではソフトウェア，あるいは人間とソフトウェアのインタラクションにおける静的および動的両方の数値に関する要求を定義する．
a) サポートされる端末の数
b) サポートされる同時ユーザ数

c) 処理される情報の量とタイプ

＜記述例＞
3.3 パフォーマンス要求
3.3.1 システムはユーザがストレスを感じない範囲において反応すること.
ただし，外部の接続システムにて一定以上の時間が経過してレスポンスが無い場合は，その旨がユーザに通知されること.
3.3.2 登録可能な PC メンバーの数は 200 人以上であること.
3.3.3 扱える投稿論文数の数には上限が無いこと
（以降 略）

3.4 論理データベース要求（Logical database requirements）:
データベースで発生する情報に関しての論理的な要求を定義する.
a) 使用される情報のタイプ
b) 使用頻度
c) アクセス処理能力
d) 実体関連モデル (Entity-Relationship model)
e) 一貫性制約
f) データ保管

＜記述例＞
3.4 設計制約
・ 同システムが利用する二次記憶装置は 500GB とする.
・ 参考までに平均的な論文 1 通のファイルサイズは 200KB 前後である.
（以降 略）

3.5 設計制限事項（Design constraints）:
その他の標準やハードウェア制限による設計制限を定義する
a) 報告フォーマット
b) データ名称
c) 会計処理手順
d) 監査記録（audit tracing）

3.6 ソフトウェア・システム特性（Software system attributes）:
（詳細は第 7 章を参照のこと）

3.7 特定要求の構成（Organizing the specific requirements）:
些細なシステムでも要求はどんどん広がってしまうものである.
3.7.1 システム・モード（System mode）: オペレーションモードによりまったく異なる振る舞いをするシステムがある.（例：コントロールシステムはモードによって異なる機能セットを持つことがある）.
3.7.2 ユーザ・クラス（User class）: ユーザの異なるクラスに対応する機能セットを持つシステムがある（例：エレベータシステムは保守要員，通常の客，消防隊員などによって異なる機能で対応する）.

3.7.3 オブジェクト（Objects）：実世界のエンティティで，システム内の存在に相対するものである．
3.7.4 特徴（Feature）：外部的に期待されているサービスで，入力から期待される結果までの流れを要求するシステムによって提供される．
3.7.5 刺激（Stimulus）：刺激の点で機能を記述することで最もよく構成されるシステムがある．（例：自動飛行機着陸システム）．
3.7.6 応答（Response）：応答の生成サポートに関する全機能の記述によって最もよく構成されるシステムがある．（例：人事システム）
3.7.7 機能的階層構造（Functional hierarchy）：3.7.1 以下の説明で全機能がうまく構成できない場合，全体の機能性は共通入力，共通出力，あるいは共通内部データアクセスなどのいずれかによる機能階層で構成できるかもしれない．

3.8 追記（Additional comments）

4 補足情報（Supporting information）：
SRS は，以下を含むとさらに使いやすくなる
a) 内容のテーブル
b) インデックス
c) 付録

4.1 Table of contents and index：
非常に重要で全般的な構造化された実践に役立つ
Appendixes：
付録は SRS では常に考慮する必要があるわけではない．
a) サンプル入力/出力フォーマット，コスト分析結果，ユーザ・サーベイ結果
b) 背景情報は SRS を読むときの助けになる
c) ソフトウェアで解決されうる問題の記述
d) コードについての特別なパッケージ作成指示や，メディア（媒体）についてのセキュリティ，エクスポート，初期読み込みなどの指示．

第5章　要求仕様の品質特性の測定

5.1　要求仕様の品質特性の測定

本章では，要求仕様の品質特性の測定について，実際に例を挙げて述べる．要求仕様には，システム全体の要求仕様書（システム要求仕様書），システム中でソフトウェアで実現する部分の要求仕様書（ソフトウェア要求仕様書）があるが，ここではソフトウェア要求仕様書の品質の測定のみに限定するが，妥当性や追跡可能性など，他の該当文書を含めないと測定できないような品質もある．また，IEEE Std 830-1998[5] に準拠しているソフトウェア要求仕様書の 3 節（要求事項についての節）のみを対象とし，この節は箇条書き形式の自然言語で記述されているとする．例題として用いた要求仕様書を付録にあげる．以下の節で，品質測定の際に要求文数を数える必要があるが，簡単のため，最下位レベルの 1 箇条書き項目を 1 つの要求文と数えた．例の要求仕様書はこのやりかたで数えると全部で 66 の要求文（ただし「無し」と書かれている部分は除いた）がある．

前章で述べた IEEE Std 830-1998 にリストアップされている品質特性について計測可能な手段を与えることによりソフトウェア要求仕様書の品質特性を数量化できる．なお，ここで与える計測可能な手段 (数式) は一例に過ぎない．

5.2　妥当性

要求仕様書が妥当とは，要求仕様書中に述べられているすべての要求は，開発されるソフトウェアが満たすべきものであることを意味している．したがって，ソフトウェアが満たすべき以外の事項が書かれていれば，その仕様書の妥当性は低いことになる．妥当性の数量化は，

$$\text{妥当性} = \frac{\text{要求仕様書中でソフトウェアが真に満たすべき要求文の数}}{\text{要求文の総数}}$$

しかしながら，要求文がソフトウェアが真に満たすべき文であるかどうかを，ツールや手続きによって検証することは難しく，プロトタイプなどを用いて顧客や利用者自身によって確認してもらうことになるのが普通である．従って，なんらかの近似に基づく数量化が必要になる．

ソフトウェアの要求仕様書以外に，システム要求仕様書など，より上位の仕様書が作成される場合がある．その場合これらの文書の記載項目との比較を行い，ソフトウェア要求仕様書中の要求が，上位の仕様書のどの記述に意味的に対応づくかにより，妥当性を数量化することができる．

$$\text{妥当性} = \frac{\text{上位の文書の記載と意味的に対応づく要求文の数}}{\text{要求文の総数}}$$

このような上位の仕様書のほかにも標準規約や法令などの文書との比較も考えられる．この近似では，これらの上位の文書の記載が，正しくソフトウェアが真に満たすべき要求のみであるという仮定に基づいている．

例題で見てみると，下表にあげた要求文が妥当でないと考えられる．

妥当でない個所（項目番号，語句）	説明
3.2.4.8 PC メンバーからの苦情等のメールを自動的に破棄されることが望ましい	PC メンバーからのメールには，割り当て論文が自分が著者になっているといった正当な理由で査読できないなどのメールもあるはずである．このような場合プログラム委員長は論文割り当てをやり直さなければならない．従って，このようなメールを自動的に破棄してしまうのは，真に満たすべき要求とは考えられず妥当ではない．
3.2.5.6 返答がない査読結果のリスト	査読結果の返答がない場合，査読結果自身はないため，査読結果のリストは作れない．論理的に達成できない要求は，真に満たすべき要求とは考えられず妥当ではない．

真に満たす要求ではない文が 2 つあるので，

$$\text{妥当性} = \frac{66-2}{66} = 0.97$$

となる．

5.3　非あいまい性

要求仕様中に述べられているすべての要求が一意に解釈できる場合，要求仕様はあいまいでないといえる．もしある要求が何通りにも解釈できる場合は，その要求仕様はあいまいとなる．

あいまい性を生じさせる要因として，自然言語自身が持っている要因と，文の読み手の背景知識が異なるため読み手によって異なる複数の解釈を生じてしまう要因とがある．

要求仕様書の非あいまい性は，仕様書中のあいまいな文の出現頻度を数えることによって数量化でき，以下の式で表すことができる．

$$\text{非あいまい性} = 1 - \frac{\text{あいまいな要求文の数}}{\text{要求文の総数}}$$

どの文があいまいであるかどうかを，誰が判定しても同じという意味で客観的に，かつ正確に判定することは難しい．それは，文を解釈する人間の側に解釈に使用する知識の差があるからであり，ある人にとってはあいまいでなくても別の人にとっては複数の解釈が生じてくることもある．従って，現実的にはあいまいな文となる可能性の高い文を判別する手法を考えるのがよい．

自然言語自身が持っているあいまい性は，自然言語の文法から来るものと，語句自身の客観的な意味が不明確であることに起因するものとがあり，構文的には以下のような文があいまいな要求文となっている可能性が高いと考えられる．

1. 語句の係り受け関係が複数あるような文，つまりコンピュータで構文解析を行ったときに，構文解析木が複数得られるような文．例えば,「返答していない PC メンバーと論文リスト」といった語句があったとき,「返答していない」は「PC メンバー」のみに係るとする場合と,「PC メンバーと論文リスト」つまり「PC メンバー」,「論文」両方に係るとする場合が考えられる．前者は，まだ査読結果を「返答していない」PC メンバーおよび（返答されているいないに関わらず）論文全部のリストを表し，後者は査読結果を「返答していない」PC メンバーおよび査読結果が「返答されていない」論文のリストを指している．

2. 主語や目的語などが省略されている文．

3. 指示語を含む文．

4. 「使いやすく」,「見栄えよく」など，それを達成したかどうかを判断する際の判断基準が，読み手によって変わるような語句を含む文．このような語句には客観的な意味の定義がなく，特に非機能要求や品質要求の記述に含まれることが多い．例題では，パフォーマンス要求の項目 3.3.1 に出現している「ユーザーがストレスを感じない範囲」という語句が，ユーザーによってその時間が異なるためあいまいであると考えられる．

5. 範囲や境界を表す語が含まれる文．例えば，付録の例にはないが,「扱える PC メンバーの数は 200 人までとする」という文が含まれていた場合，扱える範囲が 200 人を含むのかどうかがあいまいになる場合がある．

これらに該当する文を数え上げ，文全体に対して占める割合によって，あいまいになる可能性を数値化することができる．もちろん，これらに該当しているからといってその文が本当にあいまいであるとは限らない．

例題でみてみよう．上記に該当する個所を以下に示す．

あいまいな個所（項目番号，語句）	説明
3.2.2.8 論文投稿のメールのようであるが	論文投稿のメールの「よう」の判断基準があいまいである．
3.2.2.9 不正な形式とはPDFファイルの破損およびアジア圏フォントの混入，著者情報のフォームの記載漏れ	接続詞「および」がどこまで係るのかがあいまい．PDFファイルが破損しておりかつアジア圏フォントが混入しているとも解釈できるし，次の「,（コンマ）」まで「および」が係っているとすると著者情報フォームの記載漏れまで含まれていなければいけないとも解釈できる．常識的には，これら3つの条件のうちどれかが満たされれば「不正な形式」とすべきである．
3.2.5.4 結果を報告したと思われるPCメンバー	「思われる」という言葉を使用しているため，結果を報告したPCメンバーとそうでないPCメンバーとの判断基準があいまいである．
3.3.1 ストレスを感じない範囲	ストレスを感じる，感じないが人によって異なるの判断基準が人によって異なるため，その判断基準があいまいである．
3.3.1 外部システムの反応遅れ	時間にしてどれだけ遅れれば，ここでの「反応遅れ」と判断されるのかの基準があいまいである．

