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　宇宙機のソフトウェアにおいて、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っている要求漏れに着目した。実際に宇宙機（人工衛星やロケット）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ている。この分類を行うことで、欠陥の傾向や，プロセス間の影響が明確になる。
キーワード　アスキー版 pLaTeX 2_ε，タイピングの注意事項

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 can largely obstruct the chance of missions success. In this research, we focus on the omitted requirement which is considered to be a major cause of software faults. From the two perspectives of fault classification and standard development process, we classified software faults that we retrieved from actual spacecraft project(artificial satellite and rockets). By carrying out this classification, it becomes clear which kind of process leads to more faults.
Key words　pLaTeX 2_ε class file, typesetting

1. はじめに

ソフトウェア開発の際に，ソフトウェアが原因である不具合が発生することがある．その原因は大きくわけて 2 つある．1 つ目は “ソフトウェアを作る際の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” のパターンであり，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よる誤りはツールなどを用いることで容易に発見できる．一方で，後者に関しては，ツールが存在しなく，仕様書等にも記載されていないため発見が困難である．また，実際に起きている不具合の割合としてはソフトウェアのプログラム実装時の誤りによる不具合の割合は非常に少ないことが既存研究で述べられている [1]．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もカバーする必要がある．

不具合の要因の 1 つである欠陥に関する分類法の研究として，Avizienis らが提案した Basic concepts and taxonomy of dependable and secure computing があげられる [2]．しかし，要求が不完全である事象に対しては欠陥の分類など静的な分類だけでは困難である．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，実際に宇宙機と呼ばれる，人工衛星やロケットなどで発生した事例をもとに，ソフトウェアの不具合を洗い出し，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ると考えられる．

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，一度飛ばし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重要な原因となることがある．よって，宇宙機のソフトウェア開発においては，いかに失敗しないものであるか，すなわち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となっている．しかし，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．

要求漏れに関する研究として，要求仕様書に着目した Alshazly らによる研究が存在する [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，より広い観点からの考察が必要があると考える．よって本研究では，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．

静的な欠陥による分類法と，動的な開発プロセスを用いることで，要求が不完全であることに起因するソフトウェア問題の分類法を提示する．Avizienis らが提案した分類法を用いて欠陥を分類を行った [2]．プロセスとして宇宙航空研究開発機構（Japan Aerospace eXploration Agency. 以下 JAXA と記述する）で使用されている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

先述したように，本研究ではソフトウェアの不具合の分類のために，2 つの指標を用いる．1 つ目は，不具合の原因となっている欠陥に基づく分類である．2 つ目は，開発プロセスである．この異なる軸の 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を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．その正しくないサービス状態のことを Failure（障害）という．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．誤りは，内部状態であるため，我々は直接体感することはできない．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．私たちが実際に “不具合だ” と感じるのは障害であり，本研究では障害の原因となっている欠陥について着目した．

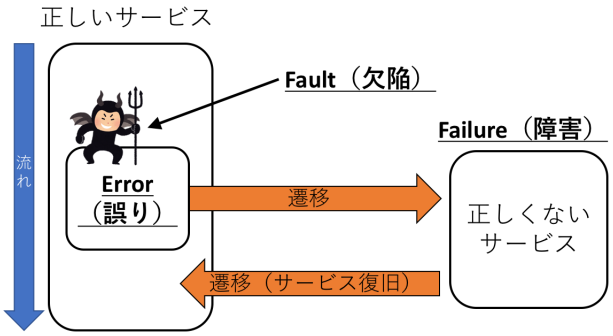

図 1　不具合に関する欠陥，誤り，障害の関係

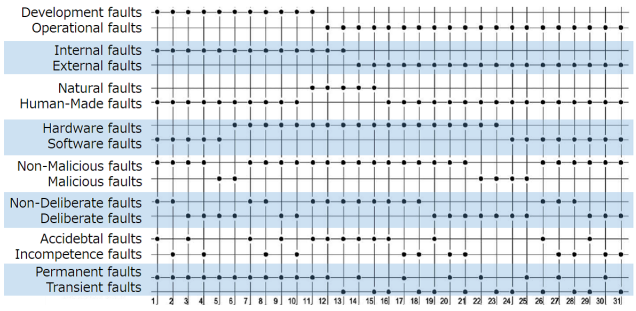

図 2　欠陥の分類結果

既存研究において，Avizienis らは，一般的な不具合について分類を行った [2]．特に，図 2 に示すように不具合全体の誤りの原因となる欠陥について着目している．彼らは，独立した 2 値の値を持つ 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 通りの観点から欠陥を網羅的に分類している．8 つの観点とは，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(Accidebtal 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．本論文では，分類された 31 個の欠陥を図 2 の左から順に分類 1，分類 2 と呼ぶことにする．

