社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

開発標準プロセスを用いた不完全なソフトウェア要求に対する問題検出の分類法

宮村　純真† 　川口　真司†† 　石濱　直樹†† 　柿本　和希†† 　飯田　　元†
片平　真史††

† 奈良先端科学技術大学院大学 情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
†† 宇宙航空研究開発機構　〒 305–8505 茨城県つくば市千現 2–1–1
E-mail: †miyamura.toma.mo6@is.naist.jp

あらまし　宇宙機（人工衛星やロケット）の開発および運用において，ソフトウェアの不具合はミッションの成功に対して大きな妨げとなっている．本研究ではソフトウェア不具合の大きな原因となる，要求漏れに着目した．実際に宇宙機で発生したソフトウェアの不具合を元に，欠陥の分類と開発標準プロセスの 2 つの観点から不具合の分類を行った．この分類を行うことで，欠陥の傾向や，プロセス間の影響度合いが明確になることが期待される．
キーワード　ソフトウェアプロセス，不具合，要求漏れ，欠陥の分類

Classification of problem detection for incomplete software requirements using the development standard process

Toma MIYAMURA†, Shinji KAWAGUCHI††, Naoki ISHIHAMA††, Kazuki KAKIMOTO††,
Hajimu IIDA†, and Masafumi KATAHIRA††

† Graduate School of Information Science, Nara Institute of Science and Technology Takayama-cho 8916-5, Ikoma-shi, Nara, 630-0192 Japan
†† Japan Aerospace eXploration Agency　Sengen 2–1–1 Tukuba-shi, 305–8505 Japan
E-mail: †miyamura.toma.mo6@is.naist.jp

Abstract　Software faults in spacecraft software leads to mission failures. To identify software faults, we focus on the incomplete software requirement, on reprehensive reason for the software faults. Based on two perspectives of software fault and development software process, we present a classification method for software faults which are extracted from actual spacecraft project (e.g., artificial satellite and rockets). We expect that this approach can help to identify the tendency of faults and reveal which certain kind of process leads to more faults.
Key words　Software process, Bug, Incomplete software requirement, Classification of faults

1. はじめに

システム開発の際に，ソフトウェアが原因で発生する不具合は大きく 2 つのパターンに分けることができる．1 つ目のパターンは “求められている要求は正しいが，ソフトウェアの実装が誤っている” パターンであり，2 つ目が “実装は要求通りであるが，要求が不完全である” パターンである．前者のソフトウェアの実装における不具合はツールなどを用いることで発見できる．また，ソフトウェアの実装における不具合の割合は非常に少ないことが既存研究で明らかにされている [1]．一方で，後者は，ツールが存在せず，要求仕様書等にも記載されていないため発見が困難である．よって，ソフトウェアの信頼性を高めるためには，ソフトウェアの実装による不具合だけでなく，要求漏れによる不具合についても考慮する必要がある．

Avizienis らは，不具合全体の要因の 1 つである欠陥に関する分類法を提案している [2]．しかし，既存手法における欠陥の分類は汎用的な観点に基づいており，分類結果に基づいて発生した欠陥に対する具体的な対策を考慮するのは困難である．そこで，提案手法では開発者が採用している開発プロセスに着目して分類することで，開発プロセス上のどこで欠陥が混入したかが明確になり，具体的な対処が実施可能になると考えた．よって本研究では，動的である開発工程（プロセス）に着目するこ

とで，要求が不完全であることに対して，対処策が発見できると考えた．

本研究では，宇宙機と呼ばれる，人工衛星やロケットなどで実際に発生した事例に対して，欠陥の分類と開発工程であるプロセスの 2 つの観点から不具合の分類・分析を行った．本分析により，対象システムにおいて，どのような欠陥に基づく不具合が多いのか，また異なるプロセスにおいて影響を及ぼしているのはどれであるかが明確になり，対策を講ずることが期待される．

