NAIST-IS-MT1651105

**修士論文**

# 要求に起因する欠陥の分類と ソフトウェア開発プロセス定義の組み合わせに基づく ソフトウェア開発組織における改善手法の提案

宮村 純真

2018 年 2 月 1 日

奈良先端科学技術大学院大学
情報科学研究科

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

宮村 純真

審査委員：

| | |
|---|---|
| 飯田 元 教授 | (主指導教員) |
| 松本 健一 教授 | (副指導教員) |
| 片平 真史 教授 | (副指導教員) |
| 石濱 直樹 准教授 | (副指導教員) |
| 川口 真司 准教授 | (副指導教員) |

# 要求に起因する欠陥の分類と ソフトウェア開発プロセス定義の組み合わせに基づく ソフトウェア開発組織における改善手法の提案*

宮村 純真

**内容梗概**

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，欠陥の分類とソフトウェア開発プロセスの２つの観点から不具合の分類を行った．欠陥の分類には Avizienis らが提案している 8 つの観点による分類を用いた．ソフトウェア開発プロセスとして 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り，今後の対策を議論するにあたり指標として活用することが期待される．今回の分析法を JAXA のプロジェクトで適用したところ，欠陥の起点となるプロセスを発見できたことにより，今後新たなプロジェクトにおいて活用することが可能であると考える．

**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

*奈良先端科学技術大学院大学 情報科学研究科 修士論文, NAIST-IS-MT1651105, 2018 年 2 月 1 日.

# Proposal for improvement in software development organization based on classification of problem detection for software requirements and software development process definition*

Toma Miyamura

**Abstract**

Software faults in spacecraft (e.g., artificial satellite and rockets) software leads to mission failures. To identify software faults, we focus on the incomplete software requirement, which is a major reason causing software faults. Based on the two perspectives of software fault and development software process, we present a classification method for software faults which are extracted from actual spacecraft project . For classifying defects, we apply classification based on the eight perspectives proposed by Avizienis. In addition, the perspective of development software process is based on the development standard provided by JAXA. We expect that this approach can help to identify the tendency of faults and reveal which certain kind of process leads to more faults.

**Keywords:**

Software process, Bug, Incomplete software requirement, Classification of faults

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1651105, February 1, 2018.

# 目 次

# 図 目 次



# 表 目 次

# 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．例えば，小田らによるコンパイラによって検出できなかったバグを見つける研究がある [14]．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 [5]．一方で，後者は，要求仕様書等にも記載されていなため発見が困難であり，不具合を発見するツールも存在しない．しかし，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 [4]．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，本論文では従来の汎用的な欠陥の分類に加え，開発者が採用しているソフトウェア開発プロセスに着目して分類する手法を提案する．静的である欠陥の分類と，動的であるソフトウェア開発プロセスに着目することで，ソフトウェア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り，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，対策を考える上での議論に指標の一つとなることが期待される．

今回の調査では，前提条件を明確にするプロセスや，要求仕様書の作成を行うプロセスにおいて不具合が多いことが確認され，欠陥の傾向も調べることができた．また，不具合の起点となるプロセスが 3 つ発見された．

本論文の構成を以下に示す．2 章では言葉の定義や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. 準備

本章では，様々な意味で理解することのできる原因や不具合について，言葉の定義を行うことで，意味の差異が発生することを防ぐ．

## 2.1 関連用語

### 2.1.1 原因の定義

原因とは，結果と対になり，因果性や因果関係とよばれ，原因と呼ばれる範囲においては，哲学や心理学など分野により異なる．一般的には，ある物事や状態を引き起こしたもとになった出来事を原因と呼ぶ．John は，原因を” 積極的と消極的のいずれを問わず，条件を全て集めた総和である “と定義している [9]．すなわち，原因とは事象（例えば事故など）が発生する十分条件の集合として考えている．また，Nancy は事象の原因を “それらの条件がすべて揃えば事象が発生する条件” と定義している [7]．すなわち，原因とは事象が発生する必要条件の集合として考えている．本研究では，原因が不具合に必ず関わる Nancy の定義と同様に，原因を事象の必要条件として定義する．

### 2.1.2 不具合

図 1 に不具合に関する欠陥，誤り，障害についての関係を示す [4]．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．正しいサービスの中になんらかの理由で混入してしまった Fault（欠陥）が原因となり，Error（誤り）と呼ばれる内部状態が発生する．誤りは内部状態であるため，開発者らは直接認知することはできない．また，システムに冗長性を持たせるなど工夫することで，誤りの影響をなくすことが可能となる．一方で誤りが遷移し，ユーザが意図してないサービスを提供されることを Failure（障害）という．