Avizienis 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いない．

本研究では，Avizienis らの分類からソフトウェア開発時に

プロセス	詳細な内容
主ライフサイクルプロセス	開発プロセス
	運用プロセス
	保守プロセス
支援ライフサイクルプロセス	文書化プロセス
	構成管理プロセス
	品質保証プロセス
	検証プロセス
	妥当性確認プロセス
	共同レビュープロセス
	アセスメントプロセス
	問題解決プロセス

表 1　主ライフサイクルプロセスと支援ライフサイクルプロセス

生じるのに着目した．すなわち，Development faults であり Software faults である，図 2 に示す分類 1〜分類 5 までの 5 通りとなった．さらに，対象としている不具合では，悪意をも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 の有無，Capability の有無であり，4 通りとなっている．Dellberate fault であり Accidental faults を分類 1，Dellberate fault であり Incompetence faults を分類 2，Non-Dellberate fault であり Accidental faults を分類 3，Non-Dellberate fault であり Incompetence faults を分類 4 とした

3.2　開発プロセス

欠陥を分類するために 1 つの指標として，JAXA が提供しているソフトウェア開発標準を使用した．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表 1 に示すように，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，もう 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）に分類されている．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り，大きく 8 つ（文書化プロセス，構成管理プロセス，品質保証プロセス，検証プロセス，妥当性確認プロセス，共同レビュープロセス，アセスメントプロセス，問題解決プロセス）に分類されている．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のあやまりに基づく欠陥であるため，開発プロセスのなか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）も分析対象とした．それぞれの詳細な内容を表 2 に示す．

4.　データの適用及び結果

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する．また，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　使用したデータ

本調査では，JAXA にて実際に発生した単純なプログラムミスなどでない 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 方 法

本実験では不具合データを欠陥の種類により分類を行い，次に，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．これはすべて私が手作業で行ったものである．

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が何であるかを明確にした

手順 2　明確になった欠陥に対して，欠陥分類のどれに該当するのかをしらべ，ラベリングを行った

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

手順 4　マッピングされたデータを集計，分析を行った

4.3　適 用 結 果

4.3.1　欠陥の抽出および欠陥分類のマッピング

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 件の欠陥を抽出した．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 で述べた分類 1〜4 のどれに当てはまるかを私が確認し，分類した結果を表 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，Dellberate fault であり Accidental faultsa である分類 1 は 31 件，Dellberate fault であり Incompetence faults である分類 2 は 22 件，Non-Dellberate fault であり Accidental faults である分類 3 は 13 件，Non-Dellberate fault であり Incompetence faults である分類 4 は 44 件であった．

4.3.2　集　　計

上記の結果より欠陥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．その結果について 5 章で述べる．

5.　結果の考察

本章では，分析した結果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

5.1.1　結　　果

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 し示す．グラフの横軸は表 2 に記載している JAXA が使用し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す．青色が欠陥の分類 1，オレンジ色が欠陥の分類 2，灰色が欠陥の分類 3，黄色が欠陥の分

フェーズ	対応番号	詳細な内容
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
	1.2	開発計画の文章化
コンピュータシステム要求分析	2.1	要求を抽出する
	2.2	要求仕様書の作成
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
	3.2	各構成品目に要求を割り当てる
	3.3	実現可能性を評価
	3.4	設計根拠と前提条件を明らかにし，評価
	3.5	上位とのトレーサビリティを評価
	3.6	インターフェース要求を抽出
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
	4.2	個別に識別子を付与
	4.3	データ及びデータベースに対する仕様を含める
	4.4	異常検知及び処理に関する仕様を含める
	4.5	インターフェース仕様に関して合意を得ること
	4.6	上位との整合性を得る
	4.7	実現可能性の評価
	4.8	元からあるものを使うときは整合性などを確認
	4.9	前提条件，制約を明確に
	4.10	検証可能性を評価
	4.11	試験計画可能性を評価
	4.12	環境やハードウェアの影響がある場合確認の評価

