

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、どのような不具合が多いのか、どのプロセスにおいて不具合が発生しやすいのかが明確になる。
キーワード アスキー版 p_LA_TE_X 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^{††}

[†] 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 hinder the success chance of missions. 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. By carrying out this classification, it becomes clear which kind of process leads to more faults.

Key words p_LA_TE_X 2_ε class file, typesetting

1. はじめに

ソフトウェアを開発すると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。その原因として大きくわけて2つあり、1つ目は”要求が正しいが、ソフトウェアの実装がおかしい”パターン、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で発見が用意であり、実際に起きる割合としてソフトウェアのプログラムミスは非常に少ない割合となっている[1]。信頼性を高めるためには、ソフトウェアの実装だけでなく、要求漏れに関してもカバーする必要がある。

本研究では、実際に宇宙機で発生した事例からソフトウェア

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、どのような不具合が多いのか、どのプロセスにおいて不具合が発生しやすいのかが明確になる。

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章ではデータを適用し、5章で考察し、6章でまとめを述べる。

2. 背景

宇宙機と呼ばれる、人工衛星やロケットなどは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、開発期間がながい、実環境でテストできないなどの特徴がある[2][3]。よって、

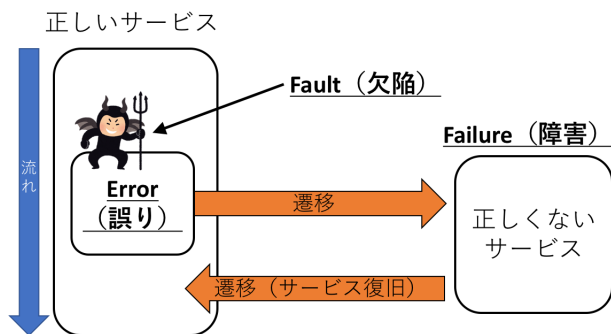


図1 不具合に関する欠陥、誤り、障害の関係

宇宙機のソフトウェア開発において、高信頼性は必要不可欠です。これを実現するために、既存の研究だけでは不十分である。また、高信頼性システムでは障害として顕著化しないような対策が必要となる。

この問題を解決するために本研究では、欠陥による分類法だけでなく開発プロセスを用いて、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。欠陥による分類法として Basic concepts and taxonomy of dependable and secure computing [4] を用いた。プロセスとして宇宙航空研究開発機構（JAXA）で使用されているソフトウェア開発標準を用いた。[5]

不具合の中でも欠陥に関して分類している研究はいくつかあり、Basic concepts and taxonomy of dependable and secure computing [4] や、Detecting defects in software requirements specification [6] などで行われている。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。よって私は、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對して、対処策が発見できるのではないかと考えた。

3. 提案する分類の枠組み

本研究で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ついての分類である。2つ目は、開発の工程を示している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、見えていなかった不具合の傾向の発見つつながることが期待できる。この章では、それぞれの指標について説明をする。

3.1 欠陥の分類

まず初めに、不具合とは、欠陥とは何かについて説明をする[4]。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事である。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。誤りが生じるきっかけや、原因となるものが Fault（欠陥）である。今回はその欠陥について着目した。

先行研究[4]では、一般的な不具合について整理されている。特に、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。図2 2値の値を持つ8つの観点を用いて、合計256通り

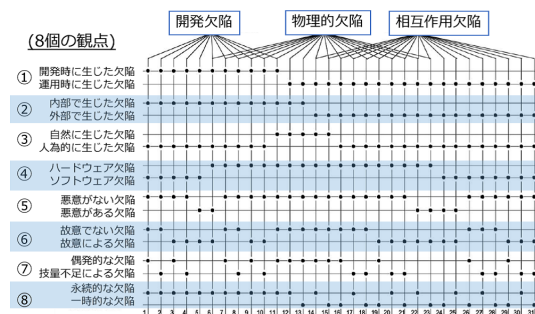


図2 欠陥の分類結果（要修正）

から言葉の定義上存在しないものを除いた、31通りで欠陥を網羅的に分類している。一方で、原因となる欠陥が分類できたとしても、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”に対処することは困難である。