本論文の構成を以下に示す．2 章では研究の背景について説明し，3 章で提案する分類の枠組みについて述べる．4 章では実際にデータを適用した結果を述べ，5 章で考察，6 章でまとめや今後の課題について述べる．

2. 背　　景

宇宙機開発の特徴として，一度打ち上げてしまうと容易に修理ができない，製品が単発，開発期間が長い，実環境でのテストを行うことができないなどがあげられる [3] [4]．宇宙機の開発では，ソフトウェアが不具合の主な原因となる場合がある．したがって，宇宙機のソフトウェア開発では，いかに不具合が発生しないかを示す高信頼性が強く求められる．高信頼性システムでは不具合が障害として顕著化しないような対策が必要である．しかし，不具合を減らす対策を行うためには，既存の静的な分類だけの対策では不十分である．

対策の例として，Alshazly らは要求仕様書における欠陥を探し出す手法を提案した [5]．しかし，要求が不完全である原因は要求仕様書のみに限らないため，より多くの工程からの考察が必要である．そこで本研究では，要求仕様書を作成する過程を含んだ，動的な開発プロセスに着目した．

既存の静的な欠陥による分類法と，動的な開発プロセスに着目することで，要求が不完全であることに起因するソフトウェアにおける欠陥の分類法を提示する．欠陥の分類法として次節で述べる Avizienis らの提案した分類を [2]，開発プロセスとしてソフトウェア開発標準を用いた [6]．

3. 提案する分類の枠組み

本研究ではソフトウェアにおける不具合の分類のために，以下の 2 つの指標を用いる．

- 不具合の原因となっている欠陥に基づく分類
- 開発プロセス

観点の異なる 2 つの指標を組み合わせることで，従来研究では発見されなかった不具合の傾向の発見や，不具合を生じさせないための対策について講じることが期待できる．本章では，それぞれの指標について記述する．

3.1 欠陥の分類

まず，不具合および欠陥について説明する [2]．図 1 に示すように，不具合とは本来提供されるべき正しいサービスが何らかの影響で正しく提供されないことを示す．正しくないサービス状態のことを Failure（障害），障害につながりうるシステムの内部状態を Error（誤り）という．誤りとは，内部状態であるため，開発者は直接体感することはできない．また，誤りが生じるきっかけや原因となるものが Fault（欠陥）である．本研究では障害の原因となる欠陥について着目した．

Avizienis らは一般的な不具合全体について，図 2 に示すように誤りの原因となる欠陥に着目している [2]．彼らは，独立した 2 つの値を持つ 8 つの観点，すなわち $2^8 = 256$ 通りの観点を提案し，そのうち言葉の定義上存在しないものを除いた，計 31 通りに欠陥を分類している．8 つの観点と独立した 2 つの値とは，以下に示すものである．

- Phase of creation or occurrence (Development faults, Operational faults)
- System boundaries (Internal faults, External faults)
- Phenomenological cause (Natural faults, Human-Made faults)
- Dimension (Hardware faults, Software faults)
- Objective (Non-Malicious faults, Malicious faults)
- Intent (Non-Deliberate faults, Deliberate faults)
- Capability (Accidental faults, Incompetence faults)
- Persistence (Permanent faults, Transient faults)

言葉の定義上存在しないものとは，Malicious faults であり Non-Deliberate faults であるもの等を示す．彼らの研究では原因となる欠陥全体の分類はできた一方で，高信頼性システムに致命的な影響を与える “要求の不完全さ” については考慮されていない．本研究では，Avizienis らの分類のうち，ソフトウェア開発時に生じるものに着目した．すなわち，Development faults と Software faults である．また，対象としている不具合では悪意を持って欠陥を入れることはないと判断し，Malicious faults は存在しないものとした．よって対象とする観点は Intent と Capability の有無であり，表 1 に示す 4 通りとなっている．