表 2　開発プロセスの内容

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

表 3　障害に対する欠陥の分類

類 4 である．

5.1.2 考　　察

このグラフから，ソフトウェアよりも上位のシステムからの要求に大きく依存し，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考えらえる．図 3 では，プロセス 4.9 やプロセス 4.1 の項目の値が高い．この個所で共通することとして，このプロセスではソフトウェア以外の部分が大きくかかわってくることである．また，類似している項目として，プロセス 4.12 のハードウェアや環境を評価する項目の欠陥も多い値となっている．さらに，ここで欠陥ををカバーするテストの項目である 4.11 でも欠陥が多くなっており，多くの不具合が検出されていると考えられる．

5.2 Intent に関する比較

5.2.1 結　　果

Intent に関する差を比較した結果を図 4 に示す．すなわち Intent に着目した場合に欠陥に関連するプロセスにどのような差が生じるのかを示している．グラフの横軸は表 2 に記載している JAXA が使用しているソフトウェア開発標準のプロセス

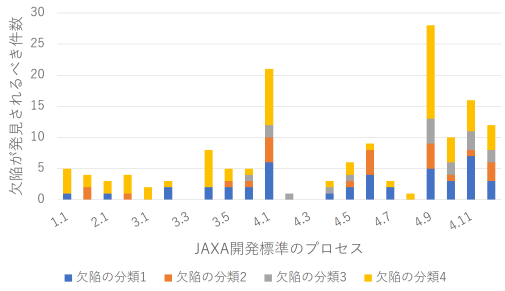

図 3　各プロセスにおいて発見されるべき欠陥数

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-Dellberate fault（青色）は欠陥の分類 1 と分類 2 を足したものとなっており，Dellberate fault（オレンジ色）が欠陥の分類の 3 と分類 4 を足したものとなっている．

5.2.2　考　　察

図 4 で着目すべき項目は Non-Dellberate fault と Dellberate fault の差が大きいプロセスである．すなわちプロセス 4.6 である．この項目は上位との整合性を確認するフェーズである．青色の値が多いことから，開発者が塾考したにも関わらず，なお間違いが起きていることがわかる．この理由として，確認すべき上位の成果物がなにであるかを考えた時に，プロセス 4.1 の成果物が候補としてあがる．このフェーズにおいて，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チェックしたことになり，結果としてこのフェーズの Non-Dellberate fault の項目が Dellberate fault に比べ高い割合であったと考えられる．

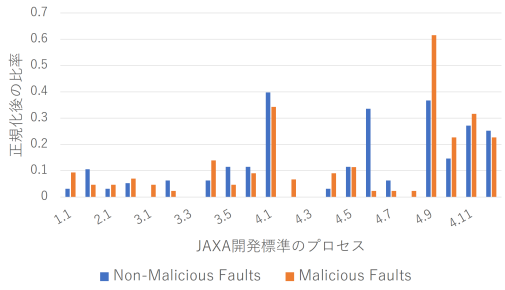

図 4　Dellberate fault と Non-Dellberate fault の比較

5.3　Capability に関する比較

5.3.1　結　　果

Capability に関する差を比較した結果を図 5 に示す．すなわち Capability に着目した場合に欠陥に関連するプロセスにどのような差が生じるのかを示している．グラフの横軸は表 2 に記載している JAXA が使用し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（青色）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（オレンジ色）が欠陥の分類の 2 と分類 4 を足したものとなっている．

5.3.2　考　　察

図 5 で着目すべき項目は Accidental faults と Incompetence faults の差が大きいプロセスである．すなわちプロセス 4.11 である．プロセス 4.11 の Accidental faults の値が Incompetence faults の値にくらべて大きい理由として，5.1 節と同様に不確定要素の影響があり，思いもしなかったことが起きたのではないかと考えられる．

