ソフトウェア品質の
定量的な評価枠組みに関する研究

A Study on Software Quality
Quantitative Evaluation Framework

2020 年 2 月

津田　直彦

Naohiko Tsuda

ソフトウェア品質の
定量的な評価枠組みに関する研究

A Study on Software Quality
Quantitative Evaluation Framework

2020 年 2 月

早稲田大学大学院　基幹理工学研究科

情報理工・情報通信専攻　高信頼ソフトウェア工学研究

津田　直彦

Naohiko Tsuda

目次

図目次

表目次

謝辞

本研究を進めるにあたり，多くの方々に御指導，御協力，御支援頂きましたそのすべての方々に深く御礼申し上げます．

本論文の主査を快くお引き受けいただき熱心な御指導を賜りました早稲田大学 鷲崎 弘宜 教授に深く御礼申し上げます．また，副査を快くお引き受けいただき様々な観点からご助言を賜りました早稲田大学 深澤 良彰 教授，早稲田大学 本位田 真一 教授，東京工業大学 林 晋平 准教授に深く御礼申し上げます．

第 2 章の研究を実施するにあたり，品質測定評価のための 21 製品のデータを提供いただいた 12 社のソフトウェア製品提供元各位に深く御礼申し上げます．また，測定評価の一部作業を実施いただいた複数の外部評価機関各位に深く御礼申し上げます．製品募集や進行にあたり多くのご支援をいただいた CSAJ 殿に深く御礼申し上げます．ユーザテストをはじめとする測定評価に関する作業について，早稲田大学グローバルソフトウェアエンジニアリング研究所，鷲崎研究室および情報理工学科における次の学生に深く御礼申し上げます：橋本慧氏，中井秀矩氏，青木耀平氏，角田武人氏，礎良輔氏，野寄祐樹，渡邊泰宏氏，先崎祐一郎氏，細野将揮氏，山田龍平氏，志村千万輝氏．また，研究機会をいただいた IPA 殿に深く御礼申し上げます

第 3 章の研究を実施するにあたり，品質測定評価のための製品データ提供をいただいたソフトウェア製品提供元に深く御礼申し上げます．

最後に，日頃から応援してくださった私の家族に心より深く感謝します．

1. はじめに

1.1. 背景

ソフトウェア品質の向上は現代社会における重要課題であり [1], 開発組織においては,その特徴を体系的かつ定量的に評価して把握することが欠かせない. 本論文では, 集積済みの実データとの比較(ベンチマーキング)に基づいて, 別のソフトウェア要素の品質を評価する形式の枠組みについて論ずる.

評価枠組みの主要な構成要素としては,品質モデルと評価方法が挙げられる.

品質モデルとは, 評価すべき品質の特徴を定義した分類体系である. ソフトウェアの品質を妥当かつ多面的に捉えることができるため,評価枠組みを構築する際の初期案として役立つ. ただし, 分類体系という性質上, 品質モデルは汎用的かつ抽象的な観点を与えるにとどまっており,具体的な評価方法については枠組みの運用者が決める必要がある.

品質の評価方法については, 開発組織を越える場合と, 開発組織内の場合とで, 主体的に評価枠組みを運用する組織が異なると考えられる. そして, それぞれの場合について, 本節の以降で述べる背景と課題がある.

尚, ソフトウェアを開発する組織については様々な形で捉えることができ, プロジェクト単位や企業単位,あるいは外注を含めて複数の企業が携わる形がありうる. 本論文では, 集積済みの実データ(ベンチマークデータ)の詳細を追跡可能であるかという側面から,品質評価枠組みの運用における開発組織の違いを考慮している. そして, そのような意味で,「開発組織を越える」「開発組織内」という表現を用いている. 例えば, 商業製品の開発元であれば異なる企業間で開発組織を越えた測定値データを素のままで共有することは難しいと考えられる. 一方で, 特定の企業内の近しい部門間では一定の手続きを要してデータを共有できることが比較的多いと考えられる. また, 外注を含めて複数の企業が携わるケースについては,二種類の場合に切り分けることが難しいが, 納品時の最終成果物の粒度では,開発の上流でソフトウェアの品質に責任を負う組織がデータの詳細を追跡できることが比較的多いと考えられる.

(a).開発組織を越えた評価：

開発組織を越えた評価では, 複数組織の製品を扱う性質上, 第三者的な外部組織がデータの集積およびベンチマークの作成を担う. また, 評価対象や評価結果の詳細については, 自組織の内容のみ閲覧できるのが自然である. そのため, 開発組織においては, 自製品のおおよその位置づけを知るための相対的な判断材料としてベンチマークを用いる.

製品や組織を超えて共通に適用可能かつ,様々な品質を多面的に定義した枠

組みとしては,ISO/IEC の国際規格 SQuaRE シリーズ [2]が存在する. しかし, 国際規格という性質上, SQuaRE では汎用的かつ抽象的に品質の特徴を定義するにとどまっており [3] [4], 複数の組織から統一的にデータを集積してベンチマークを作ることが難しかった. そのため, 従来の評価枠組みや実態調査では, 具体的に扱われる特徴が部分的であり, 特に利用者の実感を表す特徴などを明示的に含んでいない [5] [6] [7] [8] [9] [10]. 結果として, 品質間の多面的な分析が難しく,品質保証活動における効果的な意思決定が妨げとなっている.

(b).開発組織内での評価：

開発組織内での評価では, 自製品を扱う性質上, 自組織でデータの集積およびベンチマークの作成を担う. したがって, 評価方法の詳細とソフトウェア要素の詳細について照らし合わせることができる. そのため, 開発組織においては, ベンチマークを用いる際に自組織の都合に応じた調整が可能である.

通常のプロジェクトでは,開発組織は必ず最終成果物としてプログラムソースコード(以下コードと略す)を管理している. そのため, コード用の種々の解析ツールを利用可能であり [11] [12] [13] [14], ソフトウェア要素の中では比較的に品質評価を実施しやすい. これらのツールではソフトウェアの特徴量に対してデフォルトの評価基準を提供しているが,リスクの許容水準を表す閾値を任意で変更できる場合が多い. しかし, 自動評価の精度を向上させるための閾値導出手法群については,その性能と性質の違いが必ずしも十分に明らかとされていない. 結果として, 適切な手法の選択が難しく, 品質評価基準の効果的な調整の妨げとなっている.

上述の通り,開発組織を越えた評価と開発組織内での評価のそれぞれについて, 運用上の課題が存在する. そのため, ソフトウェアの品質評価枠組みを効果的に利用できる開発組織や環境が制限されている.

1.2. 本論文の取り組み

本論文では,ソフトウェア製品の品質評価枠組みの運用における上述の課題の解決に向けて, 以下の二つの研究に取り組んだ. また, これらの業績は [15](a)と [16](b)で対外的に公開している.

(a).開発組織を越えた評価についての取り組み：

国際規格に基づく網羅的なソフトウェア品質評価枠組みの構築と実適用：

第三者が利用可能な形で提供されているパッケージ製品やクラウドアプリケーションなどを評価対象として，国際規格群 SQuaRE シリーズの品質測定量の測定評価方法を網羅的に具体化した．そして，12 社の開発組織と協力して 21 製品のデータを集積し，製品品質と利用時の品質の間の関係について，その限られた一端を明らかにした．そして，産業界への提言をまとめると共に，実現した枠組みと品質ベンチマークを公開した．

期待される用途は，ソフトウェア開発における様々な段階での品質評価の活用であり，品質の要求定義・改善・取捨選択において判断材料として使うことができる．一部の企業では既にこの枠組みを適用し，品質特性単位の評価結果を製品の品質改善に役立てている [17]．

(b).開発組織内での評価についての取り組み：

ソースコードの保守性評価を対象とした特定コンテキストにおける閾値導出手法の比較：

コードの保守性の内，機能的な不具合を伴わないコードの構造上の問題に焦点を当て，その高精度な自動評価を目的とする枠組みを整理した．そして，1 社の開発組織と協力して共通のプラットフォームで開発されているコードのデータを集積し，複数の閾値導出手法の性能と性質を比較した．その際，枠組みと閾値導出手法については，コードの保守性以外についても汎用的に適用可能なものを扱った．

期待される効果は，品質評価基準の調整における適切な手法の示唆であり，開発組織において独自の評価基準を構築する際の枠組みとして使うことができる．協力した開発組織では特定のソフトウェアシリーズを対象として枠組みを適用し，コードの規模と複雑度について高精度かつ一定の妥当性のある評価基準を構築できた．

本論文が提供するこれらの枠組みの相乗的な利用方法としては，以下の形態が考えられる．まず，開発組織においては，(a)の枠組みを利用することで産業界における自製品のおおまかなポジションを品質計画の早期に参照することが期待できる．そして，それを判断材料として組織内で重視する品質を定め，(b)の枠組みによって緻密な品質評価を実現することが期待できる．

1.3. 本論文の位置付け

私の知る限りで,定量的なソフトウェア品質評価枠組みについての関連研究を図 1-1 に整理した. この図では, 横軸は実現される評価の詳細性を, 縦軸は実現される評価の多面性を表している.

品質評価枠組みの実現においては,詳細な評価の提供と多面的な評価の提供がトレードオフの関係にあると考えられる.明示的に複数の側面に対応している枠組みにおいては,ソフトウェアの様々な情報を網羅的に扱おうとする都合上, 個々の品質特性の評価の具体化が難しい [6] [15]. 一方で, 詳細な評価を意図する枠組みにおいては,開発組織において通常は必ず存在するコードを分析対象とするものが多く [10] [9] [5] [8], その他の情報源に依存する品質特性の扱いを欠いている. 結果として, 多面性と詳細性の双方を一定程度確保しようとする枠組みにおいては, 専門家の参加が必要となっている [7].

私はこれらの性質の違いから,開発組織を越える品質評価と開発組織内での品質評価では, 用途と運用上の課題が異なると考えた. そして. 双方の違いを考慮して二つの研究に取り組んだ. これらは, 図 1-1 における左上(第 2 章)と右下(第 3 章)に位置する研究となっており, 双方を取り進めることは, 将来的に一定の多面性と詳細性を確保した品質評価枠組みを実現することにつながると考えた.

多面的な評価を意図する枠組みにおいては,多数の品質特性を要約して総評を提供する都合上, 測定値を一定の規則でスコア化し, さらに把握可能な単位まで集約する変換を実施している.これらの変換においては間隔尺度ではない測定値を集約することもあるため, スコアの大小は目安として用いられる.

このような制約があるものの,多面的な評価はソフトウェア開発の品質計画における判断材料としての活用が期待されている.そのため複数の先行研究があるが, 従来の評価枠組みや実態調査では扱われる品質特性が部分的であり, 特に利用者の実感を表す特徴などが明示的に含まれていない ［5］［6］［7］［8］［9］［10］.

そこで, 私は第 2 章において, 国際規格 SQuaRE の品質測定量の測定評価方法を網羅的に具体化した. そして, 12 社の開発組織と協力して 21 製品のデータを集積し, 品質ベンチマークの作成および分析に取り組んだ.

詳細な評価を意図する枠組みに関して,特にコードの保守性の評価において,測定対象とするソフトウェア要素に対する具体的な解釈を提供する手法が研究されている. これらの手法は測定値さえあれば適用可能なため, 他の品質特性についても再利用可能である.

解釈方法としては,集積したデータ全体の測定結果からパーセンタイルなどで導出した閾値を用いる方法が広く知られている [18] [19]．また，より高精度な評価を実現するために,コードの品質状態に関する教師データを用いる手法も提案されている [20] [21] [22]．ただし，これらの手法では，前提とする解釈方法のモデルが異なる上に,性質の異なるモデル同士での性能の比較が十分になされていない．

そこで，私は第 3 章において，静的解析ツールで測定可能なコードの構造上の問題に焦点を当て，その高精度な自動評価を目的とする枠組みを整理した．そして，1 社の開発組織と協力してデータを集積し，複数の手法の性能と性質を比較した．

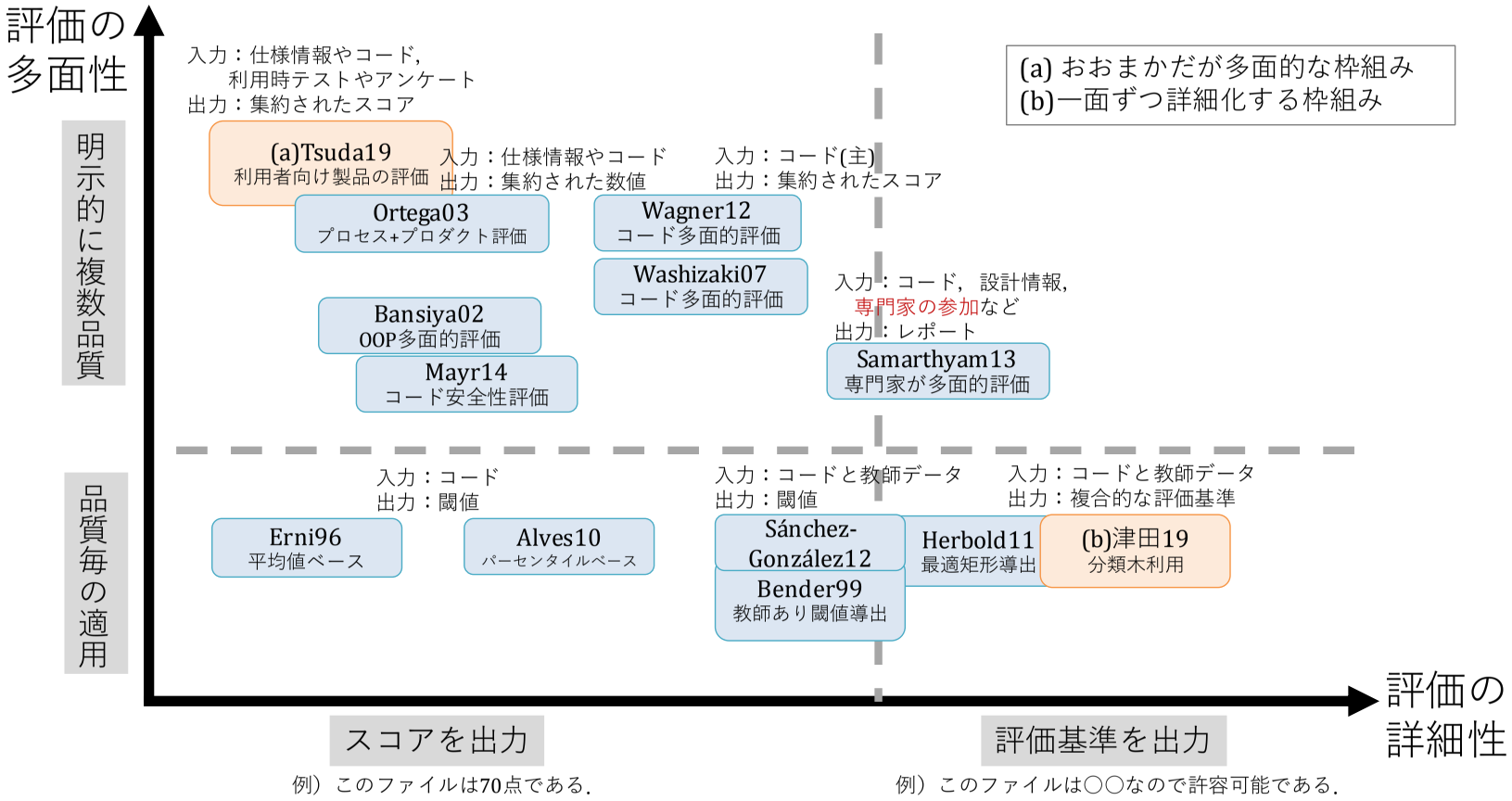

図 1-1：定量的な品質評価枠組みについての関連研究

1.4. 本論文の構成

本論文の構成は以下の通りである.

第 2 章では, 開発組織を越えた評価のための, 国際規格に基づく網羅的なソフトウェア品質評価枠組みを提案する. そして, 実際の製品への適用結果を通じて, 知見と今後の課題について述べる.

第 3 章では, 開発組織における特定コンテキストのソースコードの保守性評価を対象として, 評価基準を構築するための枠組みを整理する. そして, 枠組みで利用可能な閾値導出手法群の性能比較を通じて, 知見と今後の課題について述べる.

最後に, 第 4 章で全体のまとめと今後の展望を述べる.

2. 国際規格に基づく網羅的なソフトウェア品質評価枠組みの構築と実適用

2.1. 本章のはじめに

2.1.1. 背景

ソフトウェア品質の向上は現代社会における重要課題であり [1], 製品の開発組織では, 様々な品質の多面的な把握が欠かせない. 考慮すべき品質の種類としては, 開発者の観点としての製品品質の他, 利用者の実感としての利用時の品質が挙げられる.

これらの品質特性を多面的に測定評価するための枠組みとして, ISO/IEC が国際規格 SQuaRE シリーズ [2]を定義し, 一般的に重要と考えられる品質特性(特徴の分類)と品質測定量(特徴を定量的に表現したもの)をまとめている. しかし, それらの定義は国際規格という性質上, 特定のドメインや製品によらず共通に適用可能であることが意図されており, 汎用的かつ抽象的なものにとどまっている [3] [4]. そのため, 利用にあたっては具体化が必要であった [23].

結果として, 製品や組織を越えて網羅的な測定データを蓄積することが難しく, 品質特性間の関係は依然として不明瞭であり, 品質保証活動における効果的な意思決定の妨げとなっていた. 従来の具体化された品質評価枠組みや実態調査では, 機能適合性や信頼性といった特定の製品品質に評価項目が集中しており, 特に利用時の品質(有効性や効率など)の多くを明示的に含んでいなかった [5] [6] [7] [8] [9] [10].

2.1.2. 本研究の取り組み

本研究では, 第三者が利用可能な形で提供されているパッケージ製品やクラウドアプリケーションなどを対象に, 製品や組織を超えて共通に適用可能な形で SQuaRE の品質測定量の測定評価方法を具体化した.

さらに, 同枠組みを 21 個のソフトウェア製品群に適用することで, 品質特性別の傾向, 品質特性間の関係, 利用時品質・製品品質間の関係, そして製品コンテキストと品質特性の関係の一端を明らかとした. そして, それらの結果を総合的な品質評価枠組み WSQF: Waseda Software Quality Framework [15] として公開した. (http://www.washi.cs.waseda.ac.jp/wsqb/)

本研究の貢献を以下に整理する．これらの知見と枠組みにより，開発組織における品質計画の改善が期待される．

- SQuaRE の品質測定方法の具体化による網羅的な品質評価枠組みの構築．
- 21 製品のデータに基づく品質ベンチマーク．
- 製品品質と利用時の品質の間の関係の分析と考察．
- 具体的な測定データに基づく，産業界への提言．

以降の本章の構成は以下の通りである．

- 2.1.3 節：研究実施体制
- 2.2 節：技術的背景および関連研究．
- 2.3 節：品質評価枠組みの提案．
- 2.4 節：実態調査．枠組みの適用結果および考察．
- 2.5 節：妥当性への脅威．
- 2.6 節：成果と展望のまとめ．

2.1.3. 研究実施体制

本研究は IPA から 2015 年度ソフトウェア工学分野の先導的研究支援事業（RISE）の委託を受けて実施した．各利害関係者の役割を以下に示す．

- グローバルソフトウェアエンジニアリング研究所:
  - 委託研究の研究責任者である鷲崎教授が全体統括を行った．
  - 本論文の著者である津田が中心となってソフトウェア品質測定評価枠組みの確立，製品群への適用，結果の分析を実施した．具体的には，以下の作業を担当した．
    - 製品品質の測定用の記入様式の作成．
    - ソースコード分析環境の用意．
    - ソフトウェア製品のユーザテスト用のタスク設計作業の一部．
    - ソフトウェア製品の提供企業への測定評価方法の説明．
    - 枠組みの適用結果の回収．
    - 分析手段の実装．
    - 個々の品質の傾向と品質間の分析．
  - 著者以外が主体的に確立した枠組みの要素は以下の通りである．
    - 製品の不具合発見数を予測するためのツールの実装．
    - 利用時の品質を測定するためのユーザアンケート雛型の作成．
    - 利用時の品質を測定するためのユーザテスト実施プロセスの定義．

- 研究レビュー委員会:
  品質測定評価対象のソフトウェア製品の選定，測定評価，結果の分析を円滑かつ効果的に進めるための助言をした．専門家として，ISO/IEC JTC1/SC7/WG6 のプロジェクトエディタならびにコンビーナとして当該領域を国際的にリードする東基衛教授，込山俊博氏の参画を得た．さらに下記 CSAJ より中野正氏，鈴木啓紹氏の参画を得た．
- コンピュータソフトウェア協会（CSAJ）：
  研究チームと協力して品質測定評価対象のソフトウェア製品の提供協力企業を募集および選定し測定評価の円滑な進行を支援した．
- 評価機関(外注):
  研究チームからの委託を受けて品質測定評価を部分的に実施した．
- 学生アルバイト:
  ソフトウェア製品の測定評価作業の一部を実施した．

2.2. 技術的背景

ISO/IEC の国際規格群である SQuaRE シリーズ [2]では，ソフトウェア製品の品質に関する参照モデルを定義している．そして，品質同士の影響関係に対して以下の前提を置くことで,開発プロセスにおける品質保証に関して示唆を与えている．

- 上流工程において確保される静的な内部品質が，下流工程における動的な外部品質に影響する．
- 下流工程において確保される外部品質が，運用時に評価される利用時の品質に影響する．

しかし，品質同士の影響関係は限定された範囲でしか明らかにされておらず，開発組織における品質保証活動では既成の評価枠組みが意思決定の判断材料として十分に活用されていない [10]．

SQuaRE では評価すべき品質の特徴の分類として，二種類の品質モデル「製品品質」「利用時の品質」を定義している [24]．品質モデルは二層構造で定義されており，一層目を品質特性，二層目を副特性と呼んでいる．

また，SQuaRE では一部の品質の特徴を定量的に表現したものとして，品質測定量も策定している [25] [26]．しかし，それらの測定量の定義は汎用性を重視した結果として抽象的であり [3] [4]，直ちに利用可能な具体的な様式や指示を含んでいない [23]．

本研究では，これらの測定量の測定方法について，製品や組織を超えて共通に適用可能かつ,様々な品質特性を多面的に捉えられる形で具体化することに取り組む．

2.2.1. 関連研究

ソフトウェア製品の品質評価枠組みは複数提案されており， QMOOD [5], Ortega らのスイート [6], MIDAS [7], Mayr による品質モデル [8], Quamoco [9], 我々の過去のスイート [10]がある．これらの枠組みは設計モデルやソースコードなどの測定評価に対応している．しかし，満足度などの利用者の実感を含む利用時の品質までを網羅するものではなかった．

組織を超えたソフトウェア開発の実態調査としては，IPA データ白書 [27]や，Cusumano らの調査 [28], Jones らの調査 [29], および Russo らの調査 [30]などがある．これらの研究における調査範囲は，主に製品品質の一部の関係分析にとどまっており，利用時の品質までを含めての網羅性を欠く．

SQuaRE が定義する規格の一部に準拠した品質評価制度として，日本国内ではコンピュータソフトウェア協会（CSAJ）が PSQ 認証制度 [31]が実施している．PSQ 認証が準拠する「ISO/IEC 25051:2014」(JIS X 25051:2016) [32]では，パッケージ製品やクラウドアプリケーションを対象として，品質要求事項を抽象的に定義している．

2.3. 品質評価の枠組み

本研究の目的は，ソフトウェア製品の品質評価枠組みについて，製品や組織を超えて共通に適用可能かつ，様々な品質特性を多面的に捉えられる形で実現することである．評価対象としては，第三者が利用可能な形で提供されているパッケージ製品やクラウドアプリケーションなどを想定している．

期待される用途は，開発・保守・運用中あるいは運用検討中のソフトウェア製品に適用し，評価結果を品質の要求定義・改善・取捨選択における判断材料として役立てることである．実際に，一部の企業では既にこの枠組みを適用し，品質特性単位の評価結果を製品の品質改善に役立てている [17].

品質評価枠組みの全体像を以下に述べる(図 2-1).