本研究では、この分類からソフトウェアに関係し、さらに開発時に生じたものに着目した。その結果31通りの分類が、図2に示す1～5番までの5通りとなった。さらに、”悪意がある欠陥”は存在しないものとした。よって対象とするのは Intent の有無、Capability の有無の4つである。

3.2 開発プロセス

本研究では宇宙航空研究開発機構（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 ISO12207 * を元に作られたものである。表1に示すように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、もう一つが、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、3つに分けられ、さらにそれらが細かく分けられている。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、8つに分けられおり、こちらも同様に細かく分けられている。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問題を減らす”という目的があったため、表2に示す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。

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

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 HW の影響がある場合確認の評価

表 2 詳細な開発プロセス（内容要変更）

4. データの適用及び結果

この章では本研究で使用したデータについて説明する。また、2つの指標をどのように使ったのかについて述べる。

4.1 使用したデータ

今回 JAXA から48件提供の実際に発生した不具合情報のレポートを提供してもらった。このデータは機密事項な為、詳細には記述しない。また、この48件のデータ1つにつき1つの障害を示しているが、欠陥は1つとは限らない。

4.2 適用方法

本実験では以下の手順で適用した。

- 不具合1つ1つに対して、欠陥が何であるかを明確にした
- 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マッピングした
- マッピングされた欠陥に対して、それがどのプロセスで見つけるべきであったかを確認した
- 確認したデータを集計した

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

今回の48件の不具合から110件の欠陥を抽出することができた。すなわち、1件の不具合に対して複数の欠陥が存在している。この欠陥を分類した結果を表3に示す。分類区分とは図2で示した分類1～4に対応している。Dellberate fault であり Accidental faults である□は31件、Dellberate fault であり Incompetence faults であるは件22、Non-Dellberate fault であり Accidental faults であるは件13、Non-Dellberate fault であり Incompetence faults であるは件44であった。各不具合に対する結果の分類の細かなデータは付録にて掲載する

4.3.2 集計

上記の結果よりどのプロセスで発見すべきだったのかをマッピングした。一つの欠陥につき、複数のプロセスで見つけることになるものもある。マッピングした結果は付録に掲載する(図A-1)。1つの欠陥に対して複数の個所が該当することもある。プロセスにより分類した結果を□～□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が見えてくる。

5. 考察

本章では、JAXA のデータから導かれた結果について考察を行う。

5.1 積み上げグラフについて

図3のグラフは横軸に表2に記載している JAXA が使用し

分類区分	欠陥の種類	件数
□	Dellberate fault であり Accidental faults	31
□	Dellberate fault であり Incompetence faults	22
□	Non-Dellberate fault であり Accidental faults	13
□	Non-Dellberate fault であり Incompetence faults	44
合計		110

表3 各欠陥に対する不具合の数□ or 分類1

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である。また縦軸は不具合の件数となっており、青色が欠陥の分類□、オレンジ色が欠陥の分類□、灰色が欠陥の分類□、黄色が欠陥の分類□となっている。

このグラフで着目すべき個所は合計値が多い、プロセス4.9やプロセス4.1である。プロセス4.9とはソフトウェア要求分析フェーズの前提条件・制約事項の抽出である。これらの共通することとして、ソフトウェア以外の部分での未確定要素に大きく左右されるところである。すなわち、他のハードウェアや、環境などの影響で細かく決めきれずに多くの欠陥が検出されたと考えられる。この事実が正しいことを示すように、プロセス4.12のHWや環境を評価する項目の欠陥も多い値となっている。また、ここで漏れたとしても、テストでカバーできればよい。しかし、4.11試験の計画可能性の評価の値も高く、不具合として検出できないものが多くなっていると考えられる。

5.2 □+□について(タイトル要変更)

図4のグラフは横軸に表4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である。また縦軸は欠陥の数で正規化したものとなっている。青色は欠陥の分類□と□を足したものとなっており、オレンジ色が欠陥の分類の□と□を足したものとなっている。すなわち、Dellberate fault か Non-Dellberate fault においてどのような差があるのかがこのグラフから見えてくる。

このグラフで着目すべき個所は青色とオレンジ色の差が大きいところである。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。青の値が多いことから、しっかりと考えた上で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げることができる。このフェーズは不具合が多いことは先ほど示した。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青色の項目がオレンジに比べ高い割合であったと考えられる。

