

# ロボットシステム開発における



## OSS 利用の品質保証ガイドライン



2020 年 6 月

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

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. 対象読者 .....	4
1.3. 委員名簿 .....	5
2. ロボットシステムの開発プロセス .....	7
3. OSS 利用の目的と課題 .....	11
3.1. OSS 利用の目的 .....	11
3.2. OSS 利用の課題 .....	13
4. OSS 利用の品質保証活動 .....	20
4.1. 選定段階 .....	20
4.2. プロトタイプ開発段階 .....	26
4.3. 製品開発段階 .....	29
4.4. 保守段階 .....	33
5. 終わりに .....	36
6. 参考文献 .....	37
付録 .....	39
OSS を使用している時の保守の例 .....	39
自社で公開する OSS 保守の例 .....	41
OSS に対するセキュリティ対策の例 .....	45



# 1. はじめに

## 1.1. 位置づけ

本書は企業のロボットシステム開発において、ROS や OpenRTM に代表されるオープンソース・ソフトウェア（以下 OSS: Open Source Software）を利用する際に実施すべき品質保証活動のガイドラインである。

一般的に企業の製品開発では 1990 年代から OSS の採用が増え始め、現在ではクラウドサーバ上のビジネスプラットフォームからスマートフォン等のコンシューマ機器に至るまで広く普及が進んでいる。既に OSS を積極的に利用している業界においては、その利用方法や取り扱いについて一定の知見が蓄積されており、IPA（独立行政法人 情報処理推進機構）等からもガイドライン [1] が提示されている。

一方で日本国内のロボット業界は 1970 年代から産業用ロボットを中心に発展し、ロボットメーカー各社は自社の商品力強化や顧客の囲い込みを目指して、独自のシステム開発と品質保証に取り組んできた。その過程で OSS を積極的に取り込む動きは見られなかったが、近年は市場で要求されるロボットシステムの複雑化、IT システムとの連携の必要性、海外を中心とする安価なロボットシステムとの競争などの影響により、様々な OSS に対するニーズが高まってきている。更に従来と異なる未活用領域にロボットを導入する場合には、ロボットシステムに対する要求自体が最初に定まらない事も多い。そのような開発では、システムの開発と現場実証を繰り返すために、再利用性の高いソフトウェアを組み合わせ素早くシステムを構築する必要がある。このような背景も、誰もが自由に入手して利用できる OSS に対するニーズを後押ししている。

しかしながら顧客に提供するロボットシステムの責任を負う企業の立場からは、品質状況が明らかでない OSS を受け入れ難い場面も多い。ロボット業界では、OSS を利用する際にどの程度の品質保証活動を行えば良いのかといった知見も共有されていないため、OSS の必要性は感じているが利用に踏み切れないという、ある種のジレンマに陥っている状況である。このような状況を踏まえ

て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では、ロボットシステムの開発に適した OSS 利用時の品質保証に関するガイドラインを提示する必要があると考えた。

## 1.2. 対象読者

本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 や OpenRTM 等の OSS の利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・システムのプロジェクト管理者
- ・ ロボット製品・システムの品質保証担当者

### 1.3. 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
オブザーバ	会津大学 (株)アックス	屋代 眞 竹岡 尚三
委員	川崎重工業(株) 川崎重工業(株) (国研)産業技術総合研究所 JREロボティクスステーション(責) (株)セック (株)セック THK(株) THK(株) (株)東芝 (株)東芝 (一財)日本品質保証機構 ネットワンシステムズ(株) パナソニック(株) (株)日立製作所 (株)日立製作所 (株)本田技術研究所 (株)本田技術研究所 (株)安川電機 (株)YOODS (株)YOODS	亀山 篤 中道 大介 安藤 慶昭 大野 誠一郎 中本 啓之 建部 貴隆 三好 崇生 近藤 裕紀 平山 紀之 山本 大介 櫛引 豪 山崎 治郎 岡本 球夫 中村 亮介 吉内 英也 小川 直秀 鳥井 豊隆 平田 亮吉 原田 寛 平泉 一城

(敬称略、五十音順)