3.2 開発プロセス

欠陥を分類するための指標として，宇宙航空研究開発機構（Japan Aerospace eXploration Agency. 以下 JAXA と記述す

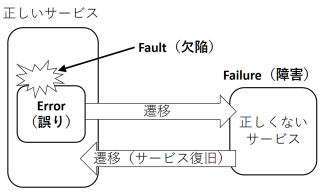

図 1　不具合に関する欠陥，誤り，障害の関係

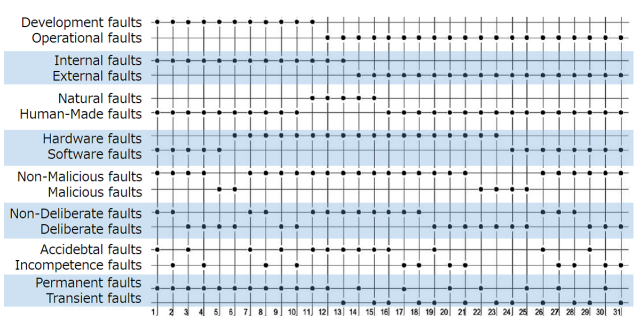

図 2　Avizienis らの提案する欠陥の分類

表 1　対象とする欠陥

分類 1	Deliberate fault であり Accidental faults
分類 2	Deliberate fault であり Incompetence faults
分類 3	Non-Deliberate fault であり Accidental faults
分類 4	Non-Deliberate fault であり Incompetence fault

る）が提供しているソフトウェア開発標準 [6] を使用した．これは ISO/IEC 12207 [7] や JIS X0160-1996 [8] などを元に作られたものである．ソフトウェア開発標準は大きく 2 つのプロセスから成り立っている．1 つ目が主ライフサイクルプロセスであり，2 つ目が支援ライフサイクルプロセスである．主ライフサイクルプロセスは，ソフトウェア開発に直接かかわってくるソフトウェアライフサイクルのプロセスであり，大きく 3 つのプロセス（開発プロセス，運用プロセス，保守プロセス）に分類される．支援ライフサイクルは，主ライフサイクルプロセスを支える，他のプロセスから呼びだされるプロセスである．本研究では，開発が行われているソフトウェアを対象とするため，主ライフサイクルプロセスの中の開発プロセスに着目した．また，本研究が対象とする欠陥は要求漏れに基づく欠陥であるため，開発プロセスの中でもソフトウェア要求分析より上位のプロセス（プロセスの開発準備，コンピュータシステム要求分析，コンピュータシステム方式設計，ソフトウェア要求分析など）も分析対象とした．それぞれの詳細な内容を表 2 に示す．以下グラフ中では対応 ID を，本文中では略称を使用する．

4.　不具合の分類

本章では本研究で使用した JAXA から提供された実際の不具合データについて説明した後に，先述した 2 つの指標をどのように使用したのかについても述べる．

4.1　調 査 対 象

本調査では，JAXA にて実際に発生した単純なソフトウェアの実装における不具合でない 48 件の不具合情報のレポートを対象とした．このレポートにはどのプロジェクトで起きたものなのか，どのような不具合であったか，その原因はなにか等がまとめられている．また，この 48 件のデータは 1 つにつき 1 つの障害を示しているが，障害 1 つにつき欠陥が 1 つであるとは限らない．