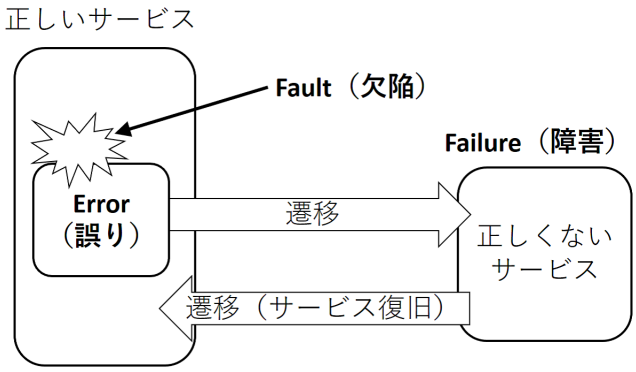


図 1　不具合に関する欠陥，誤り，障害の関係

### 2.1.3 プロセス

本研究で用いるプロセスとは一般的に使用されているウォータフォール型のソフトウェア開発プロセスのことである．ここではその 1 例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）が使用しているソフトウェア開発標準を用いた [11]．詳細な内容に関しては 3.2 章で示す．

## 2.2 現状の超高信頼性に関する問題点

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言われており [5]，高信頼性が求められるソフトウェアでは要求が大きな問題となりやすい．今回高信頼性の一例として宇宙機開発をあげる．

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をすることができない，製品開発が単発であり，開発期間も長い，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 [13][12]．

宇宙機の開発において，ソフトウェアの不具合による事故は，被害額が多いだけでなく人命にも関わる場合がある．例えば O リングの硬化により爆発したスペースシャトルチャレンジャー事故では 7 人の死者が，オペランドエラーにより無人ロケットが爆発したアリアン V 事故では 80 億 US ドル以上が失われた [1]．また，これらの事故の発生からソフトウェア開発に影響が与えられた．スペース

シャトルチャレンジャー事故ではIV&Vの強化がされ，アリアンV事故ではソフトウェアの設計・検証方法の見直しが行われた．一方で，宇宙機のソフトウェア開発では，このような不具合が発生しないためにも，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[3]．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，順序が存在しているプロセスを“動的なソフトウェア開発プロセス”と呼び，要求仕様書を作成する過程を含んだ，動的なソフトウェア開発プロセスに着目した．また，既存の順序が存在しない欠陥の分類を“静的な欠陥による分類”と呼ぶ．

既存の静的な欠陥による分類法と，動的なソフトウェア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章で述べるAvizienisらの提案した分類を[4]，ソフトウェア開発プロセスとしてソフトウェア開発標準を用いた[11]．

## 2.3 関連研究

静的な欠陥の分類に関する研究として，Avizienisらは欠陥について8つの観点による分類を行なっている[4]．Avizienisらは一般的な不具合全体を整理した上で，誤りの原因となる欠陥に着目し合計31通りに分類を行なっている．詳細な分類を行なっている一方で，具体的な対策法までは考慮されていない．また，この分類に関して詳細な内容は4章にて示す．

次に，欠陥に対する分析手法としてSTAMP(Systems Theoretic Accident Model and Processes)の一種であるSTAMP/CAST(Causal Analysis based on STAMP.)などがあげられる[6]．STAMPは事故モデルである．事故モデルとは，過去の事故を理解する，または未来の事故を防止するのに使用される．またSTAMPは故障モード影響解析(Failure Mode and Effect Analysis:FMEA)に比べ，分析時間に関するコストを減らしたものである[6]．またCASTは分析の行い方であり，事

故がなぜ起きたのかを追求していくものとなっており，今後不具合を発生させないことに着目している [6][8].

要求仕様書の不具合に対する対策の例として，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 [3]．この研究では，欠陥の基本要素に対して分析を行い，IEEE STD 830-1998[2] を元にチェックリストを作成したものである．

# 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
- ソフトウェア開発プロセス

既存の静的な欠陥による分類法と，動的なソフトウェア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．

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．本研究では欠陥の分類法として Avizienis らの提案した分類を [4]，ソフトウェア開発プロセスとして，JAXA が使用しているソフトウェア開発標準 [11] を用いた．本章では，それぞれの指標について詳述する．

## 3.1 欠陥の分類

Avizienis らは一般的な不具合全体について，誤りの原因となる欠陥に着目している [4]．彼らは，図 2 に示すように，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．彼らが定義した 8 つの観点と独立した 2 つの値を表 1 に示す．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．図 2 の左側には，8 つの観点のうち独立した 2 つの値を示し，該当する箇所を・で示している．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．