## 2. ロボットシステムの開発プロセス

ロボットシステムの開発に限らず、各企業には何らかの開発プロセスが存在し、製品開発の段階ではそのプロセスに従った品質保証を要求されるものと想定する。IPA が定義する共通フレーム 2013 [2]に登場するシステム開発の「V 字モデル」は、開発プロセスをモデル化した典型的な例である。他にも組込みソフトウェアに着目したものであれば、同じく IPA から ESPR (組込みソフトウェア向け開発プロセスガイド) [3]が発行されている。また開発を遂行する組織の成熟度やアセスメントに着目したものであれば、CMMI(Capability Maturity Model Integration) [4]等のプロセスも存在する。

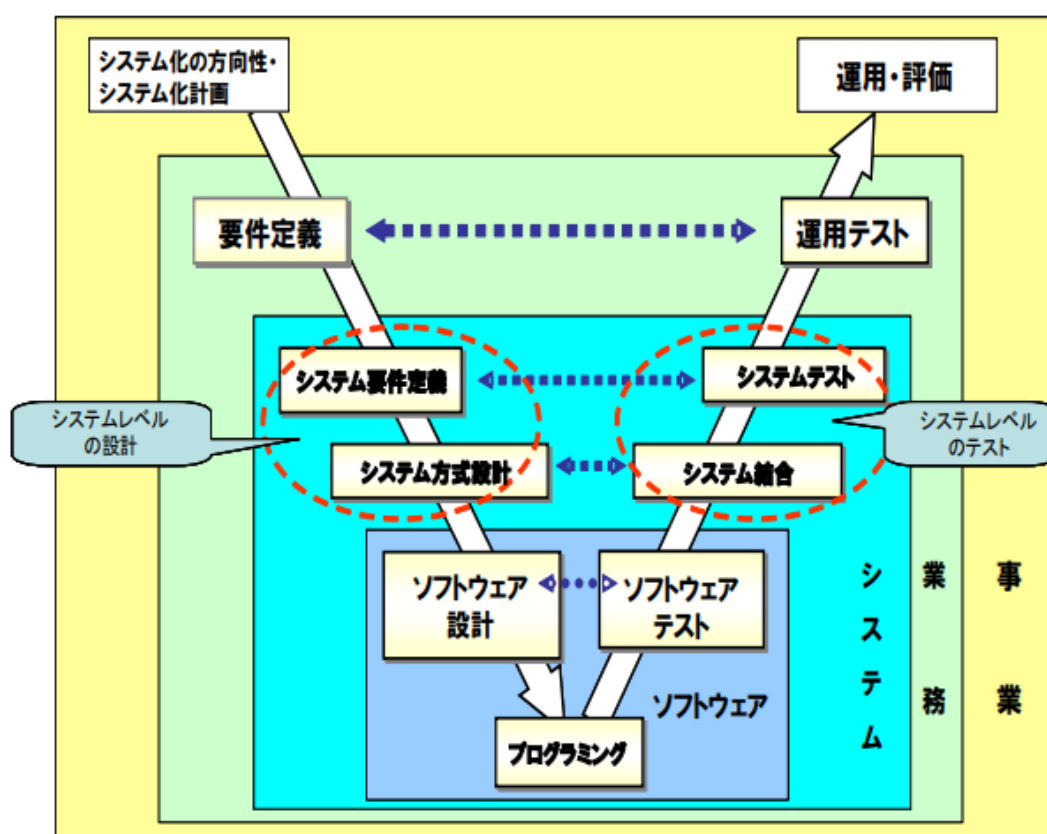


図 1 共通フレーム 2013 の V 字モデル



ただし既存のロボットシステムに全く新しい機能を追加する場合や、今まで稼働実績のない未活用領域への投入を想定する場合に、最初から上に挙げたような開発プロセスに従って開発を始める事は考えにくい。まずハードウェア・ソフトウェア共に幾度かのプロトタイプ開発を経て、システムに対する要求事項を固めながら進めていくはずである。一般に、ソフトウェアに較べるとハードウェアのライフサイクルは長く、仕様が決まってから実際に仕上がるまでに年単位になることも多い。従ってハードウェアができた時には、OS やファームウェアが古くなり、最新のソフトウェアが載らないこともある。そのような事態をなるべく避けるために、予算はかかるが、例えば四半期毎にハードウェア(例えば、検証機、量産先行機、量産機の 3 種)をずらしながら並列で立ち上げることで、できるだけ早くソフトウェアの進化に追いつこうという試みもある。この時、ソフトウェアの開発は、構成管理ツールを最大限に活用しながら、同時に 3 機種へのリリースを繰り返すので、ソフトウェアは 3 つの V 字モデルに跨って開発が進むことになる。そのような開発の例として、ロボットイノベーション WG のロボット安全設計開発調査検討委員会では、自律型生活支援ロボットの安全ガイド [5]の中で「スリーフェーズ開発プロセス」を提唱している。これはロボットシステムのユーザと開発者との関係に着目し、両方でコンセプト検証フェーズと現場実証フェーズを複数回に渡って繰り返しながら段階的にシステム仕様を固め、その後で業務実装に移行していく開発モデルである。