4.2　分 類 手 順

本調査では不具合データを欠陥の種類により分類を行った後に開発プロセスによる分類を行った．詳細な手順について，以下に示す．

手順 1　不具合（障害）1 つ 1 つに対して，その原因となっている欠陥を明確にする

手順 2　明確になった欠陥に対して，Avizienis らが提案している欠陥分類のどれに該当するのかを調べ，ラベリングを行う

手順 3　ラベリングされた欠陥に対して，それがどの開発プロセスで発見されるべきであったかを確認し，マッピングを行う

手順 4　マッピングされたデータを集計し，分析を行う

なお，これらはすべて第一著者が手作業で行ったものである．

4.3　分 析 結 果

不具合データの分析の結果，4.1 節で述べた 48 件の障害から計 110 件の欠陥を明確にした．複数の欠陥により障害が引き起こされたものもあるため，欠陥の数は障害の数より多い値となっている．また，識別した欠陥を 3.1 節で述べた分類 1〜4 のどれに当てはまるかを第一著者が確認し，ラベリングした結果を表 3 に示す．この表では，不具合事例に対して分類 1〜4 に該当するものがいくつあったのかを示している．このデータを集計したところ，3.1 節で述べた表 1 に示す分類 1 は 31 件，分類 2 は 22 件，分類 3 は 13 件，分類 4 は 44 件であった．合計 110 件の欠陥それぞれがどのプロセスで発見すべきであったのかを確認しマッピングを行った．マッピングの結果，1 つの欠陥につき，複数のプロセスで見つけるべきだったものも存在していることが分かった．さらに，プロセスを基に分類した結果を分類 1〜4 に対してそれぞれ集計し，グラフ化を行うことでそれぞれの特徴を発見した．結果について 5 章で述べる．

5.　分類結果と考察

本章では，4 章で分析した結果および，欠陥の総数や 3.1 節で示した Intent や Capability の違い，欠陥の起点となるプロセスについて考察を行う．

5.1　各プロセスにおいて発見されるべき欠陥数の比較

各プロセスに対して，発見されるべき欠陥数をカウントしたものを図 3 に示す．グラフの横軸は表 2 に記載した，JAXA で使用されているソフトウェア開発標準のプロセスに対応する番号を，縦軸は不具合の件数を示し，分類ごとに色分けを行っている．

図 3 より，ソフトウェアよりも上位のシステムからの要求にソフトウェアのプロセスが大きく依存し，より上位の曖昧さに起因して誤った要求分析が行われた結果，多くの欠陥が検出されたと読み取ることができる．その根拠として，図 3 では，“前提条件の明確化” や “SW 要求仕様書” の項目の値が高くなっている．これらの項目の共通点として，これらのプロセスではソフトウェア以外のシステムが大きく関わってくることがあげられる．また，類似している状況の項目として，“HW の評価” の欠陥も多い値となっている．また，欠陥をカバーするテストの項目である “試験についての評価” でも欠陥が多くなっており，上位システムの要求から受ける影響によりテストが正しく評価されていないと考えられる．

