



ロボット革命イニシアティブ協議会  
Robot Revolution & Industrial IoT Initiative

# ロボットシステム開発プロセス・ 品質管理調査検討委員会 中間報告書

平成 31 年 3 月

## 概要

平成 30 年度の5月~3月までの活動の報告である。ロボット開発の委員会対象分野で抱える課題の抽出経緯とその結果、及び、抽出した課題から後期の主テーマとした ROS 等の OSS(オープンソースソフトウェア)の製品実装時の課題、特に企業が求めるソフトウェア品質確保にガイドを与えるための ROS の品質調査・分析結果である。

ロボット革命イニシアティブ協議会  
ロボットイノベーション WG  
ロボットシステム開発プロセス・品質管理調査検討委員会

# 目次

1. はじめに.....	3
1.1. 位置づけ.....	3
1.2. 委員会の目的と平成 30 年度のテーマ.....	3
1.3. 委員会活動内容.....	4
1.4. 対象読者.....	5
1.5. 委員名簿.....	6
2. 活動の概要.....	7
2.1. アンケートに基づく課題の抽出.....	7
2.2. OSS の実態調査.....	8
2.3. OSS の構造解析・静的解析.....	10
3. OSS の実態調査結果.....	11
3.1. Navigation Stack.....	11
3.2. Movelt!.....	12
4. OSS の構造解析・静的解析結果.....	14
4.1. 構造解析結果.....	14

4.2. 静的解析結果.....	17
5. 考察.....	21
6. 終わりに.....	24
7. 参考文献.....	27

# 1. はじめに

## 1.1. 位置づけ

本報告書は、RRI の SWG<sup>1</sup>で議論すべき内容と NEDO のプロジェクト<sup>2</sup>で議論すべき内容とが一致をみたため、RRI の WG3 の SWG1 の下で共同活動した品質管理・開発プロセス調査検討委員会からの中間報告である。

## 1.2. 委員会の目的と平成 30 年度のテーマ

本委員会のミッションは、ロボット開発下の品質管理とプロセスに関しての、必要とされる分野のガイドライン化や抱える技術課題の提言、育成・技術方策等の提言である。委員会の中ではテーマを決め打ちせずに各委員が抱える品質と開発の課題を純粹に聞くことから始めた。議論の結果、今回は、ROS (Robot Operating System)をはじめとするオープンソースソフトウェア (OSS: Open Source Software) の製品実装の方法論を固めることとした。これを取り扱った論拠には大義があるが、長くなるので、補足資料「プラットフォーム化の必要性」(最終報告書にて掲載)にて述べる。

今年度は、先に述べた課題抽出までの議論内容とロボットの OSS の代表である ROS の品質調査の結果を報告する。調査結果からの対策など平成 31 年度の計画は 6 章にて述べる(ソースコードをどこまで直すのか、ドキュメントはどうするのかなど)。

---

<sup>1</sup>「プラットフォームロボットを軸とした誰もが使いこなせる「Easy to Use」な ロボットの実現等を検討する」ミッションである RRI のロボットイノベーション WG (WG3)のプラットフォームロボット サブ WG (SWG1)

<sup>2</sup>「ロボット活用型市場化適用技術開発プロジェクト(NEDO)」中の研究開発項目「ロボットのプラットフォーム化技術開発」

### 1.3. 委員会活動内容

ロボットが抱える技術課題が外からも中からも良く見えない(絞り込めていない)ところが大きな問題である。今回も課題の特定に多くの時間を要している。有効な議論が開始できるところまでにも多くの時間を要した。日本のロボット技術の発展に当たっては、技術課題化する能力を各技術者が持たなければならない。そのため、課題抽出に敢えて時間をかけて活動した。

また、当初はロボットの開発プロセス全般についてのガイドを期待されていた様であるが、取り扱っていない。理由は次の通り。ロボットのソフトウェア開発プロセスをヒアリングした結果、SPICE でアセスメントすれば多くのロボット開発企業ではレベル 0 か 1 になるであろう。組織能力成熟度は他の電機産業界から見て高いとは言えない。加えて危機感もなく、各企業でプロセス改善活動も取り組まれていないため、委員会活動にはなり得ず、勉強会にしかならないためである。既に他の産業分野が取り組んできた開発プロセス全般のガイドをまず参照されたい<sup>3</sup>。ちなみに、経験知として成熟度が低い組織の改善実行はプロセスより技術から入った方が、改善が上手くいくことが多い。よって、各委員が抱える課題から具体的な技術課題を抽出して今回のテーマとした。

企業からは「ROS は製品実装できない<sup>4</sup>」、一方、ROS コミュニティ関係者からは「ROS は製品実装可能な品質であるのになぜ使わない<sup>5</sup>」というかみ合わない話は何回も聞いてきた。解析も調査もせずに、空中戦のみで進めても前に進まないの、今回実際に手を動かして現物調査を実施した(3章)。ROS を触ったこともない方が多数を占めていたことも現物調査を決めた理由の一つである。構造解析・静的解析も実施した(4章)。企業経験のノウハウも入れているので、静的解析結

---

<sup>3</sup> ソフトウェアプロセスや品質管理のガイドラインはロボットに限らず IPA が 2004 年より取り組み、多くの有効なガイド [4]を出している。

<sup>4</sup> 企業からは、「ROS は使わないと生きていけないが、品質、品質保証、保守が問題。」、ROS を製品実装したと言っても品質・機能が問題でほとんど作り変えたと聞いている。」「日本メーカーで ROS を搭載している製品はほとんどない。」というアクションもない、愚痴を多数の方から聞いた。