- 品質モデルについては，「ISO/IEC 25010:2011」で定義されているソフトウェアの製品品質(内部品質および外部品質)と利用時の品質を用いた．合計で 13 個の品質特性と 42 個の副特性が含まれている．
- 品質の測定方法については，「ISO/IEC 25023:2016」と「ISO/IEC 25022:2016」における抽象的な測定量を副特性単位で網羅し，合計で 79 個の測定方法を具体化した．その際，「ISO/IEC 25051:2014」や PSQ 認証制度を参考に，製品を問わずにおおむね共通に得られる属性を用いることとした．
- 品質の評価方法については，測定値を後述の計算式でスコア化して，品質

副特性および品質特性単位で集約する方法を用いた．これにより，品質特性間の関係，および製品品質と利用時の品質間の関係を分析可能とした．

- 評価結果については，集積して品質ベンチマークとしてまとめた．これにより，各品質特性について，品質ベンチマークにおける各製品のポジションの相対評価を可能とした．本研究では，品質ベンチマークとポジションの分析を通じて，品質改善のための診断レポートを製品開発元に提供した．診断においては，形式化されたアドバイスと，製品に応じて人的に導出したアドバイスを含んでいる．レポートの例については [33]を参照されたい．

以降の節では，測定方法と評価方法の詳細について説明する．

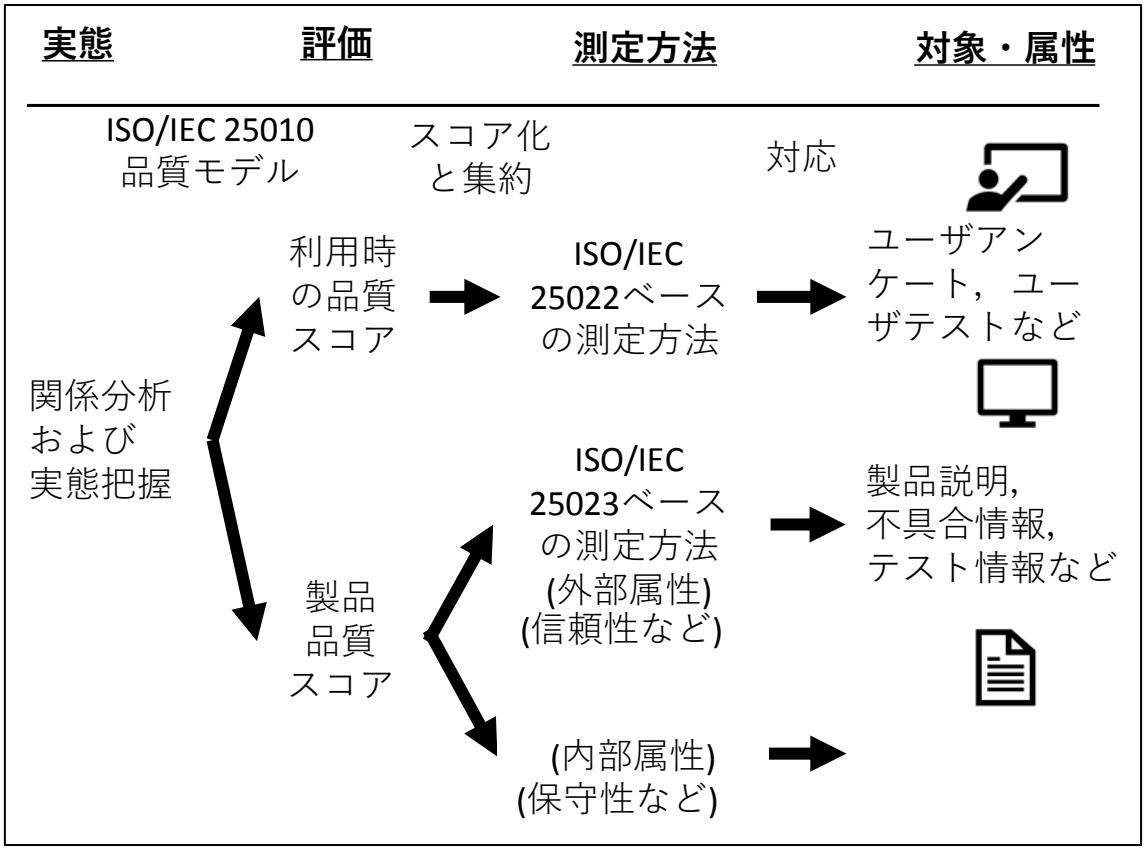

図 2-1：品質評価枠組みの全体像

2.3.1. 製品品質の測定方法

製品品質について，その品質副特性を網羅するように合計で 62 個 の測定方法を定義した．測定方法の一覧を表 2-1 に示した．

測定方法の具体化にあたり，「ISO/IEC 25010:2011」の品質特性を網羅的に評価可能とすることを目的とし，「ISO/IEC 25023:2016」における抽象的な測定量を初期候補とした．そして，測定に必要なデータの記入様式や，データの受領形式を策定した．その過程では，「ISO/IEC 25051:2014」や PSQ 認証制度を参考とし，SQuaRE およびその JIS 化を担当した国内の委員から助言を得た．

- データの記入様式については，全ての品質特性の測定方法で用いた．ソフトウェア製品の内部属性と外部属性を多面的に評価するために，製品仕様や開発時の試験記録を記入項目とした．例えば，セキュリティの副特性である否認防止性の測定方法(SNo.1.1)ついては，経路の種類数とそのうちの署名済みのものの記入を求めている．
- 開発時の発見欠陥数の記録については，機能適合性・信頼性・保守性の測定方法の一部で用いた．また，信頼性の一部の測定方法では信頼性成長モデル [34]を用いた．
- ソースコードの静的解析の結果については，保守性の副特性であるモジュール性と再利用性の測定方法で用いた．静的解析ツール Understand を用い，関数の複雑度，クラスの結合度，およびクラス内の凝集性の欠如を測定した．

品質の測定方法については，測定可能なデータありきで検討すると，本来捉えようとしていた性質を見失う可能性がある [35]．そこで，目標指向の枠組みである Goal-Question-Metric (GQM) 法 [36]を用いて検討を進めた．結果として，「ISO/IEC 25023:2016」の測定量のうち一部を対象外としつつも，全ての副特性について 1 個以上の測定方法を定義した．

例として，測定方法(SNo.1.1)の定義を図 2-2 に示す．図では，セキュリティの副特性である否認防止性について，経路のデジタル署名に着目して目標を設定している．そして，目標の達成可否を判断するために複数の側面から質問を設定し，最終的な測定結果が分かりやすくなるように一つの測定方法にまとめあげている．

62 個の測定方法を 21 個の製品に適用した結果，57 個の測定方法について 1 個以上の製品で測定可能であった(表 2-2)．このことから，評価枠組みは一定の網羅性を確保していると考えられる．しかし，全体(21 製品×62 測定方法)のうちの測定率は 36%にとどまった．未測定の原因の多くは，根拠となるデータの開発時の未記録であった．

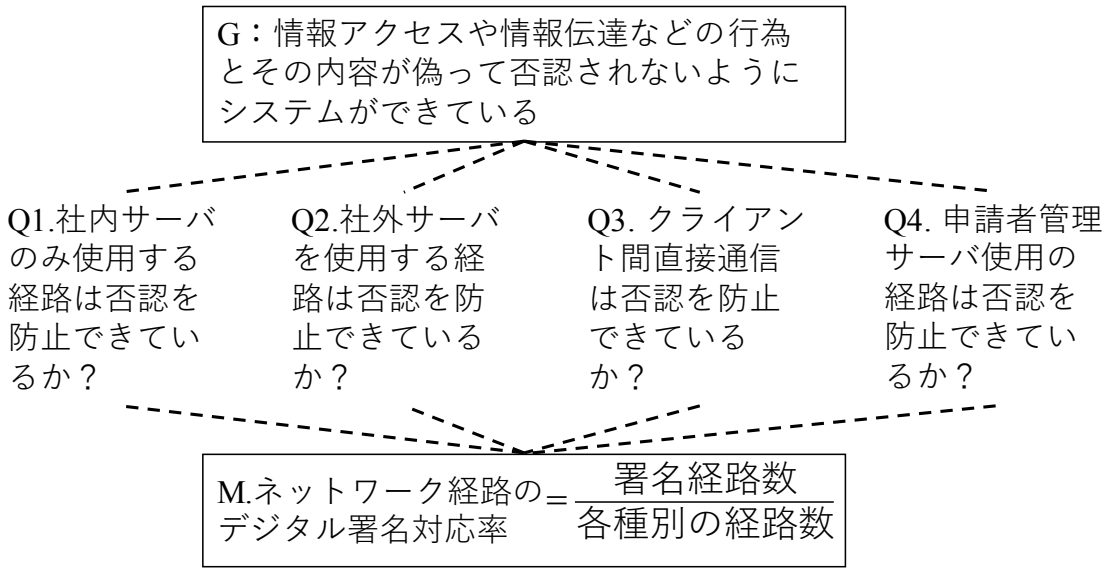

図 2-2：GQM 法の適用例(G:目標，Q:質問，M:測定法)(測定方法は SNo.1.1)

表 2-1：製品品質の測定方法の一覧（Y は 1 製品以上で可測を示す）

ID	可測	品質特性	副特性	測定方法
FCp.1.1	Y	機能適合性	機能完全性	要求実装率
FCr.1.1	Y	機能適合性	機能正確性	深刻不具合除去率
FAp.1.1	Y	機能適合性	機能適切性	システム試験数目標達成率
FAp.1.2	Y	機能適合性	機能適切性	ユーザの意図に則す度合い
PTb.0.1	Y	性能効率性	時間効率性	時間効率性の試験有無
PTb.1.1	Y	性能効率性	時間効率性	応答時間　平均
PTb.2.1	Y	性能効率性	時間効率性	応答時間　実測対目標

ID	可測	品質特性	副特性	測定方法
PTb.3.1	Y	性能効率性	時間効率性	ターンアラウンドタイム　平均
PTb.4.1	Y	性能効率性	時間効率性	ターンアラウンドタイム　実測対目標
PTb.5.1	Y	性能効率性	時間効率性	スループット　目標達成率
PRu.0.1	Y	性能効率性	資源効率性	資源効率性の試験有無
PRu.1.1	Y	性能効率性	資源効率性	CPU 使用率　最大値
PRu.2.1	N	性能効率性	資源効率性	メモリ 使用率　最大値
PCa.0.1	Y	性能効率性	容量満足性	容量満足性の試験有無
PCa.2.1	Y	性能効率性	容量満足性	ユーザ同時アクセス可能数　目標達成率
CCo.1.1	Y	互換性	共存性	他製品を共存させて試験する意図の有無
CIn.1.1	Y	互換性	相互運用性	ファイル形式のインポート/ エクスポート両対応率
UAp.2.1	Y	使用性	適切度認識性	機能の動画説明　対応率
ULe.1.1	Y	使用性	習得性	機能の説明　記載率カタログ

ID	可測	品質特性	副特性	測定方法
ULe1.2	Y	使用性	習得性	機能の説明　記載率 マニュアル
UOp.6.1	Y	使用性	運用操作性	機能の Undo 対応率
UEp.1.1	Y	使用性	ユーザエラー防止性	機能での入力内容チェック　対応率
UIn.1.1	Y	使用性	UI 快美性	UI の使いやすさ度合い
UAc.3.1	Y	使用性	アクセシビリティ	機能の聴覚ハンディキャップ配慮率
UAc.4.1	Y	使用性	アクセシビリティ	機能の視覚ハンディキャップ配慮率
UAc.5.1	Y	使用性	アクセシビリティ	言語の対応度合い
RMa.1.1	Y	信頼性	成熟性	不具合除去率（単体試験）
RMa.1.2	Y	信頼性	成熟性	不具合除去率（結合試験）
RMa.1.3	Y	信頼性	成熟性	不具合除去率（システム試験）
RMa.2.1	N	信頼性	成熟性	MTBF　目標達成率
RMa.3.1	Y	信頼性	成熟性	不具合発見率（単体試験）
RMa.3.2	Y	信頼性	成熟性	不具合発見率（結合試験）

ID	可測	品質特性	副特性	測定方法
RMa.3.3	Y	信頼性	成熟性	不具合発見率（システム試験）
RMa.3.4	Y	信頼性	成熟性	不具合発見率（チケットベース）
RMa.4.1	Y	信頼性	成熟性	試験実施率　（結合試験）
RMa.4.2	Y	信頼性	成熟性	試験実施率　（システム試験）
RAv.0.1	Y	信頼性	可用性	運用試験実績
RAv.1.1	Y	信頼性	可用性	運用実時間　対　規定時間
RAv.2.1	N	信頼性	可用性	システムダウン時間　実際対目標
RFt.1.1	Y	信頼性	障害許容性	fault-pattern テストケース(結合試験)
RFt.1.2	Y	信頼性	障害許容性	fault-pattern テストケース(システム試験)
RRe.1.1	N	信頼性	回復性	システムダウン回復時間　実際対目標
SCo.1.1	Y	セキュリティ	機密性	データのアクセス権限管理　対応率
SCo.2.1	Y	セキュリティ	機密性	データの暗号化　対応率

ID	可測	品質特性	副特性	測定方法
SIn2.1	Y	セキュリティ	インテグリティ	データの破損防止策対応率
SNo.1.1	Y	セキュリティ	否認防止性	ネットワーク経路のデジタル署名対応率
SAc.1.1	Y	セキュリティ	責任追跡性	データのアクセスログ対応率
SAu.1.1	Y	セキュリティ	真正性	ログイン機能での認証方式　対応率
MMo.1.1	Y	保守性	モジュール性	クラスの結合度
MMo.2.1	Y	保守性	モジュール性	関数のサイクロマティック複雑度
MRe.1.1	Y	保守性	再利用性	クラスの凝集性の欠如
MAn.1.1	Y	保守性	解析性	データのアクセスログ対応率
MMd.3.1	Y	保守性	修正性	不具合除去率（単体試験で発見分）
MMd.3.2	Y	保守性	修正性	不具合除去率（結合試験で発見分）
MMd.3.3	Y	保守性	修正性	不具合除去率（システム試験で発見分）
MTe.1.1	Y	保守性	試験性	モジュールの単体試験実施率
PAd.0.1	Y	移植性	適応性	複数環境の試験有無

ID	可測	品質特性	副特性	測定方法
PIn.0.1	Y	移植性	設置性	インストールの試験有無
PIn.1.1	N	移植性	設置性	インストール時間　平均
PIn.2.1	Y	移植性	設置性	インストーラ提供形態対応率
PIn.2.2	Y	移植性	設置性	インストールオプション　対応率
PRe.1.1	Y	移植性	置換性	追加学習必要度合い

表 2-2：製品品質の測定方法の測定状況

品質特性	(Y)1 製品以上で可測	(N)全製品で測定不可
機能適合性	4	0
性能効率性	10	1
互換性	2	0
使用性	9	0
信頼性	13	3
セキュリティ	6	0
保守性	8	0
移植性	5	1

2.3.2. 利用時の品質の測定方法

利用時の品質について，その品質副特性を網羅するように合計で 17 個の測定方法を定義した．測定方法の一覧を表 2-3 に示した．

測定方法の具体化にあたり，「ISO/IEC 25010:2011」の品質特性を網羅的に評価可能とすることを目的とし，「ISO/IEC 25022:2016」における抽象的な測

定量を初期候補とした．そして，ユーザテストの実施方法，およびユーザアンケートの様式，測定に必要なデータの記入様式を策定した．その過程では，「ISO/IEC 25051:2014」や PSQ 認証制度を参考とし，SQuaRE およびその JIS 化を担当した国内の委員から助言を得た．

- ユーザテストには有効性と効率性が該当しており，測定量の多くは利用者の操作に基づいていた．本研究では，実際の利用者に代わりに我々がテストを実施した．限られた時間内で効率よく製品をテストするために，製品開発元が機能一覧と正常系テスト項目を用意し，我々が異常系テスト項目を用意した．製品の主要機能の選定時や，我々が対象領域やテスト項目への理解を深める際には，外部評価機関の支援を受けた．
- ユーザアンケートには満足性，リスク回避性，および利用状況網羅性が該当しており，測定量の多くは利用者の実感に基づいていた．本研究では，その測定のために，我々が共通形式のアンケート様式を作成し，製品開発元から実際の利用者へ配布および回収した．アンケートの雛型は表 2-5 に示した．

品質の測定方法については，GQM 法を用いて検討を進めた．結果として，「ISO/IEC 25022:2016」の測定量のうち一部を対象外としつつも，全ての副特性について 1 個以上の測定方法を定義した．

17 個の測定方法を 21 個の製品に適用した結果，全ての測定方法について 1 個以上の製品で測定可能であった(表 2-4)．このことから，評価枠組みは一定の網羅性を確保していると考えられる．しかし，全体(21 製品×17 測定方法)のうちの測定率は 24%にとどまった．未測定の原因として，製品開発元の都合等によりユーザテストとユーザアンケートを実施できない場合が多かったことが挙げられる．

表 2-3：利用時の品質の測定方法の一覧（Y は 1 製品以上で可測を示す）

ID	可測	品質特性	副特性	測定方法
Ef.1.1	Y	有効性	(なし)	タスク完了率
Ef.3.1	Y	有効性	(なし)	タスク当たりエラー数
Ef.4.1	Y	有効性	(なし)	エラーが発生したタスクの率
Ef.5.1	Y	有効性	(なし)	エラーを起こした被験者の率
Ey.1.1	Y	効率性	(なし)	タスクにかかった時間の平均

ID	可測	品質特性	副特性	測定方法
Ey.5.1	Y	効率性	(なし)	タスク中の総アクションの無駄でないアクションの率
SUs.1.1	Y	満足性	実用性	製品に対する満足度
SUs.1.2	Y	満足性	実用性	ネットプロモータースコア
SUs.2.1	Y	満足性	実用性	機能に対する満足度
STr.1.1	Y	満足性	信用性	信用度合い
SPl.1.1	Y	満足性	快感性	快感度合い
SCo.1.1	Y	満足性	快適性	快適度合い
REc.0.1	Y	リスク回避性	経済リスク緩和性	経済的損失の無さ
RHe.0.1	Y	リスク回避性	健康・安全リスク緩和性	健康や人命への影響の無さ
REn.0.1	Y	リスク回避性	環境リスク緩和性	環境への影響の無さ
CCm.0.1	Y	利用状況網羅性	利用状況完全性	主要な目的以外での製品利用
CFl.0.1	Y	利用状況網羅性	柔軟性	非主要目的での製品利用時タスク達成度合い

表 2-4：利用時の品質の測定方法の測定状況

品質特性	(Y)1 製品以上で可測	(N)全製品で測定不可
有効性	4	0
効率性	2	0
満足性	6	0
リスク回避性	3	0
利用状況網羅性	2	0

表 2-5：ユーザアンケートの雛形

回答方法：設問に対して，あなたがもっともふさわしいと感じる選択肢に「○」をつけてください．（　　）についてはお感じになられていることをできるだけ具体的にご記入ください．	
Q1	この製品に対する**満足度**はどのくらいですか
	1: 満足していない　2. あまり満足していない　3. 少し満足していない 4. 少し満足している　5. そこそこ満足している　6. 満足している
Q1.1	この製品をほかの人に薦めたいと思いますか
	1:全く思わない ～ 5.どちらともいえない ～ 10:非常にそう思う
Q2	この製品において実用上問題を感じることがありますか
	1: 頻繁に感じる　2. しばしば感じる　3. あまり感じない　4: 感じない
Q3	この製品の利用目的は達成できましたか
	1: できなかった　2.あまりできなかった　3. 少しできた　4: できた
この製品の各機能に関して以下の設問にお答えください	
Q4.1	当製品の機能「　　」を利用したことはありますか
	1. 毎日利用　2. 週 2～3 回利用　3. 月 2～3 回利用　4. 年数回利用 5. 利用したことはない
Q4.1.1	機能「　　」を利用したことがある人のみお答えください． 機能 A の満足度はどのくらいですか?
	1: 満足していない　2. あまり満足していない　3. 少し満足していない 4. 少し満足している　5. そこそこ満足している　6. 満足している
Q4.2	当製品の機能「　　」を利用したことはありますか
	1. 毎日利用　2. 週 2～3 回利用　3. 月 2～3 回利用　4. 年数回利用 5. 利用したことはない
Q4.2.	機能「　　」を利用したことがある人のみお答えください．

1	機能 B の満足度はどのくらいですか?
	1: 満足していない　2. あまり満足していない　3. 少し満足していない 4. 少し満足している　5. そこそこ満足している　6. 満足している
この製品の**信用性**についてお聞かせください. ここでいう製品の信用性とは,ユーザーが意図した通りの動作・結果を得られることの度合のことを意味します.	
Q5.1	この製品はあなたが意図した動作を行いますか
	1: そう思わない　2. あまりそう思わない　3. 少しそう思う 4: そう思う
Q5.2	この製品の動作の結果は,あなたの意図に則していますか
	1: そう思わない　2. あまりそう思わない　3. 少しそう思う 4: そう思う
Q5.3	この製品を利用するにあたり,セキュリティに関する問題(例えば情報漏えいや攻撃に関する危険,不正なアクセス権設定)を感じたことがありますか
	1: 頻繁に感じる　2. しばしば感じる　3. あまり感じない 4:感じない
以下について,この製品を利用する際に感じる度合いをお聞かせください	
Q6.1	この製品を使うことに夢中になる
	1. 感じる　2. 少し感じる　3. あまり感じない　4. 感じない
Q6.2	この製品を使っていてイライラする
	1. 感じる　2. 少し感じる　3. あまり感じない　4. 感じない
Q6.3	この製品を使うことにより,創造性がかきたてられる
	1. 感じる　2. 少し感じる　3. あまり感じない　4. 感じない
Q6.4	この製品を使うのは難しい
	1. 感じる　2. 少し感じる　3. あまり感じない　4. 感じない
Q6.5	この製品を使うことで業務に対するモチベーションが上がった
	1. 感じる　2. 少し感じる　3. あまり感じない　4. 感

	じない
Q6.6	この製品を使っていて，使い方や機能に不安を感じる
	1. 感じる　　2. 少し感じる　　3. あまり感じない　　4. 感じない
この製品を使う際の疲れ具合について教えてください	
Q7.1	目に疲れは感じますか
	1.使用するたび感じる　　2. たまに感じる　　3.あまり感じない 4. 感じない　　5.覚えていない
Q7.1.1	どのような時に感じますか
	（ ）
Q7.2	手や腕の疲れは感じますか
	1.使用するたび感じる　　2. たまに感じる　　3.あまり感じない 4. 感じない　　5.覚えていない
Q7.1.2	どのような時に感じますか
	（ ）
利用状況・環境についてお答えください	
Q8.1	この製品の主な利用目的はなんですか
	（ ）
Q8.2	上記以外の目的でこの製品を利用したことはありますか?ある場合は具体的にどんな目的で利用したか教えてください
	（ ）
Q8.2.1	Q8.2 で Yes と答えた方のみお答えください. Q8.2 でお答えいただいた利用目的はこの製品でどのくらい達成できましたか
	1.達成できなかった　　2. あまり達成できなかった　　3. 少し達成できた 4.達成できた
Q8.3	この製品を利用するときにマニュアルを利用のたびに見る必要があったなど追加学習が必要でしたか?
	1. 必要だった　　2. 少し必要だった　　3. どちらかといえば必要

	だった 4. どちらかといえば必要ではなかった　　5. あまり必要ではなかった 6. 必要ではなかった	
Q9	この製品を利用するにあたって，以下について感じたことがありますか? 感じたことがある場合，その頻度はどのくらいですか?	
	経済的損失(不正な財務状況の提示，情報漏えい など)	1. 利用の度感じる　　2. 週 2～3 回感じる 3. 月 2～3 回感じる　　4. 年数回感じる 5. 感じたことはない
	健康への影響(頭痛や肩こり など)	1. 利用の度感じる　　2. 週 2～3 回感じる 3. 月 2～3 回感じる　　4. 年数回感じる 5. 感じたことはない
	人命や身体の危険（カルテ登録ミスや誤ったシミュレーション結果を出力した など)	1. 利用の度感じる　　2. 週 2～3 回感じる 3. 月 2～3 回感じる　　4. 年数回感じる 5. 感じたことはない
	環境への影響（異常量の二酸化炭素の排出,消費電力の増加 など)	1. 利用の度感じる　　2. 週 2～3 回感じる 3. 月 2～3 回感じる　　4. 年数回感じる 5. 感じたことはない
Q-In-1	この製品のメニューやボタンなどのユーザーインターフェースについて使いやすいと思いますか	
	1. 感じる　　2. 少し感じる　　3. どちらともいえない 4. あまり感じない　　5. 感じない	
Q-In-1.1	(Q-In-1 で 4 or 5 と答えた方のみお答えください) 具体的にどのような部分が使いにくいですか	
	A. レイアウト要素（フォント，画面（ウインドウやパネル)，ボタン，メニュー 等) （)	

	B. カスタマイズ要素（色，サイズ，配置&並び 等） （ ）

2.3.3. 測定値の評価方法

測定値の評価プロセスを図 2-3 に示す．まず，全ての測定方法の測定値をベンチマークの製品群に応じる形でスコア化する．次に，この測定方法のスコアを関連する副特性の単位で集約する．最後に，副特性のスコアを関連する品質特性の単位で集約する．

測定値のスコア化においては,測定方法ごとの値域の違いによらず一定の妥当性で測定値を正規化することを目的として,パーセンタイルのランク値を用いた．ランク値は測定値の分布における順序を 0 から 1 の数値で表す．例えば，上位 30%に位置する測定値のランク値は 0.7 となる(図 2-4)．

また，測定方法によって，値の大小のどちらが望ましいかが異なる．本研究では，小さい方が望ましい測定方法については数え出しの順序を逆転し，下位 30%の位置でランク値が 0.7 になるようにした．参考として，双方の例を以下に示す．

- セキュリティにおける，通信経路がデジタル署名によって改ざんから保護されている割合を示す「SNo.1.1 ネットワーク経路のデジタル署名対応率」は大きいほうが望ましい．
- 保守性における，関数の制御フローグラフにおける線型独立な経路数を示す「MMo.2.1 関数のサイクロマティック複雑度」は同じ要求を満足する限り小さい方が望ましい．