自然言語の持つあいまいさ以外に，文の読み手の背景知識が要因となり，あいまいな意味解釈をもたらしてしまうことがある．例えば，顧客やユーザは問題領域のエキスパートであるが，要求分析者はそうではない．逆に要求分析者はコンピュータの専門家であるが，顧客・ユーザはそうではない．エキスパートが読むとあいまいではないが，その領域をよく知らないステークホルダが読むと，正確な意味がわからず，複数の解釈を行うことがある．以下の例は，文や語句としてはあいまいではないが,「プログラムの委員長業務」の文脈で複数の解釈が考えられる場合である．

あいまいな個所（項目番号，語句）	説明
3.2.6.2 返答のまだないPCメンバー	プログラム委員長業務を知らない要求分析者にとっては,「返答のまだないPCメンバー」は，全く委員長に返答を返していない，つまり１編も査読結果を返していないメンバーと解釈するであろうが，ユーザであるプログラム委員長にとっては，査読割り当て分すべてを返したPCメンバーを知ることに意味があるため，査読結果を部分的に返してはいてもまだすべてを返してはいないメンバーも含むと解釈するであろう．

以上の例では，

$$\text{非あいまい性} = 1 - \frac{3+1}{66} = 0.94$$

となる．なお，ここでは要求項目3.3.1は2つのあいまいな語句を含んでいるが，文の数ということで，1つとカウントした．

5.4 完全性

要求仕様書が完全であるとは，

- 機能，性能，設計制約，属性，外部インターフェースに関する要求はすべて記載されている．特に，システム要求で触れられている外部のシステムに対する要求はすべて記載されていなければならない．
- すべての状況において，可能な入力データすべてに対してソフトウェアがどう応答するかが記載されている．特に正当な入力値と不当な入力値の両方に対する応答が記載されていなければならない．
- 要求仕様中の図や表に対するラベルと参照，および要求仕様中の用語の定義と単位の定義が記載されている．なお，定義がないためにその解釈をめぐってあいまい性が生じることもよくある．

これらの項目の抜けの割合を計算することにより，要求仕様書の完全性は以下のように数量化できる．

$$\text{完全性} = 1 - \frac{\text{抜けがある文の数}}{\text{要求文の総数}}$$

本来は，完全性とは顧客やユーザの真のニーズが漏れなく仕様書に書かれているかどうかを示す指標と考えたほうが自然であるが，顧客やユーザ自身も自分の本当のニーズがなにかわかっていないことが多いため，直接測定することは難しい．上記のような項目が顧客やユーザのニーズをすべて表現しているととらえ，これらの記載が抜けがなく要求仕様書にあるかどうかで，間接的に測定していると考えてもよいであろう．完全性をこのように捉えると，Davis[14] も述

べているように，5.2 節の妥当性との関係を，集合の包含関係で書くと図 5.1 のようになる．図中で集合 A と B が同じ集合となる，つまり C = A = B となれば，その要求仕様書は妥当かつ完全である．A − C の部分が，顧客やユーザの真のニーズでありながら，仕様書には記載されていない部分，いわゆる「抜け」になる．B − C が，顧客やユーザのニーズではない，つまりソフトウェアが必ずしも満たさなくてもよい要求でありながら要求仕様書に記載されている部分になる．この部分が大きければ妥当性は低くなる．

例題をみてみよう．上記のような抜けを列挙すると，以下のようになる．

抜けのある個所（項目番号，語句）	説明
3.2.2.8 様式が不適切	「不適切」という言葉の定義がない．3.2.2.9 に定義がある「不正」と同義と思われる．
3.2.2.12 収集の開始前および終了後に届いたメールは自動的に破棄	3.2.2.1 および 3.2.2.2 に開始日時，終了日時の設定の記載があるが，日時を設定する前にメールが届いた場合や，設定後にさらに日時を変更し設定しなおした場合にメールが届いた場合にどうするかの記載がない．例えば，終了日を当初 6 月 14 日にしていたとする．15 日に届いたメールは破棄される．その後終了日を 17 日に設定しなおしたとすると，16 日に届いたメールは受理されるが以前破棄された 15 日のメールはどうするのかの記載がない．これは，ある入力データに対してのソフトウェアの応答に記載がないことに該当する．
3.2.4.2 自由に対応付けられること	査読者自身がその論文の著者になっているなど，conflict を起こしている場合の対応付けをどうするかの記述がない．
3.2.4.6 PC メンバーに担当論文を電子メールで自動送付	PC メンバーへの電子メール送付が，アドレスが間違っているなどの理由で失敗した場合の措置に関する記述がない
3.3.1 反応遅れ	「反応遅れ」とは何ミリセカンドなのかという具体的な定義がない．この部分は定義がないため，あいまい性も引き起こしている．

要求仕様書に，まだ決まっていないあるいは明確になっていない事項を表すために，TBD(To Be Determined) という語句を使用することがある．これは，その要求項目の存在には分析者は気づいているという点で「抜け」よりは好ましいが，完全ではない．例題では項目 3.2.3.4 が TBD となっているため，これを加味すると，完全性は

$$\text{完全性} = 1 - \frac{5+1}{66} = 0.91$$

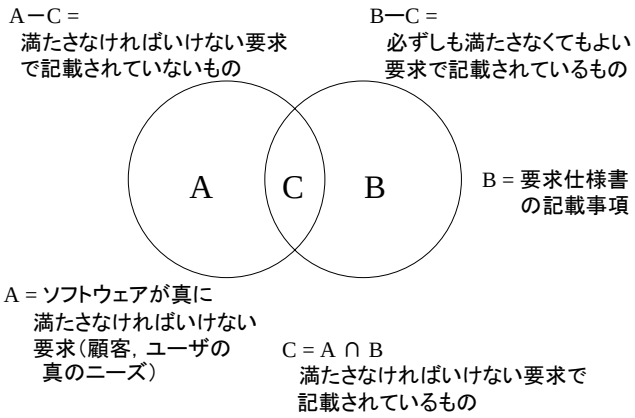

図 5.1: 妥当性と完全性との関係

5.5 無矛盾性

無矛盾性は要求仕様の中で一貫していること，つまり個々の要求が互いに矛盾しないことを表している．仕様書に含まれる矛盾としては以下のようなものが考えられる．

1. ソフトウェアの動作が矛盾
   同じ入力に対してのソフトウェアの振る舞いや出力が，複数の個所に記述されており，しかもそれらが異なっている．

2. 定義が矛盾
   語句の定義やその説明が複数箇所でなされているが，その内容が食い違っている．

3. 制約が矛盾
   制約が複数書かれているが，それらを満足する状況が存在しない

矛盾性は，以下のように数量化できる．

$$\text{無矛盾性} = 1 - \frac{\text{矛盾に関わった要求文の数}}{\text{要求文の総数}}$$

例題では，3.4 設計制約で「同システムが利用する二次記憶装置は 500GB とする」として，記憶領域に上限値を設定しているが，3.3.3「扱える投稿論文数の数には上限がないこと」と矛盾している．つまりこの 2 つの制約を同時に満足する状況は論理的には存在しない．従って，

$$\text{無矛盾性} = 1 - \frac{2}{66} = 0.97$$

となる．

5.6 重要度と安定性のランク付け

個々の要求に重要度や安定性を示す識別子がある場合に，要求仕様書は重要度と安定性のランク付けがされていることを意味する．一般に要求全てが同一の重要度を持つ訳ではなく，重要度

のランク付けは「必須な要求」か「条件つき要求」か「あってもなくてもよい要求」といったようにランクが付けられていることを意味する．ISO/IEC 12207-1995 Software Lifecycle Process, JIS X0160-1995 ソフトウェアライフサイクルによれば,

1. 必須な要求は次のような表現で終わる．(英語では shall,will が用いられる)

   「する」,「とする」,「による」,「すること」,「（の）こと」,「（し）なければならない」,「とおりとする」

2. 推奨を表現する要求は次のような表現で終わる．(英語では should が用いられる)

   「することが望ましい」,「ほうがよい」,「するのがよい」,「するとよい」

3. 限度内で許されることを表現する要求は次のような表現で終わる．(英語では may が用いられる)

   「（し）てもよい」,「差し支えない」

要求文をこれらの 3 種類のいずれかに分類した場合に，それぞれに分類された要求が上記の表現をとっているか否かで重要度のランク付けの数量化が可能となる．以下のような数量化が考えられる．

$$\text{重要度のランク付け度} = \frac{\text{重要度のランク付けが正しく表現された要求文数}}{\text{重要度のランク付けが必要な全要求文数}}$$

また，安定性は，例えば以下のように評価ができる．

$$\text{安定性のランク付け度} = \frac{\text{安定性が正しく表現された要求文数}}{\text{変更される可能性のある全要求文数}}$$

といった式が考えられる．

付録にある共通例題とした要求では,「論文リストを E-mail もしくは FTP でプログラム委員全員に配布」とあり，この要求が仕様書では欠落している．この文の替わりに「論文リストを E-mail で配布すること．E-mail が使えない場合は FTP で配布できることが望ましい」といった文を正解とする．この例を含めて下表にあげる要求文について誤ったランク付けがある．

誤ったランク付け（項目番号，語句）	説明
仕様書中に抜け	「論文リストを E-mail で配布すること．E-mail が使えない場合は FTP で配布できることが望ましい」を正解とすると，ランク付けされた文が欠落していることになる．
3.2.1.3 登録内容の更新を禁止できること	PC メンバーを更新できないのはおかしいのでは？
3.2.1.4 可能ならば登録内容の更新を禁止を解除できること	国文法を逸脱しているだけでなく，そもそも上記のように更新の禁止自体がおかしな要求なのでランク付けの誤りと判断する．
3.2.4.8 PC メンバーからの苦情等のメールを自動的に破棄されることが望ましい	この文も文法上おかしい上に，意味的にもおかしい．これもランク付けの誤りと判断する．