<sup>5</sup> ROS コミュニティ関係者からは「ROS は非常に多くの人が使ってきているから十分に製品実装可能な品質である。」「品質保証のプロセスは ROS コミュニティで定義されており、問題がない。」

「静的解析をかけたが、一般的な製品ソフトウェアに比べて良いくらいだ。静的解析をかけてもコードの不具合はでない。」との意見があった。

果の世の中の平均点よりは出来は良い。一つ一つを具体的にどうするのが決まる様に更に深く解析を進めなければならない<sup>6</sup>。

なお、ROS の製品実装の考えがこのようなかみ合わない議論になっている大きな原因の一つは、コミュニティ内の情報提供が企業側から少なく、研究開発者からの情報に限られていることと推定する<sup>7</sup>。OSS を使うが改良した版をフィードバックしないという日本の企業がモラルの低い行為をしているからである。自分の傘は使わず公共の傘を毎回借りて、壊れたら捨てて傘は提供しないのと同じである。「OSS 活用の成熟度」が低い。企業の OSS 活用レベルを向上させないと OSS を上手く使いこなせずにソフトウェアとシステム構築で日本は世界に取り残される危険がある、という提言を行っておく。

今回の委員会活動においては、ROS のソースの調査などを行い、委員の方に多くの時間を割いて頂いた。佃副委員長、亀山リーダー、平山リーダーをはじめ委員の方には感謝申し上げます。

## 1.4. 対象読者

本報告書の対象読者としては、以下を想定している。特に企業の製品開発において、ROS 等の OSS の活用を検討しているマネージャと技術者を主な対象者とする。

- ・ ロボット製品・システムの開発者
- ・ ロボット製品・システムの研究者
- ・ ロボット製品・システムの品証保証担当者

---

<sup>6</sup> ここはこの様な論拠からこの様に点検して救う、どこはどのような判断から捨てる、といった結論。

<sup>7</sup> 推定した論拠は、「社外秘やセキュリティを理由に会社内で改良した情報は企業の管理システムを通すと外に出せない」と多くの委員と NEDO プロジェクト参加者が語ったこと。及び、実際に ROS の検証を行った時に乱数で毎回ランダムに動くなど製品であれば当然持つべき機能が存在しないことからである。

## 1.5. 委員名簿

委員長	(国研)新エネルギー・産業技術総合開発機構	増田 昌庸
副委員長	イーソル(株)	佃 明彦
委員	川崎重工業(株)	亀山 篤
	川崎重工業(株)	中道 大介
	(国研)産業技術総合研究所	Geoffrey Biggs
	(株)セック	中本 啓之
	THK(株)	椎木 靖人
	THK(株)	三好 崇生
	(株)東芝	平山 紀之
	(株)東芝	山本 大介
	トヨタ自動車(株)	高橋 太郎
	(一財)日本品質保証機構	櫛引 豪
	パナソニック(株)	岡本 球夫
	(株)日立製作所	中村 亮介
	(株)日立製作所	吉内 英也
	富士ソフト(株)	橋詰 政人
	富士通(株)	神田 真司
	(株)安川電機	平田 亮吉
	(株)YOODS	原田 寛
	(株)YOODS	平泉 一城

(敬称略、五十音順)

## 2. 活動の概要

本委員会では、平成 30 年度に以下の活動を実施した。本章ではそれぞれの活動概要を報告する。

- ・ アンケートに基づく課題の抽出(5月～10月)
- ・ OSSの実態調査(11月～3月)
- ・ OSSの静的解析・構造解析(1月～3月)

### 2.1. アンケートに基づく課題の抽出

品質管理・開発プロセス面において、各委員が抱える課題を纯粹に聞くことから始めて、深堀をしていった。製品開発に OSS を利用する際の目的、課題、課題を解決するための取り組み等について参加委員にアンケートを実施し、得られたアンケート結果を元に委員会で議論を深めていった。アンケートは以下のテーマで 4 回継続して実施した。

表 1 委員会で実施したアンケート

回数	アンケートのテーマ	回答者数
1	製品の品質保証・品質管理で抱えている課題	10
2	製品の品質保証・品質管理で抱えている課題(質問追加)	14
3	協調領域と OSS の活用	6
4	商用製品搭載ソフトウェアの出荷条件	7

第 1 回と第 2 回のアンケートでは、開発コスト削減や新規技術の取り込みを目的として OSS の採用を検討しているが、以下のような課題がなかなかクリアできないため、各企業で採用に踏み切れない実態が明らかになった。

- ・ OSS の品質を評価するプロセスや指標、検査項目の明確化
- ・ OSS を使用した場合のコスト面、品質面におけるメリットの明確化
- ・ OSS を使用した製品の保守に対する体制の確立

第 3 回のアンケートでは、ロボット開発における協調領域と競争領域の分類は各社で見方が分かれるものの、協調領域においては、以下のような取り組みが有効であることが挙げられた。

- ・ OSS や標準プラットフォームの活用によりインターフェースを標準化して普及を図る
- ・ OSS や標準プラットフォームの開発を他社と共同で進めることでコスト削減を図る

第 4 回のアンケートでは、商用製品に搭載するソフトウェアの出荷条件について、以下のような意見が挙げられた。

- ・ OSS を特別扱いせず、自社製ソフトあるいは購入ソフトと同等の評価が必要  
(評価項目、評価条件、評価結果とエビデンスを求めるケースもある)
- ・ OSS をブラックボックスとして扱う場合は入出力に対する評価基準の明確化が必要

これらのアンケート結果について委員会での議論を重ねながら、EU で行われている ROSIN の Software Quality Assurance 活動 [1]なども参照した。企業の製品開発においては、OSS の品質を掌握しておくことの重要性を改めて確認した。