また，ランク値に関する制約を以下に示す．これらは今後の課題である．

- 測定値が特定の範囲に収まるほど望ましい場合を考慮していない．
- ランク値は基本的には 0 から 1 の間の数値になるが，分布に最大値をとる測定値が複数存在する場合には,上位 100%に位置する値が不在となるため，ランク値の最大値が 1 にならない．

スコアの集約方法を以下に示す．

- 副特性のスコア：その副特性に関連する測定値のスコアの平均値とする．
- 品質特性のスコア：まず，その品質特性に関連する副特性のスコアの平

均値(x とする)をとる．x の計算は製品単位で実施する．そして，ベンチマークの製品群に応じる形で x を正規化したものを，その製品の品質特性のスコアとする．

- 正規化の方法：ある品質特性について考える．ベンチマークの製品群において計算した x の値の集合を X とする．X の最小値と最大値は 0 と 1 になるとは限らない．最終的な評価結果のわかりやすさを重視して，最小値と最大値が 0 と 1 になるように X を変換したものを，その品質特性のスコアの集合 newX とする．この変換は以下の式により実現できる．

$$\mathrm{newX} = \frac{\mathrm{X} - \min(\mathrm{X})}{\max(\mathrm{X}) - \min(\mathrm{X})}$$

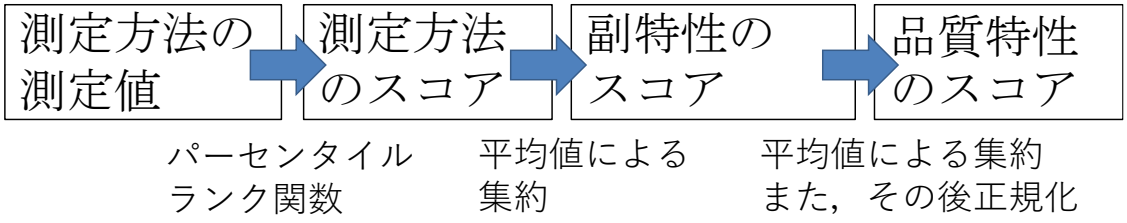

図 2-3：スコアの計算過程

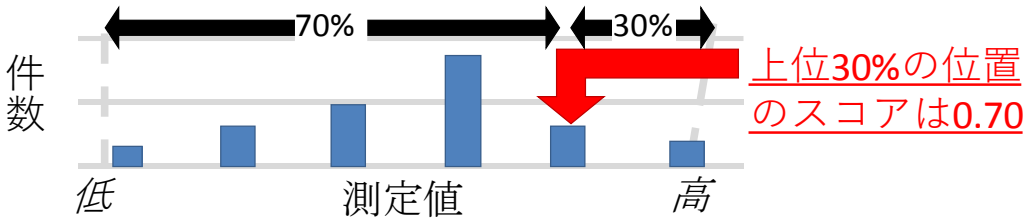

図 2-4：パーセンタイルランク関数による測定値のスコア化

2.4. 実態調査

製品や組織を超えた品質の実態が不明であることが,品質向上の取り組みの効果的な検討と実施の妨げとなっている．そこで我々は CSAJ の協力を経て様々な製品開発元から 21 個のソフトウェア製品(表 2-6)について枠組みを適用し，その実態調査をした．分析した事項を以下に示す．詳細は以降の節で述べる．

- 品質特性別の傾向．
- 品質特性間の関係．
- 製品品質と利用時の品質間の関係．
- 製品の欠陥発見パターンと品質特性の関係．
- 製品のコンテキストと品質特性の関係．

表 2-6：パッケージ製品数およびクラウド製品数のドメイン別集計

製品ドメイン	パッケージ製品数	クラウド製品数
グループ支援	4	1
データ集計	5	-
会計	3	1
セキュリティ	2	1
数値計算シミュレーション	3	-
エンドユーザ向けサービス	-	1

ユーザテストによる測定は，9 個の製品で実施できた．そのうち，7 個はパッケージ製品，2 個はクラウド製品であった．また，製品ドメインについて，以下の内訳であった．

・グループ支援のパッケージ製品が 2 個．
・データ集計のパッケージ製品が 2 個．
・会計のパッケージ製品が 1 個．
・セキュリティのパッケージ製品が 1 個，クラウド製品が 1 個．
・数値計算シミュレーションのパッケージ製品が 1 個．
・エンドユーザ向けサービスのクラウド製品が 1 個．

ユーザアンケートによる測定は 3 個のパッケージ製品で実施できた．製品ドメインについて,1 製品はグループ支援製品.2 製品はデータ集計であった．

・あるグループ支援製品では，7 社のユーザ組織から 7 名分の回答を得た．
・あるデータ集計製品では，16 社のユーザ組織から 16 名分の回答を得た．
・あるデータ集計製品では，7 社のユーザ組織から 7 名分の回答を得た．

2.4.1. 品質特性別の傾向

品質特性スコアの分布を図 2-5 に示し，それぞれの結果と考察を以下に述べる．なお，図に表示されている表記 Pa および Pb に関する情報については以降の節で述べるため，本節には関係しない．

機能適合性，性能効率性，信頼性，移植性，効率性：

- 結果）なだらかに広がる分布形状から，これらの品質の程度が製品により異なることが分かる．
- 考察）製品ごとの異なる品質要求に応じた結果と推測でき，自然なものと捉えられる．

互換性：

- 結果）スコアが低い群と高い群に二極化している．データ交換などの互換性に通じる仕組みを一部の製品において考慮していないことが原因として挙げられる．ただし，SQuaRE から具体化できた測定法が 2 個にとどまっており，その影響も原因として考えられる．
- 考察）互換性は品質モデルが「ISO/IEC 9126-1:2001」から「ISO/IEC 25010:2011」へ改訂される際に品質特性として格上げされたものである．国際規格側について，具体化して実適用可能な測定量の拡充が望まれる．

使用性：

- 結果）スコアが低い方に製品が集中している．
- 考察）原因としては，使用性を十分に考慮できていないこと，あるいはエンドユーザ対象ではないといったことから意図的に考慮していないことが考えられる．スコアの低い製品については，品質要求との対応関係を確認し，意図通りでない場合は機能の使用性を改善する取り組みが必要と考えられる．

セキュリティ：

- 結果）スコアが低い群と高い群に二極化している．一部の製品で，暗号化や破損防止などの高セキュリティ化に通じる仕組みを考慮していないことが原因である．
- 考察）近年のソフトウェア開発においては，企画時に他製品との接続を想定していない製品でも保守や派生の中で接続が求められる可能性がある．そのため，製品に必要なセキュリティについて慎重な検討が必要と考えられる．

保守性：

- 結果）スコアが低い方に広範囲で製品が集中している．
- 考察）全体的に保守性を十分に考慮できていない製品が多いと考えられる．製品寿命や品質要求との対応関係を確認し，意図通りでない場合は設計・実装上の複雑さを抑えるといった保守性向上の取り組みが必要と考えられる．

有効性：

- 結果）スコアが低い群と高い群に二極化している．ユーザテスト実施時にタスクを達成しにくい製品が一部見られた．
- 考察）製品の価値を判断する立場は利用者にあるため，利用時の品質が低い製品についてはユーザ中心の製品デザインといった利用者視点のタスク実行のしやすさの考慮が必要と考えられる．

満足性，リスク回避性，利用状況網羅性：

- 結果）測定評価できた製品数が 3 製品と極めて限られたため，意味のある傾向を得られなかった．
- 考察）多くの製品について開発者視点の製品品質が重視される一方で，利用者視点の利用時の品質の把握が困難な状況にある可能性がある．これらの品質特性の測定評価には，ユーザアンケートのような利用者の実感を直接的に把握する取り組みが不可欠である．

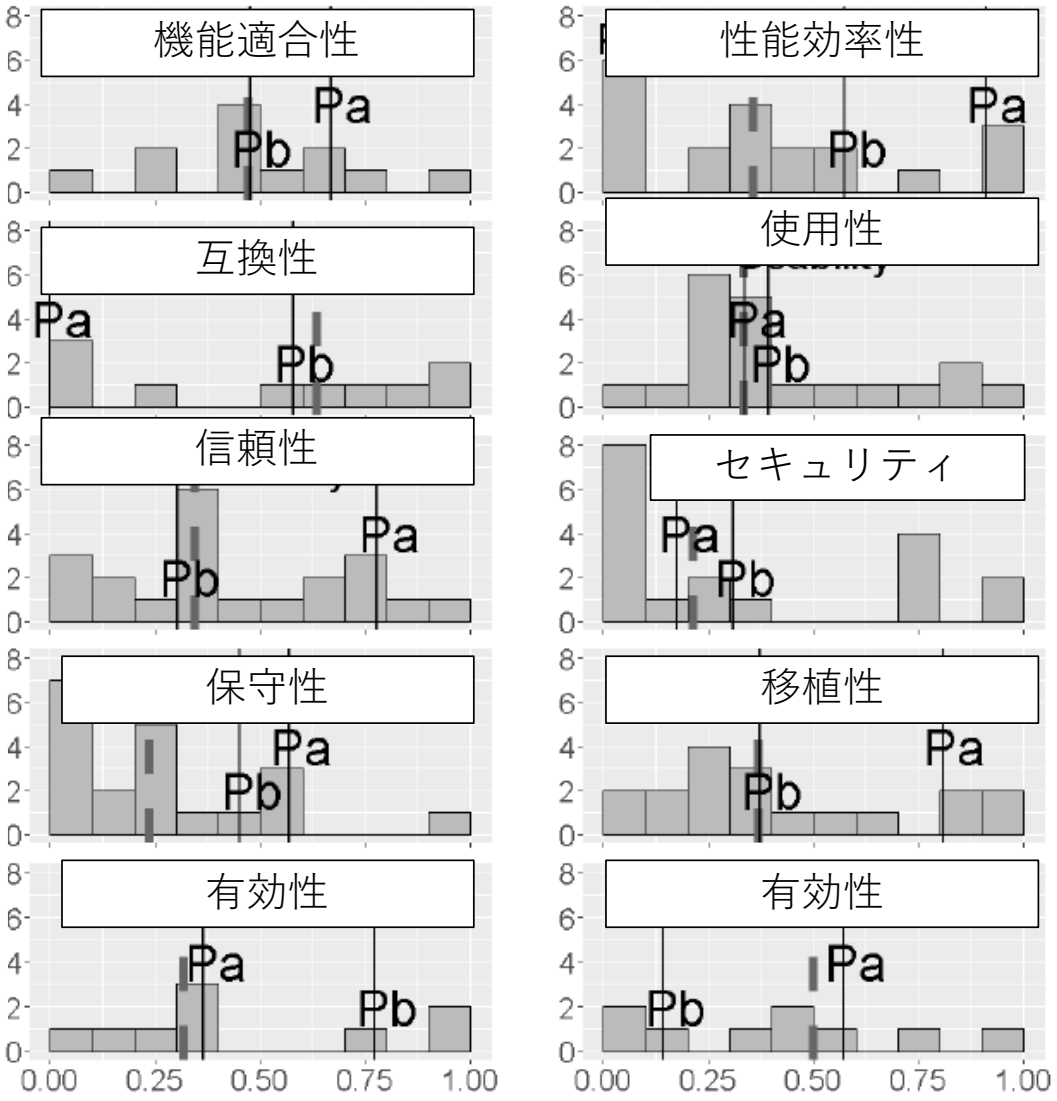

図 2-5：品質特性別の傾向(点線は中央値，Pa と Pb は着目する二製品の位置)

2.4.2. 製品開発元からのフィードバック

二社の製品開発元から 2 個の製品(Pa と Pb)について，評価枠組みを適用した結果に関するフィードバックを受け取った.品質評価の結果がおおむね開発状況と一致し，製品開発元での改善につながったため，評価枠組みはある程度有効であったと考えられる.

製品開発元には診断レポートの一部として,品質特性スコアの分布に製品のポジション表示を加えた図を提供している．Pa と Pb のポジション表示を加えたものを図 2-5 に示す．製品開発元からのフィードバック内容について分析した結果を以下に示す.

評価結果と開発者の意図の一致する箇所について：

両方の製品について，優先度が高い品質特性ではスコアも高かった.

Pa の性能効率性について，スコアが中央値よりも高いが，製品開発元での優先度も高かった．この結果は開発者の意図と一致しており，Pa ではデータ処理機能の強化のために性能効率性を最優先事項としていた.

Pa のセキュリティについて，スコアが中央値よりも低いが，製品開発元での優先度も低かった.この結果は開発者の意図と一致しており，セキュリティ技術の一部のみが使用されていた.

Pb のセキュリティと有効性について，スコアが中央値よりも高いが，製品開発元での優先度も高かった.この結果は開発者の意図と一致しており，クラウドサービスにより競合する製品との差別化が強化されていた.

Pa と Pb の互換性について，スコアが中央値よりも低いが，製品開発元での優先度も低かった.この結果は開発者の意図と一致しており，システムの動作条件(OS 等)の制限による開発の高速化を意図していた.

評価結果と開発者の意図が異なる箇所について：

Pb の信頼性について，製品開発元での優先度は高いものの，スコアは低かった．評価枠組みによって示されたこのギャップは，Pb の製品開発元において暗黙的な問題の発見に繋がった.信頼性について副特性単位でスコアを確認した結果，成熟性のスコアが低い原因として，試験密度と障害検出率について目標レベルが達成されていなかったことがわかった．そして，開発チームは，プラットフォームごとのテスト環境が高条件ではないことを発見した.

Pa と Pb の移植性について，製品開発元での優先度は低いものの，スコアは高かった.評価枠組みによって示されたこのギャップの原因を調べるために,我々はベンチマークを分析して移植性の副特性の傾向を確認した.ほとんどの

製品（18/21）では置換性が測定されておらず，また製品間での適応性のスコアの差が非常に小さい（IQR == 0）．そのため，移植性のスコアに対して設置性が支配的な影響を与えている．その結果，Pa と Pb，様々なインストールオプションを提供している点が測定方法 PIn.2.2 によって高評価され，移植性において高いスコアを得ていた．今後の課題としては，このような，製品間で測定不能または違いの少ない品質副特性を特定し，評価枠組みとして測定方法を強化することが考えられる．

2.4.3. 品質特性間の相関関係

21 製品のうち測定評価できたものについて品質特性スコア間の相関関係を図 2-6 に示す．図に記載した数値はスピアマンの順位相関係数の大きさである．また，p 値が低いものは偶然に高い相関係数が算出された可能性が低い（つまり統計的に有意）と考えられるため，アスタリスクを記載して区別した．アスタリスクの個数については，p 値が 0.05 未満かつ 0.01 以上の場合には 1 個，0.01 未満の場合には 2 個とした．

統計的に有意と見られた関係についての結果と考察を以下に述べる．

機能適合性と使用性：

- 結果）負相関．機能適合性が高いほど，使用性が低い傾向にあった．相関係数は-0.73**であった．
- 考察）原因としていくつかの可能性が考えられる．
  - 製品が当初企画した通りの機能仕様の満足を重視した結果として，副作用として利用者目線での使いやすさを損なってしまった可能性がある．
  - 利用者目線での使いやすさを重視した結果として，一部の機能を作り込めなかった可能性がある．
  - エンドユーザ向けの側面よりも機能性を重視する製品では，品質要求として使用性を重視していない可能性がある．

移植性と使用性の正相関について：

- 結果）正相関．移植性が高いほど，使用性が高い傾向にあった．相関係数は 0.47*であった．
- 考察）移植性の高い製品は様々な環境に向けて構築されているため，移植先の検討に併せて多面的な観点から実現する機能が見直される場合がある．その結果，高い使用性の製品を作り込むことに繋がった可能性がある．

なお,相関関係が因果関係とは必ずしも一致しない点には留意する必要がある．特定の品質特性スコアの増減によって，意図的に別の品質特性スコアを増減できるとは限らない．本節の考察は相関関係の原因の推察にとどまる．

	性.	互.	使用.	信.	セ.	保.	移.	有.	効.
機能適合性	0.27	0.19	-0.73**	0.35	-0.05	0.47	0.25	-0.23	0.49
性能効率性		0.44	0.24	0.36	-0.17	0.36	0.33	0.32	-0.10
互換性			0.04	0.15	-0.06	0.36	-0.07	-0.14	0.05
使用性				0.16	-0.25	0.11	0.47*	-0.13	-0.13
信頼性					0.30	0.38	0.42	-0.08	0.11
セキュリティ						-0.06	0.20	0.64	-0.34
保守性							0.25	-0.29	0.01
移植性								-0.17	0.63
有効性									0.03
効率性									

図 2-6：品質特性スコア間の相関係数(スピアマン順位相関)

2.4.4. 品質特性間の偏相関関係

相関関係が示された品質特性の組み合わせについて,一次偏相関の分析をした．一次偏相関分析では，2 個の変数の関係について，他の変数 1 個からの影響を制御(除去)する．その結果，以下の関係が擬似相関であると示された．

機能適合性と使用性：

- 結果）擬似相関関係．有効性と効率性のいずれかを制御した場合，機能的適合性と使用性の間に統計的に有意な相関はなかった．
- 考察）利用時の品質を交絡要因とした擬似相関関係に見えるが，有効性と効率性は直接的な交絡要因ではないと考えられる．図 2-6 の通り，有効性と効率性は機能適合性と使用性のいずれとも相関関係を持たない．そのため，これらふたつの利用時の品質特性とふたつの製品品質特性を仲介する隠れた交絡要因が存在する可能性がある．

交絡要因の可能性として，有効性と効率性，および機能適合性と使用性に関する要素について述べる．

本研究では，有効性と効率性はユーザテストによって測定されており，以下のドキュメントに基づいてテスト項目を準備していた．

- ソフトウェア製品の機能のリスト．
- ソフトウェア製品のユーザマニュアル．
- ソフトウェア製品のテストケースの仕様書

上記のドキュメントの成熟度と複雑度は，ユーザテストのパフォーマンスや製品の機能的適合性と使用性に影響している可能性がある．推測される問題は，これらのドキュメントの分析の難しさであり，通常は組織を越えて統一された形式で記録されていないと考えられる．

これらのドキュメントの形式のある程度統一については，製品品質と利用時の品質の間の交絡要因の分析に役立つ可能性のある要素として，今後の調査課題のひとつとする．

移植性と使用性：

- 結果）擬似相関関係．ほとんどの品質特性のいずれかを制御した場合，移植性と使用性の間に統計的に有意な相関はなかった．機能的適合性，性能効率性，互換性，信頼性，有効性，効率性(つまりセキュリティと保守性以外)が影響した．
- 考察）多数の交絡変数があり，擬似相関の原因の特定が困難である．

セキュリティと保守性のみが交絡要因とならなかった．擬似相関に関する直接的な考察とはならないが，その原因について以下に考察を述べる．

- セキュリティについては SQuaRE が意図する方針で開発されていない製品が存在し，それが品質特性スコア同士の関係分析に影響した可能性がある．SQuaRE では接続されたシステムがデータを交換する将来の IoT 時代を考慮して，データを保護するソフトウェア機能(暗号化など)を評価するように設計されている．一方で，一部の製品ではデータの保護の手段として，他製品との共存回避や，ローカルでの製品利用を推奨している．
- 保守性に移植性ついて評価される活動に違いがあり，それが品質特性スコア同士の関係分析に影響した可能性がある．ソフトウェアの保守は長期的な活動だが，一方でソフトウェアの移植とインストールは通常は一時的な活動と考えられる．

2.4.5. 欠陥発見状況タイプと品質特性

欠陥票の時系列データを得られた 9 製品について信頼性成長モデルを適用した．その結果，欠陥発見状況は安定タイプが 3 製品，漸増タイプが 3 製品，爆発タイプが 3 製品であった．各タイプの性質を以下に述べる．

- 安定タイプ）十分な欠陥を発見できていると考えられる．
- 漸増タイプ）テストを実施するたびに欠陥が徐々に発見されており，リリース後も欠陥が発見される可能性があると考えられる．
- 爆発タイプ）今後も多くの欠陥が発見されると考えられる．

また，欠陥発見状況タイプの例を図 2-7 に示した．

欠陥発見状況タイプごとの品質特性スコアの分布を箱ひげ図で図 2-8 に示す．なお，満足性，リスク回避性，利用状況網羅性については，測定評価できた製品数が限られたため省略した．品質特性別に結果と考察を以下に述べる．

機能適合性，信頼性，有効性：

- 結果）安定タイプにおいてスコアが比較的高い．
- 考察）機能適合性，信頼性，有効性が高い製品では，開発の過程で十分なテストが実施され，安定的に欠陥が発見されていると考えられる．

性能効率性，互換性：

- 結果）爆発タイプにおいてスコアが比較的低い．
- 考察）性能効率性，互換性が高い製品では，試験環境の都合からテストで十分に欠陥を発見できない可能性があり，今後も欠陥が発見される可能性がある．

他の品質特性：

- 結果）顕著な違いは見られなかった．
- 考察）対象とした製品数の数が限られていることも原因と考えられる．

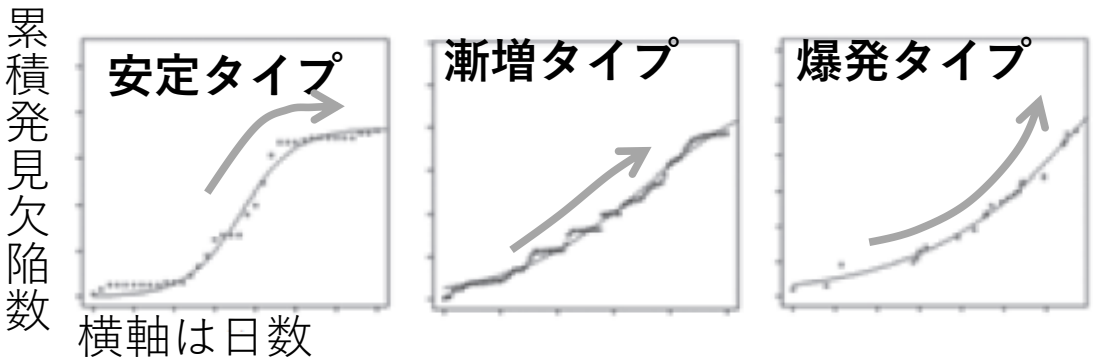

図 2-7：欠陥発見状況タイプの例

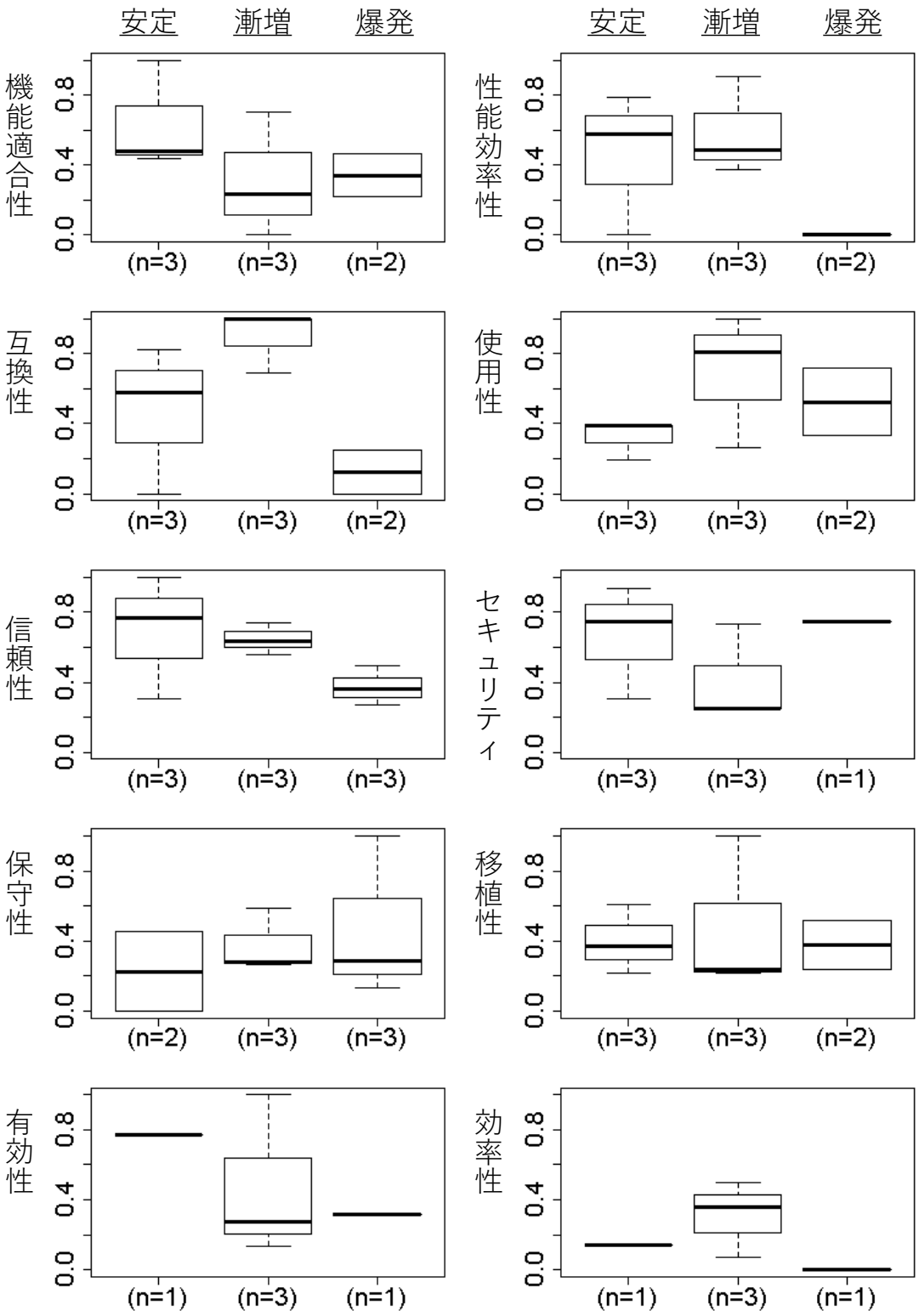

図 2-8：発見状況タイプで層別した品質特性スコアの箱ひげ図

2.4.6. プロジェクトのコンテキストと品質特性

製品のコンテキスト情報として，ドメイン・対象開発期間・開発形態・提供種別･機能数･プログラムソースコードの行数を製品開発元から得た.ただし，製品によっては一部の情報が得られていない.