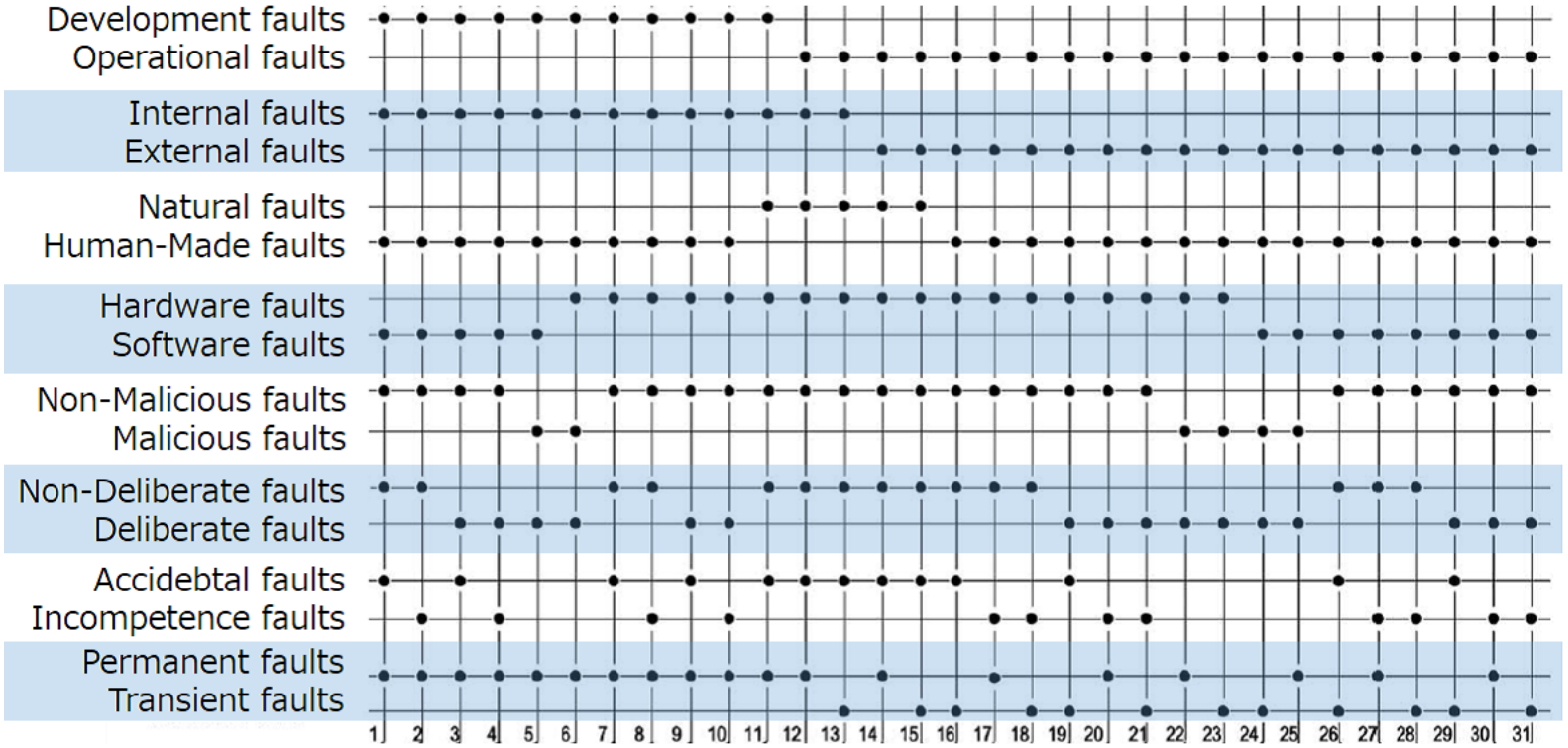


図 2　Avizienis らの提案する欠陥の分類

表 1 欠陥の観点と 2 つの値

| 観点 | 1 つ目の値 | 2 つ目の値 |
|---|---|---|
| item Phase of creation or occurrence | Development faults | Operational faults |
| item System boundaries | Internal faults | External faults |
| item Phenomenological cause | Natural faults | Human-Made faults |
| item Dimension | Hardware faults | Software faults |
| item Objective | Non-Malicious faults | Malicious faults |
| item Intent | Non-Deliberate faults | Deliberate faults |
| item Capability | Accidental faults | Incompetence faults |
| item Persistence | Permanent faults | Transient faults |