## 2.2.OSS の実態調査

OSS を企業の製品開発で使用することを想定して、一般的なソフトウェア開発の V 字モデルに当てはめると、各工程でどのような課題が存在するのかを調査した。

委員からの要求が圧倒的に多かったため、調査対象の OSS はロボットの開発で広く普及している ROS とし、ROS の中でも特に利用者が多いと思われる Navigation Stack [2]と MoveIt! [3]を選択した。調査対象のパッケージは以下の通りである。

表 2 Navigation Stack の調査対象パッケージ

パッケージ	機能	担当委員
amcl	自己位置推定	セック 中本委員
base_local_planner global_planner move_base	経路計画	東芝 平山委員/山本委員
map_server	地図配信	富士通 神田委員
costmap_2d	コストマップ	日立製作所 吉内委員/中村委員

表 3 MoveIt!の調査対象パッケージ

パッケージ	機能	担当委員
MoveIt!	マニピュレーション	川崎重工業 亀山委員
move_group	ROS ノード	THK 三好委員
visualization	可視化	安川電機 平田委員
perception	環境認識	YOODS 竹林委員
planning_interface	ユーザインターフェイス	富士ソフト 橋詰委員

調査は Navigation Stack と MoveIt!の 2 グループ制で、各グループに所属する担当委員でパッケージ毎の機能毎の縦割りで調査を実施した。V 字モデルの開発プロセスで分ける横割りの案もあったが、品質の評価軸の統一性を取ることが難しいため、縦割りとした。対象の OSS を自らの製品開発の工程に取り込む際に問題となりそうな点とクリアすべき課題を抽出した。各委員の観点で課題抽出することを重視したため、パッケージ毎に基準はあえて統一せず抽出した。各パッケージの調査結果については、3 章を参照のこと。

## 2.3.OSS の構造解析・静的解析

OSS を企業の製品開発で使用する際には、OSS そのものの品質保証をどのようにするかが課題の一つとなっている。ロボット用の OSS の品質保証に向けた取り組みとして、まずロボット用の OSS のソースコードを解析し、定量的にソフトウェアの品質を把握することを試みた。解析対象の OSS は ROS のコアパッケージと Navigation Stack および MoveIt! を選択した。これらの OSS をインストールするための OS は、Ubuntu 16.04 とした。

表 4 解析対象

解析対象	バージョン
ROS コアパッケージ	Kinetic
Navigation Stack	1.16.2
MoveIt!	0.9.15

ソフトウェアの品質を把握するにあたり、ソースコードを解析し特定のコーディングルールに違反している箇所を抽出する「静的解析」と、ソフトウェア構造の複雑性を評価する「構造解析」の 2 つの解析手法を適用した。静的解析には Rogue Wave 社の Klocwork (バージョン 2018.3)、構造解析には Headway Software Technologies 社の Structure101 (バージョン 4.2) というツールを用いた。

調査は国立研究開発法人産業技術総合研究所が実施した。各パッケージの解析結果については、4 章を参照のこと。

## 3. OSS の実態調査結果

本章では、2.2 章で挙げた OSS について、ソースコードの静的解析および、構造解析を行った結果を報告する。

### 3.1. Navigation Stack

Navigation Stack は自律型移動ロボット向けのフレームワークである。ロボットが認識可能な地図上で自己位置を推定、目標位置までの経路を計画、周辺の障害物を検出してコストマップを生成し、目標位置まで自律的に移動することを目的とするものであり、移動ロボット向け OSS としては最も使用されているものの一つとなる。今回その Navigation Stack の中でもよく使用される主要なパッケージに関して、各開発工程における問題点について実態調査を行った。

要求分析、アーキテクチャ設計、詳細設計及び実装の V 字モデルの設計工程に関して、どのパッケージについても共通する問題として各機能の正確な挙動、利用条件、性能指標に関するドキュメントが存在しないことが挙げられる。各機能、各 API の機能目的自体は Navigation Stack の ROS Wiki 上に、ソースコード内のコメントやそれを元に作成されるドキュメントに記載はあるが、具体的な挙動についてのコメント、ドキュメントはほぼ存在しない。そのため、ソースコードを読み解く以外にパッケージを理解する手段はなく、期待する自律移動動作の結果を得るために、どのパッケージを選択し、選択したパッケージ内でどのクラス、API を使用すればよいか、また各クラスのパラメータを変更した場合にどのように挙動が変わるのかを理解するには ROS に対する高度なスキルが要求され、各設計工程を実施する上での大きな障害となる。現状の対応としては、ソースコードの解析、関連する論文の調査や関連するコミュニティへの Q&A などで解決することになるが、相当な知識が必要かつ運も必要で確実に成果が得られるかわからない。主要な機能、API について利用しやすい形でのドキュメントがあれば開発の大きな助けになる。ドキュメントを整備し、維持する仕組みの構築は重要な課題であると考え、これらは OSS では対応が難しいところがあるが、ロボットの実用的な制御とその製品化のためには必要な情報であると考え、どのように進めれば仕組みを構築できるかは今後の検討課題である。一方で、これらのパッケージをブラックボックスとして使用する分にはパッケージの詳細を理解しなくても十分利用する価値があるという意見もあった。

単体テストについては、pull request に対して Jenkins を使用した CI テストの結果を参照することができる。CI テストは単にビルド成否のみ記載されているものもあれば、テストコードの実行による結果まで記載されているものもあり、パッケージによりまちまちである。テストコードが存在するパッケージについてもドキュメントは存在せず、テスト項目やカバー率に関する情報を得るにはパッケージ本体およびテストコードのソースコードを読み解く以外に方法はない。更に言えば、単体試験は内部設計書の十分性を確認するものであり、コードから作る試験は単体試験とは言えない。