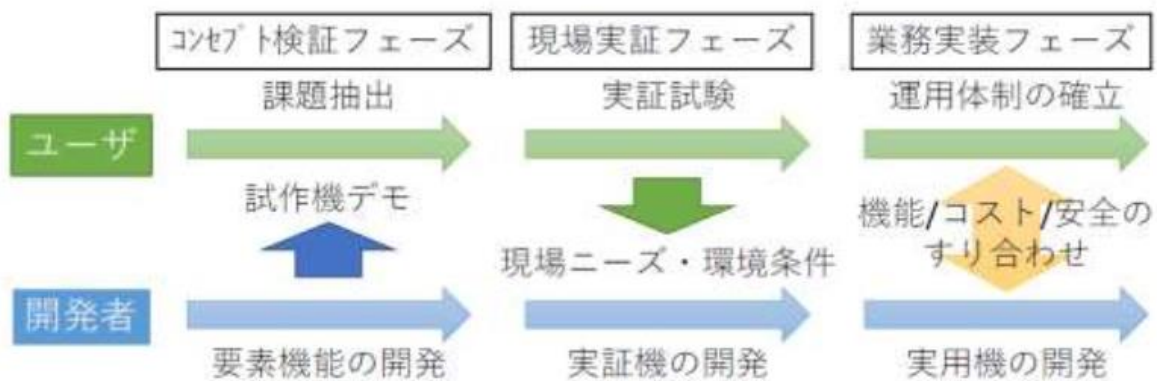


Fig. 2 3 Phase Service Robot Development Process

図 2 スリーフェーズ開発プロセス

このようなロボットシステムの開発において OSS 利用のニーズが生じた場合には、OSS を導入して品質を確保するための活動を、開発プロセス上の適切な時期に実施する必要がある。なぜなら適切な時期に適切な活動が行われていない場合、後の段階で製品開発の大きな遅れを生じるリスクに繋がりがねないからである。例えば研究開発の段階で共同研究先と OSS の使用に関して明確なガイドラインを示していなかった為、意図せずに紛れ込んでしまったものが製品開発時に発覚し、その代替ソフトウェアを急遽開発する羽目になったり、あるいはライセンス自体には問題がなかったものの、実は大きなバグを残したまま、コミュニティが非活性となってしまった OSS をうっかり使ってしまう、致命的な手戻りが生じたりする等の例がある。

本書では OSS を導入する際の活動を、「選定」「プロトタイプ開発」「製品開発」「保守」という 4 段階に分ける。この 4 段階を上で紹介した「スリーフェーズ開発プロセス」および「V 字モデル」を組み合わせると、下図のような位置付けになる。4 章以降で「選定」「プロトタイプ開発」「製品開発」「保守」の各段階で実施すべき活動内容について述べる。