Avizienis らが提案した欠陥の分類は，不具合全体を対象としたものである．本研究では，要求が不完全であることに起因するソフトウェアの不具合を減らすことに着目している．そのため，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 であり，Software faults である項目である．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．よって本研究では，Development faults であり，Software faults であり，Malicious faults でないものを対象とする．これより，対象とする観点は Intent と Capability の有無であり，表 2 に示す 4 通りとなっている．

分類 1 は，開発者が気付くことがなく入れられた欠陥であり，不注意に導入された欠陥である．すなわち，開発者がすべきことを忘れていた欠陥等が該当する．

表 2 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 |

分類 2 は，開発者が気付くことなく入れてしまった欠陥であり，能力不足による欠陥である．すなわち，開発者がすることが分からなかった欠陥等が該当する．分類 3 は，考えた上での欠陥であり，不注意に導入された欠陥である．すなわち，開発者は正しく行なっていたつもりであるが，不注意などでたまたま混入した欠陥等が該当する．分類 4 は，考えた上での欠陥であり，能力不足による欠陥である．すなわち，開発者は正しく行なっていたが，能力不足によって混入した欠陥が該当する．

## 3.2 ソフトウェア開発プロセス

欠陥を分類するための指標として，JAXA が提供しているソフトウェア開発標準 [11] を使用した．これは ISO/IEC 12207[10] や JIS X0160-1996[15] 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，人工衛星・探索機・ロケット・地上システムなどに関わるソフトウェアの開発・運用・保守に関する活動が適用範囲であり，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（ソフトウェア開発プロセス，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ソフトウェア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，ソフトウェア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）を分析対象とした．それぞれの詳細な内容を表 3 に示す．以下グラフ中では対応 ID を，本文中では ID と略称を使用する．

表 3 ソフトウェア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 | 上位との整合性確認 |
| | 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 | 既存物との整合性の確認 |
| | 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 |

# 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不具合なのか，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，複数の欠陥が原因で1つの障害が発生することもあるため，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ソフトウェア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，それがどのソフトウェア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

## 4.3 分析結果

本分類ではまず初めに，あるプロジェクトで発生した不具合に関しての不具合データから，原因となる欠陥が何であったのかを1つの不具合ごとに調査を行っ

た．不具合データの分析の結果，4.1 章で述べた 48 件の障害から計 109 件の欠陥を明確にした．

次に欠陥 1 つ 1 つに対して 2 章で示したどの欠陥の分類に該当するのかを確認した．複数の欠陥により障害が引き起こされた不具合もあるため，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 章で述べた分類 1 から 4 のどれに当てはまるかを確認し，ラベリングした結果を表 4 に示す．この表では，不具合事例に対して分類 1 から 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 章で述べた表 2 に示す分類 1 は 30 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 109 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 から 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．

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

# 5. 分類結果に対する考察

本章では，4 章で分析した結果および，欠陥の総数や 3.1 章で示した 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 3 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，分類ごとに色や柄分けを行っている．

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，プロセス 4.9 の “前提条件の明確化” やプロセス 4.1 の “SW 要求仕様書の作成” 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ハードウェアなど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，プロセス 4.12 の “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プロセス 4.11 の “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．