以上，4 文が誤った重要度のランク付けと判断でき，正しいランク付けはないところから

$$\text{重要度のランク付け度} = \frac{0}{4}$$

となる．

5.7 検証可能性

要求仕様書が検証可能とは「ソフトウェア製品がその要求を満たしていることを計算機や人手によって，有限の費用でチェックできるプロセスが存在すること」を意味する.「うまく」や「しばしば」といった定性的な表現があると検証ができないので，

$$\text{検証可能性} = 1 - \frac{\text{定性的な表現箇所の数}}{\text{定量的に表現すべき箇所の数}}$$

といった式で検証可能性を表すことができる．

例題の要求仕様書では，下表に示した最初の 2 つの要求文に問題がある．

検証に関連する個所（項目番号，語句）	説明
3.3.1 ユーザーがストレスを感じない範囲...，外部システムの反応遅れ	これらは定性的な表現であることから検証可能性を損ねている．
3.3.3 論文数には上限が無いこと	これは検証不可能な表現である．
3.3.2 PC メンバーの数は 200 人以上であること	200 人を超えてもシステムが正しく動作するかどうかを確認すれば良いので検証可能である．

以上から

$$\text{検証可能性} = 1 - \frac{3}{4}$$

となる．また，あいまいな要求は検証可能性が低いと見なすことができる [14] ので，非あいまい性でもって検証可能性を表すことができる．検証するのが困難な要求，例えば「放射能漏れが発生した場合に，本システムは半径 20km 以内の人間の致死率を 80 %以内に抑えること」といった要求はテストできない [14]．Davis 等は以下の式を検証可能性として与えている．この式は単位が与えられていないのであいまいだが，検証のコストや時間がかかるほど 0 に近づく．

$$\text{検証可能性} = \frac{\text{全要求文数}}{\text{全要求文数} + \text{要求検証のための総コスト} + \text{要求検証のための総時間}}$$

5.8　変更可能性

要求仕様書の構造やスタイルを保持したまま，任意の要求を容易に，完全に，矛盾なく変更できる場合，要求仕様書は変更可能という．変更可能であるためには

- 要求仕様書に，目次，索引，クロスリファレンスが付けられていること．
- 要求が冗長でないこと．つまり同じ要求が 2 箇所以上に表れないこと．
- 要求が互いに依存しないこと．
- 複雑な要求を 1 つの要求として表現せずに，個々の要求を分離して別々に表現すること．

が望ましい．冗長な要求文の数や互いに依存する要求文の数を使って数量化すると，それぞれ以下のようになる．

$$\text{変更可能性} = 1 - \frac{\text{冗長な要求文の数}}{\text{全要求文数}}$$

$$\text{変更可能性} = 1 - \frac{\text{依存する要求文の数}}{\text{全要求文数}}$$

同様に個々の要求を 1 つの要求として表現しているかどうかの度合を用いて以下のように数量化できる．

$$\text{変更可能性} = 1 - \frac{\text{複数の要求を 1 つの要求文とした文数}}{\text{全要求文数}}$$

Davis 等は目次と索引が要求仕様書に付けられていることが変更可能性に大きく影響を与えると考え，目次と索引が用意されている場合を 1，そうでない場合を 0 とする指標を示している [14].

例題とした要求仕様書では 3 章の詳細な要求文間で依存する文が散見される．

依存する個所（項目番号，語句）	説明
3.2.1.2 必ず 200 人以上の PC メンバーを登録できること 未査読の論文に対する要求文 3.2.5.6「締め切り後にもかかわらず返答がない査読結果のリストをテキストもしくは CSV ファイル形式で出力できること」 3.2.4.7「3.2.4.6 と同様に」という表現	この文と 3.3.2「PC メンバーの数は 200 人以上であること」は互いに依存している． 左の文と 3.2.6.2「返答していない論文のリストを出力できること」も依存している． 同じように他の要求文の識別番号を明示した要求文が全部で 4 文存在しているがこれらは互いに依存している．

以上から機能要求と性能要求文の 66 文中 6 文が依存すると見なすと

$$\text{変更可能性} = \frac{39}{66}$$

となる．

5.9 追跡可能性

追跡可能性は次の 2 種類に分けられる．

1. 後方追跡可能性：各要求から，要求仕様書に先だって作成された文書中の各要求の起源について書かれた箇所を参照できること．
2. 前方追跡可能性：各要求が名前と参照番号を有し，要求仕様書を元にして作成された全ての文書（例えば設計仕様書やソースコード）から参照できること．

従って，要求仕様書単独では定義できないが，要求文ごとに追跡可能かどうかを数えることによって数量化する．

$$\text{・前方追跡可能性} = \frac{\text{名前と参照番号を持った要求文数}}{\text{全要求文数}}$$

$$\text{・後方追跡可能性} = \frac{\text{要求の起源を参照可能な要求文数}}{\text{全要求文数}}$$

$$\text{・前方追跡可能性} = \frac{\text{要求仕様書を基に作られた文書から参照可能な要求文数}}{\text{全要求文数}}$$

Davis 等は (1) 章節番号を付与していること，(2) 段落ごとに 1 要求のみが書かれていること，(3) 各要求にユニークな番号が振られていることを，追跡可能性を向上させる要因として，これらの達成度を追跡可能性の指標としている [14].

例題の要求仕様書では

- 目次やクロスリファレンスは用意されていないが，3.2 節の機能要求と 3.3 節の性能要求に対しては要求文単位に節番号が振られている．これは全要求に識別番号が振られていると判断できる．

- しかしながら 3.2.1.4 は二つあるので識別不可能であることから，全 66 文中 43 文が識別可能と判断できる．

以上から

$$\text{前方追跡可能性} = \frac{43}{66}$$

となる．

第6章　要求仕様書から予測されるソフトウェアの品質特性

ソフトウェアの品質に関しては，ISO/IEC9126 (JIS X 0129-1994) [85] において品質特性が定義されている．ソフトウェアにおいて，ISO/IEC9126 のある特定の品質特性を高めるためには，ソフトウェア開発の要求定義，設計，実装，テスト，保守の各段階で成果物の品質を高めることが必要であると考えられる．そこで，本ワーキンググループでは，要求定義段階の成果物である要求仕様書の品質と，最終成果物であるソフトウェアの品質にどのような相関関係が存在するかを議論することとした．

要求仕様書の品質とソフトウェアの品質の相関関係が明らかになることで，以下の効果が考えられる．

- ソフトウェアのある品質特性を高めることを考えた場合，要求仕様書のどのような品質特性に注意すればよいかがわかる
- 要求仕様書のある品質特性が高いにもかかわらず，その品質特性と相関のあるソフトウェアの品質特性が低い場合は，設計段階以降のプロセスに何らかの問題があることが推測できる
- 要求仕様書から最終製品の品質を予測することができる

要求仕様書の品質としては，IEEE Std 830-1998 において品質特性が定義されている．そこで本ワーキンググループでは，この IEEE Std 830-1998 の要求仕様書の品質特性に着目し，要求仕様書から IEEE Std 830-1998 の品質特性項目が測定されたとして，その結果から ISO/IEC9126 品質特性を予測することを考える．

6.1　相関表の作成

ここでは，要求仕様書どおりにソフトウェアが作成されるものと仮定して議論を行った．この議論のために，IEEE Std 830-1998 の要求仕様書の品質特性を縦軸 (行項目) に，ISO/IEC9126 のソフトウェア品質特性を横軸 (列項目) にとり，セル (i, j) に要求仕様書品質特性 i がソフトウェア品質特性 j に影響を与える度合いを記入する相関表を用意した．相関関係の評価は，相関の強いものから，相関があるとは考えられないものまでの 4 段階で行うことにした．

例えば，要求仕様書の妥当性が向上すれば，最終製品の合目的性も向上すると考えた場合には,「○：相関がある」が記入されている．また，要求仕様書の追跡可能性が向上すれば必ず最終製品の合目的性も向上するとはいえない (つまり，相関はない) が，合目的性の計測はしやすくなる．このように評価として相関以外の関係がある場合は，それを具体的に記入することに

した．表 6.1 に相関表の例を示す.「◎」が「強い相関がある」,「○」が「相関がある」,「△」が「弱い相関がある」,「なし」が「相関がない」を表している．表 6.1 では，ソフトウェアの品質特性の「合目的性」については，要求仕様書の品質特性の「妥当性」と「非あいまい性」と「完全性」とは「相関がある」(○),「一貫性」と「検証可能性」とは「弱い相関がある」(△),「ランク付けされている」とは「強い相関がある」(◎)，という結論が得られたことを意味する．ただし，表 6.1 は記入例を示しているものであり，議論の結果を表すものではない.

表 6.1: 要求仕様書とソフトウェアの品質特性の相関関係記入例

	機能性					
	合目的性	正確性	相互運用性	標準適合性	セキュリティ
妥当性	○	○	△	△	○
非あいまい性	○	◎	△	△	○
完全性	○	◎	△	△	○
一貫性	△	△	△	△	○
ランク付けされている	◎	なし	なし	なし	なし
検証可能性	△	○	△	△	△
変更可能性	計測しやすい	計測しやすい	計測しやすい	計測しやすい	計測しやすい
追跡可能性	計測しやすい	計測しやすい	計測しやすい	計測しやすい	計測しやすい

最終の相関表の作成においては，参加者が個別に作成した相関表や，6.2 で説明する平均値 (表 6.2) をもとに，各自の「◎」,「○」,「△」,「なし」の評価の理由を議論し，調整を行った.

「計測しやすい」とは，相関はないと考えられるが，全く相関がないということではなく，計測はしやすくなる，という意味である．この議論の中で，要求仕様書の品質特性とソフトウェアの品質特性の間には，ある特定の値域のみ相関が見られるものがある，という仮説が得られた．また，個々のソフトウェアの品質特性と要求仕様書の品質特性との間の相関関係に，他の品質特性が影響する，ということが考えられる．例えば，ソフトウェアの品質特性の「合目的性」と要求仕様の品質特性の「一貫性」との相関については,「妥当性」と「非あいまい性」と「完全性」の 3 つと強い結びつきがあり，これら 3 つの品質が高い場合,「合目的性」と「一貫性」との正の相関が強くなる，などである．

図 6.1 のグラフは，横軸を要求仕様書の品質，縦軸をソフトウェアの品質とし，要求仕様書の品質が向上するとソフトウェアの品質はどのようになるか，ということを表したものである．図 6.1 の左側のグラフは，要求仕様書の品質特性が向上すれば，ソフトウェアの品質特性も向上する，という相関がみられるものであり，例えば要求仕様書の品質特性の「妥当性」とソフトウェアの品質特性の「合目的性」などにおける関係として，我々が想定したものである．図 6.1 の右側のグラフは，要求仕様書の品質特性が向上すると，ソフトウェアの品質特性もある特定のレベルまでは向上するが，その後，要求仕様書の品質特性が向上しても，必ずしもソフトウェアの品質特性は向上せず，相関が見られなくなる，というものであり，例えば要求仕様書の品質特性の「一貫性」とソフトウェアの品質特性の「合目的性」などにおける関係として想定したものである．図 6.1 の右側のような品質特性間の相関関係の評価では，結びつきの強い他の品質特性は高いものとする，などの前提を設け，その特定の前提をもとに議論を行った．その結果，参加者の賛否の意見が分かれることがあり，議論に多くの時間を要した．このような品質特性間の評価は「△：相関がないとはいえない」とし，どのような仮定のもとで相関関係が見られるかを議論の中で明らかにして，評価値とともに補足として記録した．