5.2　Intent に関する比較

Intent に着目した場合，欠陥に関連するプロセスにどのよう

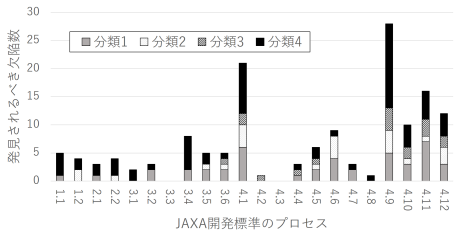

図 3　各プロセスにおいて発見されるべき欠陥数

表 2　開発プロセスの内容

プロセス	対応するプロセス ID	詳細な内容	略称
プロセスの開発準備	1.1	ソフトウェア開発計画を立案すること	開発計画立案
	1.2	開発計画の文章化	開発計画文章化
コンピュータシステム要求分析	2.1	要求を抽出する	要求抽出
	2.2	要求仕様書の作成	要求仕様書作成
コンピュータシステム方式設計	3.1	構成する品目，種別を明確にする	構成物の明確化
	3.2	各構成品目に要求を割り当てる	要求の割り当て
	3.3	実現可能性を評価	CS の実現可能性評価
	3.4	設計根拠と前提条件を明らかにし，評価	前提条件の評価
	3.5	上位とのトレーサビリティを評価	トレーサビリティの評価
	3.6	インターフェース要求を抽出	IF 要求の抽出
ソフトウェア要求分析	4.1	ソフトウェア要求仕様書の作成	SW 要求仕様書の作成
	4.2	個別に識別子を付与	識別子付与
	4.3	データおよびデータベースに対する仕様を含める	DB 仕様を含める
	4.4	異常検知および処理に関する仕様を含める	異常処理の仕様を含める
	4.5	インターフェース仕様に関して合意を得ること	IF の確認
	4.6	上位との整合性を得る	整合性の確認
	4.7	実現可能性の評価	RA の実現可能性評価
	4.8	元からあるものを使うときは整合性などを確認	整合性の確認
	4.9	前提条件，制約を明確にする	前提条件の明確化
	4.10	検証可能性を評価	検証可能性評価
	4.11	試験計画可能性を評価	試験についての評価
	4.12	環境やハードウェアの影響がある場合確認の評価	HW の評価

表 3　障害に対する欠陥の分類

不具合事例	欠陥の分類				不具合事例	欠陥の分類				不具合事例	欠陥の分類			
	分類 1	分類 2	分類 3	分類 4		分類 1	分類 2	分類 3	分類 4		分類 1	分類 2	分類 3	分類 4
1		1		1	17		1		1	33	1	1		2
2	1	1			18				1	34	1			2
3	1				19			1	1	35	1	1		1
4				1	20	1			1	36				1
5	1		1	1	21	1	1	1		37		1		2
6	1			2	22		1			38		2	1	
7	1			1	23		1			39			1	2
8	1			1	24		1			40	1			
9	1			1	25	1				41		1		
10	1			1	26	1	1		1	42		1		2
11	1			1	27	2	1	2	1	43		1	1	
12				1	28	1		1	3	44		1	1	
13				1	29	2	1		3	45				1
14	1				30	2	2	1		46	1			
15	1			2	31		1			47			1	1
16	3		1	2	32	1			1	48				1

な差が生じるのかを図 4 に示す．グラフの横軸は表 2 に記載している，JAXA で使用されているソフトウェア開発標準のプロセスに対応する番号を，縦軸は欠陥の総数に大きな差があるため正規化をした数値を示す．Non-Deliberate fault（灰色）は欠陥の分類 1 と分類 2 を足したものとなっており，Deliberate fault（黒色）が欠陥の分類の 3 と分類 4 を足したものとなっている．

図 4 で着目すべき項目は Non-Deliberate fault と Deliberate fault の差が大きい “整合性の確認” である．この項目は上位との整合性を確認するプロセスである．灰色の値が多いことから，開発者が熟考したにも関わらず，なお間違いが起きていることが読み取れる．この理由として，確認すべき上位の “SW 要求仕様書の作成” の成果物の影響であることが考えられる．“SW 要求仕様書の作成” において，不具合が多いことは 5.1 節で示した通りである．すなわち，完全でなかった成果物に対して整合性を確認したことになり，結果として “整合性の確認” の Non-Deliberate fault の項目が Deliberate fault に比べ高い割合であったと考えられる．

5.3 Capability に関する比較

Capability に着目した場合，欠陥に関連するプロセスにどのような差が生じるのかについて図 5 に示す．グラフの横軸は表 2 に記載している，JAXA で使用されているソフトウェア開発標準のプロセスに対応する番号を，縦軸は欠陥の総数に大きな差があるため正規化をした数値を示す．Accidental faults（灰色）は欠陥の分類 1 と分類 3 を足したものとなっており，Incompetence faults（黒色）が欠陥の分類の 2 と分類 4 を足したものとなっている．

図 5 で着目すべき項目は Accidental faults と Incompetence faults の差が大きい “SW 要求仕様書の作成” である．“SW 要求仕様書の作成” の Accidental faults の値が Incompetence faults の値に比べて大きい理由として，5.1 節と同様により上位のシステムにおいての不確定要素の影響があり，想定外の出来事が起きたのではないかと推測することができる．なぜなら，要求仕様書を作成するにあたり，より上位のシステムで作成された成果物の内容を反映しないといけないからである．