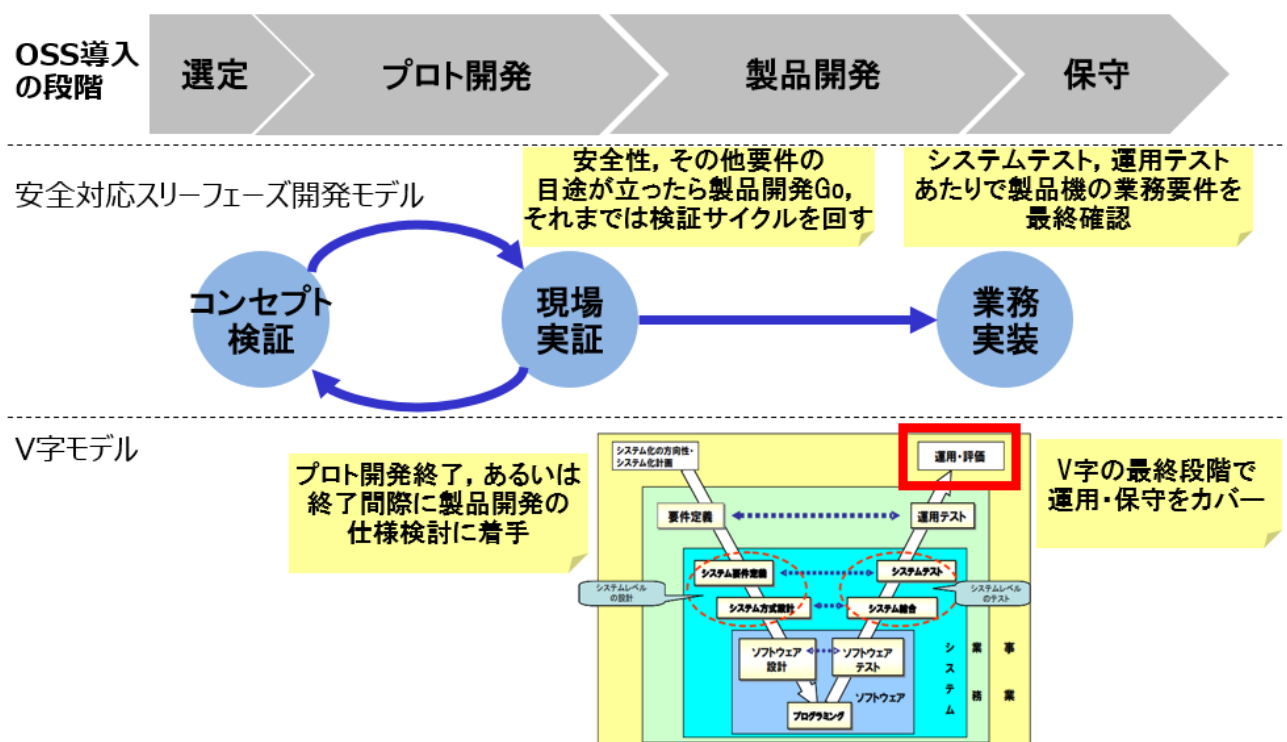


図 3 本書で扱う OSS 導入の段階と開発プロセスの対応



## 3. OSS 利用の目的と課題

### 3.1. OSS 利用の目的

ロボットシステム開発における OSS 利用の目的について説明する。

現状の産業用ロボットを未活用領域で使用するためには以下のような課題があり、その解決が必要になる。

#### ① 高度に智能化されたロボットシステムの実現が必要

未活用領域のうち、主にサービス領域では従来人が行っていた多能工的な作業をロボットに代替させることが望まれている。こうした作業の自動化を従来のティーチプレイバック方式で実現するとなると膨大な時間が必要になる。そのためビジョン、多数のセンサや AI を活用し、ロボットが自律的に作業を実施することが必要となる。

#### ② 共通のプラットフォーム上での機能開発が必要

従来はティーチングペンダント＋ロボット言語等、メーカーごとに異なる手段でロボットシステム構築がされてきた。ロボットメーカーが独自に機能を開発し、機能間の連携はそのロボットメーカーが提供するクローズドな環境の中で実現されてきた。しかし、未活用領域では AI、クラウド、センサ技術をはじめとする多種多様かつ高度な技術が必要になり、一社で実現するのは大変困難である。それぞれの技術に習熟した社員を揃えることは育成など含め多大なコストが必要になる。そのため、企業、研究機関、スタートアップの力を結集し協調すべきところは協調し開発をすすめる必要がある。しかし、それぞれの技術をそれぞれの組織個別の独自システムの上で開発してしまうと、実際に集結させる上でやはり大変なコストがかかることになる。そのため、共通のソフトウェアプラットフォーム上で開発を行い、各機能、技術の相互接続性を確保する必要がある。

この課題の実現手段として OSS の利用がある。

ロボット分野においては、ROS や RT ミドルウェアといった、ソフトウェアをコンポーネント単位で取り扱うソフトウェアプラットフォームがある。その上では、マニピュレーション用のライブラリである MoveIt!、AMR (Autonomous Mobile Robot)用のソフトウェアパッケージ群である NavigationStack、画像処理・点群処理のための OpenCV や PCL (Point Cloud Library)、あるいは AI のための様々なライブラリ等、機能別のソフトウェアパッケージ群が利用可能である。また、開発作業を支援する様々なコマンド群やツール群、Gazebo や Choreonoid, V-REP といったシミュレータ、可視化や操作のための rviz や RTSystemEditor といったツールなどが OSS で提供されている。