コンテキスト情報の一部と平均スコアの関係を図 2-9 と図 2-10 に示す.コンテキストの種別のうちで顕著であった結果と考察を以下に述べる.

ドメイン別の傾向：

- 結果）互換性とセキュリティについてドメインによって平均スコアに顕著な差が見られた.エンドユーザ向けサービス製品においてセキュリティは高く，数値計算シミュレーション製品において低い結果となった.
- 考察）重視される品質特性の違いに起因すると考えられる．あるいは，具体化できた測定方法が限られていたことも原因として考えられる.

提供種別がパッケージ製品の傾向：

- 結果）パッケージ製品はクラウド製品に比べて，セキュリティ・性能効率性・有効性・効率性が低い結果となった.
- 考察）近年のソフトウェア開発においては，企画時に他製品との接続を想定していない製品でも保守や派生の中で接続が求められる可能性がある.パッケージ製品においてネットワーク接続や利用環境の変化を考慮する場合，特にセキュリティへの注意が必要であると考えられる.

提供種別がクラウド製品の傾向：

- 結果）クラウド製品についてはパッケージ製品に比べて信頼性，保守性，移植性がやや低い.
- 考察）SQuaRE の測定量でクラウド環境を明示的に考慮しておらず，結果として保守性と移植性のスコアが低かった可能性がある．これらについては，今後の対応強化が期待される.
  - SQuaRE では 1990 年代までのソフトウェア製品の形態と開発方法を念頭においており，アジャイル開発やプラットフォームとしてのクラウドに対する考慮を欠いている可能性がある.
  - クラウドでは DevOps に代表される運用体制と連携した機能拡張や，仮想化システムの標準への対応といったクラウド環境特有の考慮が必要とされている.

その他の傾向：

規模，期間，開発形態のそれぞれにおいては，品質に顕著な傾向の違いが見られなかった．

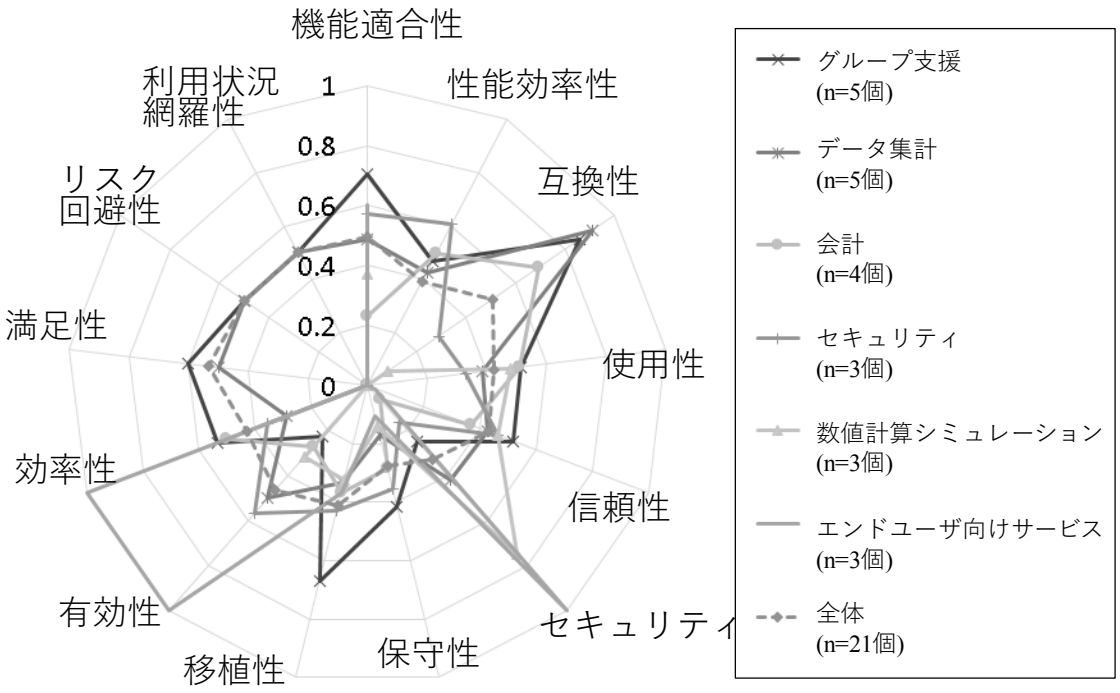

図 2-9：ドメインで層別した品質評価スコア平均

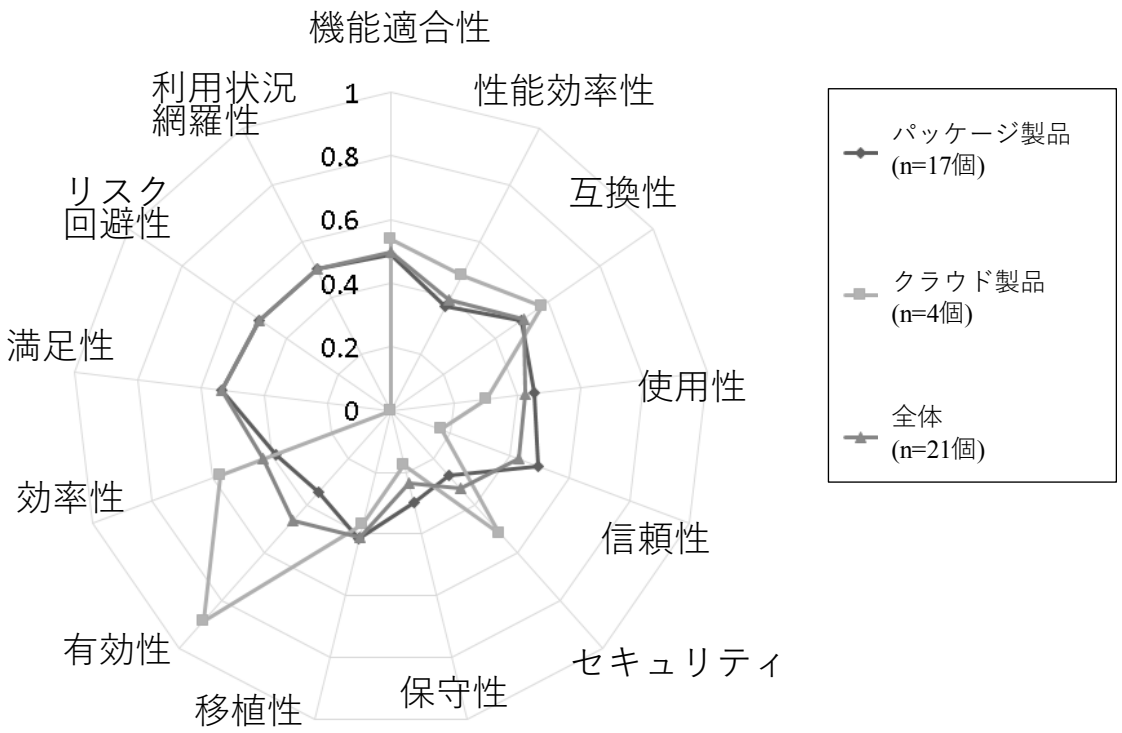

図 2-10：提供種別で層別した品質評価スコア平均

2.4.7. 結果の要約と提言

実現した評価枠組みの 21 製品への適用結果から，以下の提言をまとめた．

提言 1：

IoT/IoE 時代に重要となる品質であるセキュリティと互換性が低いパッケージ製品が一部に見られた．一部の製品ではデータの保護の手段として，他製品との共存回避や，ローカルでの製品利用を推奨している．しかし，近年のソフトウェア開発では，企画時に他製品との接続を想定していない製品でも保守や派生の中で接続が求められる可能性がある．そのため，パッケージ製品についても慎重な検討が必要と考えられる．

ソフトウェアの開発プロセスや方法，取り巻く環境が変革されつつあるため，これらの品質に対する意識の変革の必要性は増していくと考えられる．

提言 2：

利用時の品質である有効性について二極化の傾向が見られた．製品の価値を判断する立場は利用者にあるため，ユーザ中心の製品デザインといった利用者視点のタスク実行のしやすさの考慮が重要と考えられる．

提言 3：

製品品質と利用時の品質の間の関係について，一部のソフトウェアドキュメントが交絡要因となって機能的適合性，使用性，有効性，効率に影響する可能性がある．ソフトウェア製品の機能一覧，ユーザ向けマニュアル，テストケースの仕様書については追跡可能な形で記録されていることが望ましい．

また，信頼性成長モデルにより特定される欠陥発見状況が安定タイプの製品は有効性が高く，逆に漸増あるいは爆発タイプの製品は有効性が低い傾向にあった．

提言 4：

21 製品の協力を得たが，品質特性単位で見ると実際に測定評価できた製品数は半数に満たないものがほとんどであった．この原因としては，各製品開発元において，様々な品質特性の観点から評価可能とするための根拠となるデータを記録していないことが挙げられる．SQuaRE の品質測定量の一部は開発時の活動に対する目標値を参照するが，多くの製品では利用可能ではなかった．

多面的な品質測定評価を可能とするためには，試験密度や不具合発見数といった開発時の活動に関する目標値の設定が重要である．

2.5. 妥当性への脅威

内的妥当性への脅威：

相関分析の結果は，機能的適合性と使用性について負の相関関係を，使用性と移植性について正の相関関係を示した．しかし，偏相関分析の結果は，これらの相関関係が交絡効果によるものである可能性を示した．そのため，使用性の増加は常に機能的適合性の低下や移植性の向上に繋がるとは限らない．

これらの脅威の低減に向けては，擬似相関関係の交絡因子について調査することが対策として考えられる．

ユーザテストについて，我々は実際の利用者に代わって異常系テスト項目を定義し，正常系テスト項目と合わせて両方を実行した．その際には，対象領域やテスト項目への理解を深めながら進めており，我々は少なくとも初心者相当の製品利用者の視点を確保できていたと考えられる．

しかし，ユーザテスト結果には実際の利用者が持っている幅広い使用法や様々な観点が含まれない可能性がある．本研究では，製品のユーザテストを試験環境で実施していた．また，製品の操作方法の習熟度について，本研究のテスターが平均的な実際の利用者を下回っていた可能性がある．

これらの脅威の低減に向けては，各製品の実利用者もしくは一定時間をかけて熟練したテスターを準備することが対策として考えられる．

外的妥当性への脅威：

本研究では 21 個の製品群を調査対象として分析を行った．これらの限られたデータは品質の実態を明らかとする上で，一定の特徴を把握するにあたっては役立ったと考えられるが，統計的に十分な数量とは言い難い．そのため，本論文の分析結果があくまでも今回取得したデータの範囲内で推定したものである点に留意する必要がある．また，調査対象の製品群は産業界全体を代表するものではなく，主にパッケージ化されたソフトウェア等のプロバイダーが存在する領域の製品群に限定されている．

これらの脅威の低減に向けては，調査対象の製品データを拡充し，分析結果に継続的に修正を加えることが対策として考えられる．

構成概念妥当性への脅威：

品質測定方法の具体化の際には，SQuaRE 及びその JIS 化を担当した国内委員からの助言を得た．加えて，製品への実適用を通じて提案する枠組みを具体的に利用できることを確認した．また，二社の製品について枠組みと品質ベンチマークを活用できた［17］．しかし，これらは枠組みの妥当性を完全に保証するものではない．

この脅威の低減に向けては，枠組みの妥当性をより明らかとするために，枠組みが直接に扱っていない要素と品質評価結果を比較することが対策として考えられる．例えば，各製品の品質要求項目や製品リリース後の保守工数が挙げられる．

2.6. 本章のおわりに

本研究では，ソフトウェア製品の品質評価枠組み WSQF を構築し，21 製品への適用結果を分析することで，製品品質と利用時の品質の間の関係について，その限られた一端を明らかにした．そして，ベンチマークデータの分析結果に基づいて，産業界への提言をまとめた．

WSQF でソフトウェア製品の品質をベンチマークと比較可能とすることにより，開発・保守・運用あるいは運用検討時において，品質の要求定義・改善・取捨選択といった意思決定の判断材料を得られる．実際に，一部の企業では既にこの枠組みを適用し，品質特性単位の評価結果を製品の品質改善に役立てている［17］．

今回の実態調査はソフトウェア製品の品質実態の限られた一端を明らかとすることにつながったと考えられる．しかし，開発時の目標値の記録不足といった理由から，品質測定方法によってはデータを得られる製品数が限られた．

今後の課題は，更なる品質実態の調査に向けて，データ量を増やし，継続的に関係分析結果を更新することである．現在まで，2 製品の追加データを得ている．データ数が一定数に達した時点で，より包括的に品質特性間やその構成要素間の関係分析を試みる予定である．

3. ソースコードの保守性評価を対象とした特定コンテキストにおける閾値導出手法の比較

3.1. 本章のはじめに

3.1.1. 背景

ソフトウェア品質の保証は現代社会における重要課題であり [1], 開発組織においては,問題となりうる箇所を未然に把握することが欠かせない.その際,プログラムソースコード(以下コードと略す)は最終成果物として通常は必ず存在するものであり重要な判断材料となる.

コードの品質評価について,製品や組織を越えて共通に参照できる知見としては, コードスメル [37], 発展性欠陥 [38], 不可視の技術的負債 [39]などがある. これらは, 機能上の直接の不具合ではないコードの構造上の問題の分類体系であり,主にコードの保守性評価基準を作成する際の初期案として役立つ.

定量的かつ直感的に解釈可能な評価基準を作る場合,コードの規模や複雑度などのメトリクスを説明変数として,リスクの許容水準を表す閾値と組み合わせて条件式を作る. 例えば, あるメトリクス測定値がある閾値以下ならば“問題なし”とみなす. あるいは, 複数の条件の論理積(AND 結合)や論理和(OR 結合)によって評価する.

自動評価の精度については,評価対象のコードと評価基準の内容の組み合わせ次第であることが知られており, 多くの測定評価ツールでは, 利用者が任意の閾値を設定できるようにしている. そのため, 開発組織においては, プログラミング言語やアプリケーションドメインなどのコンテキストに応じた閾値を設定することで, 自動評価の精度を改善できる.

しかし, 自動評価の精度を向上させるための閾値導出手法群については, その性能と性質の違いが必ずしも十分に明らかとされていない. 結果として, 適切な手法の選択が難しく, 品質評価基準の効果的な調整の妨げとなっている.

3.1.2. 本研究の取り組み

本研究では, コードの保守性の内, 機能的な不具合を伴わないコードの構造上の問題に焦点を当て, その高精度な自動評価を目的とし, 教師あり学習を用いて評価基準を調整するための枠組みを整理した. そして, 協力企業 1 社で共通のプラットフォームで開発されているコードのデータを集積し,複数の閾値

導出手法の性能と性質を比較した.

我々の枠組みでは,まず既存の体系を参照することでコードの評価基準の初期案とし，次に重視する評価観点と測定用のメトリクスを選定する．そして,対象コンテキスト考慮して測定値の解釈を変更することで,評価基準を調整できる．また，調整においては閾値導出手法として機械学習を用いるが，複数のメトリクスの影響関係を反映することができる分類木学習アルゴリズムを用いることとした.

そして，少量の教師データにより，高精度かつ，開発者の認識に反しない評価基準を得られることが実用的であるとみなして，以下の研究課題(Research Questions: RQ)に取り組んだ.

- RQ1:我々の枠組みにより，実用的な評価基準を得られるか?
- RQ2:少量の教師データを入力とした場合でも,分類木学習により他の閾値導出手法に比べ高精度な評価基準を導出できるか?
- RQ3:教師データの量は，自動評価の精度にどう影響するか?

本研究の貢献を以下に述べる．これらの知見と枠組みにより，開発組織における品質評価の緻密化が期待される.

- コードの保守性のうち，機能的な不具合を伴わないコードの構造上の問題に焦点を当て，その高精度な自動評価を目的とし，コードメトリクスの最適な解釈と閾値を導出するための実用的な枠組みを提案した．枠組みはプログラミング言語や評価対象の粒度に依存しない形に一般化しているため，汎用的に利用することが可能である．また，測定データを用意すれば，コードの保守性以外についても枠組みを汎用的に利用することが可能である.
- 評価実験 1:建機制御用の C++組込みソフトウェア群を題材にソースコードファイル単位で我々の枠組みを適用し，適切に閾値をカスタマイズ可能か確認した.
- 評価実験 2:分類木学習アルゴリズム C5.0 と他の手法のそれぞれによる自動評価の精度を比較した．その際，学習時の教師データの数の違いが精度に与える影響も分析し，他の手法では教師データの増加に対して精度の改善が小さいことを確認した.

以降の本章の構成は以下の通りである.

- 3.2 節：技術的背景および関連研究.
- 3.3 節：品質評価枠組みの定義.
- 3.4 節：実験環境と設定.

- 3.5 節：実験 1．RQ1 の実験方法および実験結果．
- 3.6 節：実験 2．RQ2 と RQ3 の実験方法および実験結果．
- 3.7 節：考察．RQ1 から RQ3 に関する議論．
- 3.8 節：妥当性への脅威．
- 3.9 節：成果と展望のまとめ．

3.2. 技術的背景および関連研究

3.2.1. 教師データを用いない手法．

教師データを用いない閾値導出手法の長所はソースコード以外の情報が不要であり，実施の手間が少ないことである．しかしながら，一般に，コードの規模や複雑度といった構造的メトリクスの分布(特に中央値)はソフトウェアのコンテキストによって異なる [40] [41]．そして，教師データ無しでは必ずしも有用な閾値を導出できないという報告もあり [42] [43] [44]，これらの手法は自動評価の精度を重視する場合には実用性が低い．

以降，本節では代表的な手法について説明する．

ごく単純には,コードメトリック一種類毎に測定値の平均値を計算して閾値とすることもできる [19]．ただし，規模や複雑度といったコードメトリクスは一般に正規分布とならないことが知られており [45] [46]，平均値よりもパーセンタイルの方が外れ値の影響を受けにくい(ロバストである)といわれている．p パーセンタイルとは，入力したメトリクス測定値を昇順に並べて小さい値から数え上げた時に全体の p%目となる値を意味する．例えば，中央値は 50 パーセンタイルに，箱ひげ図の箱の下部と上部は 25 パーセンタイルと 75 パーセンタイルに相当する．

ソフトウェアメトリクスの分析においては，Alves らの方法(Alves 法) [18] が用いられることも多い [47] [48] [49]．Alves 法では，多くのソフトウェアにおいて頻出する値を重視し,一部のソフトウェアのみで測定される極端な値の影響の低減に取り組んでいる.内部的には通常のパーセンタイル関数とは異なり，各要素(クラスやファイル)の重み(行数等)の累積パーセンテージを，要素が属するソフトウェアを考慮しながら集計している．そして，70%，80%，90%の閾値を越えると,それぞれリスクの大きさが Moderate, High, VeryHigh とみなすと仮定している．

経験則としては 80 対 20 の法則は知名度が高く [50] [51]，80%とその近辺の 70%や 90%は利用されやすい傾向にある．あるいは，中央値(50 パーセンタイル)や，箱ひげ図における異常値境界が利用される場合もある [52] [13] [11]．

3.2.2. 教師データを用いる手法

Bender らの方法(Bender 法 [20])と ROC 曲線による方法(ROC 曲線法 [21] [53]) では，任意のメトリック一種類の測定値と教師データを入力することで閾値を導出できる．しかし，コードの良し悪しは一種のメトリックの大小のみで判断することが難しく，複数のメトリクスによる多面的な解釈が重要となる [54] [35].

Herbold らは複数メトリクスを同時に考慮する閾値導出に取り組んでいるが，一つのメトリックにつき一つの閾値を設定するモデルの解釈になっており，様々な解釈に対応した一般化には至っていない [22].

このように，これらの閾値導出手法では必ずしもメトリクス間のトレードオフや冗長性が考慮されていない．結果として忙しいエキスパートの時間を割いて教師データを用意しても効果的な機械学習ができず，自動評価の精度が飛躍的には改善されない恐れがある．

これらの手法で二つのメトリクスの閾値を設定した場合の解釈モデルの例を図 3-1 に示す．この図では二つのメトリクス Metric1 と Metric2 を軸に取り，それぞれ一個ずつ閾値を設定して点線を引いている．この解釈モデルにおいて，良い評価判定となるのは二つのメトリクス全てが閾値以下となる四角形の領域である．言い換えれば，この四角形のサイズ変更により表現できる範囲でしか，解釈モデルを調整できない．

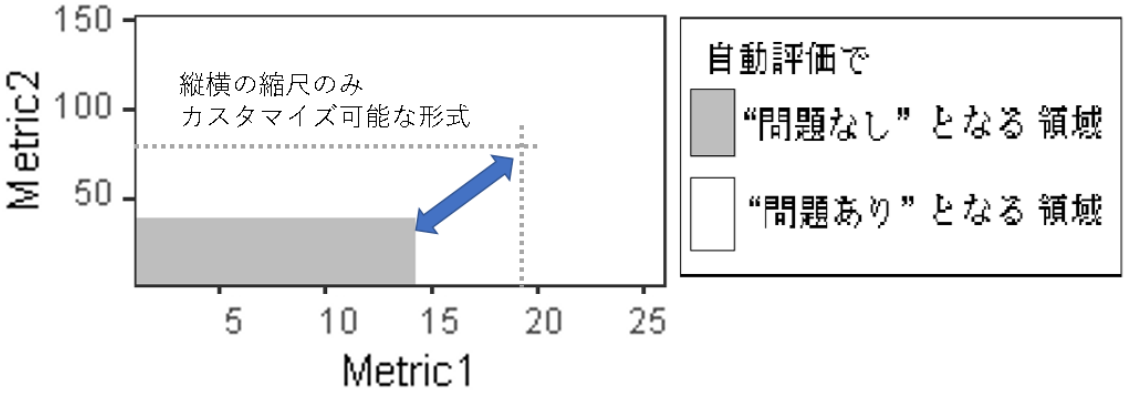

図 3-1：二種類のメトリクスに一個ずつ閾値を設定した解釈モデル

3.2.3. 教師あり学習を用いた解釈モデルの構築手法

SVM(Support Vector Machine) で二つのメトリクスの解釈モデルを構築した場合，問題の有無を判別するための境界が曲線状になる．参考として，形状の例を図 3-2 に示す．

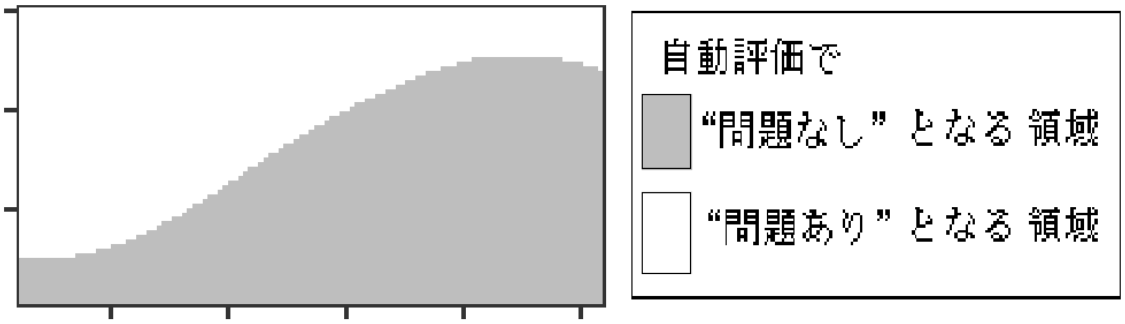

図 3-2：SVM による解釈モデルの例.

RandomForest で二つのメトリクスの解釈モデルを構築した場合，問題の有無を判別するための境界が複雑に入り組んだ形状になりやすい．参考として，形状の例を図 3-3 に示す．これらの手法では高精度な解釈モデルを構築しやすいが，内容を閾値による条件の組合せで表現しにくく，評価過程や考慮されている範囲が人間にとってわかりにくいものになりやすい．モデルのわかりやすさは，人間がそのモデルを信用できるかどうかにも影響するため留意すべきである [55]．また，これらの手法では境界の形状をパラメータで制御できるが，境界の形状との関係が非線形的であり直感的な調整が難しい．

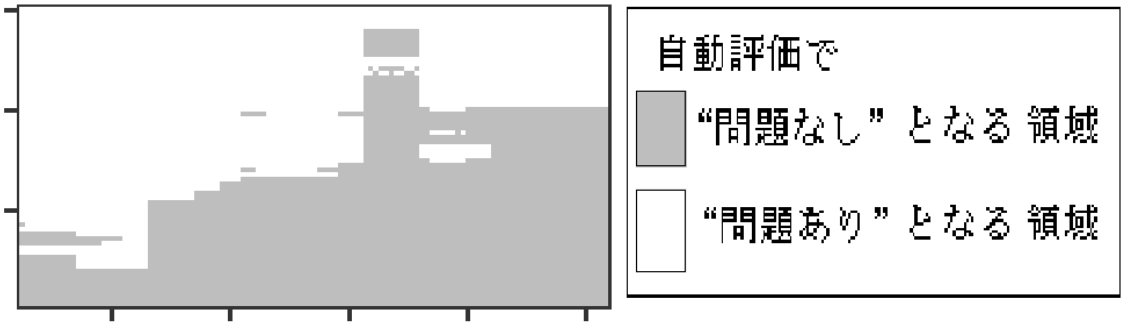

図 3-3：RandomForest による解釈モデルの例.