結合テスト、総合テストについて、単体テストの内容が不明確であることから、前工程で見つかるべきバグが後工程で見つかる可能性がある。バグに対しては ROS コミュニティへ Issue 登録して開発者に修正してもらいたいべきであるが、OSS である以上、修正版が開発工程上の期日までに提供される保証はない。修正版が期日までに提供されなかった場合の対策としては、自力で修正する、バグを起こさないような回避策を自社ソフト側で施す、回避策が困難であれば制限事項として明記し、関係者へ通知して運用で回避する、といった案が考えられる。

## 3.2.MoveIt!

MoveIt! はマニピュレータ用のモーションプランニングについてのフレームワークである。マニピュレータとそのエンドエフェクターを指定された姿勢にするため、周辺機器、ロボットと干渉しないような経路を計算しマニピュレータを制御、動作を行うことを目的とするものであり、マニピュレータ制御用 OSS としては最も使用されているものの一つとなる。今回その MoveIt! の中でもよく使用される主要なパッケージに関して、各開発工程における問題点について実態調査を行った。

要求分析、アーキテクチャ設計、詳細設計及び実装の V 字モデルの設計工程に関して、どのパッケージについても共通する問題として各機能の正確な挙動を記載したドキュメントが存在しないことが挙げられている。各機能、各 API の機能目的自体はソースコード上のコメントやそれを元に作成されるドキュメントに記載はあるが、具体的な挙動についてのコメント、ドキュメントはほぼ存在しない。そのため、期待する経路計算の結果を得るために、どのクラス、API を使用すればよいか判断できず、また各クラスのパラメータを変更した場合にどのように挙動が変わるのかが分からず各設計工程を実施する上での大きな障害となる。現状の対応としては、ソースコードの解析、関連す

る論文の調査や関連するコミュニティへの Q&A などで解決することになるが、相当な知識が必要かつ運も必要で確実に成果が得られるかわからない。もし主要な機能、API について利用しやすい形でのドキュメントがあれば開発の大きな助けになる。これらの整備とそれを維持する仕組みの構築は重要な課題であると考え。他には非機能要件についての記載がないことや異常処理についての仕様が定義されていないなど設計思想は統一されているのかに疑念を抱く様な問題点の指摘があった。これらは OSS では対応が難しいところがあるが、ロボットの実用的な制御とその製品化のためには必要な情報であると考え、それらをどのように定義するかについて今後検討が必要であると考え。

単体テストについて、各パッケージで対応状況はまちまちである。Jenkins を使用した CI テストが実行されているパッケージもあれば、Google Test を使用し実施するものや、一見して単体テスト手段が見当たらないパッケージもある。共通して言えることは、テスト仕様書がないため、どのようなテストが行われているかはそれぞれのテストモジュールを確認するしかないという点がある。また仮に使用者側のほうで単体テストを作成しようとしても、機能や API の詳細仕様を把握するのが困難なため、作成にそれなりの時間が必要になる。Jenkins を使用した CI テストなどは良い取り組みであると思われるので、それらを各パッケージへの展開しその仕様書を用意するなどの対応は有効であると考え。

結合テスト、総合テストについて、テストの結果期待される動作が行われない場合に、機能の正確な仕様がわからないことにより、使用しようとしている機能の選択が間違っているのか、使い方、パラメータの設定が間違っているのかなどの原因の切り分けが困難であるという指摘があった。現状はソースコードの確認が最も有効な手段となるが、アルゴリズムとその根拠となる理論の理解も必要なるためデバッグの難易度が高い。また仮に機能の選択が間違っているとなった場合には V 字工程の最初のほうまで手戻りが発生するリスクがあり、そのリスクを開発計画時に考慮する必要がある。これは ROS を採用する際の大きな障害となる。これらデバッグの難易度を下げるために、コンポーネントの入力/出力値の解析・可視化ツールを用意することが有効ではないかとの提言があった。

## 4. OSS の構造解析・静的解析結果

本章では、2.3 章で挙げた OSS の C++ のソースコードについて、構造解析および、静的解析を行った結果を報告する。

### 4.1. 構造解析結果

ROS のコアパッケージと Navigation Stack、MoveIt! の C++ のソースコードに対して、構造解析ツール Structure101 を用いて構造解析を実施した結果を示す。

Structure101 では、ソースコードの構造の良し悪しを、「Tangle(もつれ)」と「Fat((肥大)」という 2 つのメトリクス(指標)で評価する。Tangle は相互参照しているファイルの数で、Tangles(フォルダ) はフォルダを跨いで相互参照しているファイル数を表す。Fat は関数レベル、ファイルレベル、クラスレベルおよびフォルダレベルに算出されるメトリクスで、Fat(関数)は循環的複雑度(Cyclomatic Complexity)を表し、他のレベルでは包含する要素間(例えばフォルダレベルの場合はフォルダ内のファイル間)の依存グラフのエッジ数を表す。表 7 にメトリクスとしきい値を示す。しきい値は、過去の蓄積データから導き出された Structure101 が提唱している値であり、各メトリクスはいずれも小さい値ほど構造が良いと判断する。

表 5 構造解析のメトリクスとしきい値

メトリクス	内容	しきい値
Tangles(フォルダ)	フォルダを跨いだファイルの相互参照数	0
Fat(フォルダ)	フォルダ内の要素の依存数	120
Fat(ファイル)	ファイル内の要素の依存数	120
Fat(クラス)	クラス内の要素の依存数	120
Fat(関数)	循環的複雑度(Cyclomatic complexity)	15

これらのメトリクスに対して、各パッケージの構造解析を行い、しきい値を超えているものがどの程度の割合で存在するかを評価した。その結果をエラー! 参照元が見つかりません。に示す。