一方で，より上位のプロセスである，プロセス 1.1〜プロセス 3.6 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，このプロセスにおいてはプロセスに対して技術が求められることがわかる．さらに，この上位にあるプロセスでのミスが後々響いてくることが図 5 からわかる．例えば，同じ前提条件や制約に関するプロセス 3.4 とプロセス 4.9，要求仕様書に関する，プロセス 2.2 とプロセス 4.1 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．

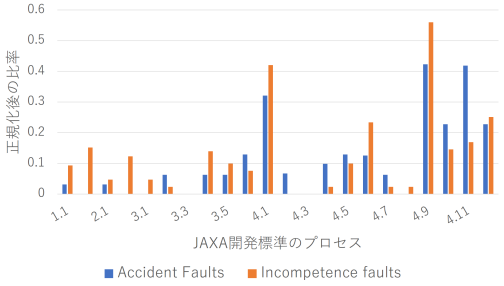

図 5　Accidental faults と Incompetence faults の比較

5.4　各フェーズの関係性について

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．

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．例として黄色のセル（横軸：プロセス 4.4，縦軸：プロセス 3.4）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プロセス 4.4 で発見されるべきものを探した．今回，プロセス 4.4 で発見されるべきものは合計で 3 件見つかった．該当プロセスで発見されるべき欠陥の個数は，この値が表の一番下の行に記述している．また 4.3.2 で示したように，1 つの欠陥が複数のプロセスで見つけることが可能だったため，プロセス 4.4 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．その値をプロセス 4.4 で発見されるべきものの総数である 3 で割り算した値を対応するセルに記述する．この例では，プロセス 4.4 とプロセス 3.4 の両方において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 “-” で記述

プロセス	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	
不具合データの個数	5	4	3	4	2	3	0	8	5	5	21	1	0	3	6	9	3	1	28	10	16	12

表 4　各プロセスへの影響度

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，この図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赤色に変更している．すなわちプロセス間の影響度が高いものが赤色となっている．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するし，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．今回の例であると，プロセス 4.4 はプロセス 3.2 やプロセス 4.1 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になる．

5.4.2　ピポットとなるプロセス

表 4 より次の点が明確になった．まず 1 つ目として，表 2 で示した工程の中でも，より上位の工程において，プロセス 1.1 の影響が大きいことである．5.4.1 で示したように，プロセス 2.1 はプロセス 1.1 の影響を非常に多く受けており，プロセス 2.2 はプロセス 1.1 やプロセス 2.1 の影響を受けている．また，プロセス 3.1 はプロセス 1.1, プロセス 2.1, プロセス 2.2 の影響が大きい．ここから言えることは，プロセス 1.1 をより完璧に仕上げることで，プロセス 2.1, プロセス 2.2, プロセス 3.1 の不具合解消に大きな影響があることである．このあたりのプロセスは上流工程の中でもかなり上のプロセスであり，大きな枠組みをきめているため，互いに影響しあうのは理解できる．

2 つ目として，表 2 で示した工程の中でも下位にあるプロセス 4.9 が大きな影響を与えていることである．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，プロセス 2.1, プロセス 2.2, プロセス 3.1, プロセス 4.2 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るプロセスがプロセス 4.9 より上のプロセスで明確になっておらず，プロセス 4.9 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．

今回の結果より他のプロセスに影響を与えるピポットとなっているプロセスが導かれる．ピポット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，プロセス 1.1，プロセス 1.2，プロセス 3.2，プロセス 3.3，プロセス 3.5，プロセス 3.6，プロセス 4.2，プロセス 4.3 である．一方で，プロセス 3.3，4.2，4.3 は今回のデータに欠陥がないまたは著しく少ないので除外した．さらにプロセス 1.2 は他に影響を与えないため，今回の不具合データから，プロセス 1.1,，プロセス 3.2，プロセス 3.5，プロセス 3.6 がピポットとなり，このプロセスを強化することがよいと考えられる．

6.　まとめと今後の課題

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．不具合データからこの分類を行うことで，不具合の傾向を知ることができ，プロセスとしてどこに力を入れるべきであるかが見えてくる．一

方で，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．

本研究では，不具合の傾向やプロセス間の影響を知ることはできたが，今後の課題としてそれを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,” 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, Jan. 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.