分類木学習アルゴリズム C4.5 [56]やその後継である C5.0 で二つのメトリクスの解釈モデルを構築した場合，閾値による条件式の組合せが得られる．参考として，形状の例を図 3-4 に示す．分類木学習では特徴量空間を閾値で再帰的に分割して最適な解釈モデルを構築するため，図 3-1 のような四角形のサイズ変更よりも表現力が高い．

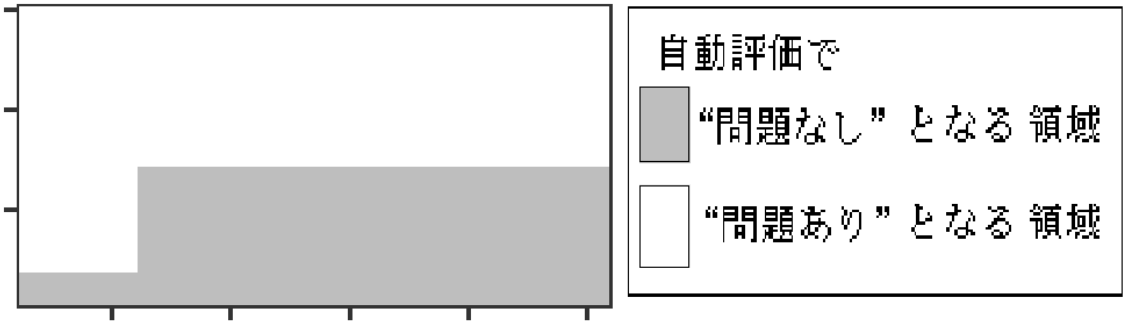

図 3-4：分類木による解釈モデルの例.

Fontana らは複数のコードスメルをメトリクスで予測することについて，SVM や RandomForest や C4.5 等の性能を比較した [57]. しかし，従来の閾値決定手法との性能の比較には踏み込んでおらず，C4.5 と従来手法との間に，どの程度の性能の差があるかは十分に明らかではない.パーセンタイル関数や Alves 法には教師データが不要という強みがあるため，C4.5 はそれより大きく優れていることを示せなければ,人手をかけてまで教師データを収集する動機付けをしにくい. 一方，従来の閾値決定手法については，性能の上限において C4.5 に劣ることは推測できるものの，その単純さから過学習をしにくい可能性もあり，教師データが少ない状況下での性能比較には意義がある.

3.2.4. コードの保守上の問題の教師データについて

Mantyla らはコードの保守上の問題の分類として，機能性欠陥と発展性欠陥を定義している [38]. 機能性欠陥とは，ソフトウェア実行時に観測されうる機能上の不具合である. 一方で，発展性欠陥とは，ソフトウェア実行時に観測できるものではなく,コードのわかりやすさや修正しやすさが通常よりも低くなってしまっている状態である.

一般に発展性欠陥の有無はテスト実行で判断できないため,教師データについては目視で与える必要がある. しかし，レビュー観点によってはドメインに精通したエキスパートに任せなければならず [38]，実施する際のコストが大きい. 結果として，開発組織においては，エキスパートが多忙であることを前提に [58],限られた少量の教師データを効果的に利用することが課題となる.

また，発展性欠陥についての既成のデータセットの種類には限りがあり [59],自組織の対象コンテキストと類似したプロジェクトのデータを再利用できるとは限らない. この点については，発展性欠陥と類似した概念であるコー

ドスメル [37]と不可視の技術的負債 [39]についても同様である.

3.3. 品質評価枠組みの定義

本節ではコードのわかりやすさの評価基準(解釈モデルおよび閾値) をカスタマイズするための実用的な枠組みを定義する. 以降, 用語の定義, 枠組みの概要, 実施手順, 枠組みで扱われる解釈モデルの例, の順に説明する.

3.3.1. 用語の定義

枠組みで取り扱う用語を以下のように定義した.

評価基準：

「ある観点でコードを評価するための定量的な基準」.

評価基準としては, 多数のメトリクスを説明変数とするものや, 複数のメトリクス測定値を合成するものもありうる. しかし, それらの評価基準は評価過程や考慮されている範囲が人間にとってわかりにくいものになる場合がある. そのため, 本研究で扱う評価基準は, 説明変数を合成しておらず, その解釈を閾値を用いて形式表現可能なものに限定した. 例えば,「ファイル行数<=100 AND 関数定義数<=10 ならば問題なし」は二種類のメトリクスの条件式を組み合わせた評価基準である.

逆に,「ファイル行数+10*関数定義数<=100 ならば問題なし」といった複数のメトリクス測定値を合成する条件式や, 重回帰, SVM, RandomForest などは対象外とした.

解釈モデル：

「閾値的な評価基準で用いる解釈の形式表現について, 閾値が固定されずに抽象化されているもの」.

例えば「ファイル行数<=T1 AND 関数定義数<=T2 ならば OK」という解釈モデルでは閾値 T1,T2 は固定されておらず, 利用前にこれらを定めて具体化する必要がある. 複数の既存のコード評価ツールにおいて, 解釈モデルに相当する設定機能が用意されている [13] [11].

3.3.2. 枠組みの概要

本研究ではコードのわかりやすさの高精度な自動評価を目的とし,その評価基準をカスタマイズするための実用的な枠組みを整理した. 枠組みでは, エキスパートがレビュー可能なコードが少量となることを踏まえ,レビュー対象をコンテキストを代表するソフトウェア群の一部に絞る. そして, 複数のメトリクスの影響関係を分類木学習によって考慮することで,そのコンテキストに適した解釈モデルと閾値の組み合わせを導出できる.

次に, 枠組みの汎用性と課題について述べる. 本研究では, C++のソフトウェア群を題材に評価実験をしているが,我々の枠組み自体はプログラミング言語に依存しない形に一般化している. そのため, 枠組みの実施手順については汎用的に利用することが可能である. ただし, 枠組みで導出したコードメトリクス閾値とその解釈モデルは対象とするコンテキスト向けにカスタマイズされているため, 同一のコンテキスト(プログラミング言語とアプリケーションドメイン)でのみ共有することを制約とする. その際, アプリケーションドメインの同一性については開発組織が経験的に判断するものとする.コンテキストの同一性の判定方法の具体化と評価については今後の課題である.

我々の枠組みの全体像を図 3-5 に示す. この図は実施手順を構成する以下の七つの Steps における前後関係と入出力関係, および閾値導出が可能となるタイミングを示している. その詳細は 3.3.3 節で個別に説明する.

- Step1) コンテキスト代表のソフトウェア群の選択.
- Step2) コード評価の GQM モデルの定義.
- Step3) 静的解析によるメトリクス測定.
- Step4) レビュー対象ファイルの選択.
- Step5) エキスパートによるレビューの実施.
- Step6) 分類木学習による解釈モデルと閾値の導出.
- Step7) 妥当性の議論.

Step1 から Step3 までを完了すると, パーセンタイル関数や Alves 法を利用できる状態になる. Step6 までを完了すると, 教師あり学習を利用できる状態になる. 最後に, Step7 では導出された解釈モデルおよび閾値の妥当性を議論する. Step7 の結果に応じて, 必要であれば以前の Step からやり直す.

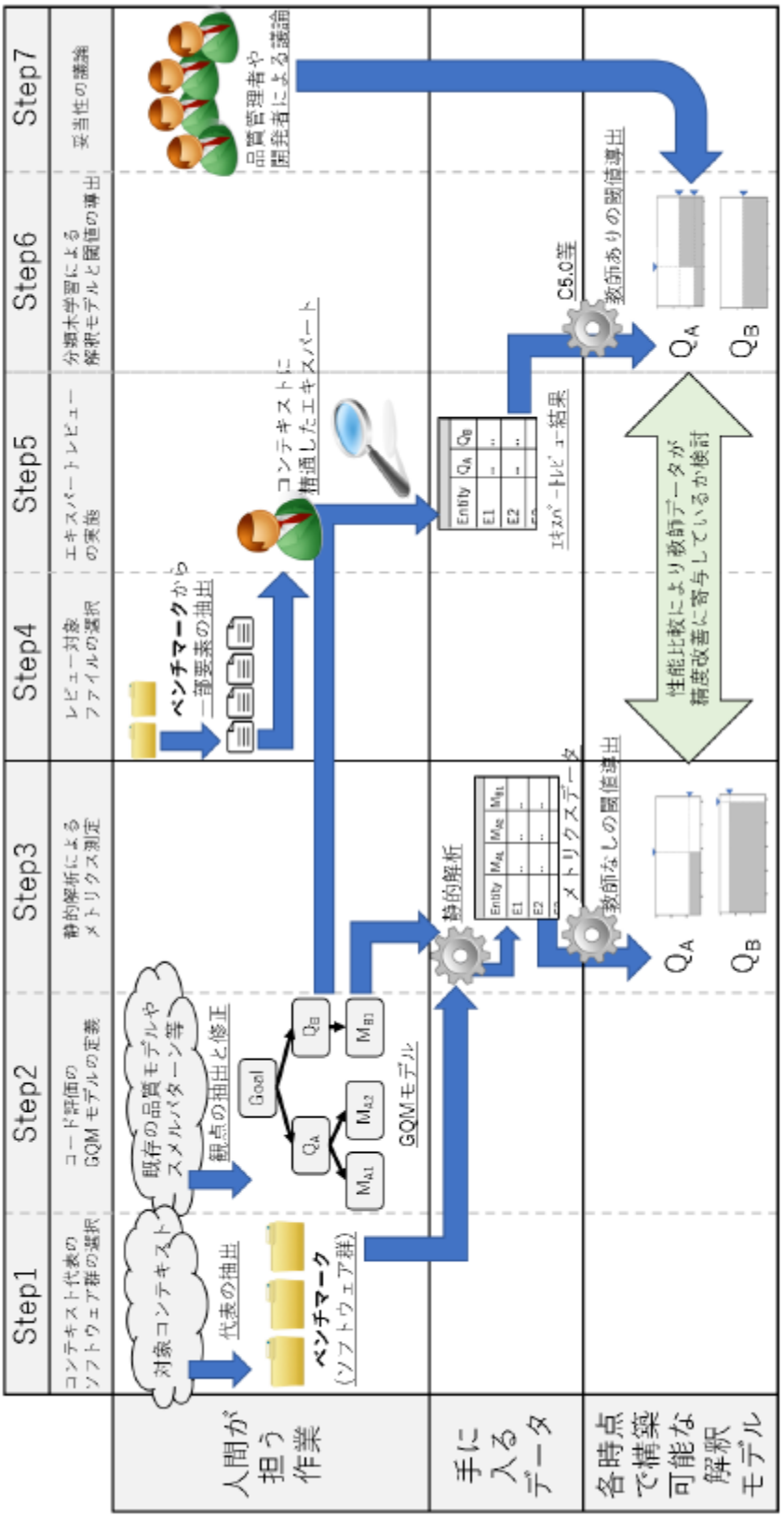

図 3-5：我々の枠組みの全体像

3.3.3. 実施手順の詳細

Step1) コンテキスト代表のソフトウェア群の選択

- 開発組織においてコードの自動評価を導入したいコンテキストを特定し，そのコンテキストを代表するソフトウェア群を用意する．

以下，制約について述べる．代表としたソフトウェア群は他のコンテキストまでに広げた一般性は欠きやすい．そのため，我々の枠組みで導出した評価基準は，同一のプログラミング言語とアプリケーションドメインのソフトウェアでのみ共有することを制約とする．その際，アプリケーションドメインの同一性については開発組織が経験的に判断するものとする．コンテキストの同一性の判定方法の具体化と評価については今後の課題である．

組込み製品ベンダ等，複数の製品機種を長期間に渡り開発・保守をしている組織では，近いコンテキストのソフトウェアが蓄積されやすく，Step1 を実施しやすいと考えられる．一方で，OSS(Open Source Software)については産業用ソフトウェアとでは多くの種類のメトリクスで分布が異なる傾向にあると報告があり [60]，注意して対象を選ぶことが望ましい．

Step2) コード評価の GQM モデルの定義

- 保守性の評価方法や，保守上の問題のパターン集・カタログ等を参考にして，対象コンテキストにおいてコードを評価する際の観点とメトリクスを定義する．

コードをどのように評価するかは，Goal-Question-Metric(GQM)モデルで定義するとわかりやすい [61]．具体的には，「コードがわかりやすく，開発者にとって保守性が高いこと」を目標(Goal)とし，達成可否について複数の質問(Question)を設け，定量的に回答するための測定方法(Metrics)を定義する．例えば，クラス(ファイル)が担う責務量が大きすぎないか，という質問を設けたら，

- Metric1：ファイル内の関数定義数．
- Metric2：ファイルのコード行数．

といったメトリクスを関連付けていく．GQM モデルでは閾値までは定義していないので，大枠のみ他のコンテキストに流用することは許容できる．例として，我々が本研究で定義した基礎的なメトリクスを表 3-1 に，GQM モデルを表 3-2 に示した．

採用する Question や Metrics の定義をしていく際には，ソフトウェア品質評価の国際規格や [24]，発展性欠陥の分類 [38]，コードスメルのパターン [62] [63]，既存の GQM モデル [10] [64]等が参考になる．これらの体系的枠組みには多量な観点とメトリクスを含まれているので,試験的導入の際には自組織が特に評価したい項目を抽出してカスタマイズすべきである．

表 3-1：基礎的なソースコードメトリクス．

メトリクス	説明
LOC	コードファイルの行数を，コメント行や空行も含めて数えたもの．Lines Of Code の略.
DN	ある関数について，その制御フローのネストの深さの最大値．ネストのある箇所とみなすのは{if, for, while, do, switch}を使用している箇所とする．Depth of Nest の略.
CC	ある関数について,そのサイクロマティック複雑度.計算方法は「制御フローの分岐の数+1」となる.Cyclomatic Complexity の略.

表 3-2：GQM モデルの例: Goal(目標)，Question(質問)，Metrics(測定法)．

Goal	Question	Metrics	ファイル単位メトリクスの説明
G: コードがわかりやすく書かれており，開発者にとって保守性が高いこと.	Q1: ファイルの責務の量は適切か?	FUN	ファイル内で定義されている関数の数．Number of Function definitions の略.
		ELOC	ファイル内のコード行数を,処理実行の主体以外の行を無視しながら数えたものであり，無視されるのは空行，コメント行，括弧のみの行，変数宣言のみの行等である．Executable LOC の略.
	Q2: 関数の処理が複雑すぎないか?	AveDN	ファイル内の関数の DN の平均値．Average of DN の略.
		MaxDN	ファイル内の関数の DN の最大値．Maximum of DN の略.
		AveCC	ファイル内の関数の CC の平均値．Average of CC の略.
		MaxCC	ファイル内の関数の CC の最大値．Maximum of CC の略.

Step3) 静的解析によるメトリクス測定

- GQM モデルで定義したメトリクスを測定する．測定対象は代表としたソフトウェア群の各ソースコードファイルとする．測定ツールの種類は問わない．

具体例として,代表的なソースコード静的解析ツールである Understand の説明をする．Understand は複数のプログラミング言語(C/C++，C#，Java 等)に対応しており，行数やサイクロマティック複雑度を測定可能である．
メトリクス測定値は CSV ファイルとして出力できる．また，API を用いれば独自メトリクスの測定も可能である．

なお，静的解析ツールを利用する際には，ツール毎のメトリクスの測定仕様の違いに注意すべきである．例えば結合度の測定仕様の違いとして，Integer や String などのプリミティブクラスも数えるか，外部ライブラリのクラスも数えるか，明示的でない依存関係も数えるかなどが挙げられる．また，行数やサイクロマティック複雑度といったファイル単体で完結して測定可能なメトリクスについても,複数の測定パターンが採用されている場合もあるため同様に注意すべきである．

Step4) レビュー対象ファイルの選択

- 代表としたソフトウェア群から一部のソースコードファイルを抽出し，次の Step5 でのエキスパートによるレビューの対象とする．

全体のレビューは難しく,限られた範囲で教師データを集めることが現実的となる.観点によってはドメイン知識を有するエキスパートでないと難しい上に [38]，エキスパートには種々の業務が集中しやすい [58]からである．

抽出すべきファイルの数についてはいくつかの知見がある．Rahaman らは現実的かつ効果的な量として全体の 5%から 20%を提案している [65].また，Fontana らは 100 件程度でもコードスメルの機械学習において有用であったと報告している [57]．全体として大規模なソフトウェアではファイルの細分化が徹底される傾向があり,ファイルの大多数は小規模になっていると考えればレビューを実施可能な量と考えられる．また，本研究の実験結果を先に挙げると，約 22 個の教師データで学習した場合でも，80 パーセンタイルより高精度な自動評価を実現できた．

抽出すべきファイルの内容はカスタマイズ方針によるため,どのような方法がより実用的な評価基準の導出に寄与するかは明らかではない．そのため，参考として具体例を以下の述べる．各方法の評価は今後の課題である．

- 仮定を置かずに，ファイルをランダムに選択する．
- Fontana らの研究では [57], 既成ツール(iPlasma 等)により問題個所の候補を絞り込んだ．さらに，人手でレビュー可能な数になるようにランダム選択した．
- 本研究では，協力企業が組込み製品(建機)の機種展開をしていくための母体となっている複数のソフトウェアシステムを代表とした．そして，人手でレビュー可能な数を選択する際には，まずレビュアーが内容を熟知しているサブシステム群を特定した．最後に，教師データの正例と負例の偏りを低減することを意図し，問題のある個所とない箇所を同程度含んでいると想定されるサブシステム群を数個選択した．

Step5) エキスパートによるレビューの実施

- GQM モデルの各 Question をチェックリスト項目として，各ファイルをエキスパートがレビューする．
- 各 Question について，各ファイル毎に“問題あり”か“問題なし”のいずれかを記録する．

エキスパートとしてはコンテキストに精通した熟練開発者や設計者などが相応しい．M 個のファイルを N 個の Question でレビューする場合には，M 行 N 列のテーブルを結果として記録することになる．問題の有無を二値に振り分けることが難しい場合には,何点未満なら問題ありとみなすのかを明示した上で,五段階や百点満点で採点してもよい．レビュー後に自動的に二値に変換できる．

Step6) 分類木学習による解釈モデルと閾値の導出

- 分類木学習アルゴリズムにより，メトリクス測定値の解釈モデルを構築する．
- 各 Question について，それぞれ独立に実施する．

分類木学習アルゴリズムについて，C4.5 系列のものは複数の統計環境において実装および提供されている．本研究では統計言語 R で実装されている C5.0 アルゴリズムを用いた．

Question が N 個の場合には，N 個の解釈モデルを構築することになる．こ

の方式では，個々の Question 毎に関連するメトリクスの影響関係を考慮しつつ，簡潔なモデルを構築できる．現時点では，複数の Question をまたいだ影響関係は考慮していない.この制限の緩和や解決については今後の課題とする.

Step7) 妥当性の議論

- コンテキストに詳しい開発者や品質管理者を交え，解釈モデルの妥当性を議論する.
- もし許容できない内容の場合，以前の Step からのやり直しを検討する.

コンテキストに詳しい開発者や品質管理者を交えることで,解析したデータに含まれない背景情報も含めた検討ができる ［54］［35］．積極的で示唆のある議論をするためには，以下の二つの要素が重要であると我々は考えた.

- 性能指標：コードの自動評価がエキスパートのレビュー結果と一致している度合いが定量化されていないと比較や議論をしにくい.
- 可視化：解釈モデル(分類木)をテキストのまま見ても，特徴量空間上に分布する実データとの対応関係は直感的に理解しにくい.

性能指標については，Precision，Recall，F 値(F1 値とも呼ばれる)などがある．これらについて，問題のあるファイルを陽性とみなして計算する．同じ F 値の手法同士については再現率と適合率を比較したトレードオフの議論が可能であるため，開発組織が目的に応じて判断する．例えば，再現率が低くとも適合率が高い場合は,問題個所の検出漏れがある代わりに検出した範囲内では正答率が高い．一方で，再現率が高くとも適合率が低い場合は，問題個所の検出漏れが少ない代わりに，誤検出が多く含まれる.

可視化について，もし分類木が最終的に含むメトリクスが 2 個以下なら，二次元平面で表現できる.この平面上に実際のメトリクス測定値の散布図を重ねることで,分類木の内容について対象コンテキストの開発者の認識に反しないかを議論できる.確認手順の具体化と評価，説明変数が三つ以上の場合の効果的な可視化方法については今後の課題である．統計解析言語 R において contour 関数や ggplot2 パッケージの geom_tile 関数を用いれば，自動評価における“問題あり”と“問題なし”の境界を塗り分ける処理を実装できる.

また，教師無し手法と教師あり手法の性能比較により，教師データが評価基準の精度改善に寄与しているかを確認することも議論の際には有用である.もし，性能(F 値等)の差が小さい場合には，教師データ作成者が元々単純な解釈をしていることや,機械学習のチューニングパラメータが不適切であること等が理由として考えられる．前者の場合，教師データを増やし続けても導出される評価基準の精度は一定のままである可能性があり,教師データの作成におい

て他の観点を優先する方が有用である可能性がある.

3.3.4. メトリクス測定値の様々な解釈モデルの例

本節ではメトリクス測定値の解釈モデルの例について述べる. 例えば, 全体的には測定値の大きさでコードの構造上の問題を判断できるものの, その関係が単調ではなく, 一定以上になるとむしろ問題は少なくなるという場合がある. 一部のクラスに処理を意図的に集中させる場合や, 特定のコードが例外的に許容されている場合が知られている [66] [67].

また, 他のメトリクスが別のメトリクスの判断に影響する非直交な関係性もありえる [54] [35]. 例えば, コード規模の目標を「行数の削減」ひとつだけ(One-track-metrics [54]の状態)にしてしまうと, 開発者はその達成のために, 一行あたりのステートメントを複雑化させてしまう可能性があり(負のホーソン効果 [35]), 結果として行数以外の構造的特徴も考慮しなければならなくなることがある.

以下, 我々の枠組みで構築されうる解釈モデルについて, 四つのパターンを例として説明する. 実際には, 例示するような図形を左右上下反転や回転したパターンや, 複数が同時に起こるパターンもありえる.

もし分類木学習によって得られた解釈モデルが開発者にとって不自然であれば, 教師データの偏りや過学習が原因の可能性もあるが, 一方でエキスパートが独自の暗黙知を使っていることが原因の可能性もある. そのため, 我々の枠組みを適用することで, エキスパートのレビューにおける判断を定量的に表現し直すことができ, 暗黙知を発見することに繋がる.

以下, 解釈モデルの 4 つのパターンを図 3-6 から図 3-9 に例示する. これらの例では, 二つのメトリクスを説明変数とする場合を対象としている.

図 3-6 は二つの説明変数のうち, 片方のみで閾値が設定されたパターンを示す. 我々の枠組みでは, N 個のメトリクスを説明変数として学習していても, そのうちの一部のみで十分な精度の自動評価ができる場合には, このような解釈になりうる. 説明変数同士の相関が強い場合では, いずれかのメトリックのみで解釈をモデル化でき, 冗長さを低減できる

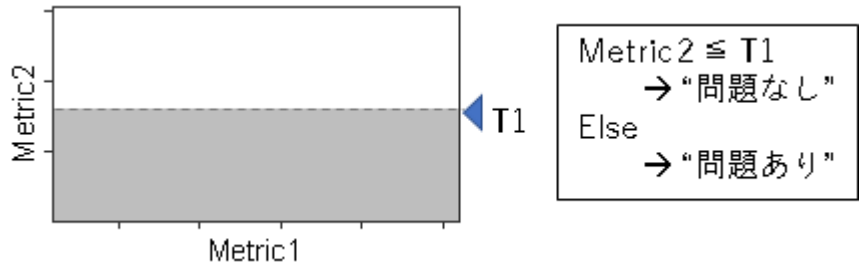

図 3-6：一つのメトリックのみで閾値を設定しているパターン.

図 3-7 は判別境界が中抜き状になるパターンを示す．あるメトリックの値が一定範囲にあるかどうかが重要な場合にはこの解釈になりうる.行数や複雑度などは，大きすぎても小さすぎても実装の不自然さの兆候であり，ゴルディロックの原理と呼ばれて研究されてきた [68] [69].