こうした OSS は世界中の多くの組織、開発者が参加するコミュニティが存在しており、OSS 利用に対するフィードバック、それに基づく機能通過・拡張が自律分散的に行われ一種のエコシステムを形成している。

自動運転の分野においても、Autoware と呼ばれる ROS をベースとした自動運転用コンポーネント・ツール群がパッケージ化されており、多くの技術者がこうした共通のプラットフォーム上で技術開発と共有を行い、自動運転技術の向上に寄与している。

上記に挙げた機能、ソフトウェアを仮に OSS を利用せずに開発・設計を行うと数年単位の開発期間が必要になる。OSS を利用した場合、仮に十分な評価期間を計画したとしても、それよりはるかに短い時間で製品化を行うことができる。

まとめると、OSS を使用する目的は、ロボットに必要な多種多様な機能を、開発コストを抑え実現することであるといえる。

## 3.2. OSS 利用の課題

OSS を利用することで様々なメリットが得られる一方、企業のロボット製品開発で OSS を利用する際には、解決すべき課題が存在する。本節では、OSS 利用上の課題を示し、以降の章で、各課題に対する対策を講じる段階、具体的な対策を説明する。

- i. OSS の制約による OSS 選定時の問題
- ii. 品質が保証されない
- iii. サポートが保証されない
- iv. 法的リスクが存在する

### 3.2.1. OSS の制約による OSS 選定時の問題

「OSS の利用の目的」で記載した通り、OSS を使用する目的は、ロボットに必要な多種多様な機能を、開発コストを抑え実現することである。OSS を選定する際には、実現すべきシステム要件が決まっており、その機能の一部を、既の実現済みの OSS を利用することで、開発期間、開発コストを短縮することを目的としている。しかし、システム要件に合致する OSS を選定するのが困難な場合がある。システム要件は、開発する製品のターゲットにより決まってくるものであるが、OSS の開発方針は開発者の事情により決定されるため、開発するロボットシステムにすぐに適用できる状態になっているとは限らず、利用者にとって導入の障害となるような制約を OSS が有していることも多い。以下に例を示す。

- ・ 動作 OS やミドルウェア、利用しているライブラリなど、依存ソフトウェアのバージョンが固定的である

- ・ ミドルウェアを利用している OSS で、アルゴリズム部分が分離されていない

これらの制約により、開発するロボットシステムに組み込む際に発生しうる問題の例を以下に示す。

- ・ 周辺システムとの連携のため、RT ミドルウェアベースのソフトウェアにしたいが、利用したい OSS が ROS を前提としている。
- ・ Python3 で統一したいが、利用したい OSS は Python2 で作られている。
- ・ 依存ソフトウェアのセキュリティアップデートが行われているが、バージョンに依存したコードがあり、アップデートを行うのが厳しい。

システム開発においては、通常、利用する OSS の制約の影響によってシステムの仕様を変更することはできない。そのため、OSS の利用上の制約により、システムに適用できない場合には、異なる OSS を探す、もしくは、自社開発をして当該 OSS の利用を停止するか、システム要件を満たすように OSS を改変する必要がある。

### 3.2.2. 品質が保証されない

ロボットを製品として公開、販売するうえで、開発企業はロボットを制御するソフトウェアの品質を保証する必要がある。ここでの「品質」とは、「定義した機能要件、非機能要件に適合する」とこととする。

ソフトウェアの品質保証に必要なのは、ソフトウェアの開発過程で品質が作り込まれていることを監視すること、および、完成したソフトウェアに対して品質を確認することである。「2 ロボットシステムの開発プロセス」において例として挙げた既存の開発プロセスにおいては、品質確保のためのプロセスが定義されている。製品化を見越したシステムは、こうした開発プロセスに則って開発されるが、OSS の開発プロセスに規定はなく、品質確保の記録が残されていない、もしくは、実施されて