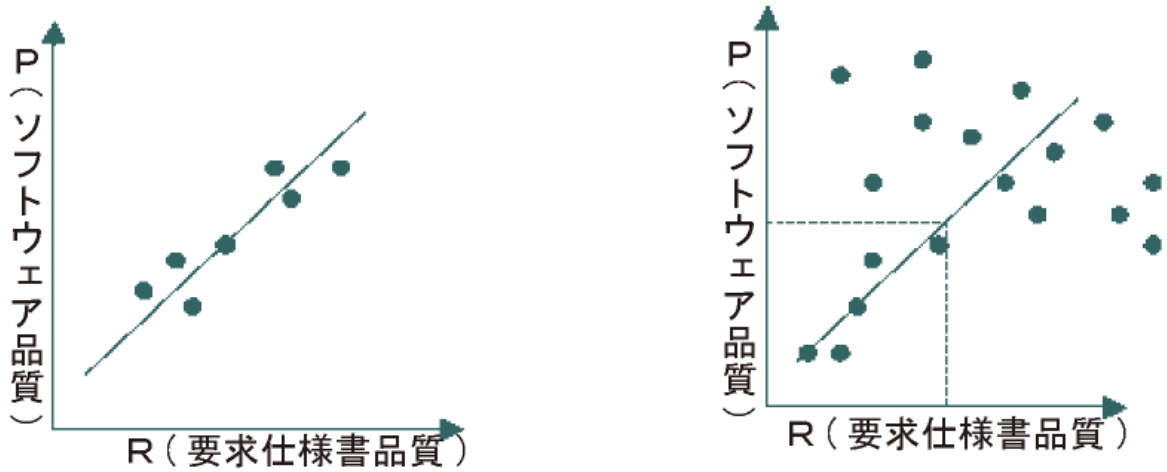

図 6.1: 要求仕様書とソフトウェアの品質の相関

6.2　一般的な相関表

相関表の作成にあたり，まず，特定のシステムに限定しないで議論を行った．この際に作成された相関表を表 6.2 に示す．この表は，参加者が個別に作成した相関表について,「◎：強い相関がある」を 3 点,「○：相関がある」を 2 点,「△：弱い相関がある」を 1 点,「なし：相関がない」を 0 点として平均値を取ってまとめたものである．従って，各セルの評価値が高ければ，それだけ品質特性間の相関関係が強いということである．例えばこの表において，ソフトウェアの品質特性の「合目的性」については，要求仕様書の品質特性の「妥当性」の評価値が 2.17 で，やや強い相関がある，と多くの人が考えていたと言える．また,「ランク付けされている」の 1.17 のような，3 点と 0 点との中間の値となっている評価値は，やや相関があると多くの人が考えていたか，相関の評価が人によって差があったものであり,「変更可能性」の評価値が 0.33 で，相関はほとんどない，と多くの人が考えていたと言える．

そして，この表 6.2 をもとに議論を繰り返し,「◎」,「○」,「△」,「なし」の評価をしなおした．その結果を表 6.3 に示す．なお，この表のセル中の () 内の番号は，6.2 に述べる補足の () 内の番号と対応する．

表 6.2: 要求仕様書とソフトウェアの品質特性の相関関係の討論参加者の平均値

	機能性				
	合目的性	正確性	相互運用性	標準適合性	セキュリティ
妥当性	2.17	1.75	0.83	0.75	1.08
非あいまい性	2.00	1.92	1.08	1.00	1.25
完全性	2.25	1.92	1.08	1.08	1.75
一貫性	1.67	1.67	0.83	0.75	0.92
ランク付けされている	1.17	0.17	0.33	0.17	0.42
検証可能性	1.25	1.17	0.75	0.67	0.83
変更可能性	0.33	0.33	0.42	0.42	0.33
追跡可能性	0.56	0.42	0.33	0.25	0.25

	信頼性			使用性		
	成熟性	障害許容性	回復性	理解性	習得性	運用性
妥当性	0.42	0.25	0.25	1.25	0.75	0.75
非あいまい性	0.92	0.67	0.50	1.08	0.83	0.75
完全性	1.42	1.75	1.00	0.67	0.50	0.67
一貫性	1.00	0.50	0.42	0.67	0.75	0.67
ランク付けされている	0.17	0.33	0.25	0.33	0.42	0.17
検証可能性	1.17	1.17	1.00	0.67	0.25	0.25
変更可能性	0.50	0.25	0.42	0.33	0.50	0.50
追跡可能性	0.33	0.25	0.42	0.50	0.25	0.25

	効率性		保守性			
	時間効率性	資源効率性	解析性	変更性	安定性	試験性
妥当性	0.75	0.67	0.67	0.42	0.50	0.75
非あいまい性	0.83	0.67	0.92	1.00	1.08	1.25
完全性	0.75	0.58	0.75	1.00	1.08	1.00
一貫性	0.42	0.42	1.17	1.25	1.25	1.08
ランク付けされている	0.50	0.33	0.25	0.08	0.17	0.67
検証可能性	0.50	0.42	1.25	1.33	1.50	2.25
変更可能性	0.33	0.33	0.83	1.83	0.83	0.58
追跡可能性	0.25	0.25	2.17	1.92	1.42	1.92

	移植性			
	環境適用性	設置性	規格適用性	置換性
妥当性	0.58	0.50	0.67	0.50
非あいまい性	0.42	0.42	0.42	0.50
完全性	0.75	0.67	0.75	0.67
一貫性	0.42	0.42	0.42	0.42
ランク付けされている	0.25	0.08	0.08	0.17
検証可能性	0.08	0.25	0.42	0.17
変更可能性	0.58	0.67	0.33	0.42
追跡可能性	0.58	0.42	0.42	0.67

表 6.3: 要求仕様書とソフトウェアの品質特性の相関関係の議論結果

	機能性				
	合目的性	正確性	相互運用性	標準適合性	セキュリティ
妥当性	○	◎	△ (*9)	△ (*12)	○ (*15)
非あいまい性	○	◎	△	△	○
完全性	○	◎	△	△	○ (*16)
一貫性	△ (*1)	△ (*6)	△	△	○
ランク付けされている	◎ (*2)	なし (*7)	なし (*10)	なし (*13)	なし (*17)
検証可能性	△ (*3)	○ (*8)	△ (*11)	△ (*14)	△ (*18)
変更可能性	計測しやすい (*4)	計測しやすい	計測しやすい	計測しやすい	計測しやすい
追跡可能性	計測しやすい (*5)	計測しやすい	計測しやすい	計測しやすい	計測しやすい

	信頼性		
	成熟性	障害許容性	回復性
妥当性	なし	なし	なし
非あいまい性	△ (*19)	△ (*19)	△ (*19)
完全性	○ (*20)	○ (*20)	○ (*20)
一貫性	△ (*21)	△ (*21)	△ (*21)
ランク付けされている	なし (*22)	なし (*22)	なし (*22)
検証可能性	△ (*23)	△ (*23)	△ (*23)
変更可能性	計測しやすい (*24)	計測しやすい (*24)	計測しやすい (*24)
追跡可能性	なし (*25)	なし (*25)	なし (*25)

	使用性		
	理解性	習得性	運用性
妥当性	1.25(保留)(*26)	1.25(保留)(*26)	1.25(保留)(*26)
非あいまい性	なし (*27)	なし (*27)	なし (*27)
完全性	○ (*28)	○ (*28)	○ (*28)
一貫性	△ (*29)	△ (*29)	△ (*29)
ランク付けされている	なし (*30)	なし (*30)	なし (*30)
検証可能性	△ (*31)	△ (*31)	△ (*31)
変更可能性	なし	なし	なし
追跡可能性	△ (*32)	△ (*32)	△ (*32)

	効率性		保守性			
	時間効率性	資源効率性	解析性	変更性	安定性	試験性
妥当性	なし	なし	△ (*36)	△ (*41)	△	△ (*53)
非あいまい性	△	△	△	△	○ (*48)	◎
完全性	○ (*33)	○ (*33)	△ (*37)	△ (*42)	○ (*49)	なし (*54)
一貫性	△	△	△ (*38)	△ (*43)	○ (*50)	○
ランク付けされている	なし (*34)	なし (*34)	△ (*39)	なし・△ (*44)	なし (*51)	△ (*55)
検証可能性	△	△	○	なし・○ (*45)	△ (*52)	◎
変更可能性	計測しやすい	計測しやすい	△ (*40)	なし・○ (*46)	なし	なし
追跡可能性	なし (*35)	なし (*35)	◎	○・なし (*47)	◎	◎ (*56)

	移植性			
	環境適用性	設置性	規格適用性	置換性
妥当性	なし	なし	なし	なし
非あいまい性	なし	なし	なし	なし
完全性	なし (*57)	なし (*57)	なし (*57)	なし (*57)
一貫性	なし	なし	なし	なし
ランク付けされている	なし (*58)	なし (*58)	なし (*58)	なし (*58)
検証可能性	なし	なし	なし	なし
変更可能性	なし	なし	なし	なし
追跡可能性	なし	なし	なし	なし

また，表 6.3 の評価以外にも，各項目間の相関関係の評価を確定するにあたり，その根拠となった情報や，評価の際の仮定や制限事項，参加者の意見が分かれた場合の各々の意見などの補足を併記し，本表の利用者に対し十分な情報を提供できるよう考慮している．その補足をソフトウェアの品質特性ごとに下記に示す．例えば，ソフトウェアの品質特性の「機能性」のうちで，副特性の「合目的性」に対して，要求仕様書の「一貫性」と「ランク付けされている」,「検証可能性」,「変更可能性」,「追跡可能性」との関係に関して，相関表の作成時の議論の結果，補足が併記されている．これらの補足を以下に示す．以下の補足において，要求仕様書の品質特性の () の番号は，表 6.3 のセルに付記されている () の番号と対応する．また，(意見) と書かれているものは，参加者の一部から出た意見である．

(1) 機能性

ソフトウェアの品質特性の「機能性」では，全ての副特性に対して補足が併記されている．機能性のうち，正確性に関しては，技術的要素 (設計) が強くかかわる特性である，という補足が付いている．