一方で，より上位のプロセスである，“開発計画立案” から “IF 要求の抽出” において Accidental faults に比べて Incompetence faults の値が目立つ結果となっている．よって，これら開発の初期段階であるプロセスでは 1 つの小さなミスが，後のプロセスにおいて大きく表れる．したがってこのプロセスにおいては実行する技術が求められることが考えられる．これは図 5 からも読み取ることができる．例えば，同じ前提条件や制約に関する “前提条件の評価” と “前提条件の明確化”，要求仕様書に関する，“要求仕様書作成” と “SW 要求仕様書の作成” など，プロセスの下位に行くにつれ欠陥の数が多くなっている．

5.4 各プロセスの関係性について

表 4 は，各プロセスがどのプロセスに影響を与え，どのプロセスから，影響を受けているのかを示したものである．行も列もプロセスを表しており，表 2 を基に対応する番号を表記した．この表の数値は，4 章で得たデータを次節で述べる手順に従い詳細に分析したものである．

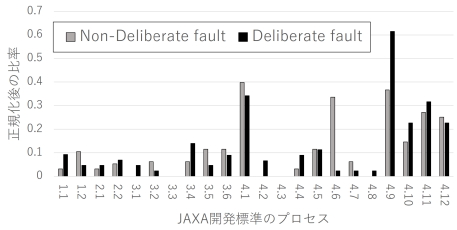

図 4　Deliberate fault と Non-Deliberate fault の比較

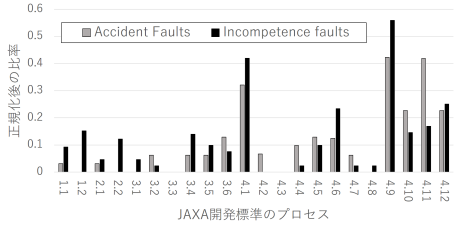

図 5　Accidental faults と Incompetence faults の比較

5.4.1 表の作成方法

表 4 で示した各プロセスへの影響度の求め方について述べる．例として濃い灰色のセル（横軸：プロセス 3.4，縦軸：プロセス 4.4）について取り上げる．まず初めに，4.2 節で示した手順 3 で作成されたマッピング結果から “異常処理の仕様を含める” で発見されるべき欠陥を探した．その結果，発見されるべき欠陥は合計で 3 件見つかった．該当プロセスで発見されるべき欠陥の総数は，この値が表の一番下の行に記述している．また 4.3 節で示したように，1 つの欠陥が複数のプロセスで見つかる可能性があったために，“異常処理の仕様を含める” で発見すべき欠陥のうち他のプロセスであるならどこで発見すべきだったのかをプロセスごとに調べカウントした．その値を “異常処理の仕様を含める” プロセスで発見されるべき欠陥の総数である 3 で割った値を対応するセルに記述する．この例では，“異常処理の仕様を含める” で見つけるべき欠陥の数は 1 つであったため，$1 \div 3 \simeq 0.3$ となっている．これを全プロセスに対して行ったものが，表 4 である．対象となる不具合が存在しかなったセルについては “-” と記述している．表 4 より，各プロセスの依存関係，すなわちどのプロセスがどのプロセスに対して影響を与えているか，あるいは影響を受けているのか分析を行うことが可能となる．

表 4 では，行も列も同じプロセスの場合は黒塗りに，計算した値が 0.5 以上の場合は背景色を薄い灰色に変更している．すなわちプロセス間の影響度が高いものが薄い灰色となっている．また，黒いセルより上側にあるものはそのプロセスより上位のプロセスに該当し，下側にあるものはそのプロセスより下位のプロセスに該当する．今回の例では，“異常処理の仕様を含める” は “要求の割り当て” や “SW 要求仕様書の作成” からの影響を多く受けており，逆に大きく影響を与えているものはなかったことが分かる．