いない場合もあり、品質を客観的に判断することができない点が利用するうえでの課題となる。本項では、特に以下の点について記載する。

- (1) 要件(機能、非機能)が明確になっていない
- (2) 開発プロセスに沿って品質を確認された記録がない
- (3) 不具合が存在する

#### (1) 要件(機能、非機能)が明確になっていない

前述した通り、ソフトウェアの品質は、「定義した機能要件、非機能要件に適合する」ことを示す指標であるが、OSS には、そもそも機能要件、非機能要件が明らかになっていないものが多い。例えば、ロボット用のミドルウェアである「ROS」のパッケージは、外部設計、内部設計、マニュアル等が不足していることを、2018 年度の RRI 調査検討委員会で調査した。また、ソフトウェアを利用するためには、ソフトウェアの非機能要件についても明らかになっている必要がある。論文等、詳細が記載されている資料を公開しているソフトウェアもあるが、“精度”や“処理速度”に関する資料が不足しており、ソフトウェアを動かして評価してみないと、製品に適用できるか判断できないソフトウェアは多い。

#### (2) 開発プロセスに沿って品質を確認された記録がない

ソフトウェアを自社開発する場合、通常、以下に示すような品質を向上するための活動を実施する。

- i. 設計レビュー、設計審査
- ii. コードレビュー、ツールを利用したコード解析
- iii. テスト(単体テスト、結合テスト、適格性確認)



共通フレーム 2013 では、「検証プロセス」において、作業結果が仕様(要求事項)を適切に反映しているかを確認する。また、「共同レビュープロセス」において、「合意目標に対する進捗、及び利害関係者を満足させる製品の開発を確実にするために、利害関係者と共通理解を維持する目的として、効果的なレビュー」を行うこととしている。商用のソフトウェアでは品質を保証する主体が存在するため、これらの記録を参照できなくても、一定のレベルで行われているものと考えられる。特に安全性が求められるようなソフトウェアでは、機能安全などの認証を取得していれば、これらのプロセスを行っていることが保証される。一方、OSS には品質保証をする主体がなく、品質確保のための活動が行われたかどうかを確認した記録がない、もしくは公開されていないため、品質を客観的に判断することができない。

### (3) 不具合が存在する

「要件が明確でない」「品質を確認した記録がない」という点を説明したが、品質を確保する活動が不足しているため、不具合が存在しているソフトウェアもある。GitHub 等で公開されている OSS では、不具合報告が"Issue"として挙がっているが、Issue が未対応となっているものも多い。また、OSS を静的コード解析ツールにかけると、数多くの指摘が抽出される。静的コード解析ツールでは、例えば変数の初期化漏れといったような直接的な不具合や、将来的に不具合となりうる可能性がある危険なコードを抽出することができる。NEDO のロボット活用型市場化適用技術開発プロジェクトにおいて、Klocwork を利用して、ROS の品質調査を行ったところ、6.6 件/KLOC の潜在不具合の可能性が指摘された。[6]

### 3.2.3. サポートが保証されない

一般的に製品を開発、販売する場合、製品の保証期間内の障害に対しては、瑕疵対応の義務がある。通常、ソフトウェアシステムは、開発するソフトウェア、ソフトウェア製品(サードパーティー製)、OSS らの組み合わせで構築され、インテグレータがシステムの責任を負うことになる。システ

ムに障害が発生した場合、その原因がソフトウェア製品にある場合であれば、ソフトウェア製品の開発元、もしくは販売元がサポート契約の範囲内において、障害に対応する。

一方、多くの OSS では、サポートする義務がないことを条項を含んだライセンス形態での提供となっており、OSS を保証する主体が存在しないため、不具合や改善要望を報告してもすぐに反映されることはほぼない。また、開発自体が停止するリスクもある。そのため、障害が発生した場合には、システムの責任を負う会社が自社で対処する必要があるが生じる。有名な OSS の中には、有償でサポートを提供する会社がついているものもあるが、そのような OSS はごく一部である。開発元のサポートが得られない場合、自社で対応する必要があるが、対応する技術力がなければ、利用を継続することは困難である。

※ ただし、利用した OSS をメンテナンスできる技術力がある場合には、ソースコードは公開されているため、保守可能となる、というのはメリットになる。一方、ソフトウェア製品の場合、サポートが廃止された時点で保守が不可能となってしまうデメリットもある。

### 3.2.4. 法的リスクが存在する

第三者が開発したソフトウェアを利用して製品を開発する場合には、利用するソフトウェアの著作権者が定めるライセンスに従う必要がある。商用のソフトウェアはライセンスが明確であるが、OSS のライセンスの種類は多く、多様な利用条件が定められている。オープンソースを促進することを目的とするオープンソース・イニシアティブ(OSI)が承認しているオープンソースライセンスは、100 種類近く存在する。同種類のライセンスであっても、条項の一部が異なるバージョンが複数あるライセンスも存在するため、注意が必要である。