(1.1) 合目的性

- (*1) 一貫性
  - 妥当性，非あいまい性，完全性の 3 つと結びつきが強く，これらが高い場合，一貫性は正の相関が強くなる．
  - 一貫性が低いと合目的性が低くなる可能性がある．
- (*2) ランク付けされている
  - 問題の仮定の解釈で，書かれたことが全部作られると解釈した場合は関連がないが，機能を取捨選択を行って作るという場合には，強い相関がある．
- (*3) 検証可能性
  - 検証後に修正をすると言う仮定を置けば，相関がある．
  - 計測がしやすくなるだけで相関はない．
- (*4) 変更可能性 (計測しやすい)
  - 変更可能性の定義によれば，これが高いとわかりやすい仕様書になることが期待できる．このため，合目的性を計測しやすくなると言う意味である．
- (*5) 追跡可能性 (計測しやすい)
  - 要求仕様書の内容が最終製品へ追跡ができれば，合目的性の検証がやりやすくなると言う意味である．

(1.2) 正確性

- (*6) 一貫性
  - 少なくとも，一貫性が低い場合は，正確性が低くなるといえる．
  - (意見) 一貫性と合目的性の議論と同様で，妥当性，非あいまい性，完全性を一緒に考えるほうがいいのではないか．
- (*7) ランク付けされている
  - 正確性に関する要求項目にランク付けがあった場合は,「ランク付けされている」は正確性に関連する．
- (*8) 検証可能性
  - 計測しやすい．
  - 合目的性よりは判断しやすいから○である．

(1.3) 相互運用性

- (*9) 妥当性
  - IEEE Std 830-1998 の 5.2 章に相互運用性に関して記述するところがあり，これらは，妥当性から一貫性がすべて高ければ相互運用性が高いと言えるだろう．ただし，非機能要求なので，設計やアーキテクチャに依存するところが多い．
- (*10) ランク付けされている
  - 正確性に関する要求項目にランク付けがあった場合は,「ランク付けされている」は正確性に関連する．
- (*11) 検証可能性
  - 合目的性の場合と同じく検証しやすいため△である．

(1.4) 標準適合性

- (*12) 妥当性
  - IEEE Std 830-1998 の 5．2 章に相互運用性に関して記述するところがあり，これらは，妥当性から一貫性がすべて高ければ相互運用性が高いと言えるだろう．ただし，非機能要求なので，設計やアーキテクチャに依存するところが多い．
- (*13) ランク付けされている
  - 正確性に関する要求項目にランク付けがあった場合は,「ランク付けされている」は正確性に関連する．

- (*14) 検証可能性
  - 合目的性の場合と同じく検証しやすいため△である．

(1.5) セキュリティ

- (*15) 妥当性
  - セキュリティは，すべてのモジュールに関係する (一方，相互運用性，標準適合性は一部のモジュールにのみ関係する)．このため，妥当性は，相互運用性や標準適合性よりも強くセキュリティに関連する．
- (*16) 完全性
  - セキュリティは，すべてのモジュールに関係する (一方，相互運用性，標準適合性は一部のモジュールにのみ関係する)．このため，完全性は，セキュリティに強く関連する．
- (*17) ランク付けされている
  - 正確性に関する要求項目にランク付けがあった場合は,「ランク付けされている」はセキュリティに関連する．
- (*18) 検証可能性
  - 合目的性の場合と同じく検証しやすいため△である．

(2) 信頼性

ソフトウェアの品質特性の「信頼性」では，全ての副特性に対して同じ補足が併記されている．この品質特性自体には，設計や実装など後段階の影響が大きい，機能性の合目的性などの評価値とは意味が異なる，などの補足が付いている．

(2.1) 成熟性，(2.2) 障害許容性，(2.3) 回復性

- (*19) 非あいまい性
  - 妥当性よりはやや関連があると言うことで△とした．
- (*20) 完全性
  - 例外事象の場合の処理が十分に書かれているため○とした．
  - IEEE Std 830-1998 に，成熟性や障害許容性，回復性についての記述項目がある．
- (*21) 一貫性
  - 完全性が高い場合，一貫性は成熟性と相関をもつ．

- (*22) ランク付けされている
  - (意見) 例外処理に頻度による優先度がついていると，優先度順にテストすることができるため成熟性が高くなる．
- (*23) 検証可能性
  - 検証可能性が高ければ成熟性 (障害許容性または回復性) が計測しやすく，それを計測して成熟性が悪ければ修正するはずである．
- (*24) 変更可能性 (計測しやすい)
  - 直接的な関連はない．
  - (意見) ただし，変更可能性が高ければ成熟性が低いときにそれを修正することが容易になるだろう．
- (*25) 追跡可能性
  - 追跡可能性が高ければ成熟性が低いときにそれを修正するときに貢献する (どの部分を修正すればいいかがわかりやすい)．

(3) 使用性

ソフトウェアの品質特性の「使用性」では，全ての副特性に対して補足が併記されている．使用性については，理解性，習得性，運用性の違いが，要求仕様書からみるとない，という補足が付いている．

(3.1) 理解性，(3.2) 習得性，(3.3) 運用性

- (*26) 妥当性
  - (意見) 正当でない要求が入っている場合は，理解性が落ちるだろう．
  - (意見) 利用性は，設計にかかわるところが多く,「要求」の寄与する割合は低い．
    (意見の合意が得られなかったので，2 つの意見を両論併記)
- (*27) 非あいまい性
  - ただし，仕様書のあいまい性が，製品マニュアルに継承されてしまう可能性がある．
- (*28) 完全性
  - (意見) 要求として (例えばエンドユーザが) 理解しやすいようにするための指定をした場合は，そのような要求がもれなく記述されていないと理解性が低くなる．
- (*29) 一貫性
  - 完全性が高い場合に一貫性の相関が強くなる．

- (*30) ランク付けされている
  - 理解性に関する要求記述にランク付けがなされていれば，理解性が高くなる．
- (*31) 検証可能性
  - 検証可能性が高いと検証者にとってわかりやすいはずである．検証者の中にはユーザもいる．
- (*32) 追跡可能性
  - 要求から追跡が取れるソフトウェアの使用性は，高くなると期待できる．
  - (習得性のみの補足) 仕様書から操作マニアルが作られる場合は，追跡可能性が高ければよい操作マニュアルができることになるため，高くなる可能性がある．

(4) 効率性

ソフトウェアの品質特性の「効率性」では，全ての副特性に対して同じ補足が併記されている．効率性については，信頼性の議論と原則的に同様，という補足が付いている．

(4.1) 時間効率性，(4.2) 資源効率性

- (*33) 完全性
  - 反応時間に関する記述等がちゃんと書かれていれば上がる．
- (*34) ランク付けされている
  - (意見) 時間効率性に関する記述が高くランク付けされていれば上がるのではないか．
- (*35) 追跡可能性
  - 完全性が高い，つまり時間効率性の要求がちゃんと書かれていれば，要求から追跡でき, 向上しやすい．

(5) 保守性

ソフトウェアの品質特性の「保守性」では，全ての副特性に対して補足が併記されている．保守性のうち，解析性では，要求仕様書を解析に使うかどうかで変わる，という補足が付いている．

(5.1) 解析性

- (*36) 妥当性
  - 妥当性が低ければ余計な機能が入っている．それが解析をしにくくする可能性がある．妥当性が高くても解析性が高いとはいえない．

- (*37) 完全性
  - 抜けている要求がバグになって入る可能性があるため，もともとない要求をベースに解析しなければならないため，解析性が低くなる可能性がある．
- (*38) 一貫性
  - (意見) 一貫性があるから解析ができるのではないか (○: 相関がある).
  - (意見) 一貫性があるからといって，見つけやすいことにはつながらない．ただし，追跡可能性が高ければ, 見つけやすいので解析がしやすい (相関なし).
    (意見の合意が得られなかったので，2 つの意見を両論併記)
- (*39) ランク付けされている
  - 重要度のランク付けだと相関はないが，安定度のランク付けがあると解析のための同定は容易になる．
- (*40) 変更可能性
  - 追跡可能性が高いと相関はある．

(5.2) 変更性

- (*41) 妥当性
  - (意見) 解析性と同じではないか.
  - (意見) 解析性を含まないと解釈するべきではないか.
  - 余計なものがあると変更の手間は増える．
- (*42) 完全性
  - (意見) 解析性と同じではないか.
  - (意見) 解析性を含まないと解釈するべきではないか.
  - 余計なものがあると変更の手間は増える．
- (*43) 一貫性
  - (意見) 一貫性がとれていても変更性は高まらないだろう.
- (*44) ランク付けされている (なし・△)
  - 訂正に場所の同定を含めるなら△，そうでないなら「なし」である．
- (*45) 検証可能性 (なし・○)
  - 訂正に場所の同定を含めるなら△，そうでないなら「なし」である．
- (*46) 変更可能性 (なし・○)

  - 本当に変更しやすいかは作りの問題．要求仕様が変更可能だときちんと作られた最終製品の変更性も高くなるとはいえない．
  - 訂正に場所の同定を含めるなら△，そうでないなら「なし」である．

- (*47) 追跡可能性 (○・なし)
  - 訂正に場所の同定を含めるなら△，そうでないなら「なし」である．

(5.3) 安定性

- (*48) 非あいまい性
  - (意見) 修正する箇所があいまいだとすると安定でなくなるから相関はある．前提条件が常に成立するとは限らないから○ではないか．
- (*49) 完全性
  - (意見) 修正する箇所があいまいだとすると安定でなくなるから相関はある．前提条件が常に成立するとは限らないから○ではないか．
- (*50) 一貫性
  - (意見) 修正する箇所があいまいだとすると安定でなくなるから相関はある．前提条件が常に成立するとは限らないから○ではないか．
- (*51) ランク付けされている
  - (意見) 変更性に対するランク付けがあるので，それが付けられていないと適切な設計がされないだろう．その結果安定性が得られないのではないか．
- (*52) 検証可能性
  - 検証可能ならば安定性を検証しやすい．

(5.4) 試験性

- (*53) 妥当性
  - 妥当性が低ければ余計な機能が入っている．それが解析をしにくくする可能性がある．妥当性が高くても解析性が高いとはいえない．
- (*54) 完全性
  - (意見) 改訂できるということは完全な要求に対して改訂されたはずである．
- (*55) ランク付けされている
  - 安定性に対するランク付けがされていると試験しやすいコードとなっている．テスト項目の順序は与えられるが強い相関はない．

- (*56) 追跡可能性
  - (意見) 解析性よりは低い相関ではないか.