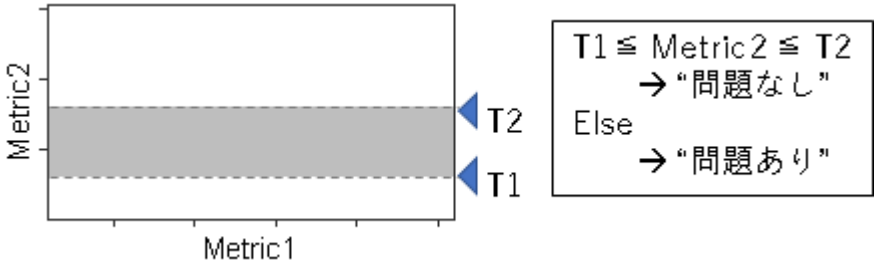

図 3-7：判別境界が中抜き状になるパターン.

図 3-8 は判別境界が階段状になるパターンを示す．あるメトリックの値の大きさ次第で他のメトリクスの判断基準が変わる場合にはこの解釈になりうる．例えば規模については，担っている機能の違うコード同士で行数や関数定義数の中心傾向が違う場合がある．また，自動生成コードやテストコード，MVC におけるコントローラクラス等では評価基準に違いが出やすいことも知られている [70].

解釈モデルがこのパターンになった場合,判断基準が変わっている箇所に注目したい．複数の評価基準において，同様の切り替わりが多く見られる場合，その区分けをサブコンテキストとして別々に枠組みを適用することも考えられる.

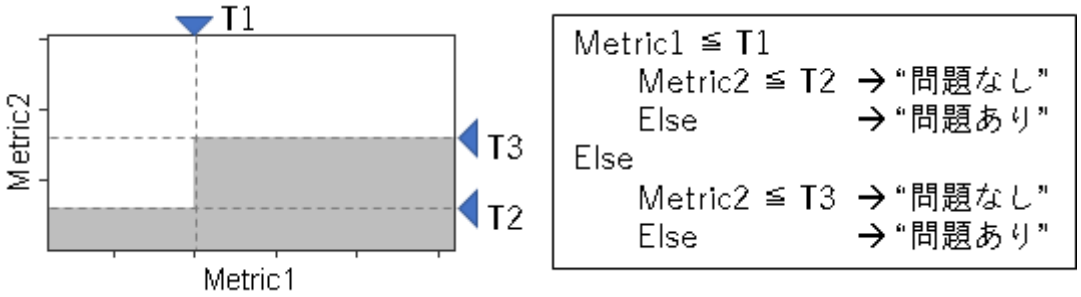

図 3-8：判別境界が階段状になるパターン.

図 3-9 は判別境界判別境界に突出した箇所があるパターンを示す．局所的に判断基準が変わることが特徴的である.突出した箇所に該当するコードの特徴が，そのコンテキストにおいて普遍的なものなのか，教師データの元であるソフトウェア特有のものであるかを注意すべきである．後者の場合，解釈モデルが同じコンテキストの他のソフトウェアにとって適していない恐れがある(過学習の恐れがある).

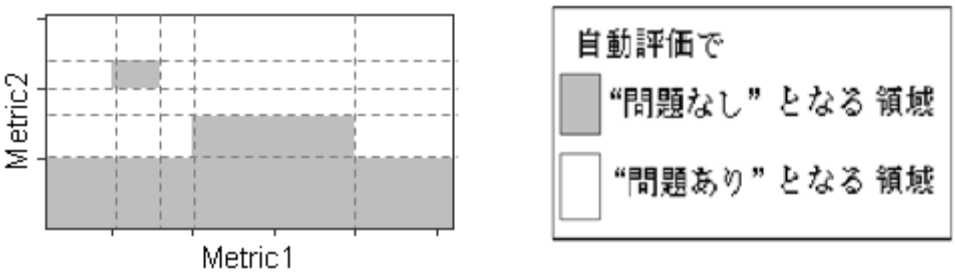

図 3-9：判別境界に突出した箇所があるパターン.

3.4. 実験環境と設定

3.4.1. 概要

我々は枠組みの実用性を評価するために，以下の研究課題に取り組んだ.

- RQ1: 我々の枠組みにより，実用的な評価基準を得られるか?
- RQ2: 少量の教師データを入力とした場合でも，分類木学習により他の閾値導出手法に比べ高精度な評価基準を導出できるか?
- RQ3: 教師データの量は，自動評価の精度にどう影響するか?

RQ1 については，実験 1 で扱うものとし，枠組みの Step6 と Step7 を実施することで取り組んだ.

RQ2 と RQ3 については，実験 2 で扱うものとし，閾値導出手法の性能を比較することで取り組んだ．比較した閾値導出手法は以下の通りである．

- 教師データを用いない手法：パーセンタイル関数と Alves 法．
- 教師あり学習を用いる手法： Bender 法(ロジスティック回帰ベース)，ROC 曲線法，C5.0 アルゴリズム．

実験においては建機制御用の C++組込みソフトウェア群を題材として，メトリクスデータと教師データを取得するために，3.3.3 節で説明した Step1 から Step5 までを実施した．結果として実験で利用した以下の要素の詳細については本節の以降で述べる．

- データセット
- GQM モデル
- 教師データとしたエキスパートレビューデータ
- 手法を比較するための性能指標

3.4.2. データセット

コード自動評価の対象コンテキストとしては，協力企業 1 社が継続的に開発・保守している共通プラットフォームから，建機制御用の組込みソフトウェアシリーズ(C++)を選んだ．そして，コンテキストの代表としては，一定の妥当性があると考えられるものとして，建機の機種展開で母体とされている 3 個のソフトウェア(Sys1，Sys2，Sys3)を選んだ．

Sys1 から Sys3 の規模を表 3-3 に示す．これらは多言語開発されていたが，85%と大部分のコードファイルが C++のものであり，残りが C 言語のものであった．本研究では C++部分のみを実験に用いた．

表 3-3：コンテキスト代表のソフトウェア群の規模の要約(C++のみ実験で使用)

ソフトウェア(システム)	C++の部分		C 言語の部分	
	ファイル数	総 LOC	ファイル数	総 LOC
Sys1	502	154,004	79	24,956
Sys2	586	181,163	86	28,272
Sys3	319	47,938	81	42,237
合計	1,407	383,105	246	95,465

3.4.3. GQM モデル

対象コンテキストで取り扱う GQM モデルを定義する際には，我々の過去の研究成果である GQM モデルを参考とした [10]．そして，保守性の内の特にコードのわかりやすさや扱いやすさに着目して複数の Question とメトリクスを抽出し，一部変更を加えた．また，メトリクスの粒度はファイル単位に揃えるものとした．

本研究では以下の二つの Question とそれに対応するメトリクスを題材として評価実験を実施した．それぞれの定義については表 3-1 と表 3-2 に示した．

- **Q1:ファイルの責務の大きさが適切であるか？）**

  ファイルが担う責務の量に関する 2 個のメトリクスを関連付けた．

  - ELOC：ファイル内の実行部分のコード行数．コメント行や空行といった，コンパイラが実行部分として認識しない行を含まない．
  - FUN：ファイル内の関数定義数．

  FUN を関連付けたのは，ファイルの責務の適切さについて，以下に示すようなエキスパートの暗黙的な判断について必要であれば評価基準が学習できるようにするためである．

  - 行数が多いファイルでも，小規模な関数への分割が十分なら許容する．

- **Q2:（ファイル内の）関数の処理が複雑すぎないか？）**

  関数の制御フローに関する 4 個のメトリクスを関連付けた．

  - AveDN：ネスト深さ(DN)のファイル内平均値．
  - MaxDN：ネスト深さのファイル内最大値．
  - AveCC：サイクロマティック複雑度(CC) [71]のファイル内平均値．
  - MaxCC：サイクロマティック複雑度のファイル内最大値．

3.4.4. エキスパートによるレビューの結果

対象コンテキストのソフトウェアアークテクチャに精通し，10 年以上の開発経験を持つエキスパート 1 名によりコードレビューを実施した．その際，Sys1 から Sys3 の全ファイルをレビューすることが困難であったため，3 個のサブシステム(Sub1，Sub2，Sub3)をレビュー対象として選定した．まずレビュアーが内容を熟知しているサブシステム群を特定した．次に，教師データの正例と負例の偏りを低減することを意図し，双方を同程度含んでいると想定されるサブシステム群を 3 個選定した．結果的には，Sub1 から Sub3 を Sys1 のみから選定することとなった(図 3-10)．

レビュー結果の内訳を表 3-4 に示す. 各 Question について問題ありと判定されるファイルは半数以下であり, Q1 では 42%(60/143), Q2 では 31%(45/143)であった. なお, 約 40%のファイルが問題ありとなっているが, これはコードの静的構造上の問題を指摘するものであり, ソフトウェアの機能的な不具合の量を意味していないことは留意点である.

また, エキスパートがレビューにかけた時間は一つのファイルにつき数分であり, 143 個のファイルのレビューに合計で約 8 時間かかった. レビューしたファイルがサブシステム群全体に占める率は約 10%(143/1407)であった.

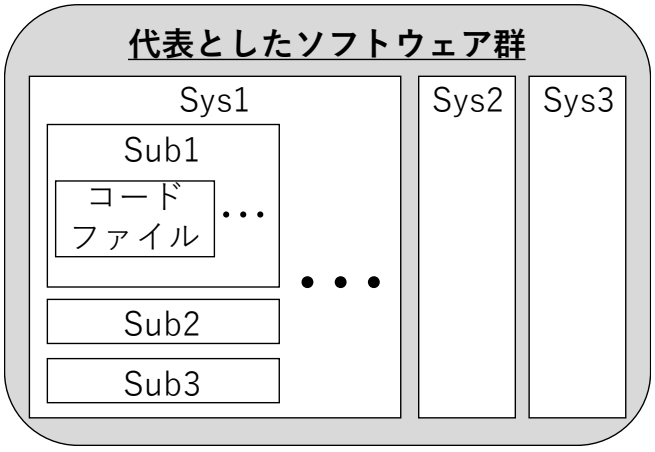

図 3-10：代表としたソフトウェア群とレビュー対象の包含関係

表 3-4：サブシステムの規模とエキスパートによるレビュー結果の要約

対象の			Q1	Q2
サブシステム	ファイル数	総 LOC	問題あり	問題あり
Sub1	57	13,952	17	14
Sub2	31	6,926	11	11
Sub3	55	8,875	32	20
合計	143	29,753	60	45

3.4.5. 性能指標

本研究の実験では, 複数の閾値導出手法を比較するための性能指標として F 値を用いた. また, 実験 2 の教師あり学習の比較では, より詳細な分析のために Precision と Recall も用いた. これらの指標を計算する際には, ファイルに問題があることを陽性として扱った.

F 値の値域は[0.0, 1.0] であり，1 に近づくほど性能が高いことを表す．F 値は Precision と Recall の調和平均であるが，Precision は偽陽性の少なさを表し，Recall は偽陰性の少なさを表す．偽陽性と偽陰性はそれぞれ自動評価の誤りを表すが，一般的にトレードオフの関係にあることが知られており，その調和平均によって総合評価することができる．

本研究における偽陽性は,エキスパートが問題なし(陰性)と判定しているにもかかわらず，自動評価では陽性と誤判定してしまったファイルを意味する．偽陰性は，エキスパートが陽性と判定しているにもかかわらず，自動評価では陰性と誤判定してしまったファイルを意味する．

3.5. 実験 1

3.5.1. 方法

3.3.3 節で説明した枠組みの Step6 と Step7 を協力企業で実施した．Step6 では，Q1 と Q2 のそれぞれの評価用分類木を構築するためのツールとして，統計解析言語 R の C50 パッケージを用いた．Step7 では，構築した分類木の妥当性を確認するために，対象コンテキストに精通する開発担当者 3 名を含む 5 名で議論した．そしてその後，構築した分類木を利用した結果について協力企業の開発者からフィードバックを得た．

3.5.2. 結果(Step6)

Q1 の分類木は FUN と ELOC の非直交な関係性に解釈を与えるものとなった．FUN の値に応じて，ELOC では二つの閾値が使い分けられている．一方で，Q2 の分類木では MaxCC のみ用いられ，他のメトリクスは除外された．

構築された分類木のテキストベースの表現を以下に示す．

- Q1) ファイルの担う責務が大きすぎないか？

  If(FUN ≦ 6)

  　　IF(ELOC ≦ 19): 問題なし

  　　ELSE：問題あり

  Else

  　　IF(ELOC ≦ 71): 問題なし

  　　ELSE：問題あり

- Q2) ファイル内の関数が複雑すぎないか？

  IF(MaxCC ≦ 7): 問題なし

  ELSE：問題あり

構築された分類木を二次元で可視化した結果を図 3-12 と図 3-13 に示した．また，これらの図におけるプロット形状と背景色の意味を図 3-11 に示した．プロットの形状は各ファイルのエキスパートによるレビューの結果を表しており，背景色は自動評価の解釈モデルを表している．それぞれの詳細を以下に示す．

- プロット形状が“X”:エキスパートが“問題あり”と評価したファイル．
- プロット形状が“O”:エキスパートが“問題なし”と評価したファイル．
- 薄い背景色：自動評価において“問題あり”となる領域．
- 濃い背景色：自動評価において“問題なし”となる領域．

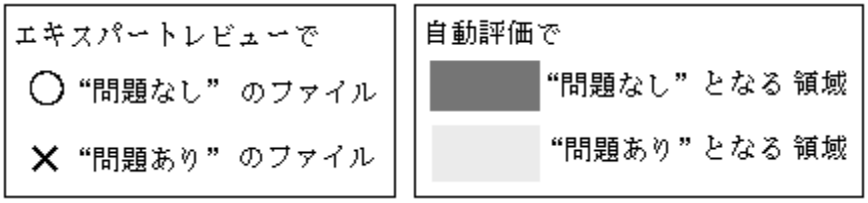

図 3-11：Q1 と Q2 の可視化についての凡例.

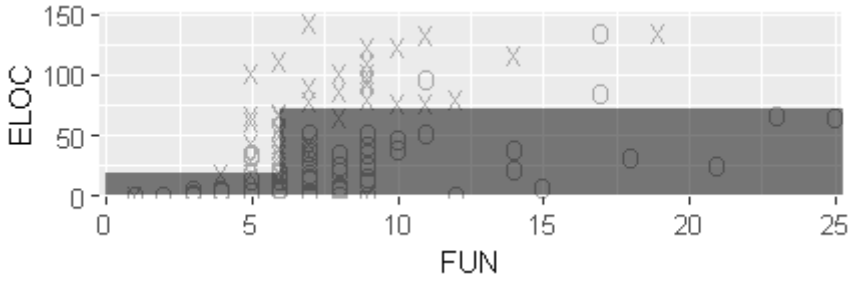

図 3-12：Q1 について構築した分類木を可視化した散布図.

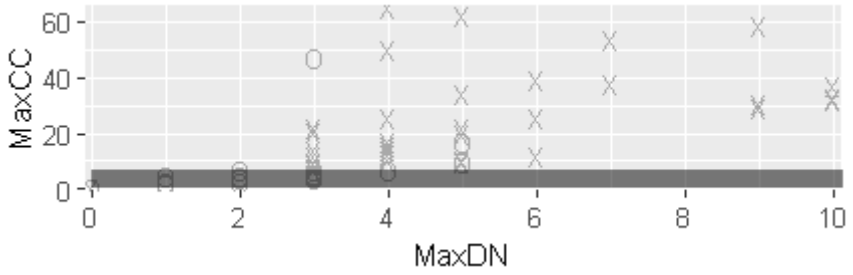

図 3-13：Q2 について構築した分類木を可視化した散布図.

3.5.3. 結果(Step7)

Q1 と Q2 の分類木について,対象コンテキストに精通する開発担当者 3 名(教師データ作成者ではない) と研究室メンバ 2 名で議論し，その内容に一定の妥当性があることを確認した．分類木の性能指標である F 値はそれぞれ 0.88(Q1)と 0.91(Q2)であり，エキスパートによるレビュー結果との一致度が高かった．また，可視化を通じた議論により，解釈モデルと閾値については対象コンテキストの開発者の認識に著しく反するものではないこと確認した．

以下に，それぞれの分類木についての要約を示す．

Q1 で得られた解釈モデルについて，対象コンテキストにおいて妥当と判断

した．このコンテキストではソフトウェアの機能が細分化されており，個々の機能を担うファイルには以下の傾向があった．

- ある程度の数の関数を持つようなファイルは，複数の処理の組み合わせを担っているため，実行コード行数がやや多い．
- コアでない周辺ファイルでは，少ない関数のみが含まれ規模も小さい．

これらの傾向から，ファイルの役割と関数定義数の間に一定の関係性が見受けられる．そのため，関数の定義数に応じて，許容される実行コード行数の大きさの判断を複数用意することは不自然ではないと考えた．F 値も 0.88 と高いため，現状ではこの分類木を採用した．ただし，今後教師データを追加して再学習することで分類木が同様の解釈モデルを維持するかどうかを確認することも視野に入れることとした．

Q2 で得られた解釈モデルは対象コンテキストにおいて妥当と判断した．このコンテキストでは多分岐の switch-case 文を使う傾向があり，ネストが浅くても実行経路が複雑になっている関数が多々あった．そのため，ネストの深さに関するメトリクスが解釈モデルに含まれなかったのは不自然ではないと考えた．また，AveCC も解釈モデルに含まれなかったが，解釈モデルで用いられている MaxCC と相関しており片方は冗長となる関係にあった．

特徴的であったのは解釈モデルの形状であった．Q1 の解釈モデルは開発者が明示的には意識していなかった階段状パターンになった．一方で，Q2 の解釈モデルは一種のメトリックと一個の閾値から成る単純な形式になった．この結果ついては，評価対象であるコードファイル単体において，以下に述べる規模と複雑度の性質の違いが影響していたと考えられる．

- ファイル単体の規模を減らすには，設計を見直してそのファイルが担う機能を外部に移動するか，アルゴリズムを見直して実行処理の行を減らす必要がある．そのため，開発者一人の判断で削減することが難しい面があり，開発における様々な局面で異なる規模感のファイルが許容されうると考えられる．
- ファイル単体に含まれる関数の構造的複雑度を減らすには，関数分割をすればよい．また，関数分割はそのファイル内で完結させることが通常は可能と考えられる．

これらの理由から，少なくともこのコンテキストにおいては，人間によるコードファイルの規模の解釈は多面的なものに，構造的複雑度の解釈は単純なものになりやすい可能性があると判断した．

3.5.4. 評価基準の妥当性に関するフィードバック

協力組織の開発者からのフィードバックでは，Step7 で構築した分類木を用いた Sys2 と Sys3 の評価結果について一定の妥当性が認められた．

3.6. 実験 2

3.6.1. 方法

実験 2 では主に，教師あり学習を用いる閾値導出手法を比較した．加えて，教師データを用いるからには越えるべきと期待されるターゲットとして教師データを用いない手法とも比較をした．取り扱う閾値導出手法は以下の五つである．C5.0 以外の方法では図 3-1 で説明した形式でメトリクスの解釈モデルを構築した．

- 教師データを用いない閾値導出手法: パーセンタイル関数，Alves 法．
- 教師あり学習を用いる閾値導出手法: Bender 法，ROC 曲線法，分類木学習アルゴリズム C5.0．

また，図 3-14 に示すように，閾値導出手法に応じて以下のデータを入力とした．

- 教師データを用いない手法では，代表となるソフトウェア群の全体に基づいて閾値を決めることが一般的であるため，1407 個の C++ファイルのメトリクスデータを入力とした．
- 教師あり学習を用いる手法では，エキスパートによるレビューの対象とした 143 個の C++ファイルのメトリクスデータと教師データを入力とした．

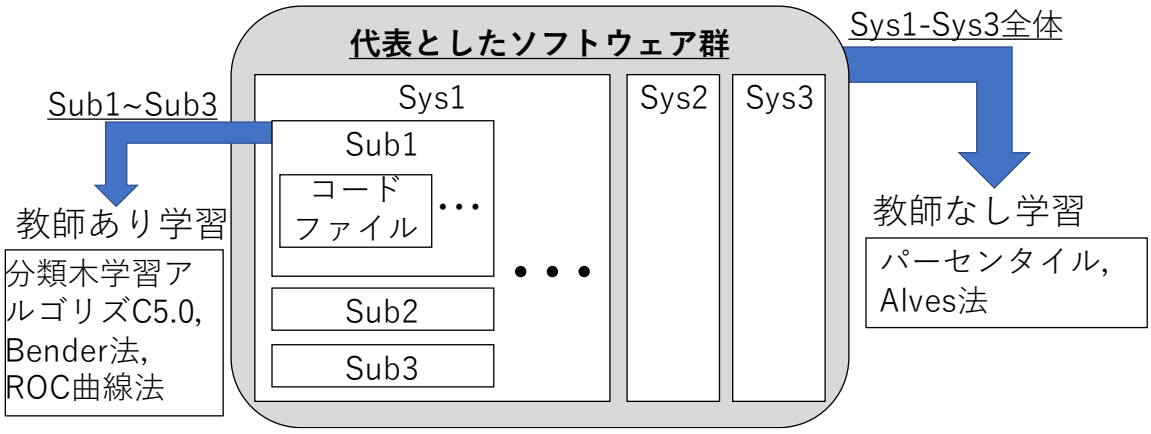

図 3-14：実験におけるベンチマークデータの使い分け

そして，教師あり学習を用いる手法については，それぞれ，以下の三つの規則での実験を 100 回繰り返し，平均値を測定結果とした．

- 規則 1) 143 個の教師データから 72 個(50%) をランダム選択して学習用候補とし，残りをテストデータとする．
- 規則 2) 学習用候補からさらに n%をランダム選択して学習に用いる．その際，n の値は 10 から始め，5 刻みで 100 までを対象とする．
- 規則 3) 規則 3 のランダム選定では正例と負例が必ず一件以上含まれるように調整する．学習データに正例か負例の片方しか存在しない場合には，正例と負例を隔てる閾値の導出ができないため，ランダム選定をし直す．

また，パーセンタイル関数，Alves 法，Bender 法では，リスクの大きさを仮定して設定する引数の値次第で導出される閾値が変わる．そのため，本研究では以下のように表記してそれぞれを区別した．

- パーセンタイル関数の引数に 10 パーセントを指定したものは P10 と表記する．
- Alves 法の引数に 10 パーセントを指定したものは A10 と表記する．
- Bender 法の引数に 0.10 を指定したものは B10 と表記する．

3.6.2. 結果（教師データを用いない閾値導出手法ついて）

本節では実験結果を要約する折れ線図を示し，その読み方を説明する．図から読み取れる事象と，それに対する考察は 3.7.2 節および 3.7.3 節で説明する．

パーセンタイル関数と Alves 法のそれぞれの方法において自動評価を実施することで F 値を測定した．パーセンタイル関数と Alves 法では，リスク境界として仮定するパーセンテージ次第で導出される閾値が変わる(自動評価の結果も変わる)．本実験では，10%毎のパーセンテージそれぞれを仮定した場合の F 値を測定し，折れ線を引いて図示するものとした．

Q1 と Q2 の測定結果をそれぞれ図 3-15 と図 3-16 に示した．この図の横軸と縦軸はそれぞれ，パーセンテージと F 値の大きさを意味する．

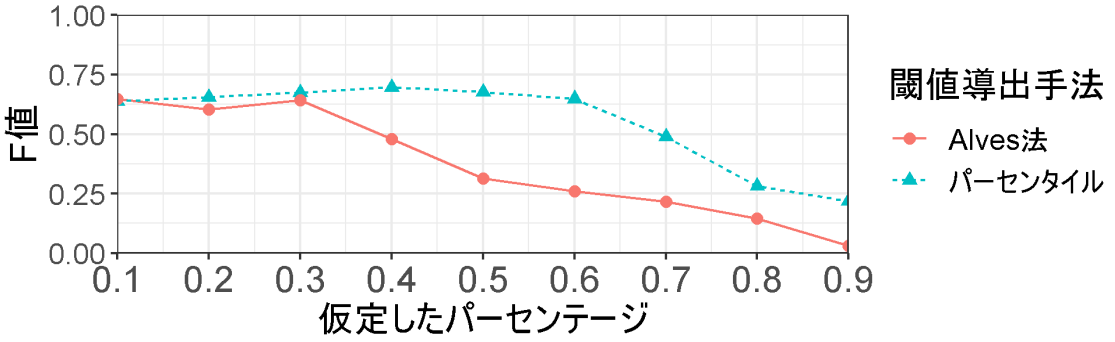

図 3-15：パーセンタイル関数と Alves 法の F 値（Q1）.

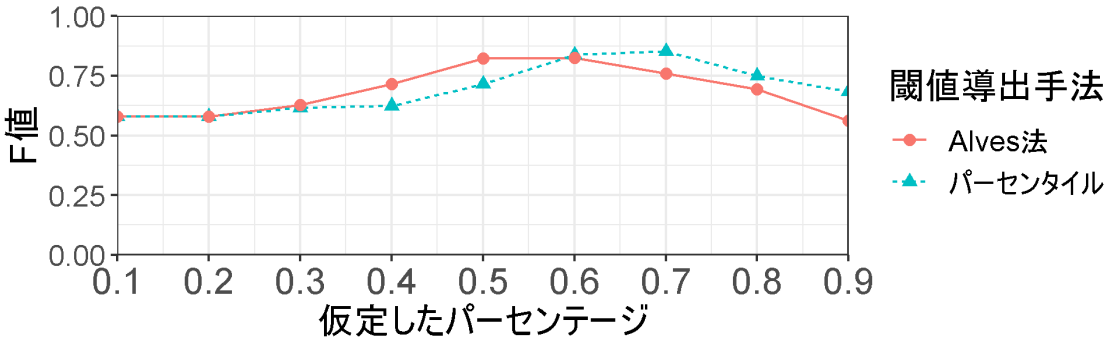

図 3-16：パーセンタイル関数と Alves 法の F 値（Q2）.