表 6 構造解析結果

解析対象パッケージ	Tangles (フォルダ)	Fat (フォルダ)	Fat (ファイル)	Fat (クラス)	Fat (関数)
ROS コアパッケージ	1%	0%	47%	0%	0%
Navigation Stack	0%	0%	28%	0%	1%
MoveIt!	1%	0%	45%	0%	0%

構造解析の結果、各パッケージとも、Fat(ファイル)の超過率が大きいですが、その他のメトリクスについてはいずれも概ねしきい値内に納まっていることが分かった。全体的には、相互参照、複雑度ともに低く、保守性が高いソースコードと言える。しかしながら、各メトリクスのしきい値を超過している対象を確認した結果、以下のことが分かった。

- Tangles(フォルダ)

ROS コアパッケージと MoveIt!でフォルダを跨いだ相互参照が多いファイルが見られるが、パラメタや共通的な処理であり、構造上の欠陥ではないと判断する。対象数も少なく、問題視する必要はないと考える。

- Fat(フォルダ)

しきい値を越えているものはなく、問題ないと判断する。

- Fat(ファイル)

Fat(ファイル)の数値が高いものは、ファイル内の要素(パラメタや関数)の依存関係(参照)が多いものであり、ソフトウェアの更新時に当該ファイルを更新する際の工数(影響範囲分析の難易度増、テストケース数の増)が大きくなるという傾向がある。

ファイルの中で宣言している変数が多いだけで、構造上は問題がないものが多いが、Fat(ファイル)の数値が 1000 を超えているものが 121 ファイル(ROS パッケージ:61 ファイル、Navigation Stack:6 ファイル、MoveIt!:54 ファイル)あり、これらはソースコードの構造が複雑(1 ファイルあたりのソースコードの行数が 1000 行を超える、内部で扱っている数式が複雑など)であり、潜在的な不具合が残存している可能性があり、品質上の懸念がある。amcl や move\_base、moveit\_core などの主要パッケージにも該当するファイルがあり、有識者に

よる詳細なコードレビューと、コードレベルのユニットテストを実施し、品質の検証が必要である。

- Fat(クラス)

しきい値を超えているものはほとんどなく(Navigation Stack で 1 件、MoveIt! で 2 件)、構造上も大きな問題はない。

- Fat(関数)

Fat(関数)は循環的複雑度を示す数値であり、ネストが深いなど、ソースコードが複雑になっているものを示す。統計的には、しきい値を超えているものは少なく見えるが、全体で 541 関数(ROS パッケージ: 160 関数、Navigation Stack: 77 関数、MoveIt!: 304 関数)がしきい値を超えていた。中でも、しきい値が 50 以上のものが 92 関数(ROS パッケージ: 23 関数、Navigation Stack: 14 関数、MoveIt!: 55 関数)あり、それらは、条件分岐やネストが複雑であり、潜在的な不具合が残っている可能性がある。ROS の各パッケージは、コードレベルのユニットテストが行われておらず、カバレッジの評価も行われていない。複雑なソースコードにも関わらず、コードの検証が不十分というリスクのある状態である。しきい値が 50 を超えている関数に関しては、ソースコードを元に、フローチャートなどの設計書の作成と、それに基づくコードレベルでのユニットテストによる検証が必要であると考えられる。

ソフトウェア構造の問題は、ソフトウェアの読解が困難になり、品質を確保するためのテストケースも多くなることから、品質への影響が大きい。これは、現時点での品質だけではなく、未来の品質にも影響を及ぼす。前述のソースコードの構造が複雑なものは、当該パッケージのバージョンアップの際に、不具合を作りこむ危険性も高く、今後将来に渡り ROS 関連のパッケージを使い続けるためには、前述したような対策を今から打っておくことが重要である。

## 4.2. 静的解析結果

ROSのコアパッケージと Navigation Stack、MoveIt!の C++のソースコードについて、Klocworkで静的解析を実施した。それぞれのパッケージについて、ツールによりソースコード上に問題があると指摘が出た件数および、各パッケージの規模[KLOC](コメント除く実行行数)、ソースコードの指摘密度[件/KLOC](コメント除く実行行数 1000 行あたりの指摘件数)の結果をエラー! 参照元が見つかりません。に示す。

各パッケージのソースコードのうち、チュートリアルやテストコード、ツールなどは、解析対象から除外し、ロボットのアプリケーションとして利用されるもののみを解析対象とした。また、Klocworkでは、重大/エラー/警告/レビューの4段階のレベルで指摘が出るが、ここでは、プログラム上の欠陥につながる可能性があるもののみを抽出するため、重大とエラーのレベルの指摘件数のみカウントした。警告/レビューは、致命的な欠陥ではなく、ソースコードのメンテナンスのためには改善した方がよいというものであり、今回は解析の対象外とした。

表 7 静的解析結果

解析対象パッケージ	指摘件数 [件]	規模 [KLOC]	指摘密度 [件/KLOC]
ROS コアパッケージ	1,696	256	6.6
Navigation Stack	298	33	9.0
MoveIt!	709	135	5.3
OpenCV3(参考値)	2,145	1,007	2.1

画像処理の分野でよく利用される OSS である OpenCV3 を同じように Klocwork で静的解析したところ、指摘密度は 2.1 件/KLOC であった。OpenCV3 も品質が良いとは言えない数値であるが、OpenCV3 と比較して、ROS コアパッケージの指摘密度は 3.1 倍、Navigation Stack は 4.3 倍、MoveIt! で 2.5 倍であり、OpenCV3 よりもプログラム上の欠陥が内在しており、製品実装には品質改善が必要な可能性がある、と評価できる。