5.3 □+□について(タイトル要変更)

図5のグラフは横軸に表5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である。また縦軸は欠陥の数で正規化したものとなっている。青色は欠陥の分類□と□を足したものとなっており、オレンジ色が欠陥の分類の□と□を足したものとなっている。すなわち、Accidental faults か Incompetence faults においてどのような差があるのかがこのグラフから見えてくる。

このグラフでも着目すべき個所は青色とオレンジ色の差が大きいところである4.11である。ここの Accidental faults の値が Incompetence faults の値にくらべて大きい理由として、ここでも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、1.1～3.6において青色に比べてオレンジ色の値が目立つ結果となっている。よって、この辺りでは技量がかかなり重要なファクターとなっている。さらに、ここでのミスが後々響いてくることがこのグラフから読

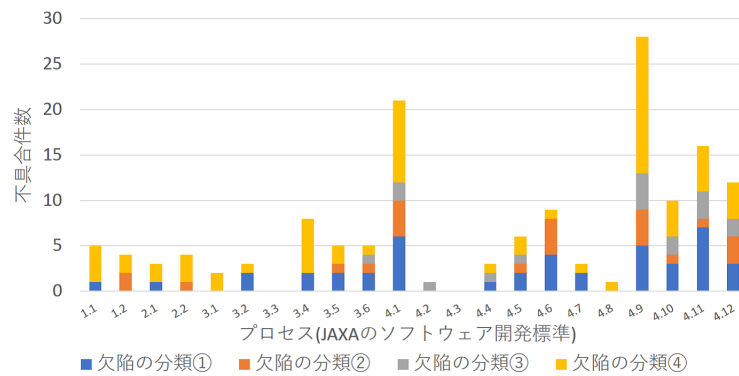


図 3 積み上げグラフ

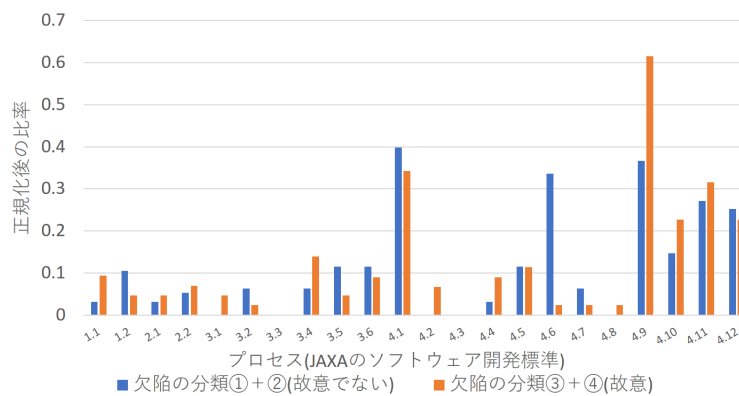


図 4 欠陥の分類①+②と③+④の比較

み取れる。例えば、同じ前提条件や制約に関する 3.4 と 4.9、要求仕様書に関する、2.2 と 4.1 など、後ろに行くにつれ影響が広がっていることが読み取れる。

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、対応する番号を表記した。この表の数値は、4 章で得たデータを詳細に分析したものである。

5.4.1 表の作成方法

黄色のセルについて、この数値をどのようにして計算し導かれたものであるかを説明する。例として黄色の箇所について説明をする。不具合の原因となるプロセスが 4.4 であるものを抽出した。今回、4.4 が原因であったものは合計で 3 件見つかり、この値が表の一番下の行に記述している。また 1 つの欠陥に対して複数のプロセスが該当しているので、そのプロセスをカウントし、その値を 4.4 のプロセス数で割り算した値となっている。これを行うことで、各プロセスの依存関係、すなわちどのプロセスに影響があるのか、どのプロセスから影響をうけているのが見えてくる。

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 以上の場合は赤で色を塗っている。すなわち関係性が高いものが赤色、関係性が薄いものが白色となっている。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。

よって、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 考察

この結果より分かったことがいくつかある。まず 1 つ目として、より上位の工程に着目する。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いる。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。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、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る。なので、互いに影響しあうのは理解できる。