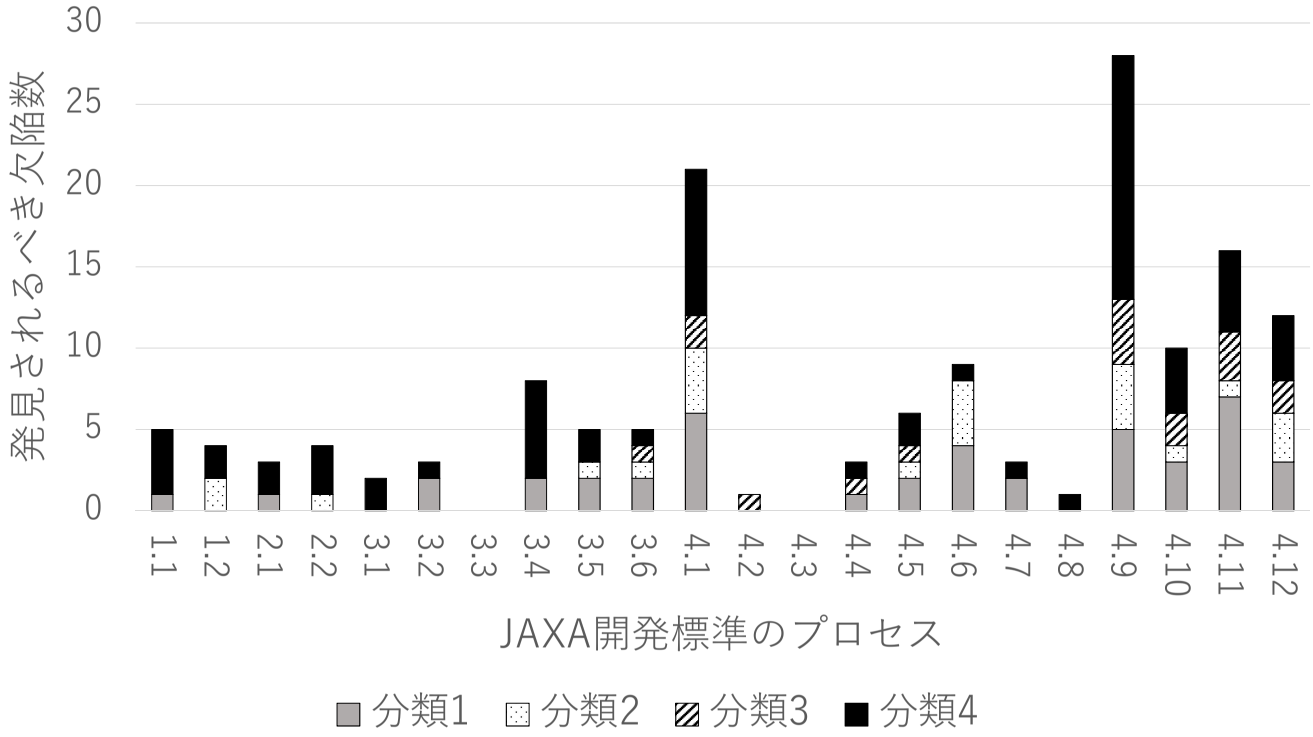


図 3 各プロセスにおいて発見されるべき欠陥数

## 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な差が生じるのかを図 4 に示す．グラフの横軸は表 3 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 の差が大きいプロセス 4.6 の “既存物との整合性の確認” 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，気付いていない間違いが起きていることが読み取れる．この理由として，確認すべき上位のプロセス 4.1 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．プロセス 4.1 の “SW 要求仕様書の作成” において，不具合が多いことは 5.1 章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，結果としてプロセス 4.6 の “既存物との整合性の確認” 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．