次に、静的解析結果の指摘の内訳をエラー! 参照元が見つかりません。に示す。

表 8 指摘カテゴリの内訳

#	指摘カテゴリ	ROS コア パッケージ		Navigation Stack		MoveIt!		計	
1	C/C++の警告	1	0.1%	5	1.7%	0	0.0%	6	0.2%
5	NULL ポインタの逆参照	1,244	73.3%	43	14.4%	48	6.8%	1,335	49.4%
8	バッファオーバーフロー	22	1.3%	12	4.0%	15	2.1%	49	1.8%
10	ポータビリティの指摘	0	0.0%	0	0.0%	0	0.0%	0	0.0%
11	メモリリーク	54	3.2%	7	2.3%	41	5.8%	102	3.8%
12	リソース処理の指摘	164	9.7%	6	2.0%	423	59.7%	593	21.9%
14	不適切なメモリ割り当て解除	4	0.2%	1	0.3%	0	0.0%	5	0.2%
15	不適切な反復子の使用法	0	0.0%	0	0.0%	0	0.0%	0	0.0%
16	到達不能コード	9	0.5%	2	0.7%	6	0.8%	17	0.6%
17	同時実行	2	0.1%	0	0.0%	1	0.1%	3	0.1%
18	問題となる可能性があるコードプラクティス	0	0.0%	2	0.7%	0	0.0%	2	0.1%
20	強力な型チェッカー	0	0.0%	0	0.0%	0	0.0%	0	0.0%
21	戻り値の型の不一致	10	0.6%	2	0.7%	2	0.3%	14	0.5%
22	戻り値の無視	0	0.0%	0	0.0%	0	0.0%	0	0.0%
23	文法上の警告の指摘	0	0.0%	0	0.0%	0	0.0%	0	0.0%
24	未使用のローカル変数	0	0.0%	0	0.0%	0	0.0%	0	0.0%
25	未初期化のデータの使用	152	9.0%	198	66.4%	169	23.8%	519	19.2%
26	未検証のユーザー入力	8	0.5%	17	5.7%	0	0.0%	25	0.9%
27	無効な算術演算です	24	1.4%	3	1.0%	3	0.4%	30	1.1%
28	禁止が必須とされているAPI	0	0.0%	0	0.0%	0	0.0%	0	0.0%
29	禁止が推奨されているAPI	0	0.0%	0	0.0%	0	0.0%	0	0.0%
31	解放済みメモリの使用の可能性	2	0.1%	0	0.0%	1	0.1%	3	0.1%
32	計算値の不使用	0	0.0%	0	0.0%	0	0.0%	0	0.0%
	計	1,696	100.0%	298	100.0%	709	100.0%	2,703	100.0%

上記より、「NULL ポインタの逆参照」、「未初期化のデータの使用」、「リソース処理の指摘」、の3つの指摘カテゴリで全体の80%を占めていることが分かる。これら3つのカテゴリの指摘について、以下に見解を述べる。

- 「NULL ポインタの逆参照」

入力データの未検証が起因となる指摘であり、プログラムの異常終了につながる重要な指摘である。ROS コアパッケージで1,244件の指摘があるが、フレームワーク的に利用される

パッケージということを考慮すると、一般的には入力値の正当性を呼び出し側で保証する(入力値のチェックをする)事が多いため、フレームワーク単体で静的解析を実施した場合は、潜在的な欠陥ではない場合も多い。したがって、指摘全件について詳細に解析を行い、ユーザーが開発したアプリケーション(ROS ノード)側で保証すべき指摘と、ROS コアパッケージ本来の指摘を切り分け、後者はソースコードの修正の必要性を評価し、対処するというアクションが必要である。また、ユーザーアプリケーション側で保証すべき指摘に関しては、注意すべき点をチュートリアルや開発ガイドとして整理し、共有する、ユーザーアプリケーションを開発した際は静的解析ツールによる解析を行うことが有効である。

- 「リソース処理の指摘」

メモリやファイルなどのリソースのリークの可能性が指摘されている。MoveIt!のソースコードを解析すると、fopen などのストリーム取得に対する終了処理(fclose)が記載されておらず、有限なリソースを開放していないことが指摘されている箇所があった。明示的な終了処理はない場合であっても、デストラクタで自動的に開放(free)しているソースコードも確認できたため、すべてがリソースリークに繋がる訳ではないが、長時間動作時に異常が発生する可能性があり、ソースコードの改修が必要である。

- 「未初期化のデータの使用」

未初期化のデータの使用に関して、ローカル変数の場合は問題が起こることはあまりないと考えるが、一部で未初期化のヒープメモリ領域を利用している可能性が指摘されており、CERT(ID:EXP33-C)やCWE(ID:CWE-457)で報告されている問題が潜在している可能性がある。これらについても、予期せぬ動作や異常終了につながる可能性があり、ソースコードの改修が必要であると考えられる。

その他 20%の指摘も含め、ツールによる指摘は、あくまで「プログラム上の欠陥である可能性がある」というだけであり、研究開発用途では、このまま利用していても、大きな問題にはならない。しかしながら、人と共存するロボット、安全性を担保しなければならないロボットなどの製品や実運用システムで ROS やその関連パッケージを利用する場合は、話は別である。今回検出された指摘全件について、本当にプログラム上の欠陥でないかを明らかにし、欠陥であった場合は、その修正と