(6) 移植性

ソフトウェアの品質特性の「移植性」では，全ての副特性に対して同じ補足が併記されている．移植性のうち，環境適用性に対して，製品の属性である，という補足が付いている．

(6.1) 環境適用性，(6.2) 設置性，(6.3) 規格適用性，(6.4) 置換性

- (*57) 完全性
  - 環境適用性 (設置性，規格適用性，置換性) に関する要求が抜けていると，完全性に関連する．
- (*58) ランク付けされている
  - 機種依存性の要求のランクが低いと環境適用性はあがる．

6.3　教務系システムの相関表

相関表の作成においては，どのような種類のソフトウェアを対象とするかによって，項目間の評価が異なるという意見が多く見られた．そこで，ソフトウェアの種類を特定して相関表についての議論を行った．このとき，各参加者の経験に基づき，情報系システムと組み込み系システムにおける相関表について議論を行った．これまでにこの議論によって得られた相関表の例を表 6.4 に示す．この表は，情報系システムのうち，大学における教務系システムでの相関表の一部である．教務系システムとして，学生の学籍や成績の管理や授業のカリキュラムの管理などを行うシステムを想定している．この教務系システムは，データベース処理を主とするシステムであり，特に以下のことが重視される．

- セキュリティ
- ユーザインタフェースの使いやすさ
- 変更のしやすさ

セキュリティに関しては，教務系のシステムは学生の連絡先や成績などの個人情報を扱うために，また，ユーザインタフェースの使いやすさについては，コンピュータのエキスパートではない多くの学生が利用するために，また，変更のしやすさについては，授業のカリキュラムの変更などが数年単位で行われるため，それぞれ重視されるものと想定した．

この表 6.4 と表 6.3 を比較すると，ソフトウェアの品質特性の「正確性」と要求仕様の品質特性の「妥当性」や「非あいまい性」,「完全性」との間の評価については，大きな違いはないと考えられる．しかし，特に「標準適合性」の「ランク付けされている」,「セキュリティ」の「ラ

表 6.4: 教務系システムの分析

	機能性				
	合目的性	正確性	相互運用性	標準適合性	セキュリティ
妥当性	◎	◎	○	◎	◎
非あいまい性	◎	◎	○	◎	◎
完全性	◎	◎	○	◎	◎
一貫性	◎	◎	○	◎	◎
ランク付けされている	◎	△	○	◎	◎
検証可能性	なし	なし	なし	なし	なし
変更可能性	なし	なし	なし	なし	なし
追跡可能性	なし	なし	なし	なし	なし

ンク付けされている」などの間の相関値に大きな違いが見られるようである．表 6.3 は，特定のシステムを前提とせずに議論し，作成された相関表である．その表 6.3 と，教務系システムを前提として議論し，作成された表 6.4 の間にも，このように大きな違いが見られる場合がある．従って，システムごとの相関表を作成することが重要であると考えられる．

6.4 相関表の利用

本章の冒頭で述べたとおり，相関表を利用することにより，ソフトウェアのある特定の品質特性を高めたい場合には，要求仕様書のどの品質特性に注意すればよいかがわかることが期待できる．つまり，ソフトウェアの要求獲得のプロセスにおいて，ソフトウェアに要求される品質特性がわかれば，その品質特性と相関の強い要求仕様書の品質特性を高めるように，要求仕様書を記述することが重要となる．

また，6.3 で示したように，どのような種類のソフトウェアを対象とするかによって，相関表が異なる可能性も高い．そこで，ソフトウェア開発の要求獲得のプロセスにおいて，開発対象のソフトウェアに特化した相関表を作成することが重要であると考えられる．本ワーキンググループでは，ソフトウェアに特化した相関表を作成する際に，表 6.3 の相関表及び補足がガイドラインとして使用されることを期待している．

以上をふまえ，相関表を利用した際の要求獲得の流れを図 6.2 に示す．要求獲得の流れとしては，下記の通りである．

1. 顧客や開発者が，ソフトウェアについて重視する製品特性を決定する．
2. 1. で決定された製品特性をもとに，表 6.3 のシステム非特化の相関表をカスタマイズし，開発対象システムに特化した相関表を作成する．
3. 2. で作成されたシステム特化の相関表をもとにして，重視すべき要求仕様書の品質特性を決定する．
4. 相関表の各評価値をもとに，要求分析手法を選択する．例えば，要求仕様書の記述があいまいであれば，形式的手法を適用する，また，システムの目的が不明確であれば，ゴール指向分析を適用し，目的を明らかにする，などが考えられる．

5. 要求分析を行い，要求仕様書の記述を行っていく．

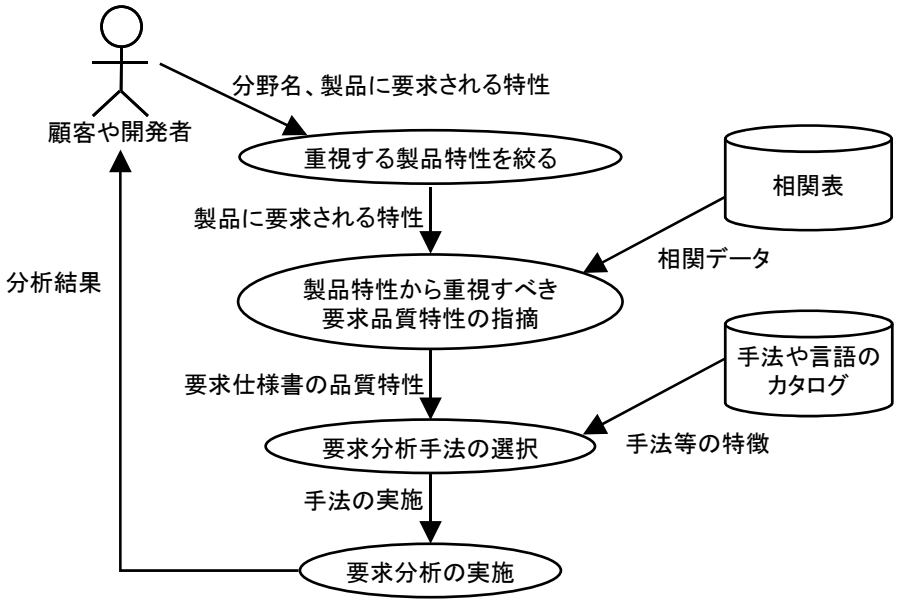

図 6.2: 相関表利用時の要求獲得プロセス

6.5　今後の課題

今後は，ソフトウェアの種類に特化した相関表について議論を深め，ソフトウェアの種類ごとの相関表を作成していくことが必要である．

また，相関表は各参加者の経験によって作成されている．従って，その正しさを何らかの形で実証することが望まれる．そのためには，実際の開発プロジェクトで要求仕様書やソフトウェア製品の品質項目を客観的に測定し，分析することも必要になってくるであろう．

第7章　おわりに

要求工学での成果をソフトウェア製品自体の品質に反映するための第一歩として，要求工学での中心的な成果物である要求仕様書と製品との関係を品質という立場から考察した結果を本稿にまとめた．

第 1 章で触れた要求工学における三つの問題点の中でも，本稿では二番目の点である「要求工学における指標」に重点をおいて，現状の技術や研究の方向性について調査・研究した．結果として，一番目の点である「技術の未成熟さ」は，たしかにその通りではあるが，利用可能な技術や研究成果は現在でも存在することを確認できた．三番目の「非機能要求の評価法」も同様である．

我々が中期的な研究目標とした点についてであるが,「仕様書自体の品質」については IEEE Std.830 標準に示された指針をもとに，いくつかの具体的なメトリクスを提案することが可能であることを示した．また,「要求品質と製品品質の関係」についてであるが，関係付けを明確化するためのパラメタが多数あるため，明確な関係を示すことはできなかった．しかし，特定のケース，例えば業務種類や開発プロセスを固定した場合を想定し，いくつかの関係を示すことができた．

1 章で言及したソフトウェア工学における品質の定義にもあるが，開発過程 (プロセス) の性質も重要な点である．本稿では要求工学プロセスについての品質には言及できなかったが，これは今後の本ワーキンググループの重要な研究課題の一つとしたい．

最後に，本稿での成果が参考となり，要求工学技術の活用が一層促進され，それによってソフトウェア開発がビジネスとして一層成功することを願う．実際，一部の業界 (生命保険業界等) では，開発されたソフトウェアや情報システムはビジネスを支援する単なるツールではなく，ビジネスを成立させる基盤そのものになりつつある．この点を鑑みても情報システムのための要求分析の役割は益々大きくなると思われる．

執筆および討論参加者

執筆者

1, 7 章 海谷 治彦 (信州大学)

2 章 妻木 俊彦 (日本ユニシス)

3 章 中村 友昭 (新日鉄ソリューションズ株式会社)

4 章 鎌田 (板倉) 真由美 (日本 IBM 東京基礎研究所)

5 章 佐伯 元司 (東京工業大学), 大西 淳 (立命館大学)

6 章 白銀 純子 (東京女子大学), 中谷 多哉子 (筑波大学)

討論参加者 (50 音順, 執筆者を除く)

所属は討論参加当時のものである.

糸賀 裕弥 (立命館大学), 長田 晃 (信州大学), 勝藤 彰夫 (アルゴ創研), 河野 仁一 (東京工業大学), 中所 武司 (明治大学), 友枝 敦 (富士総研), 橋本 正明 (九州工業大学), 林 晋平 (東京工業大学), 廣田 豊彦 (九州産業大学), 蓬莱 尚幸 (セレスター・レキシコ・サイエンシズ), 三瀬 敏朗 (松下電工/九州工業大学) 綿引 健二 (東京工業大学)
他 9 名

付 録A　要求仕様書の例

行頭の数字は行番号である．5 章の分析は以下の 3 節以降を対象としている．3 節において行番号の後に # がある行は行数として数えていない．よって分析対象となるのは 045〜110 行の 66 文となる．

[001] 1. はじめに
[002] 1.1 目的
[003] 国際会議のプログラム委員長 (PC) の業務を支援する情報システムの構築のためにこの文章は書かれた.
[004] よって，システムの設計者を含む開発グループおよびプログラム委員長自身が本文書の主たる読者となる.
[005] 1.2 スコープ
[006] 本システムは「PC 支援システム」と命名されている.
[007] その名のとおり多忙なプログラム委員長が行う業務のうち，自動化可能なものは自動化し，委員長の意思決定の助けとなる情報を自動作成することを目的とする.
[008] これによって，PC 自身の仕事量の削減，および仕事の遂行時間の短縮を可能とする.
[009] 1.3 用語定義について
[010] 1. PC 支援システム: 本文書で作成を要求している情報システム製品の名称
[011] 2. システム: 通常，PC 支援システムに同じ.
[012] 3. プログラム委員長: 本システムの利用者
[013] 4. PC: プログラム委員長に同じ
[014] 5. PC メンバー: プログラム委員会のメンバー
[015] 6. 投稿者: 論文投稿者
[016] 7. MUA: メールユーザーエージェント Outlook 等が代表的.
[017] 8. MTA: メールトランスファーエージェント sendmail 等.
[018] 1.4 参照文書
[019] 本システムの支援対象となる業務の内容は以下の二つの文書をもとに遂行されている.