3.6.3. 結果（教師あり学習を用いる閾値導出手法ついて）

Bender 法，ROC 曲線法，分類木学習アルゴリズム C5.0，それぞれの方法において自動評価を実施することで F 値，Precision，Recall を測定した．

Q1 と Q2 の F 値の測定結果はそれぞれ図 3-17 と図 3-18 に示したこの図の横軸と縦軸はそれぞれ，手持ちの教師データのうち学習に用いた個数(および割合)と F 値の大きさを意味する．以下に図を読み比べる際の留意点を述べる．

- 図 3-15 と図 3-16 の横軸は手法の引数の値を意味する．
- 図 3-17 と図 3-18 の横軸は教師データ全体のうちの使用率を意味する．

本実験では，5%毎にそれぞれの割合で学習した場合の F 値を測定し，折れ線を引いて示した．加えて，比較対象として教師データを用いない閾値導出手法で測定した F 値について特徴的な位置に直線を引いて示した．図 3-17 と図 3-18 で直線を引いたのは，P80 と一番 F 値が大きかったパーセンタイルの位置である．

P80 については，80 対 20 の法則の知名度が高く，教師データなしの場合に利用される可能性が高いと考えられるため用いた．

一番 F 値が大きかったパーセンタイルについては，Q1 では P40，Q2 では P70 であった．あらかじめ問題のあるファイルの割合のノウハウがあるならば，教師データなしでも高性能なパーセンタイルを導ける可能性があり，比較対象としての意義がある．教師あり学習を用いる閾値導出手法には，これらの二つのターゲットを越えることが期待されている．

そして，図 3-17 と図 3-18 は C5.0 が比較対象の中では最も高性能であることを示している．また，他の手法では教師あり学習を用いているものの，パーセンタイル関数の性能を下回る場合があることを示している．

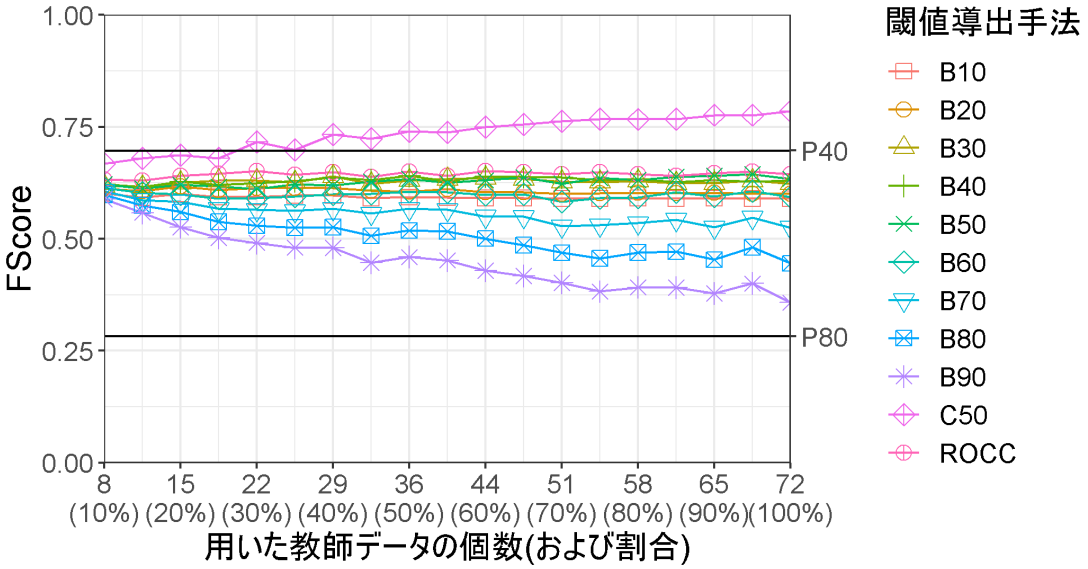

図 3-17：教師データの量と F 値の関係（Q1）.

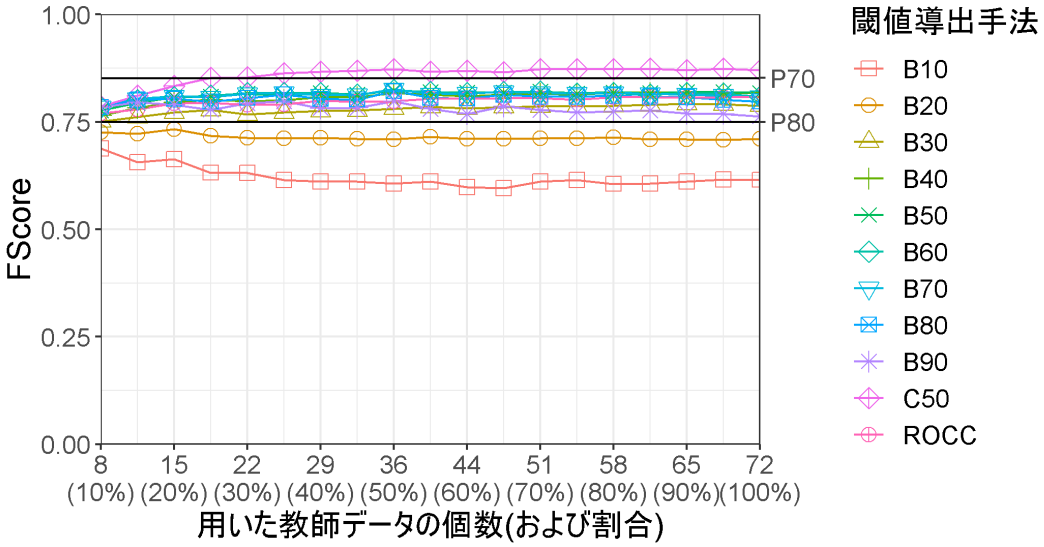

図 3-18：教師データの量と F 値の関係（Q2）.

ここまでの F 値の結果に加えて，教師あり学習の比較では，より詳細な分析のために Precision と Recall も同様に測定した．Q1 と Q2 の Precision は図 3-19 と図 3-20 に示した．Q1 と Q2 の Recall は図 3-21 図 3-22 に示した．それぞれ，F 値の図と同様の形式である．

Q1 と Q2 の双方で共通した傾向を以下に述べる．

- C5.0 では教師データを増やすにつれ，再現率と適合率が双方とも上昇する傾向であった．
- ROC 曲線法や Bender 法の B40，B50 等は再現率と適合率が双方ともほぼ一定という傾向であった．
- B10，B20 等は再現率が高い一方で適合率が低いという傾向であった．
- B90，B80 等は再現率が低い一方で適合率が高いという傾向であった．

C5.0 は他の手法に比べて Precision が高く，Recall も減少傾向ではなかったため，高い F 値が得られたと考えられる．他の手法は，Precision と Recall のどちらか一方が高くもう片方が低い，あるいはどちらも中程度という傾向であった．

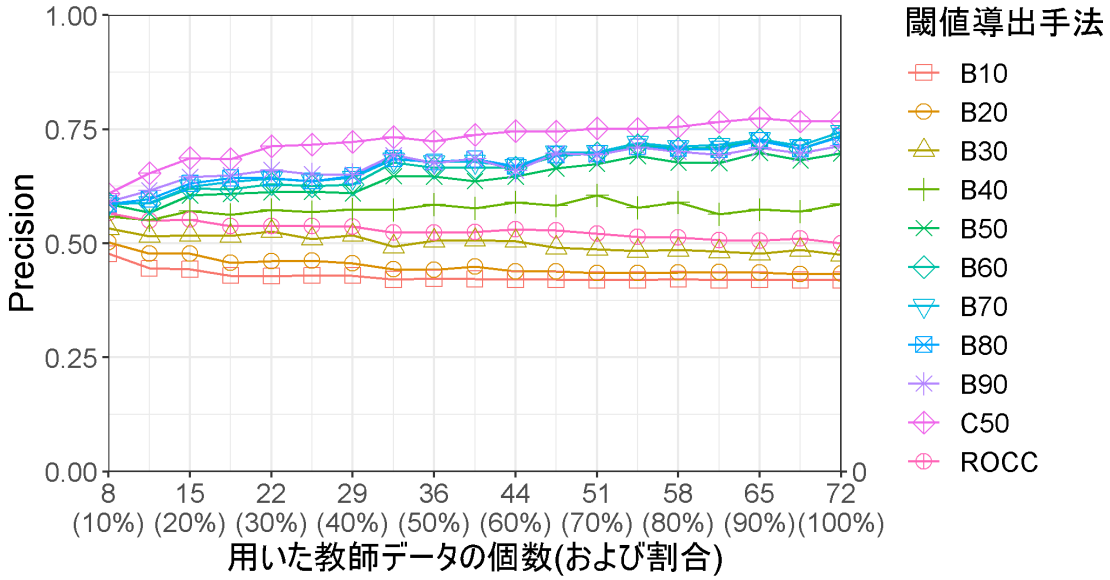

図 3-19：教師データの量と Precision の関係（Q1).

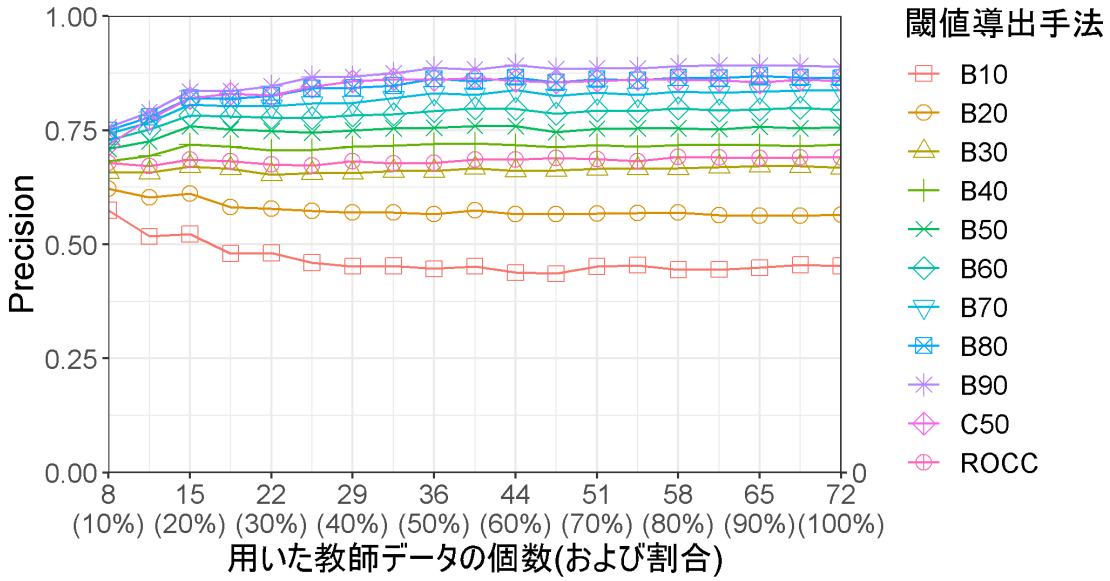

図 3-20：教師データの量と Precision の関係（Q2).

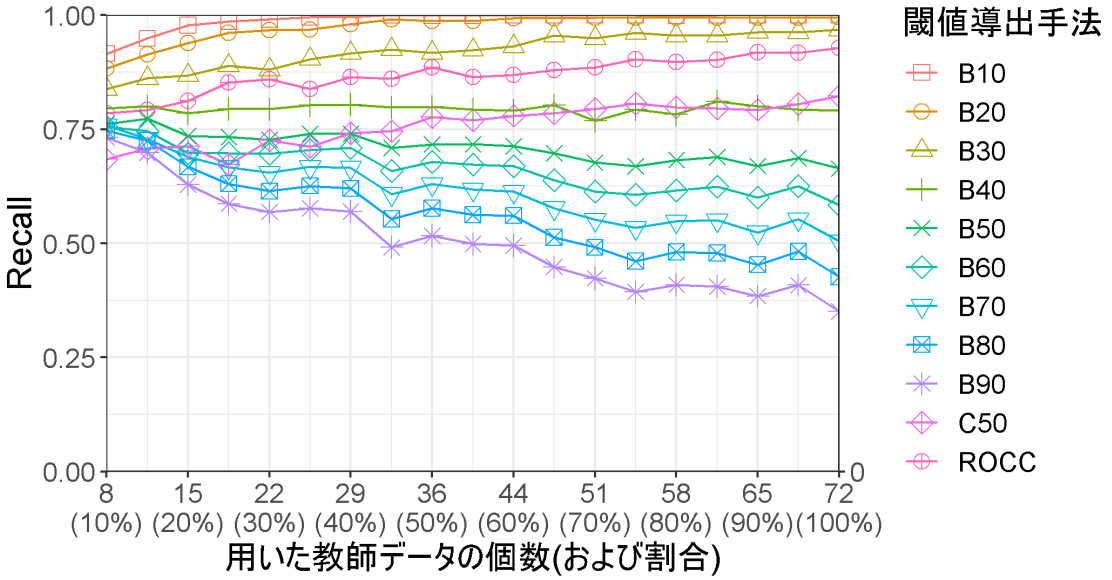

図 3-21：教師データの量と Recall の関係（Q1).

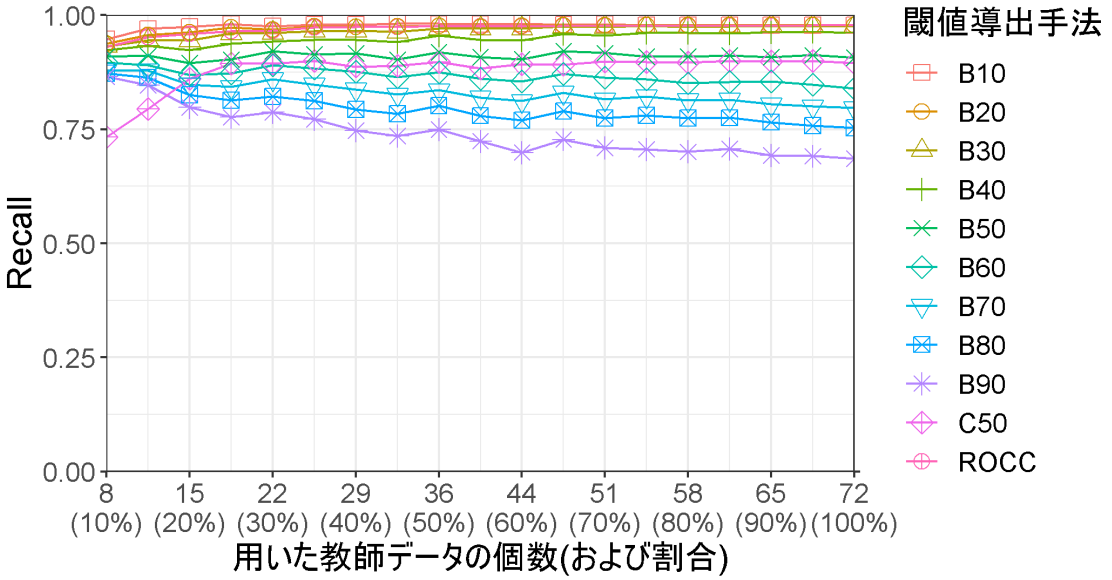

図 3-22：教師データの量と Recall の関係（Q2).

3.7. 考察

本節では RQ1 から RQ3 について，実験結果に基づいて考察する.

3.7.1. RQ1：我々の枠組みにより，実用的な評価基準を得られるか?

実験 1 の結果は，RQ1 に対して肯定的であった.

3.5.3 節で述べた通り，本研究の Step7 では，対象コンテキストに精通する

開発担当者 3 名を含む 5 名による議論を通じて，Q1 と Q2 の双方の評価基準について精度が高いことと，内容に一定の妥当性があることを確認した．

今後の課題としては，評価基準の変化について調査の余地がある．対象コンテキスト内で教師データを追加する場合や，別のコンテキストで同様の評価基準を構築する場合において，解釈モデルと閾値の変化をどのように解釈してどのように管理するかは明らかではない．

3.7.2. RQ2：少量の教師データを入力とした場合でも，分類木学習により他の閾値導出手法に比べ高精度な評価基準を導出できるか?

実験 2 の結果は，RQ2 に対して肯定的であった．

図 3-17 と図 3-18 は，C5.0 が他の閾値導出手法よりも高性能であることを示している． Q1 と Q2 のいずれにおいても，用いる教師データが 26 個(図の横軸目盛りの 35%)以上の場合，C5.0 は他の手法よりも大きい F 値を示している．また，必要となった教師データ数も十分に少ない量と考えられる．

以下，比較対象とした手法の傾向についてまとめる．

パーセンタイルについて：

教師データを用いない手法であるため，性能が低い場合が多かった．しかし，一部の設定のパーセンタイルでは ROC 曲線法と Bender 法を上回る性能を示した．特徴的な点を以下にまとめた．

- Q1 の P80 は F 値が 0.28 であり性能が低い．特に，誤って問題なしとみなしてしまう偽陰性の判定結果が 34%と多かった．P80 の閾値が大きすぎるために，リスクのあるファイルを多く許容してしまっていた．
- Q2 の P80 は F 値が 0.75 であり性能が高い．
- 他に F 値が最上位のパーセンタイルを確認すると，Q1 では P40 が 0.70 を示し，Q2 では P70 が 0.85 を示した．双方共に ROC 曲線法と Bender 法と同等あるいは上回る性能であった．

ROC 曲線法と Bender 法について：

教師あり学習をしている ROC 曲線法と Bender 法において問題となるのは，パーセンタイルを大きく差を付けて上回ることがなかった点である．さらに上述の通り，一部の設定のパーセンタイルよりも性能が下回っている．この結果については，ROC 曲線法と Bender 法の解釈モデルの制約と教師データの内容が影響していたと考えられる．影響についての考察を以下にまとめた．

- Q1 のように，エキスパートがコードの複数の特徴を複合的に解釈している場合には，メトリック一種毎に閾値を最適化する解釈モデルが適さ

ない可能性がある．この場合，C5.0 を用いることが望ましいと考えられる．

- Q2 のように，エキスパートがパーセンタイルに類する判断をしている場合には，一部の教師あり学習とパーセンタイル関数とで性能に差が出にくいと考えられる．

3.7.3. RQ3：教師データの量は，自動評価の精度にどう影響するか?

手法によって，使用する教師データの増加に対する自動評価の精度改善度合いが異なった．C5.0 では教師データを増やすことによる精度の改善が見られた．一方で，ROC 曲線法と Bender 法では，教師データを増やしても F 値があまり向上しない，あるいは逆に下がる結果になった．このことから，測定評価ツールにおいて利用可能であれば，C5.0 で導出した評価基準を用いる方がよいと考えられる．

以下，C5.0 について考察する．まず，C5.0 について観察できた事柄を述べ，次に原因を推察して知見としてまとめる．

Q1 と Q2 を比べると，教師データの割合に対する性能の変化量が異なることを図 3-17 と図 3-18 が示している．そこで，F 値の変化量を求めるために，教師データの割合について，C5.0 が他の閾値導出手法よりも高性能となり始める 35%の場合と，全教師データを用いる 100%の場合とで比較をした．その結果，F 値の差は Q1 では 0.08(=0.78-0.70)，Q2 では 0.01(=0.87-0.86) であった．教師データの増加に対して，Q1 では約 10%の精度が改善されている一方で，Q2 では 1%しか精度が改善されていない．このことと RQ2 での考察から，以下のことが推測できる．

- Q1 のように複合的な解釈の学習では，より多くの教師データが役立つ．
- Q2 のように元々が単純な解釈の場合では，非常に少ない教師データでも十分となりうる．

今回の実験結果から，評価対象の観点(GQM の Question) によって十分となる教師データの数が異なると推測できる．例えば，処理の複雑度の低減は関数を分割することで比較的一律にできるが，ファイルが担う規模(必要な機能)の削減は突き詰めるとアルゴリズムや設計の見直しが必要となる．そのため，コード規模の適切さの評価(Q1)は複雑さの評価(Q2)よりも比較的に難しく，エキスパートが複合的な解釈をする可能性がある．

以下，他の手法について考察する．

ROC 曲線法と Bender 法について，教師データの増加に対して F 値の改善が小さい，あるいは逆に下がっていることを図 3-17 と図 3-18 が示してい

る．

- ROC 曲線法や B40，B50 等については，再現率と適合率が双方ともほぼ一定という傾向であった．これらの手法ではメトリクスの複合的な解釈の最適化ができずに性能が伸び悩んでいたと考えられる．
- B90，B80 等については，再現率が低い一方で適合率が高いという傾向であった．特に Q1 の B90 では教師データを増やすにつれて F 値が下がっていた．Bender 法の引数 p0 を 0.90 に設定すると，楽観的に問題個所を少なく見積もることになる．そのため，極端に大きな閾値が導出され，問題ありとみなすべきファイルの多くを許容してしまったと考えられる．
- B10，B20 等については，再現率が高い一方で適合率が低いという傾向であった．特に， Q2 の B10 では教師データを増やすにつれて F 値が下がっていた．Bender 法の引数 p0 を 0.10 に設定すると，悲観的に問題個所を多く見積もることになる．そのため，極端に小さな閾値が導出され，問題なしとみなすべきファイルの多くを問題個所として誤検出してしまったと考えられる．

3.7.4. 関連研究との比較

3.2.4 節で述べた Fontana らの研究 [57]では，4 種類のコードスメルの自動評価について我々の実験よりも高い F 値が出ていた．本研究との差の原因としては，機械学習時に説明変数としたメトリクスの数の影響が考えられる．

Fontana らは 60 個以上のメトリクスを使用して評価基準を学習していた．一方で我々は，人間が直感的に解釈可能な評価基準の導出を意図し，GQM(Goal-Question-Metrics) 法で説明変数となるメトリクスの候補を選定し，Q1 と Q2 でそれぞれ 2 個と 4 個のメトリクスを使用していた．メトリクスを限定する方法の評価については今後の課題である．

3.8. 妥当性への脅威

内的妥当性への脅威：

教師データを作成したエキスパートが 1 人だけであり，レビュー実施時の集中力や先入観といった個人的要因の影響を排せていない点は内的妥当性への脅威である．しかし，このエキスパートは対象企業において 10 年以上の開発経験を持つ設計者であったため対象コンテキストには十分に精通していたため，教師データの内容には一定の妥当性があったと考えられる．

また，評価基準が対象コンテキストの開発者の認識に著しく反していないこ

とをケーススタディで確認しており,教師データには一定の妥当性があったと考えられる．枠組みの Step 7 において，対象企業の開発者 3 名と研究室メンバ 2 名によって，評価基準を可視化して判別境界付近のデータを確認した．

外的妥当性への脅威：

本研究では，結果的に，教師データを作成するためのレビュー対象ファイルを Sys1 のみから選定していた．そのため，機械学習時に過学習をし，Sys2 と Sys3 に適さない評価基準が導出される恐れがあった．一方で，いずれのシステムも同一の組織によって開発されており,一定の共通性が認められると考えられる．そして，同組織の開発者からのフィードバックでは，導出した分類木を用いた Sys2 と Sys3 の評価結果について一定の妥当性が認められた．このことから,同一のコンテキストで評価基準を共有するという我々の想定にも一定の妥当性があったと考えられる.より具体的な実証実験については今後の課題である．

本研究では，元々の教師データが 143 個と少量であった．そこで，サンプルの割合を変動させるのに十分な学習データを確保するために，2-fold 交差検証に類する形で実験をした．しかし,2 回しか機械学習を実施しない 2-fold の形式では，10-fold の形式に比べてデータの偏りの影響を低減しにくいという欠点がある．本研究ではこの妥当性への脅威を低減するために，の実験を 100 回繰り返して，その平均値を測定結果とした．

3.9. 本章のおわりに

本研究では，コードの保守性の内，機能的な不具合を伴わないコードの構造上の問題に焦点を当て,閾値に基づく評価基準を統計的に導出するための手順を枠組みとして整理した．さらに，従来の一般的な閾値導出手法と分類木学習の実用性を比較するために,少量の教師データにより高精度かつ開発者の認識に反しない評価基準を導出できるかという観点で評価実験をした．

枠組みにおいて利用した分類木学習アルゴリズム C5.0 について得られた知見は以下の通りである．

- 少量の教師データでも高精度かつ開発者の認識に反しない評価基準を導出できた．80 パーセンタイルや ROC 曲線法等を上回る性能を期待できる．
- 教師データを増やすことで自動評価の精度が改善した．一方で，Bender 法と ROC 曲線法では，顕著な改善は見られなかった．

- コードの複雑度の評価基準よりも，コードの規模の評価基準の方が多数の教師データを高精度化のために必要とした．ファイル単位での，規模と複雑さの許容判断の難しさの違いが影響していた可能性がある．

本研究は，高精度かつ開発者の認識に反しない実用的な評価基準を得るための方法について，限られた一端を明らかとすることにつながったと考えられる．しかし，今回の評価実験の結果は一つの企業における一つのコンテキストにおいて限定的に得られたものである．