各ライセンスでは、OSS を利用するソフトウェアのソースコード公開義務、利用する OSS の明示義務、商用利用の可否、といったことを利用条件として定めている。ソースコードの公開義務が発生してしまう条項を含む OSS を利用すると、本来、公開する想定ではなかった、自社で培ってきた技術を公開する必要があるが生じてしまい、技術を流出してしまうリスクが存在する。

また、ライセンスが定める条項に合っていない利用方法を行った場合、訴訟に発展するケースもある。訴訟まで至った場合、企業は多額の賠償金を請求され金銭的な損失を被るうえに、市場の信用を損なうことになり、大きなダメージを負うこととなる。そのような事態を防止するためにも、OSSを利用する際にはライセンスを正しく把握したうえで、OSSの利用判断を行う必要がある。さらに、利用するOSS内部で別のOSSを利用していることもあるため、利用するOSSがライセンスを遵守しているか、といった観点での調査も必要である。

さらに、特許についても注意が必要なケースも存在する。OSSが採用しているアイデアやアルゴリズムが特許を取得されていて、かつ、ライセンスにより、特許の利用を許可されていない場合もある。そのようなOSSを不用意に利用してしまうと、特許侵害になる可能性もあるため、留意が必要である。

ライセンス、特許に関するより詳細な説明は、ロボットソフトウェアライセンス・特許調査検討委員会で策定している「オープンソースを活用したロボット開発のためのライセンス・特許ガイドライン」[7]を参照のこと。

以上で述べたように、ロボットシステムの開発においてOSSを導入するには様々な課題がある。課題と解決方法を要約すると、表1のようになる。

表 1OSS 導入時の課題と対応策

項番	OSS 導入時の課題	対応策
1	OSS の制約による OSS 選定時の問題	OSS ソフトウェア要件の整理
2	品質保証されない	品質に関する情報の収集
3	サポートが保証されない	サポート状況の調査
4	法的リスクが存在する	ライセンス等の確認



## 4. OSS 利用の品質保証活動

### 4.1. 選定段階

本節ではロボットシステム開発における選定段階の概要と、選定段階において OSS を選定する際に考慮すべき項目について説明する。

OSS 選定はシステム開発における企画段階の作業の一つで、目的はシステム開発において必要な機能を自社開発ではなく、他社品、あるいは OSS により実現するための方法を検討すること、使用するソフトウェア、ハードウェアを様々な観点から評価し、決定することである。企業においてシステム開発を行う際には、開発期間の短縮と開発費用の削減は二大課題であり、これを解決するにはすべてのソフトウェア、ハードウェアを自社開発するのではなく、十分な品質が見込め、かつシステム要件を適切に満たす他社品、あるいは OSS を積極的に製品に適用することが必要である。

OSS の導入に対し、開発者が期待する効果を端的に表すと、「自社開発と比してシステム開発が早く、低コストで実現できること」となる。自社開発をせずに OSS を導入する際の最初の作業は、適用可能な OSS に関する大まかな情報を、開発対象のシステム要件と照らし合わせて収集することである。例えばロボットのマニピュレーション制御がロボットの要件に含まれている場合、マニピュレーション制御を実現している OSS をリストアップする。複数の OSS が候補としてリストアップされることも珍しくはない。

次に、導入候補となる OSS に対し、自社製品に導入するという観点で、OSS が商用製品としての利用において問題ないかどうかを検証する。OSS 導入時の課題は 3.2 節にて述べた通りであるが、本節では OSS 導入時の各課題に対して、選定段階で留意すべき点について説明する。

## 1. OSS ソフトウェア要件の整理

OSS の利用にあたっては、OSS の動作環境や依存ソフトウェアがシステム開発に影響を与える可能性があるため、これらに関する情報収集を行う。

- 機能要件の合致度

OSS によって実現したい機能が、導入候補の OSS によってどの程度カバーされているかを精査する。OSS 選定時においては、OSS の外部仕様、API 仕様等から、実現したい機能要件のうち何項目がカバーされているかを調べる。