修正後の検証を行わない限り、ROS やその関連パッケージを品質保証された状態で、安心して利用することはできない。

ここで、今回検出された指摘 2703 件のうち、158 件の指摘をサンプリング抽出し、プログラム上の欠陥であるかどうかの評価を行った。その結果、おおよそ半数は、欠陥である可能性が高く、より詳細な調査を行った上で、修正が必要であろうことが分かった。半数というと、ROS コアパッケージ、Navigation Stack、MoveIt!という ROS の主要パッケージだけで、約 1350 件(3.2 件/KLOC)の欠陥が残存していることになる。そのうち、全てが異常終了などの致命的な欠陥であるわけではないが、これをこのまま看過することはできない。一般的に、企業が開発する組込み系の最終製品では、不具合の残存率は 0.001~0.005[件/KLOC]と言われており、それと比較すると現状のロボット用 OSS の品質は十分に高いとは言えない状況である。そのため、OSS とのコミュニティとも方針を調整した上で、既存の欠陥の可能性のある箇所の修正と、その後の維持メンテナンスの体制の構築、OSS コミュニティへのフィードバックを行っていく必要があると考える。

## 5. 考察

本委員会の平成 30 年度の活動を通じて、ROS を始めとした OSS を活用して自社製品の競争力を強化したいというニーズは高まっているものの、それらを実際に各社の開発プロセスに取り込もうとすると、人材の育成、ドキュメントの整備、コード品質の向上、更にはコミュニティの形成など、乗り越えなくてはならない課題が多く存在することが判明した。

2 章のアンケート結果では、委員の心理が一種のジレンマに陥っているようにみえた。海外を始めとした競合は OSS を活用して早く安く良い製品を出してきそうだが、OSS の品質課題が解決できず自社製品では採用に踏み切れないということである。これは一見矛盾している。OSS の品質が本当に悪ければ競合製品は脅威にならないはずであるし、OSS を使った競合製品が脅威になるのであれば自社でも入手して使えばいい。本当の困りごとは OSS の品質を見極めた上で、あるところは改善しながらうまく使う、あるところは使うのを諦めて作り直すといった、判断の基準やプロセスが定まっていないことであると考えられる。このような判断は、第三者が作ったガイドラインを参照すれば誰でもできるという類のものではない。OSS の品質をしっかりと把握しながら、自社にとってのリスクと効用を理解して判断を下さないと意味がない。本委員会のような議論の場を通じて、そのような判断ができる人材を育てていく必要がある。

3 章の調査結果では、ROS パッケージの仕様書や使い方を示すドキュメントに関する課題が多く抽出された。特にパッケージの公開から期間が経過して内容が古くなっているものや、元々の質や量が不足している例が散見され、企業の製品開発で使用するには設計書とコードの不一致等、品質的に放置できない部分である。また今後はクラウド等で稼働する AI や ICT などの情報システムと、実空間で稼働するロボットシステムを連携させたい場面が増えてくると見込まれるが、そのような現場を扱う技術者は、必ずしもロボティクスの専門家とは限らない。そのような人たちにとって使用しやすい、ハードルの低い環境を用意する必要があり、開発プロセスの基本であるドキュメントの整備が求められる。

4 章の構造解析結果では、解析対象の ROS パッケージの構造は概ね良好であるが、一部のファイルや関数の複雑度がしきい値を超えていた。ソースコードの複雑度は保守性や可読性を下げ、

現在の品質だけでなく将来の品質にも悪影響を及ぼす。不具合が明らかになる前からレビューを実施して構造を見直すなど、予防的な改善活動が必要である。

また同じく4章の静的解析結果では、解析対象の ROS パッケージは他の OSS と比較して指摘密度が高いという傾向が確認された。静的解析は全てのコードの誤りを検出できるわけではないため、潜在的な不具合は更に多く存在している可能性もある。静的解析はソースコードの誤りの検出と修正に用いるだけでなく、ソースコードの品質の測定に用いることができる。指摘密度が高いものは指摘された以外にも欠陥が多い、指摘内容を見て、修正できるレベルでないコード品質のものは使えない、である。

例外はあるにしろ、組込み系ソフトウェアでは、「障害を発生させうる欠陥がないこと」が企業側の出荷条件になることが多く<sup>8</sup>、出荷時にその証明が求められる。構造解析や静的解析結果は品質基準を満たしていることの証明の論拠の一つとして用いるものである。購入ソフトウェアや OSS など一定の欠陥を認めざるを得ない場合でもその潜在欠陥密度は推定が必要で、その潜在欠陥数から想定した障害発生頻度に対して保守体制を構築しなければならない。

ROS の品質に関してもこの証明を如何にするかは各企業に委ねざるを得ないが、製品実装できるまでの品質になるまでの道のりは示さなければならないし、最低限の修正はせざるを得ない。

今回の調査で発見した品質面の課題は、既に多くの先行調査で分析されているように、企業の製品開発者と OSS の開発者の間で品質保証に対する取り組み方が異なっているために生じている。OSS の開発者に対して最初から企業レベルの品質保証活動を期待することは合理的でないため、まず企業の開発者が不足しているエビデンスを作成するなど、OSS を自社の開発プロセスに取り込むための活動が必要である。もし対象の OSS が協調領域で使われるものであれば、複数の企業で OSS ユーザーのコミュニティを形成し、品質保証活動の一部を共同で実施することも可能である。また対象の OSS を長期的に活用していくことを考えると、OSS の開発者に対しても企業

---

<sup>8</sup>企業で用いられている出荷時の品質基準(残存欠陥密度)の一例を挙げると、組込み系で 0.001~0.005 件/KL、エンタープライズ系で 0.02~0.05 件/KL 程度である。

の要望や改善点をフィードバックし、徐々に OSS 自体の品質を向上させていく活動も有益である。そのような活動は個々の企業が都度行うよりも、複数の企業がまとまって取り組む方がより合理的だと考えられ、その様な枠組み(企業体、協会、コミュニティ等)の設立も検討しなければならない。