2 つ目として、最も欠陥の多かったプロセス 4.9 も関してである。他のプロセスが 4.9 に影響をもたらしているのはプロセス 2.1、プロセス 2.2、プロセス 3.1、プロセス 4.2 である。上記で示したように、1.1 の解決で 2.1.3.1 の欠陥が減ることが予想される。また、プロセス 3.2 やプロセス 4.2 は何かの影響を受けやすいという結果は出ていない。プロセス 4.9 が他のプロセスから大きな影響を受けているわけではない。ここでプロセス 4.9 が他のプロセスから影響を受けにくいと理由として、前提条件に関する項目がここで初めてでてきたため、このフェーズより上位のフェーズから満遍なく影響が広がっているものと考えられる。

今回の結果より他のプロセスに影響を与えるピボットとなっ

ているプロセスが導かれる。それは特定の箇所から影響を受けていない項目である。候補としては以下に示すものがある。

- プロセス 1.1: ソフトウェア開発計画を立案すること
- プロセス 1.2: 開発計画の文章化
- プロセス 3.2: 要求仕様書の作成
- プロセス 3.3: 実現可能性を評価
- プロセス 3.5: 上位とのトレーサビリティを評価
- プロセス 3.6: インターフェース要求を抽出
- プロセス 4.2: 個別に識別子を付与
- プロセス 4.3: データ及びデータベースに対する仕様を含める

一方で、プロセス 3.3、プロセス 4.2、プロセス 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、プロセス 1.1、プロセス 1.2、プロセス 3.2、プロセス 3.5、プロセス 3.6 がピボットとなるプロセスであると考えられる。

6. ま と め

今回、JAXA の約 50 件の不具合データを元に分析をした。この分析により、プロセス 4.9 での不具合が最も多く、原因として Non-Dellberate fault であり Incompetence faults の割合が最も高かった。また、プロセス 4.6 では Dellberate fault と Non-Dellberate fault の間で、プロセス 4.11 では Accidental faults と Incompetence faults の間で大きな差が見られ、より信頼性をあげるためにはここに対策を入れる必要があることが分かった。また、プロセス間の影響としてプロセス 1.1、プロセス 1.2、プロセス 3.2、プロセス 3.5、プロセス 3.6 がピボットとなっており、ここに力を入れることで、連鎖的に欠陥が減ることが予想される。

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] 宇宙航空研究開発機構 石浜直樹, "ソフトウェア開発標準," <https://www.ipa.go.jp/files/000004749.pdf>.
- [3] 宇宙航空研究開発機構 古石ゆみ, "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/2009/SPI-Japan/session1A/1A3.pdf>.
- [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, Jan. 2004.
- [5] 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-0-049.pdf>.
- [6] A.A. Alshazly, A.M. Elfatraty, and M.S. Abougabal, "Detecting defects in software requirements specification," Alexandria Engineering Journal, vol.53, no.3, pp.513–527, 2014.