[020] 1. IEEE による国際会議開催の手引き
http://www.computer.org/portal/site/ieeecs/index.jsp より
Conferences -> Conference Organizer Resources のページ.
[021] 2. 大西教授による業務の手引き
http://www.selab.is.ritsumei.ac.jp/~ohnishi/RE/problem.html
[022] 1.5 以降の内容
[023] 本章を含め 3 章構成であり，PC 支援システムへの要求項目は 3 章に詳細定義されている.

[024] 2 章では 3 章で定義されたシステムの背景および周辺を説明する.
-----
[025] 2. 背景と概要
[026] 2.1 システムの展望
[027] PC 支援システムは PC が自身のパーソナルコンピュータ上で動作させるアプリケーションソフトウェアである.
[028] 本システムは MUA の機能も有しており，これらを使って，PC メンバーや投稿者との情報交換を行うことができる.
[029] MUA は暗号化しない SMTP を使用する.
[030] 2.2 主な機能
[031] PC 支援システムの主な機能は以下のとおりである.
[032] ・メールで投稿された論文を収集し，投稿者に対して受諾の返事を行う.
[033] ・事前に登録された PC メンバーがどの論文を査読すべきかの情報を保持する.
[034] ・査読結果を収集を自動的に行い，全論文の仮の順位付けを行う.
[035] 2.3 ユーザーの特徴
[036] ユーザーはおおむね博士号を持つレベルの高い知能を有するが，多忙なものが多い.
[037] 2.4 制限
[038] 本システムは，電話，FAX，郵便等による情報を扱うことはできない.
[039] 本システムは PC 自身のパーソナルコンピュータで動作するため 24 時間稼動ではない.

[040] 本システムと直接連携をとる MTA の信頼性は十分に高い.
[041] 2.5 想定事項
[042] PC 支援システムが動作するコンピュータは特定の OS を想定していないが，前述のようにユーザーと対話を行う部分，および SMTP で外部システムと交信を行う機能を有していなければならない.
[043] 2.6 将来展望
[044] 本システムの次期バージョンではウエブサーバーとの連携を行う.

----- # 3. 要求事項について
----- # 3.1 外部インタフェース要求
----- # 3.1.1 ユーザーインタフェース
[045] a. マウスおよびキーボードの入力によって機能を遂行する GUI を提供すること.
----- # 3.1.2 ハードウェアインタフェース
[046] a. 同システムを稼動させるコンピュータは 2 つ以上のボタンを持つマウスと,
[047] b. キーボードを有すること.
[048] c. 同システムを稼動させるコンピュータは MTA と通信するための装置を持つこと.
[049] d. 二次記憶装置は RAID1 に準拠し 3 箇所以上の冗長化が行われていること.
[050] e. 局地災害に対処するため冗長化された記憶装置は異なる国に配置されていること.

----- # 3.1.3 ソフトウェアインタフェース
[051] a. US-ASCII だけでなく ISO 8859-1 (Latin-1) Character Set を扱えること.
[052] b. ISO-2022-JP 等，アジア圏の言語は扱えなくともよい.

[053] c. 日付および時刻の情報を取得できること.
[054] d. MTA 側に論文投稿窓口となるメールアドレスが登録されていること.
[055] e. MTA 側に査読結果受理窓口となるメールアドレスが登録されていること.
----- # 3.1.4 通信インタフェース
[056] a. SMTP に必要な装置やソフトウェア部品が提供されていること.
----- # 3.2 機能要求
----- # 3.2.1 モード 1: PC メンバー登録
[057] 3.2.1.1 PC メンバーの情報を登録できること.
[058] 登録する項目は以下の 6 個: 名前, 所属, メールアドレス, 電話番号, 住所, 得意分野のリスト
[059] 3.2.1.2 必ず 200 人以上の PC メンバーを登録できること.
[060] 3.2.1.3 登録内容の更新を禁止できること.
[061] 3.2.1.4 可能ならば登録内容の更新を禁止を解除できること.
[062] 3.2.1.4 登録内容の更新を禁止されていなければ，自由に修正追加削除ができること.
[063] 3.2.1.5 登録内容を通常のテキストファイルおよび CSV 形式で出力できること.
----- # 3.2.2 モード 2: 論文の収集
[064] 3.2.2.1 収集の開始日時を設定できること.
[065] 3.2.2.2 収集の終了日時を設定できること.
[066] 3.2.2.3 論文投稿先のメールアドレスおよびパスワードを設定・変更できること.
[067] 3.2.2.4 上記のメールアカウント宛のメールを自動的に読み込むことができること.

[068] 3.2.2.5 読み込んだメールが論文投稿のメールであるか否か判定できること.
[069] 3.2.2.6 論文投稿メールは PDF の論文本体と著者の情報を含むフォームの二つが添付されていること.
[070] 3.2.2.7 論文投稿のメールである場合，送信者に受信が成功した旨を自動的に連絡できること.
[071] 3.2.2.8 論文投稿のメールのようであるが様式が不適切である場合は，その旨を送信者に自動的に連絡すること.
[072] 3.2.2.9 不正な様式とは PDF ファイルの破損およびアジア圏のフォントの混入，著者情報のフォームの記入漏れをさす.
[073] 3.2.2.10 全く論文投稿メールの体裁をしていないメールは自動的に破棄すること.
[074] 3.2.2.11 送信者への自動返答に失敗した場合，その旨を利用者に通知すること.
[075] 3.2.2.12 収集の開始前および終了後に届いたメールは自動的に必ず破棄すること.
[076] 3.2.2.13 フォームに埋める著者の情報については TBD.
----- # 3.2.3 モード 3: 論文収集状態の表示
[077] 3.2.3.1 収集中の統計情報を表示できること.
[078] 統計情報とは締め切りまでの時間と集まった論文の数である.
[079] 3.2.3.2 送信者への自動返答に失敗に関する詳細情報を表示できること.
[080] 3.2.3.3 収集されている論文のリストをプレインテキストおよび CSV 形式で出力可能であること.
[081] 出力項目は以下のとおり.
[082] 論文のタイトル, ページ数, キーワードのリスト

[083] 3.2.3.4 自動返答に失敗に関する詳細情報の内容は TBD.
----- # 3.2.4 モード 4: PC メンバーに担当論文をアサインおよび配布
[084] 3.2.4.1 論文 1 通当りの最小査読者数を設定できること．1 以上の整数.
[085] 3.2.4.2 モード 1 で生成した PC リストとモード 3 で生成できる論文リストを自由に対応付けられること.
[086] 3.2.4.3 論文当りの査読者数が 3.2.4.1 での設定に反しないことを完全にチェックできること.
[087] 3.2.4.4 それぞれの PC メンバーの担当論文数を列挙できること.
[088] 3.2.4.5 3.2.4.2 での対応付けの更新を禁止できること.
[089] 3.2.4.6 対応付けが禁止された状態において，担当する PC メンバーに担当論文を電子メールで自動送付ができること.
[090] 3.2.4.7 3.2.4.6 と同時に査読フォームを送付できること.
[091] 3.2.4.8 PC メンバーからの苦情等のメールを自動的に破棄されることが望ましい.
----- # 3.2.5 モード 5: 査読結果の収集
[092] 3.2.5.1 結果収集の締め切り日時を登録できること.
[093] 3.2.5.2 査読結果受理窓口となるメールアドレスを登録できること.
[094] 3.2.5.3 結果報告のメールを自動収集できること.
[095] 3.2.5.4 結果を報告したと思われる PC メンバーに対して，結果を受理した旨を自動返信できること.
[096] 3.2.5.5 報告結果の様式に不正がある場合，その旨も連絡すること.
[097] 3.2.5.6 締め切り後にもかかわらず返答がない査読結果のリストをテキストもしくは CSV ファイル形式で出力できること.
----- # 3.2.6 モード 6: 査読結果収集状態の表示
[098] 3.2.6.1 査読結果の収集率をパーセントで表示できること.
[099] 3.2.6.2 返答のまだない PC メンバーと返答していない論文のリストを出力できること.
----- # 3.2.7 モード 7: 査読結果の仮集計の生成と表示
[100] 3.2.7.1 優劣の判定基準は自由に設定できること.
[101] 3.2.7.2 優劣の判定基準に加えて論外か否かの閾値を設定できること.
[102] 3.2.7.3 収集結果に基づき論文の優劣を自動的に順序付けを出力できること.
[103] 3.2.7.4 順序付けの際，論外を非出力することが可能なこと.
[104] 3.2.7.5 出力結果はテキストもしくは CSV 形式であること.
----- # 3.3 パフォーマンス要求
[105] 3.3.1 システムはユーザーがストレスを感じない範囲において反応すること.
[106] ただし，MTA 等，外部システムの反応遅れがあった場合，その旨がわかるようにすること.
[107] 3.3.2 前述のように扱える PC メンバーの数は 200 人以上であること.
[108] 3.3.3 扱える投稿論文数の数には上限が無いこと.
----- # 3.4 設計制約
[109] 同システムが利用する二次記憶装置は 500GB とする.
[110] ちなみに標準的な 1 通の論文は 200KB 前後である.
----- # 3.5 ソフトウェアシステム属性
----- # 特に無し

----- # 3.6 その他
----- # 無し

参考文献

[1] 大西淳. 要求工学ワーキンググループ活動報告. 情報処理学会研究報告, Vol. 2001, No. 31, pp. 127–134, Mar. 2001. 2001-SE-130, ソフトウェア工学 130-18.

[2] 情報処理学会 ソフトウェア工学研究会要求工学ワーキンググループ. 要求工学における品質特性の測定について. 情報処理学会研究報告, Vol. 2001, No. 114, pp. 75–82, Nov. 2001. ソフトウェア工学研究会 2001-SE-135.

[3] IEEE Standard Glossary of Software Engineering Terminology, 2002. IEEE Std. 610.12-1990(R2002).

[4] Bertrand Meyer. On Formalism in Specifications. *IEEE Software*, Vol. 2, No. 1, pp. 6–26, 1985.

[5] IEEE, Recommended Practice for Software Requirements Specifications, Std 830-1998, 1998.

[6] Linda A. Macaulay. *Requirements Engineering*. Applied Computing. Springer, 1996.

[7] Pericles Loucopoulos and Vassilios Karakostas. *System requirements engineering*. McGraw-Hill, 1995.

[8] Soren Lauesen. *Requirements Engineering Styles and Techniques*. Addison-wesley, 2002.

[9] 大西淳, 郷健太郎. 要求工学. ソフトウェアテクノロジーシリーズ. 共立出版, May 2002.

[10] Ian Sommerville and Pete Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, 1997.