## 6. 終わりに

OSSを一括りにして扱いを決めるのは誤りだと途中で気付いた。コードの特性が異なるし、目的も異なる。MySQLの様に一社で作り設計思想も統一され品質向上に関しても取り組んでいるものもあれば、コミュニティで色々な方が色々なアルゴリズムを作っており、色々な品質レベルになっているものもある。また、LINUXの様に純粋な無償提供をして一人で品質監視を徹底しているものもあれば、ユーザーを増やすためにOSS化して保守・教育で儲けを出すビジネスモデルとしてのOSSもある。ROSは純粋な無償提供であるが、コミュニティからの貢献を取り入れるときに品質監視が十分であるのだろうか。

SW-CMMに始まり、CMMI、ISO15504(SPICE)などの組織能力・成熟度モデルをベースにきちんと設計・検証と管理をすることに、ここ20年間世界中が注力してきた。また、日本は組込みソフトウェアを強みとして、プロセス・技術・人材育成の向上に努め、更なる日本の強みとしてきた。これらの文化革命を経験した後に、ドキュメントを作ったら終わりだというOSSの文化を聞くと人依存の文化に逆戻りした様で、頭の中で整理しきれない。その状態は未だ続いている。考えれば考える程、研究開発という水に製品品質という油を注いで混ぜている様に思えて、間違えた検討をしているのではないかと自信がなくなってくる。

ROS-IやROSINも製品実装化に取り組んでいるが、日本企業の視点から見ると、製品が要求する品質までは確保されない様な動きであり、製品実装版になりそうもない。

ROSを製品実装するためにはどう扱っていくのが正しいのだろうか？ 現時点における考えは以下の①である。加えて、機能的な分類を行い②の整理で対応可能なものを検討する。ROSのメリットを鑑みた上、ROSを品質分析して使えるところは骨までしゃぶりつくして使う考えである。使えないなら、使えない定量的・定性的論拠を持って次に進む。静的解析結果の定量値だけ見ると相当品質が悪いレベルだからと言って直ぐに全部捨てるという軽率な考え方はしない。

③はこれから各企業の更なる取組が必要となろう。④は注視が必要である。過剰な品質は事業の妨害となるし、ユーザーにとっては高価格化を招くデメリットを生じる。

- ① ソフトウェアに造詣が深い企業経営者ならば、ゼロから新たにソフトウェアを作るより、世の中で使われている便利なソフトウェアを品質分析して、使える部分は改良して徹底的に使う、製品の品質にはなり得ないと判断したところ/新たに作った方が早く安いと判断したところは捨てる、である。ゼロから新たに作っても不具合は埋め込まれ、安定するまでに期間が掛かることを知っているからである。但し、設計思想がバラバラなソフトウェアは使えないリスクをはらんでいる、また、質の悪いソフトウェアは直しても、直しても品質が安定しないことが経験値としてあるため、捨てるか修正するか判断は基準を作り厳密に実施する。
- ② 一番素直で正しいやり方。信頼性が求められない製品や信頼性が求められない機能に対して、ROSをそのまま使う。外側で安全や品質が担保できる方法論があるならば、フェールセーフ・フェールソフトの機能(安全等)を独立で動作させて使う。
- ③ OSSを使いこなすためのポピュラーな方法。OSSを使うならば、その企業からOSSのコミュニティに深く入り込んでいる技術者を作り、ドキュメントはないが個人の知識・経験・技術に頼るやり方で品質を担保するやり方。品質保証としては完全ではないが、便利なソフトウェアを早く安く使いこむには一つの良いやり方である。エンタープライズ系はこのやり方が適している様に感じる。元々、サーバのハードウェアとファームウェア、ミドルウェアの組み合わせで早く安くシステムを構築することが可能であるが、品質はよろしくないため、冗長系と保守で稼働率を保ってきた。これはコストの分割払い的な欧米文化である。
- ④ 消費者や企業の品質・信頼性の概念が変化することに期待する方法。品質や信頼性に五月蠅いと言われる日本においても、感覚が変わってきたところもある。Windows PCは時々おかしくなるので、リセットや電源オフオンすることが消費者の扱いのデファクトスタンダードになった。家庭にある回線モデムやルータがハングしても通信会社にクレームを入れず、自分で電源を入れなおす人がほとんどであろう。携帯電話へ期待する品質レベルも不具合ゼロから電源入れなおしは仕方なしと文化を変えてくれた。日本人は言えるところには厳しく言うが、言っても無駄なところには黙っている様な傾向がある。海外メーカーには言わない。

以上の考えに従い、OSSを活用して、製品実装可能な共通ソフトウェア部品とすることを諦めることなく、平成31年度も取り組んでいく。NEDOプロジェクトの活動とも連携して、コード品質の整理や設計書作成などのサービスのニーズ調査と技術論の確立、ROSコミュニティとの関係構築も計画立案していく予定である。協調領域と競争領域の整理も必要に応じて実施する。実施内容の詳細は委員会で議論して決めていく。

以上

## 7. 参考文献

- [1] ROSIN, “Quality Assurance Process and Community Management in ROS,” [オンライン]. Available: <http://rosin-project.eu/wp-content/uploads/D3.1-Quality-Assurance-Process-and-Community-Management-in-ROS.pdf>.
- [2] ROSWiki, “navigation,” [オンライン]. Available: <http://wiki.ROS.org/navigation>.
- [3] MoveIt!, “MoveIt!,” [オンライン]. Available: <https://moveit.ros.org/>.
- [4] IPA, “書籍情報,” [オンライン]. Available: <https://www.ipa.go.jp/sec/publish/index.html>.