5.4.2 欠陥の起点となるプロセス

表 4 より明らかになった点を述べる．まず 1 つ目に，表 2 で示した工程の中でも，より上位の工程において “開発計画立案” の影響が大きい点である．5.4.1 節で示したように，“要求抽出” は “開発計画立案” の影響を非常に多く受けており，“要求仕様書作成” は “開発計画立案” や “要求抽出” の影響を受けている．また，“構成物の明確化” は “開発計画立案”，“要求抽出”，“要求仕様書作成” の影響が大きい．以上より，“開発計画立案” をより完璧に仕上げることで，“要求抽出”，“要求仕様書作成”，“構成物の明確化” の不具合解消に大きな影響があることが分かる．上記で示したプロセスは，上流工程の中でも極めて上位に位置するプロセスであり，システム全体に関わる枠組みを決めているため，互いに影響を及ぼしているのだと考えられる．

2 つ目に，表 2 で示した工程の中でも下位にある “前提条件の明確化” が大きな影響を与えている点である．このプロセスは欠陥が最も多かったプロセスである．しかし，特定のプロセスからの影響度が高いわけではないが，“要求抽出”，“要求仕様書作成”，“構成物の明確化”，“識別子付与” 等，多くのプロセスから影響を受けている．これは，前提条件や制約などに関す

表 4　各プロセスへの影響度

プロセス ID	1.1	1.2	2.1	2.2	3.1	3.2	3.3	3.4	3.5	3.6	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10	4.11	4.12
1.1		0.3	1	0.5	1	-		0.1	-	-	0	-		-	-	-	-	-	0.1	0.1	-	-
1.2	0.2		-	-	-	-		-	-	-	-	-		-	-	-	-	-	-	-	-	-
2.1	0.6	-		0.5	1	-		0.1	-	-	0	-		-	-	-	-	-	0.1	0.1	-	-
2.2	0.4	-	0.7		1	-		0.1	0.2	0.2	0.1	-		-	0.2	-	-	-	0.1	0.1	-	-
3.1	0.4	-	0.7	0.5		-		0.1	-	-	-	-		-	-	-	-	-	0.1	0.1	-	-
3.2	-	-	-	-	-			0.1	-	0.2	0.1	-		0.7	0.2	-	-	-	0	-	-	-
3.3	-	-	-	-	-	-		-	-	-	-	-		-	-	-	-	-	-	-	-	-
3.4	0.2	-	0.3	0.3	0.5	0.3			-	-	0	-		0.3	-	-	-	-	0.3	0.1	-	0.1
3.5	-	-	-	0.3	-	-		-		0.2	0.1	-		-	0.2	0.4	-	-	0	-	-	-
3.6	-	-	-	0.3	-	0.3		-	0.2		0.1	-		-	0.8	-	-	-	-	-	-	0.1
4.1	0.2	-	0.3	0.5	-	0.7		0.1	0.4	0.6		-		0.7	0.5	0.1	-	-	0.2	-	-	0.1
4.2	-	-	-	-	-	-		-	-	-	-			-	-	-	-	-	0	-	-	-
4.3	-	-	-	-	-	-		-	-	-	-	-		-	-	-	-	-	-	-	-	-
4.4	-	-	-	-	-	0.7		0.1	-	-	0.1	-			-	-	-	-	0	-	0.1	-
4.5	-	-	-	0.3	-	0.3		-	0.2	1	0.1	-		-		-	-	1	-	-	-	0.1
4.6	-	-	-	-	-	-		-	0.8	-	0	-		-	-		-	-	0.1	0.1	0.1	-
4.7	-	-	-	-	-	-		-	-	-	-	-		-	-	-		-	-	-	-	0.1
4.8	-	-	-	-	-	-		-	-	-	-	-		-	0.2	-	-		-	-	-	-
4.9	0.4	-	0.7	0.5	1	0.3		1	0.2	-	0.2	1		0.3	-	0.2	-	-		0.1	-	0.3
4.1	0.2	-	0.3	0.3	0.5	-		0.1	-	-	-	-		-	-	0.1	-	-	0		0.6	0.3
4.11	-	-	-	-	-	-		-	-	-	-	-		0.3	-	0.1	-	-	-	0.9		0.4
4.12	-	-	-	-	-	-		0.1	-	0.2	0	-		-	0.2	-	0.3	-	0.1	0.3	0.3	
不具合データの総数	5	4	3	4	2	3	0	8	5	5	21	1	0	3	6	9	3	1	28	10	16	12