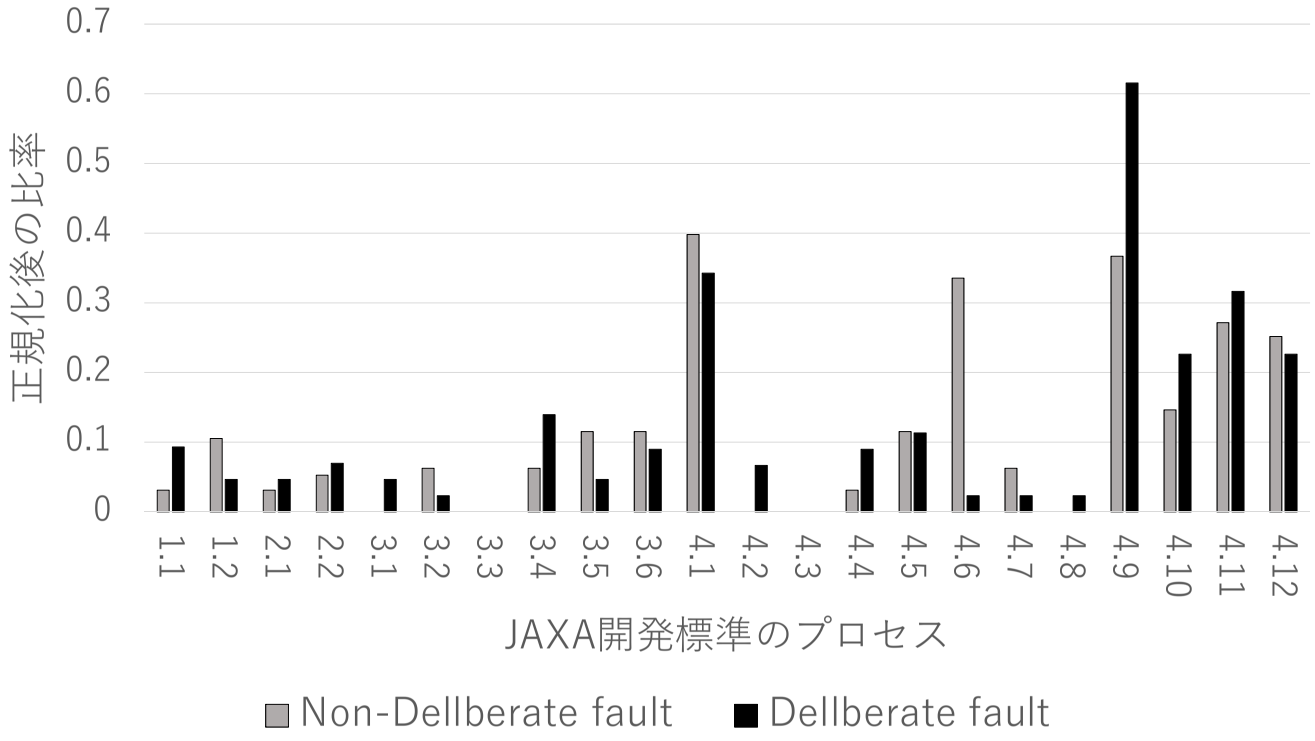


図 4 Deliberate fault と Non-Deliberate fault の比較

## 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 3 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 の差が大きいより上位のプロセスである．プロセス 1.1 の “開発計画立案” からプロセス 3.6 の “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プロセス 4.9 の “前提条件の評価” とプロセス 4.9 の “前提条件の明確化”，要求仕様書に関する，プロセス 2.2 の “要求仕様書作成” とプロセス 4.1 の “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．

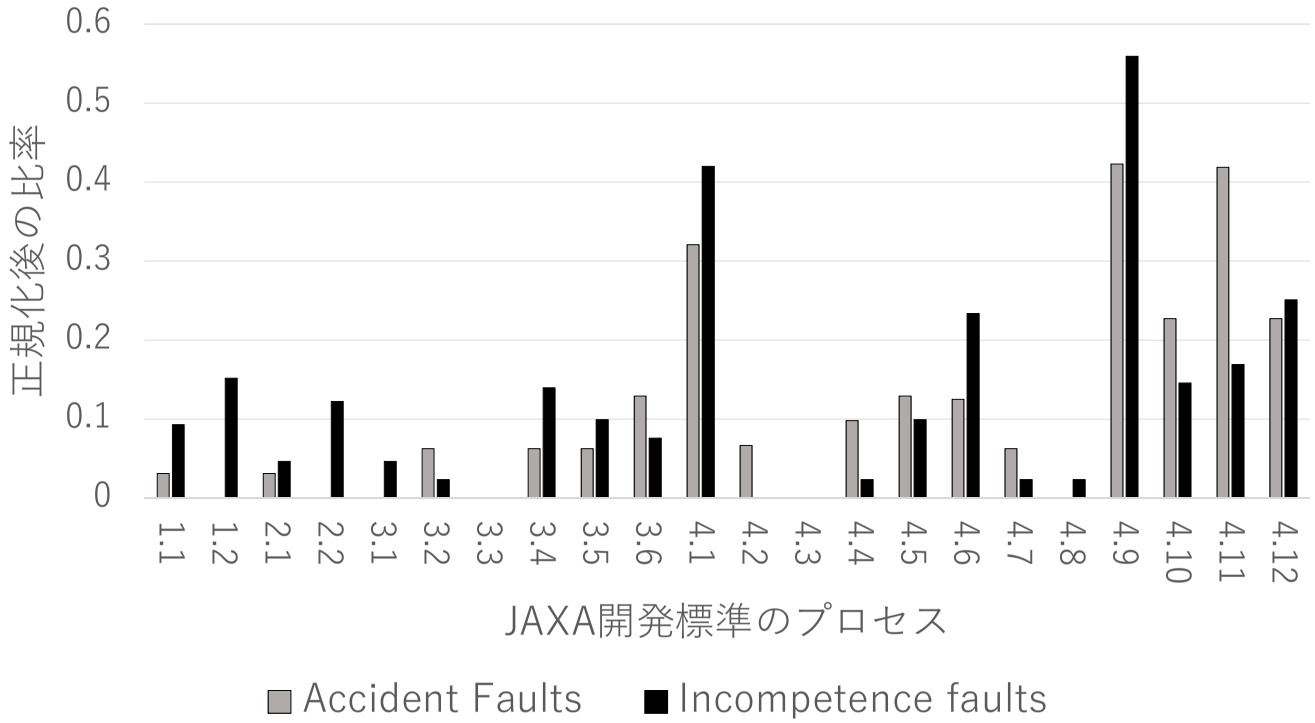


図 5 Accidental faults と Incompetence faults の比較

## 5.4 各プロセスの関係性について

表5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3を基に対応するプロセスIDを表記した．この表の数値は，4章で得たデータを次章で述べる手順に従い詳細に分析したものである．