付 録

表 3 の詳細なデータである。

図 A・1 はプロセスによって欠陥を分類した結果である。

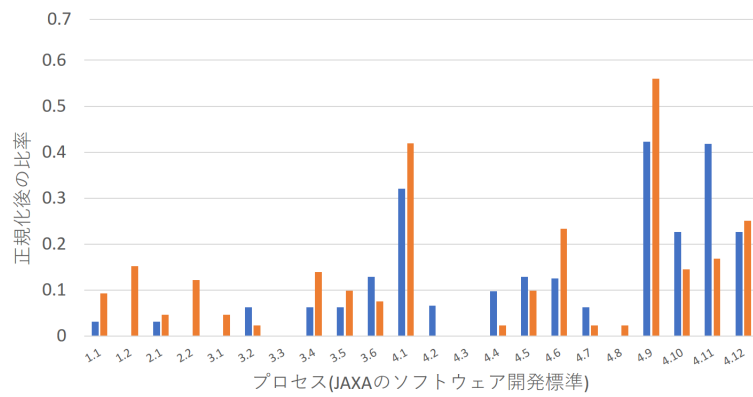


図 5 ㊦+㊦と㊦+㊦の比較

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
1.1	ソフトウェア開発計画を立案すること	0.2	0.3	1	0.5	1	-	-	0.1	-	-	0	-	-	-	-	-	-	-	0.1
1.2	開発計画の文章化	0.2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2.1	要求を抽出する	0.6	-	-	0.5	1	-	-	0.1	-	-	0	-	-	-	-	-	-	-	0.1
2.2	要求仕様書の作成	0.4	-	0.7	-	1	-	-	0.1	0.2	0.2	0.1	-	-	-	0.2	-	-	-	0.1
3.1	構成する品目、種別を明確に	0.4	-	0.7	0.5	-	-	-	0.1	-	-	-	-	-	-	-	-	-	-	0.1
3.2	各構成品目に要求を割り当てる	-	-	-	-	-	-	-	0.1	-	0.2	0.1	-	-	0.7	0.2	-	-	-	0
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	0.2	-	0.3	0.3	0.5	0.3	-	-	-	0.2	0.1	-	-	0.3	-	-	-	-	0.3
3.5	上位とのトレーサビリティを評価	-	-	-	0.3	-	-	-	-	-	0.2	0.1	-	-	-	0.2	0.4	-	-	0
3.6	インターフェース要求を抽出	-	-	-	0.3	-	0.3	-	-	0.2	-	0.1	-	-	-	0.8	-	-	-	-
4.1	ソフトウェア要求仕様書の作成	0.2	-	0.3	0.5	-	0.7	-	0.1	0.4	0.6	-	-	-	0.7	0.5	0.1	-	-	0.2
4.2	個別に識別子を付与	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0
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	-	-	-	-	-	0.7	-	0.1	-	-	0.1	-	-	-	-	-	-	-	0
4.5	インターフェース仕様に関して合意を得ること	-	-	-	0.3	-	0.3	-	-	0.2	1	0.1	-	-	-	-	-	1	-	-
4.6	上位との整合性を得る	-	-	-	-	-	-	-	-	0.8	-	0	-	-	-	-	-	-	-	0.1
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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.2	-	-	-	-
4.9	前提条件、制約を明確に	0.4	-	0.7	0.5	1	0.3	-	1	0.2	-	0.2	1	-	0.3	-	0.2	-	-	-
4.1	検証可能性を評価	0.2	-	0.3	0.3	0.5	-	-	0.1	-	-	-	-	-	-	-	0.1	-	-	0
4.11	試験計画可能性を評価	-	-	-	-	-	-	-	-	-	-	-	-	-	0.3	-	0.1	-	-	-
4.12	環境や HW の影響がある場合確認の評価	-	-	-	-	-	-	-	0.1	-	0.2	0	-	-	-	0.2	-	0.3	-	0.1
対象となるフェーズの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

表 4 各プロセスへの影響度

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			
	☒	☒	☒	☒		☒	☒	☒	☒		☒	☒	☒	☒
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 A・1 詳細な欠陥に対する不具合の数

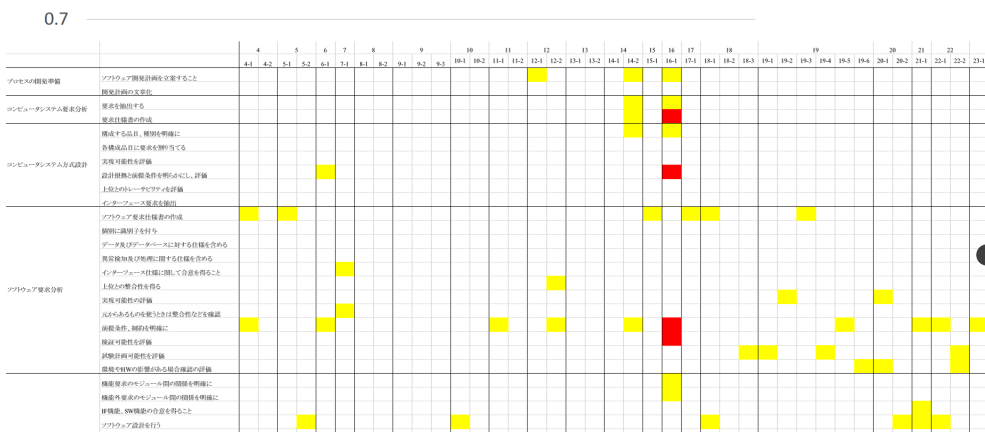


図 A・1 プロセスによる分類