今後の課題は，題材としたコンテキストにおける教師データを追加して分析を深めること，および異なるコンテキストでの調査を始めることである．今後，協力企業において異なるシステムにも枠組みを適用することを検討している．

また，評価対象がコードの構造である性質上，機械学習の教師データ（正解）をエキスパートが目視で与えているため，効率化方法としてレビュー対象の優先度付けを検討している．

4. おわりに

4.1. 要約

本論文では，ソフトウェア製品の品質評価枠組みについて，開発組織を越える場合と開発組織内の場合とでは，主体的に評価枠組みを運用する組織が異なることに着目した．そして，双方におけるベンチマークの用途を考慮して，枠組みの運用上の課題の解決に取り組んだ．

第 2 章では，開発組織を越えた品質評価に向けて，国際規格 SQuaRE に基づく網羅的なソフトウェア品質評価枠組みを提案した．そして，12 社の開発組織と協力して 21 製品のデータを集積し，製品品質と利用時の品質の間の関係について，その限られた一端を明らかにした．そして，産業界への提言をまとめると共に，実現した枠組みと品質ベンチマークを公開した．

第 3 章では，開発組織内での品質評価に向けて，評価基準を構築するための枠組みを整理した．そして，開発組織における特定コンテキストのソースコードの保守性評価を題材として，1 社の開発組織と協力して共通のプラットフォームで開発されているコードのデータを集積し，複数の閾値導出手法の性能と性質を比較した．また，枠組みと手法については，コードの保守性以外についても汎用的に適用可能なものを扱った．

これらの取り組みの結果，我々の枠組みのそれぞれを適用した開発組織においては，ソフトウェアの品質を効果的に把握することができた．

また，これらの枠組みの相乗的な利用方法としては，以下の形態が考えられる．まず，開発組織においては，第 2 章の枠組みを利用することで産業界における自製品のおおまかなポジションを品質計画の早期に参照することが期待できる．そして，それを判断材料として組織内で重視する品質を定め，第 3 章の枠組みによって緻密な品質評価を実現することが期待できる．

4.2. 今後の展望

本論文の今後の展望について図 4-1 に整理した．本論文での二つの取り組みは図における左上(第 2 章)と右下(第 3 章)に位置しており，双方を取り進めることが，将来的に一定の多面性と詳細性を確保した品質評価枠組みを実現することにつながると考えられる．

以下，それぞれについて，課題と共に対策を述べる．

国際規格に基づく網羅的なソフトウェア品質評価枠組みおよび品質ベンチマーク(第 2 章)については，開発時の目標値の記録不足といった理由から，品質測定方法によってはデータを得られる製品数が限られた．これらの課題の緩和に向けては，まずはデータの母数を増やすことや，代替測定の検討が対策として考えられる．現在まで，2 製品の追加データを得ている．データ数が一定数に達した時点で，より包括的に品質特性間やその構成要素間の関係分析を試みる．これらの活動により，枠組みの多面性と詳細性の改善を目指す予定である．

ソースコードの保守性評価を対象とした特定コンテキスト向けの枠組み(第 3 章)の評価実験においては，取り扱った範囲が一つの企業における一つのコンテキストに限られた．また，評価対象がコードの構造であった性質上，機械学習の教師データ(正解)をエキスパートが目視で与えており，用意できた件数と評価観点が限られた．これらの課題の緩和に向けては，エキスパートのレビューを効率化することが対策として考えられる．その方法のひとつとしては，レビュー対象の優先度付けが考えられる．この活動により，評価実験の外的妥当性や枠組みの多面性の改善を目指す予定である．

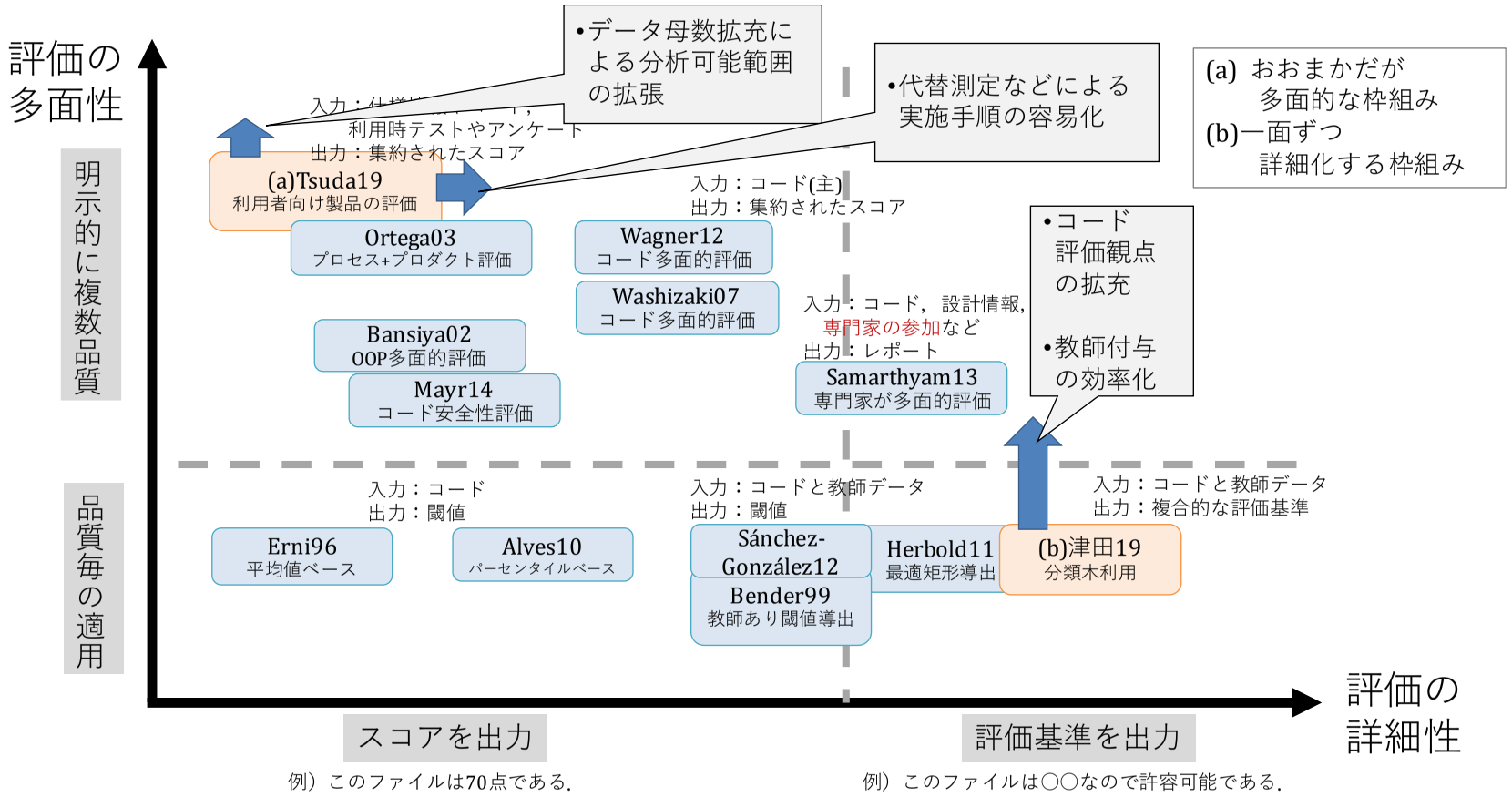

図 4-1：研究領域における今後の展望の位置付け

参考文献

[1] 東 基衛, "システム・ソフトウェア品質標準 SQuaRE シリーズの歴史と概要," *SEC journal,* vol. 10, pp. 18-22, 1 2015.

[2] ISO/IEC, "ISO/IEC 25000:2014 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE," ISO/IEC, 2014.

[3] A. Abran, A. Khelifi, W. Suryn and A. Seffah, "Usability Meanings and Interpretations in ISO Standards," *Software Quality Journal,* vol. 11, pp. 325-338, 11 2003.

[4] J. Heidrich, D. Rombach and M. Kläs, "Model-Based Quality Management of Software Development Projects," in *Software Project Management in a Changing World*, G. Ruhe and C. Wohlin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 125-156.

[5] J. Bansiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Trans. Softw. Eng.,* vol. 28, pp. 4-17, 1 2002.

[6] M. Ortega, M. Pérez and T. Rojas, "Construction of a Systemic Quality Model for Evaluating a Software Product," *Software Quality Journal,* vol. 11, pp. 219-242, 01 7 2003.

[7] G. Samarthyam, G. Suryanarayana, T. Sharma and S. Gupta, "MIDAS: A Design Quality Assessment Method for Industrial Software," in *Proc. ICSE*, San Francisco, CA, USA, 2013.

[8] A. Mayr, R. Plösch and M. Saft, "Objective Safety Compliance Checks for Source Code," in *Companion Proc. of ICSE*, 2014.

[9] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb and J. Streit, "The Quamoco Product Quality Modelling and Assessment Approach," in *Proc. ICSE*, 2012.

[10] H. Washizaki, R. Namiki, T. Fukuoka, Y. Harada and H. Watanabe, "A Framework for Measuring and Evaluating Program Source Code Quality," in *Proc. of the 8th Int. Conf. on PROFES*, Riga, 2007.

[11] N. Moha, Y. G. Gueheneuc, L. Duchien and A. F. L. Meur, "DECOR: A Method for the Specification and Detection of Code and Design Smells,"

IEEE Transactions on Software Engineering, vol. 36, pp. 20-36, 1 2010.

[12] N. Tsantalis and A. Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities," *IEEE Transactions on Software Engineering,* vol. 35, pp. 347-367, 5 2009.

[13] R. Marinescu, "Detection strategies: metrics-based rules for detecting design flaws," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, 2004.

[14] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," *IBM Journal of Research and Development,* vol. 56, pp. 9:1-9:13, 9 2012.

[15] N. Tsuda, H. Washizaki, K. Honda, H. Nakai, Y. Fukazawa, M. Azuma, T. Komiyama, T. Nakano, H. Suzuki, S. Morita, K. Kojima and A. Hando, "WSQF: Comprehensive Software Quality Evaluation Framework and Benchmark Based on SQuaRE," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019.

[16] 津田 直彦, 鷲崎 弘宜, 深澤 良彰, 保田 裕一朗 and 杉村 俊輔, "プログラムソースコードの分かりやすさの閾値に基づく評価基準とその導出手法群の評価," *情報処理学会論文誌,* vol. 60, pp. 804-820, 3 2019.

[17] K. Kojima, S. Morita, T. Hirose, M. Wakamoto, S. Kikuchi, A. Hando and H. Washizaki, "Evaluation of Impact of Software Quality Assuarance Technique on Quality Characteristics," *SEC journal,* vol. 14, pp. 50-57, 8 2018.

[18] T. L. Alves, C. Ypma and J. Visser, "Deriving Metric Thresholds from Benchmark Data," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, 2010.

[19] K. Erni and C. Lewerentz, "Applying design-metrics to object-oriented frameworks," in *Proceedings of the 3rd International Software Metrics Symposium*, 1996.

[20] R. Bender, "Quantitative risk assessment in epidemiological studies investigating threshold effects," *Biometrical Journal,* vol. 41, pp. 305-319, 1999.

[21] L. Sánchez-González, F. García, F. Ruiz and J. Mendling, "A study of the effectiveness of two threshold definition techniques," in *16th International Conference on Evaluation Assessment in Software*

Engineering (EASE 2012), 2012.

[22] S. Herbold, J. Grabowski and S. Waack, "Calculation and Optimization of Thresholds for Sets of Software Metrics," *Empirical Softw. Engg.,* vol. 16, pp. 812-841, 12 2011.

[23] I. Biscoglio and E. Marchetti, "Definition of Software Quality Evaluation and Measurement Plans: A Reported Experience Inside the Audio-Visual Preservation Context," in *ICSOFT*, 2014.

[24] ISO/IEC, "ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," ISO/IEC, 2010.

[25] ISO/IEC, "ISO/IEC 25022:2016 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Measurement of quality in use," ISO/IEC, 2016.

[26] ISO/IEC, "ISO/IEC 25023:2016 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Measurement of system and software product quality," ISO/IEC, 2016.

[27] IPA/SEC, *IPA/SEC White Paper 2007 on Software Development Projects in Japan.*

[28] M. Cusumano, A. MacCormack, C. F. Kemerer and B. Crandall, "Software Development Worldwide: The State of the Practice," *IEEE Softw.,* vol. 20, pp. 28-34, 11 2003.

[29] C. Jones, Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies, 1 ed., New York, NY, USA: McGraw-Hill, Inc., 2010.

[30] D. Russo, P. Ciancarini, T. Falasconi and M. Tomasi, "Software Quality Concerns in the Italian Bank Sector: The Emergence of a Meta-quality Dimension," in *Proc. ICSE-SEIP*, Buenos, 2017.

[31] CSAJ, *PSQ Certification System.*

[32] ISO/IEC, "ISO/IEC 25051:2014 Software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Requirements for quality of Ready to Use Software Product (RUSP) and instructions for testing," ISO/IEC, 2014.

[33] *WSQB17: Waseda Software Quality Benchmark.*

[34] K. Honda, H. Washizaki and Y. Fukazawa, "Generalized Software

Reliability Model Considering Uncertainty and Dynamics: Model and Applications," *International Journal of Software Engineering and Knowledge Engineering,* vol. 27, p. 967, 2017.

[35] H. Washizaki, "Chapter One - Pitfalls and Countermeasures in Software Quality Measurements and Evaluations," *Advances in Computers,* vol. 106, pp. 1-22, 2017.

[36] V. R. Basili, G. Caldiera and H. D. Rombach, "Goal, Question, Metric Paradigm," in *Encyclopedia of Software Engineering*, Wiley, 1994.

[37] M. Lanza and R. Marinescu, Object-oriented Metrics in Practice, Berlin: Springer-Verlag, 2006.

[38] M. V. Mäntylä and C. Lassenius, "What Types of Defects Are Really Discovered in Code Reviews?," *IEEE Transactions on Software Engineering,* vol. 35, pp. 430-448, 5 2009.

[39] P. Kruchten, R. L. Nord and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," *IEEE Software,* vol. 29, pp. 18-21, 11 2012.

[40] F. Zhang, A. Mockus, Y. Zou, F. Khomh and A. E. Hassan, "How Does Context Affect the Distribution of Software Maintainability Metrics?," in *2013 IEEE International Conference on Software Maintenance*, 2013.

[41] J. Y. Gil and G. Lalouche, "When do Software Complexity Metrics Mean Nothing? - When Examined out of Context," *Journal of Object Technology,* vol. 15, pp. 2:1--25, 2016.

[42] L. Lavazza and S. Morasca, "An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measures," in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, Ciudad, 2016.

[43] R. Shatnawi, "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems," *IEEE Transactions on Software Engineering,* vol. 36, pp. 216-225, 3 2010.

[44] M. V. Mäntylä and C. Lassenius, "Subjective Evaluation of Software Evolvability Using Code Smells: An Empirical Study," *Empirical Softw. Engg.,* vol. 11, pp. 395-431, 9 2006.

[45] G. Concas, M. Marchesi, S. Pinna and N. Serra, "Power-Laws in a Large Object-Oriented Software System," *IEEE Trans. Softw. Eng.,* vol. 33, pp. 687-708, 10 2007.

[46] R. Wheeldon and S. Counsell, "Power law distributions in class relationships," in *Proceedings Third IEEE International Workshop on Source Code Analysis and Manipulation*, 2003.

[47] F. A. Fontana, V. Ferme, M. Zanoni and A. Yamashita, "Automatic Metric Thresholds Derivation for Code Smell Detection," in *Proceedings of the Sixth International Workshop on Emerging Trends in Software Metrics*, Florence, 2015.

[48] G. Vale, D. Albuquerque, E. Figueiredo and A. Garcia, "Defining Metric Thresholds for Software Product Lines: A Comparative Study," in *Proceedings of the 19th International Conference on Software Product Line*, Nashville, 2015.

[49] L. Veado, G. Vale, E. Fernandes and E. Figueiredo, "TDTool: Threshold Derivation Tool," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, Limerick, 2016.

[50] T. J. Ostrand, E. J. Weyuker and R. M. Bell, "Where the Bugs Are," in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, Boston, 2004.

[51] S. R. Chidamber, D. P. Darcy and C. F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Trans. Softw. Eng.,* vol. 24, pp. 629-639, 8 1998.

[52] K. Lochmann, "A benchmarking-inspired approach to determine threshold values for metrics," *SIGSOFT Softw. Eng. Notes,* vol. 37, pp. 1-8, 11 2012.

[53] R. Shatnawi, W. Li, J. Swain and T. Newman, "Finding Software Metrics Threshold Values Using ROC Curves," *J. Softw. Maint. Evol.,* vol. 22, pp. 1-16, 1 2010.

[54] E. Bouwers, J. Visser and A. Deursen, "Getting What You Measure," *Commun. ACM,* vol. 55, pp. 54-59, 7 2012.

[55] M. T. Ribeiro, S. Singh and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA, 2016.

[56] J. R. Quinlan, C4.5: Programs for Machine Learning, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[57] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni and A. Marino, "Comparing

and Experimenting Machine Learning Techniques for Code Smell Detection," *Empirical Softw. Engg.,* vol. 21, pp. 1143-1191, 6 2016.

[58] A. Bosu and J. C. Carver, "Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Torino, 2014.

[59] F. Palomba, D. D. Nucci, M. Tufano, G. Bavota, R. Oliveto, D. Poshyvanyk and A. De Lucia, "Landfill: An Open Dataset of Code Smells with Public Evaluation," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, Florence, 2015.

[60] B. Robinson and P. Francis, "Improving Industrial Adoption of Software Engineering Research: A Comparison of Open and Closed Source Software," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Bolzano-Bozen, Italy, 2010.

[61] V. R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Softw. Eng.,* vol. 10, pp. 728-738, 11 1984.

[62] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

[63] M. V. Mäntylä, "Empirical Software Evolvability - Code Smells and Human Evaluations," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, 2010.

[64] A. Monden, T. Matsumura, M. Barker, K. Torii and R. Basili Victor, "Customizing GQM Models for Software Project Monitoring," *IEICE Trans. on Information and Systems,* vol. 95, pp. 2169-2182, 2012.

[65] F. Rahman, D. Posnett and P. Devanbu, "Recalling the "Imprecision" of Cross-project Defect Prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, Cary, 2012.

[66] K. Yamashita, C. Huang, M. Nagappan, Y. Kamei, A. Mockus, A. E. Hassan and N. Ubayashi, "Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density," in *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2016.

[67] C. Taube-Schock, R. J. Walker and I. H. Witten, "Can We Avoid High Coupling?," in *Proceedings of the 25th European Conference on Object-oriented Programming*, Lancaster, 2011.

[68] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis and S. N. Rai, "The Optimal Class Size for Object-Oriented Software," *IEEE Trans. Softw. Eng.,* vol. 28, pp. 494-509, 5 2002.

[69] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Trans. Softw. Eng.,* vol. 25, pp. 675-689, 9 1999.

[70] M. Aniche, C. Treude, A. Zaidman, A. Deursen and M. A. Gerosa, "SATT: Tailoring Code Metric Thresholds for Different Software Architectures," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2016.

[71] T. J. McCabe, "A Complexity Measure," *IEEE Trans. Softw. Eng.,* vol. 2, pp. 308-320, 7 1976.

研究業績

ジャーナル論文

- ○ プログラムソースコードのわかりやすさの閾値に基づく評価基準とその導出手法群の評価
  津田直彦, 鷲崎弘宜, 深澤良彰, 保田裕一朗, 杉村俊輔, 情報処理学会論文誌, Vol. 60, No. 3, 2019 年 3 月, pp. 804--820.
- ● 国際規格に基づく総合的なソフトウェア品質評価枠組みとその実製品適用による品質ベンチマーク
  鷲崎弘宜, 津田直彦, 本田澄, 中井秀矩, 深澤良彰, 東基衛, 込山俊博, 中野正, 鈴木啓紹, SEC journal, Vol. 14, No. 1, 2018 年 8 月, pp. 26-41.

国際会議論文

- ○ WSQF: Comprehensive Software Quality Evaluation Framework and Benchmark based on the SQuaRE
  Naohiko Tsuda, Hironori Washizaki, Kiyoshi Honda, Hidenori Nakai, Yoshiaki Fukazawa, Motoei Azuma, Toshihiro Komiyama, Tadashi Nakano, Hirotsugu Suzuki, Sumie Morita, Katsue Kojima, Akiyoshi Hando, Proceedings of the 41st ACM/IEEE International Conference on Software Engineering (ICSE 2019), Industry Track, IEEE, May 2019, pp. 312-321.
- ○ Machine Learning to Evaluate Evolvability Defects: Code Metrics Thresholds for a Given Context
  Naohiko Tsuda, Hironori Washizaki, Yoshiaki Fukazawa, Yuichiro Yasuda and Shunsuke Sugimura, The 18th IEEE International Conference on Software Quality, Reliability & Security (QRS 2018), July 2018, pp. 16-20.
- ○ Iterative Process to Improve GQM Models with Metrics Thresholds to Detect High-risk Files
  Naohiko Tsuda, Masaki Takada, Hironori Washizaki, Yoshiaki Fukazawa, Shunsuke Sugimura, Yuichiro Yasuda, Masanao Futakami, IEEE Region 10 Annual International Conference, Proceedings/TENCON, February 2017, pp.3813-3816.
- ● A SQuaRE-based Software Quality Evaluation Framework and Its Case

Study
Hidenori Nakai, Naohiko Tsuda, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, IEEE Region 10 Annual International Conference, Proceedings/TENCON, February 2017, pp.3704-3707.

- Initial Framework for Software Quality Evaluation Based on ISO/IEC 25022 and ISO/IEC 25023
  Hidenori Nakai, Naohiko Tsuda, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, Proceedings - 2016 IEEE International Conference on Software Quality, Reliability and Security-Companion, QRS-C, September 2016, pp.410-411.

国内会議論文

○ コードの発展性欠陥の自動評価：コンテキストを考慮したメトリクス閾値の機械学習
津田直彦，鷲崎弘宜，深澤良彰，保田裕一朗，杉村俊輔，情報処理学会第80回全国大会プログラム，2018年3月，pp. (1-179)-(1-180).

○ メトリクスと閾値による保守性・再利用性評価式の作成・更新プロセス
津田直彦，高田正樹，鷲崎弘宜，深澤良彰，杉村俊輔，保田裕一朗，二上将直，情報処理学会ソフトウェア工学研究会ウィンターワークショップ2015・イン・宜野湾，2015年01月，pp.1-2.

- 保守性・再利用性が低いファイルの予測：産業データを用いた研究
  津田 直彦,高田 正樹,鷲崎 弘宜,深澤 良彰, 杉村 俊輔,保田 裕一朗, 二上 将直, 情報処理学会第186回ソフトウェア工学研究発表会，2014年11月，pp.1-8.

講演

- コードメトリクスと閾値による保守性評価：品質要求に基づいたカスタマイズ
  津田直彦，鷲崎弘宜，深澤良彰，第24回ソフトウェア工学の基礎ワークショップ FOSE2017，ポスター，2017年11月.
- IPA RISE委託研究 2015-16年度 測定評価と分析によるソフトウェア製品品質の実態定量化および総合的品質評価枠組みの確立（報告セミナー用）
  鷲崎弘宜，津田直彦，本田澄，JISA Digital Masters Forum 2017，招待講演，2017年10月.

- 保守性に対する熟練者判断を用いたソースコード自動評価の最適化
  津田直彦, 鷲崎弘宜, 深澤良彰, 日本ソフトウェア科学会第 34 回大会, ポスター, 2017 年 09 月
- Customization Patterns for GQM Metrics-Layer: Optimization by Checklist Based Maintainability Review and Machine Learning
  Naohiko Tsuda, Hironori Washizaki, Yoshiaki Fukazawa, Shunsuke Sugimura, Yuichiro Yasuda, Masanao Futakami, 8th IEEE International Workshop on Empirical Software Engineering in Practice (IWESEP 2017), Fast Abstract, March 2017, pp.1.
- Iterative Process to Improve GQM Models with Metrics Thresholds to Detect High-risk Files
  Naohiko Tsuda, Masaki Takada, Hironori Washizaki, Yoshiaki Fukazawa, Shunsuke Sugimura, Yuichiro Yasuda, and Masanao Futakami, 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2015) Doctoral Symposium (Presentation), March 2015.
- 保守性・再利用性の低いファイル予測：プロジェクトに合わせた最適化の枠組み
  津田 直彦, 高田 正樹, 鷲崎 弘宜, 深澤 良彰, 杉村 俊輔, 保田 裕一朗, 二上 将直, 日本ソフトウェア科学会 第 21 回 ソフトウェア工学の基礎ワークショップ FOSE 2014 in 霧島, ポスター, 2014 年 12 月.
- Detect Low Maintainability and Reusability Files: Framework for Optimal Prediction in Consideration of a Project
  Naohiko Tsuda, Masaki Takada, Hironori Washizaki, Yoshiaki Fukazawa, Shunsuke Sugimura, Yuichiro Yasuda and Masanao Futakami, 6th International Workshop on Empirical Software Engineering in Practice (IWESEP 2014), Poster, November 2014.