### 5.4.1 各プロセスへの影響度の調査

表5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章で示した手順3で作成されたマッピング結果からプロセス4.4の“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章で示したように，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，プロセス4.4の“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．その値をプロセス4.4の“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，プロセス3.4の“前提条件の評価”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5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5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．影響とは，他のプロセスを解決することで，対象としているプロセスでの不具合がなくなることを指し示す．

表5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，プロセス 4.4 の “異常処理の仕様を含める” はプロセス 3.2 の “要求の割り当て” やプロセス 4.1 の “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．

表 5 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 |

### 5.4.2 欠陥の起点となるプロセス

表5より明らかになった点を述べる．まず1つ目に，表3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“開発計画立案”の影響が大きい点である．5.4.1章で示したように，プロセス2.1の“要求抽出”はプロセス1.1の“開発計画立案”の影響を非常に多く受けており，プロセス2.2の“要求仕様書作成”は1.1の“開発計画立案”やプロセス2.1の“要求抽出”の影響を受けている．また，プロセス3.1の“構成物の明確化”はプロセス1.1の“開発計画立案”，プロセス2.1の“要求抽出”，プロセス2.2の“要求仕様書作成”の影響が大きい．以上より，プロセス1.1の“開発計画立案”をより完璧に仕上げることで，プロセス2.1の“要求抽出”，プロセス2.2の“要求仕様書作成”，プロセス3.1の“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と考えられる．

2つ目に，表3で示した工程の中でも下位にあるプロセス4.9の“前提条件の明確化”が他のプロセスへ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，プロセス2.1の“要求抽出”，プロセス2.2の“要求仕様書作成”，プロセス3.1の“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るプロセスがプロセス4.9の“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，プロセス4.9の“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を導くことができた．欠陥の起点となっているプロセスとして以下の条件を考えた．

**条件1** 特定のプロセスから影響を受けていない

**条件2** 他のプロセスに関して影響を与えている

本分析の結果より，条件1を満たす候補としては，プロセス1.1の“開発計画立案”，プロセス3.2の“要求の割り当て”，プロセス3.3の“CSの実現可能性の評

価”，プロセス 4.2 の “識別子付与”，プロセス 4.3 の “DB 仕様書を含める”，プロセス 4.7 の “RA の実現可能性評価” があげられる．このうち，条件 2 を満たすものは，プロセス 1.1 の “開発計画立案”，プロセス 3.2 の “要求の割り当て”，プロセス 4.2 の “識別子付与” となる．

よって今回の不具合データからは，プロセス 1.1 の “開発計画立案”，プロセス 3.2 の “要求の割り当て”，プロセス 4.2 の “識別子付与” が欠陥の起点となり，このプロセスを強化することでより他のプロセスに関しても影響が伝わり，より多くの不具合を減らすことが期待される．

## 5.5 提案手法の議論

本研究で提案した手法は，静的な欠陥による分類法と，動的なソフトウェア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である．この分類法により，ある組織における要求漏れに起因するソフトウェア問題の傾向を知ることが可能となる．静的な欠陥による分類では，汎用的な観点における分類で欠陥について幅広く分析をしている．また，組織で使用している動的であるソフトウェア開発プロセスを用いることで，議論をすべき個所を絞ることができる．

従来の静的な欠陥による分類では，観点による分類だけであったため，分類結果をもとにプロジェクトを改善するには至らなかった．しかし，本研究で提案した動的なプロセスによる分類を組み合わせることで，着目すべき個所が明確になり，結果として議論すべき指標として活用することができる．本研究では動的なプロセスによる分類として JAXA で使用されているプロセスを用いたが，このプロセスは一般的なプロセスを基に作られている．他の一般的なウォーターフォール型のソフトウェア開発プロジェクトにおいても同様の工程が行われており，プロセスを変更することで利用することが可能である．

# 6. おわりに

本研究では，高信頼性が求められるソフトウェアでは要求が大きな問題となりやすいことから，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た．本研究で用いた分類は，静的な欠陥による分類法と，動的なソフトウェア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．

5.5 章で述べたように，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として，本論文で述べた分類法は有効であると考える．今後の課題として，本論文で提案した分類法を用いることで，プロジェクトがどのように改善するのか，影響度についての調査が挙げられる．今回の結果より，プロセス 4.9 の “前提条件の明確化” では不具合が多く検出され，プロセス 1.1 の “開発計画立案”，プロセス 3.2 の “要求の割り当て”，プロセス 4.2 の “識別子付与” では他の欠陥に影響を与えていることが分かった．この結果をプロジェクトにフィードバックすることで，プロジェクトにどのような変化があるかについて議論することが考えられる．