るプロセスが “前提条件の明確化” より上のプロセスで明確になっておらず，“前提条件の明確化” より上位のプロセスから満遍なく影響を受けているためだと考えられる．

本分析より他のプロセスに影響を与える欠陥の起点となっているプロセスが導かれる．欠陥の起点となるプロセスとは，特定の個所から影響を受けていないが他に影響を与える項目である．候補としては，“開発計画立案”，“開発計画文章化”，“要求の割り当て”，“CS の実現可能性の評価”，“トレーサビリティの評価”，“IF 要求の抽出”，“識別子付与”，“DB 仕様書を含める” があげられる．一方で，“CS の実現可能性の評価”，“識別子付与”，“DB 仕様書を含める” は今回のデータに欠陥がないまたは著しく少ないため除外し，“開発計画文章化” は他に影響を与えないため除外した．よって今回の不具合データからは，“開発計画立案”,，“要求の割り当て”，“トレーサビリティの評価”，“IF 要求の抽出” が欠陥の起点となり，このプロセスを強化することが望まれる．

6. まとめと今後の課題

本調査で用いた分類は，静的な欠陥による分類法と，動的な開発プロセスを合わせたものであり，要求が不完全であることに起因するソフトウェア問題の分類法を提示している．不具合データから本分類を行うことで，不具合の傾向を知ることができ，どのプロセスに力を入れるべきか指標として活用できる．さらに，適用した組織におけるプロセス間の影響度も知ることができ，効率よく対策を講じることができると期待される．

今回，JAXA の約 50 件の不具合データを元に欠陥の傾向やプロセス間の影響について分析を行ったが，他の組織でも使用しているプロセスを当てはめることで，同様の分析を行うことが可能であり，欠陥の傾向やプロセス間の影響について議論ができると考えている．

今後の課題として，不具合の傾向やプロセス間の影響に対する対策や手法の提示・検討があげられる．

文　献

[1] A. Ghazarian, “A case study of source code evolution,” In Proc. 2009 13th European Conference on Software Maintenance and Reengineering, pp.159–168, March 2009.

[2] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” IEEE Transactions on Dependable and Secure Computing, vol.1, no.1, pp.11–33, 2004.

[3] 宇宙航空研究開発機構，“ソフトウェア開発標準,” https://www.ipa.go.jp/files/000004749.pdf.

[4] 宇宙航空研究開発機構，“宇宙機搭載ソフトウェア開発のアセスメント,” http://www.jaspic.org/event/20 09/SPI-Japan/session1A/1A3.pdf.

[5] A.A. Alshazly, A.M. Elfatatry, and M.S. Abougabal, “Detecting defects in software requirements specification,” Alexandria Engineering Journal, vol.53, no.3, pp.513–527, 2014.

[6] 宇宙航空研究開発機構，“ソフトウェア開発標準,” http://sma.jaxa.jp/TechDoc/Docs/JAXA-JERG-0-04 9.pdf.

[7] I.A. toISO/IEC 12207-2008 Systems and S. Engineering, “Software life cycle processes. iso, geneva, switzerland,” 1995.

[8] 日本工業標準調査会，“ソフトウェアライフサイクルプロセス,” 1996.