[11] Suzanne Robertson and James Robertson. *Mastering the Requirements Process*. Addison-Wesley Pub (Sd), 1st edition, Aug. 1999.

[12] Karl E. Wiegers. *Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle*. Microsoft Pr, 2nd edition, Feb. 2003.

[13] Gerald Kotonya and Ian Sommerville. *Requirements Engineering Process and techniques*. Wiley, 1998.

[14] Alan Davis and et. al. Identifying and measuring quality in a software requirements specification. In *Proc. of 1st Int'l Software Metrics Symp.*, pp. 141–152, 1993.

[15] Shaw, M. Prospects for an engineering discipline of software. *IEEE Software*, Vol. 7, No. 6, pp. 15–24, 1990.

[16] 大西 淳. 要求工学ワーキンググループ活動報告. 情報処理学会研究報告 ソフトウェア工学研究会, Vol. 130, pp. 127–134, 2001.

[17] Robertson, S. and Robertson, J. *Mastering the Requirements Process*. Addison-Wesley, 1999.

[18] Sharp, H., Finkelstein, A. and Galal, G. Stakeholder identification in the requirements engineering process. In *Workshop on Requirements Engineering Processes (REP'99) - DEXA'99*, pp. 387–391, 1999.

[19] Zucconi, L. Techniques and experiences capturing requirements for several real-time applications. *ACM SIGSOFT Software Engineering Notes*, Vol. 14, No. 6, pp. 51–55, October 1989.

[20] Zeroual, K. An approach for automating the specification-acquisition process. In *Proc. of the 2nd International Workshop on Software Engineering and Its Applications*, pp. 349–355, 1989.

[21] Kawakita, J. *The Original KJ Method*. Kawakita Research Institute, 1996.

[22] Stewart, D. W. and Shamdasani, P.N. *Focus Groups: Theory and Practice*. Sage, 1990.

[23] Zahniser, R.A. How to speed development with group sessions. *IEEE Software*, pp. 109–110, May 1990.

[24] Shaw, M. and Gaines, B. Requirements acquisition. *Software Engineering Journal*, Vol. 11, No. 3, pp. 149–165, 1996.

[25] Goguen, J. and Linde, C. Techniques for requirements elicitation. In *Proc. of 1st IEEE International Symposium on Requirements Engineering (RE'93)*, pp. 152–164, 1993.

[26] Potts, C. Requirements models in context. In *Proc. of 3rd International Symposium on Requirements Engineering (RE'97)*, pp. 102–104, 1997.

[27] Viller, S. and Sommerville, I. Social analysis in the requirements engineering process: From ethnography to method. In *4th International Symposium on Requirements Engineering (RE'99)*, 1999.

[28] Easterbrook, S. Resolving conflicts between domain descriptions with computer-supported negotiation. *Knowledge Acquisition: An International Journal*, Vol. 3, pp. 255–289, 1991.

[29] Robinson, W. N. and Volkov, S. Supporting the negotiation life-cycle. *Communications of the ACM*, Vol. 41, No. 5, pp. 95–102, 1998.

[30] Boehm, B. et al. Software requirements as negotiated win conditions. In *Proc. of Int'l Conf. Requirements Eng.*, 1994.

[31] B. Boehm and H. In. Identifying quality-requirement conflicts. *IEEE Software*, pp. 25–35, 1996.

[32] Saaty, T. L. *The Analytic Hierarchy Process*. McGraw Hill, 1980.

[33] FIPS183. Federal Information Processing Standards Publication 183: Integration definition for Function Modeling(IDEF0), 1993.

[34] DeMarco, T. *Structured Analysis and System Specification*. Prentice-Hall, 1979.

[35] Jackson, M. *System Development*. Prentice Hall, 1983. （邦訳；システム開発ＪＳＤ法）.

[36] Rumbaugh, J. et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991. （邦訳；オブジェクト指向方法論ＯＭＴ）.

[37] Mohagheghi, P., Anda, B. and Conradi, R. Effort estimation of use cases for incremental large-scale software development. In *Proc. of ICSE 2005*, pp. 303–311, 2005.

[38] Mader, M. *Designing Tool Support for Use-Case-Driven Test Case Generation*. Di-plomarbeit FHTW Berlin, 2004.

[39] Bertrand Meyer. *Object-oriented software construction*. Prentice Hall, second edition, 1997.

[40] Kang, C., Cohen, G., Hess, A., Novak, E. and Peterson, A. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21 ADA235785, Software Engineering Institute, 1990.

[41] van Lamsweerde, A., Darimont, R. and Letier, E. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, Vol. 24, No. 11, pp. 908–926, 1998.

[42] Yu, E. Towards Modelling and reasoning support for early-phase requirements engineering. In *Proc. of 3rd Requirements Engineering (RE'97)*, pp. 226–235, 1997.

[43] Kaiya H., Horai H. and Saeki M. AGORA: Attributed goal-oriented requirements analysis method. In *Proc. of Requirements Engineering (RE'02)*, 2002.

[44] Jacobson I., Christerson M., Jonsson P. and Overgaard G. *Object-Oriented Software Engineering*. Addison Wesley, 1992. （邦訳； オブジェクト指向ソフトウェア工学 OOSE）.

[45] Grady R. Practical. *Software Metrics for Project Management and Process Improvement*. Prentice Hall, 1992.

[46] Jacobson I. Usecase and aspect - Working seamlessly together. *Journal of object technology*, Vol. 2, No. 4, pp. 7–28, 2003.

[47] A. Cockburn. *Writing Effective Use Case*. Addison Wesley, 2001.

[48] Takako Nakatani. Requirements description metamodel for use case. In *Proc. of APSEC 2001*, pp. 251–258, 2001.

[49] 玉井 哲雄. ソフトウェア工学の基礎. 岩波書店, 2004.

[50] Jackson, M. *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*. Addison Wesley, 1995.

[51] Parnas, D.L. and Madey, J. Functional documentation for computer systems. *Science of Computer Programming*, Vol. 25, No. 1, pp. 41–61, 1995.

[52] Jones,C.B. *Systematic Software development using VDM*. Pretice Hall, 2nd edition, 1990.

[53] Spivey, J.M. *The Z Notation–A Reference Manual*. Pretice Hall, 2nd edition, 1992.

[54] J. Clarke, O. Cross, and A. Peled. *Model Checking*. MIT press, 1999.

[55] Maiden, N. and Sutcliffe, A. Exploiting Reusable Specifications Through Analogy. *Communications of the ACM*, Vol. 34, No. 5, pp. 55–64, 1992.

[56] Heitmeyer, C. L., Jeffords, R. D. and Labaw, B. G. Automated Consistency Checking of Requirements Specifications. *IEEE Transactions on Software Engineering and Methodology*, Vol. 5, No. 3, pp. 231–261, 1996.

[57] Holzmann, G. The Model Checker Spin. *Transactions on Software Engineering*, Vol. 23, No. 5, pp. 279–295, 1997.

[58] Fagan, E. Design and code inspection to reduce errors in program development. *IBM systems journal*, Vol. 15, No. 3, pp. 182–211, 1976.

[59] Nuseibeh, B. Ariane 5: Who Dunnit? *IEEE Software*, Vol. 14, No. 3, pp. 15–16, 1997.

[60] Davis, A. Operational prototyping: A new development approach. *IEEE Software*, Vol. 9, No. 5, pp. 70–78, 1992.

[61] J.R. Cameron. Prototyping core functionality using JSD. In *IEE Colloquium on Requirements Capture and Specification for Critical Systems*, p. 138, 1989. 2/1-2/2.

[62] A. Ohnishi and N. Tokuda. Visual software requirements definition environment. In *Proc. of COMPSAC*, pp. 624–629, 1997.

[63] B Boehm. Software Risk Management: Principles and Practices. *IEEE Software*, Vol. 8, No. 1, pp. 32–41, 1991.

[64] Tsumaki, T. and Morisawa, Y. A framework of requirements tracing using UML. In *Proc. of the 7th Asia Pacific Software Engineering Conference*, pp. 206–213, 2000.

[65] Christel M.G. and Kang K.C. Issues in requirements elicitation. Technical Report CMU/SEI-92-TR-12, CMU, 1992.

[66] Sage, Andrew P. and Palmer, James D. *Software Systems Engineering*. John Wiley & Sons, 1990.

[67] Dubois, E. and Hagelstein, J. and Rifaut, A. Formal requirements engineering with ERAE. *Philips Journal of Research*, Vol. 43, No. 3/4, pp. 393–414, 1988.

[68] Eriksson, H. and Penker, M. *Business Modeling with UML*. John Wiley & sons, Inc, 2000. 邦訳;「UML によるビジネス・モデリング」, ソフトバンク.

[69] C Marshall. *Enterprise Modeling with UML*. Addison Wesley Longman, 2000.

[70] Tsumaki, T. and Tamai, T. A framework for matching requirements engineering techniques to project characteristics and situation changes. In *Proc. of International Workshop on Situational Requirements Engineering Processes in conjunction with the 13th IEEE International Requirements Engineering Conference*, pp. 44–58, 2005.

[71] A. Finkelstein, J. Kramer, B. Nuseibeh, Finkelstein L., and Goedicke M. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, Vol. 2, No. 1, pp. 31–58, 1992.

[72] Nuseibeh, B., Kramer, J. and Finkelstein, A. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, pp. 760–773, 1994.

[73] S. Brinkkemper, Saeki M., and Harmsen F. Meta-modelling based assembly techniques for situational method engineering. *Information System*, Vol. 23, No. 7, pp. 49–508, 1998.

[74] N.E. Fenton and S.L. Pfleeger. *Software Metrics: A Rigorous & Practical Approach(2nd ed.)*. PWS Publishing Company, 1997.

[75] T. DeMarco. *Controlling Software Projects.* Yourdon, 1982.

[76] C. Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, 2000.

[77] Standish Group. CHAOS Report, 1994.

[78] K. Ewusi-Mensah. *Software Development Failures*. The MIT Press, 2003.

[79] T.L. Woodings. Meta-metrics for the Accuracy of Software Project Estimation. In *Proceedings of the 2nd Western Australian Workshop on Information Systems Research*, 1999.

[80] J.R. Taylor. 計測における誤差解析入門. 東京化学同人, 2000.

[81] V.R. Basili and D.M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, Vol. 10, No. 6, 1984.

[82] J. Barnard and A. Price. Managing code inspection information. *IEEE Software*, Vol. 11, No. 2, 1994.

[83] T. Glib. *Principles of Software Engineering Management*. Addison-Wesley, 1988.

[84] International Standard ISO/IEC 9126-1. Software engineering - Product quality - Part 1: Quality model, 2001.

[85] ISO/IEC 9126 : Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use, 1991.