# 謝辞

本研究を進めるにあたり，ご指導，ご協力いただきました方々に心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア設計学研究室 飯田 元 教授には研究方針だけではなく，研究に対する姿勢や心構え，論文執筆，発表方法についても多くのご助言を頂きました．心よりお礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学学研究室 松本 健一 教授には研究をまとめるにあたり大変貴重なご意見を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 超高信頼ソフトウェアシステム検証学研究室 （宇宙航空研究開発機構 第三研究ユニット） 片平 真史 教授には安全性に関する幅広い見地からのご指導を頂きました．心よりお礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 超高信頼ソフトウェアシステム検証学研究室 （宇宙航空研究開発機構 第三研究ユニット） 石濱 直樹 准教授には研究方針や論文執筆などにご助言いただきました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 超高信頼ソフトウェアシステム検証学研究室 （宇宙航空研究開発機構 第三研究ユニット） 川口 真司 准教授には研究テーマの考案からデータ準備，評価，論文執筆，発表にいたるまで全ての過程に対してご助言を頂くだけでなく，つくば滞在時には様々なご配慮を頂きました．心より感謝いたします．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア設計学研究室 市川 昊平 准教授には発表方法や発表資料へのご指摘等，多くのご助言を頂きました．心より感謝いたします．

宇宙航空研究開発機構 第三研究ユニット（有人宇宙システム株式会社） 柿本 和希 様 には研究テーマの考案からデータ準備，評価，論文執筆，発表にいたるまで全ての過程に対してご助言を頂くだけでなく，つくば滞在時には様々なご配慮を頂きました．心より感謝いたします．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア設計学研究室 渡場 康弘 助教 には発表方法や発表資料へのご指摘等，多くのご助言を頂きました．心

よりお礼申し上げます.

宇宙航空研究開発機構 第三研究ユニットの皆様には，研究テーマに関する助言をいただいただけでなく，貴重なデータも頂きました．つくばでの研究活動の際には暖かく迎えていただき，非常に楽しく日々を過ごすことができました．心よりお礼申し上げます.

奈良先端科学技術大学院大学 小川 暁子 様、森本 恭代 様，仲田 綾奈様には研究の遂行に必要な事務処理など，多岐にわたりご助力いただきました．心より感謝申し上げます.

ソフトウェア設計学の皆様 には日々の議論の中で多くのことを学ばせていただきました．また皆様のおかげで非常に楽しい二年間を送ることが出来ました．心より感謝申し上げます.

末筆ではありますが，研究活動や日々の生活を支えてくれた家族に心から感謝を申し上げて，謝辞といたします.

# 参考文献


[1] 失敗百選. http://www.sydrose.com/case100/index.html.

[2] Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, pp. 1–40, Oct 1998.

[3] Amira A. Alshazly, Ahmed M. Elfatatry, and Mohamed S. Abougabal. Detecting defects in software requirements specification. *Alexandria Engineering Journal*, Vol. 53, No. 3, pp. 513 – 527, 2014.

[4] 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.

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

[6] IPA ニューヨークだより. 5月号 stamp（システム理論に基づく事故モデル）研究の現状（前編）. https://www.ipa.go.jp/files/000038950.pdf, 2009.

[7] Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley Professional, 1995.

[8] Nancy G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety (Engineering Systems)*. The MIT Press, 2012.

[9] John Stuart Mill. *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence and the Methods of Scientific Investigation*. John W. Parker, 1843.

[10] ISO/IEC: Amendment to ISO/IEC 12207-2008 Systems and Software Engineering. Software life cycle processes. iso, geneva, switzerland, 1995.

[11] 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.

[12] 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/SPIJapan/session1A/1A3.pdf.

[13] 宇宙航空研究開発機構. 宇宙航空研究開発機構 (jaxa) における信頼性向上とその対応策. https://www.ipa.go.jp/files/000004749.pdf.

[14] 小田まり子, 掛下哲郎. パターンマッチングに基づいた c プログラムの落し穴検出法. 情報処理学会論文誌, Vol. 35, No. 11, pp. 2427–2436, nov 1994.

[15] 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.

# 発表リスト

[1] 宮村純真，川口真司，石濱直樹，柿本和希，飯田元，片平真史， s 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，ソフトウェアサイエンス研究会，2018 年 1 月