- 対象ハードウェア・OS

OSS の動作環境であるハードウェアや対象 OS を確認する。OS はソフトウェア開発に与える影響が比較的大きいため、動作環境に自社開発において導入を予定している OS が動作環境に含まれているかを確認する。対象の OS が自社が前提としていない OS の場合、OS を変更して他の開発に影響が出ないかどうかを検討する。OSS によっては複数の OS を対象として開発されていることもあるが、OS ごとに開発の成熟度合いが異なる場合があるため、自社が採用予定の OS において、対象の OSS がどの程度安定して動作しているかも確認する必要がある。

- 依存ソフトウェアの動作条件

対象の OSS が、OS が標準では提供しない別のソフトウェアに依存する場合がある。例えば画像処理や通信関係のライブラリを必要とするケースが考えられる。このような場合、OSS が必要とする依存ソフトウェアの一覧と、OSS が動作を保証する、あるいは動作実績のある依存ソフトウェアのバージョンを確認する。依存ソフトウェアに関してはさらに、ソフトウェアのバージョンが指定された特定のバージョンでのみ動作するか、あるいは指定されたバージョン以降の新しいバージョンでも動作するかを調査する。複数の依存 OSS を使用する場合、それらの依存ソフトウェアが互いに干渉しないかどうか（インストール・動作時に衝突しないか）を検証する。

- 依存 OSS のシステム要件との合致度

依存するソフトウェアが OSS である場合、依存 OSS の適用可否を本節で述べる方法にて再帰的に確認する必要がある。依存 OSS の適用可否は最終的には総合的な調査を行って判断するが、依存 OSS として調査すべき項目はライセンスに関する情報であり、商用利用が可能かどうかを最優先で確認する。

- 開発ドキュメントの有無

API 仕様書等により OSS の基本的な使用方法が公開されているかを調査する。API 仕様書は関数の呼び方、引数、返値等に関する情報であるが、API 仕様書だけでは OSS の使用方法を把握することは難しい。そこで、API 仕様書に加えて、OSS の基本的な使い方を伝えるチュートリアルやサンプルコードが存在するかを合わせて確認する。

- 性能要件の達成度

システムが満たすべき性能要件において、OSS が関係する要件のうち、OSS が達成可能な要件の数を確認する。OSS の性能に関する情報が公開されていない場合、達成可能な要件数は OSS 選定時においては 0 とし、プロトタイプ開発にて性能試験を実施する必要がある。

- OSS 改変可否と改変コスト

OSS が満たすべき機能要件、性能要件の数が不十分だったり、あるいは未達の要件の中に開発対象のシステムにおいて必須の要件が含まれたりする場合、OSS の改変を検討する必要がある。OSS 改変可否を判断するとともに、改変が必要な場合、改変にかかるコストの概算から、OSS 自体の適用可否を判断する材料とする。

## 2. 品質に関する情報の収集

OSS の品質・性能に関する情報は非公開、または整備されていないことが多く、十分な量の情報を得ることは簡単ではないが、可能な範囲で下記項目の情報収集を試みる。

- 仕様・設計ドキュメント

基本仕様・詳細仕様に関するドキュメント、あるいはそれに類する情報の有無を確認する。ドキュメントがある場合は内容を精査する。

- ソースコードの静的解析・構造解析

OSS はソースコードが公開されているため、開発言語に合わせた静的解析ツールを用いて、事前にソースコードが内包するバグ、潜在的な問題点やその分量を把握する。また、構造解析を行うことにより、関数やクラスの関係性を図示して、ソースコードの複雑度を確認する。

- テストの実施結果

テストの実施結果の有無を確認する。Jenkins 等の継続的インテグレーション(CI: Continuous Integration)システム等で単体テストが自動化されている場合、テストの実施日時を確認する。テストの結果抽出されたバグの件数などが公開されている場合、ソースコードの規模と合わせてバグ密度を算出し、ソフトウェア品質を判断する一つの指標として用いることができる。また、テストコードが公開されている場合、テスト密度やテストの網羅性を判断することができる

- 実案件での適用例

OSS が適用された他の商用・非商用システムの情報を収集することで、他社の当該 OSS に対する信頼度を知ることができる。また、OSS が適用されたシステムの性能、規模から、当該 OSS の性能に関する参考値を入手できる

- バグフィックスの状況

発生したバグとその対策状況が公開されている場合、既知の問題とその修正状況からソフトウェア品質に関する情報を入手できる

### 3. サポート状況の調査



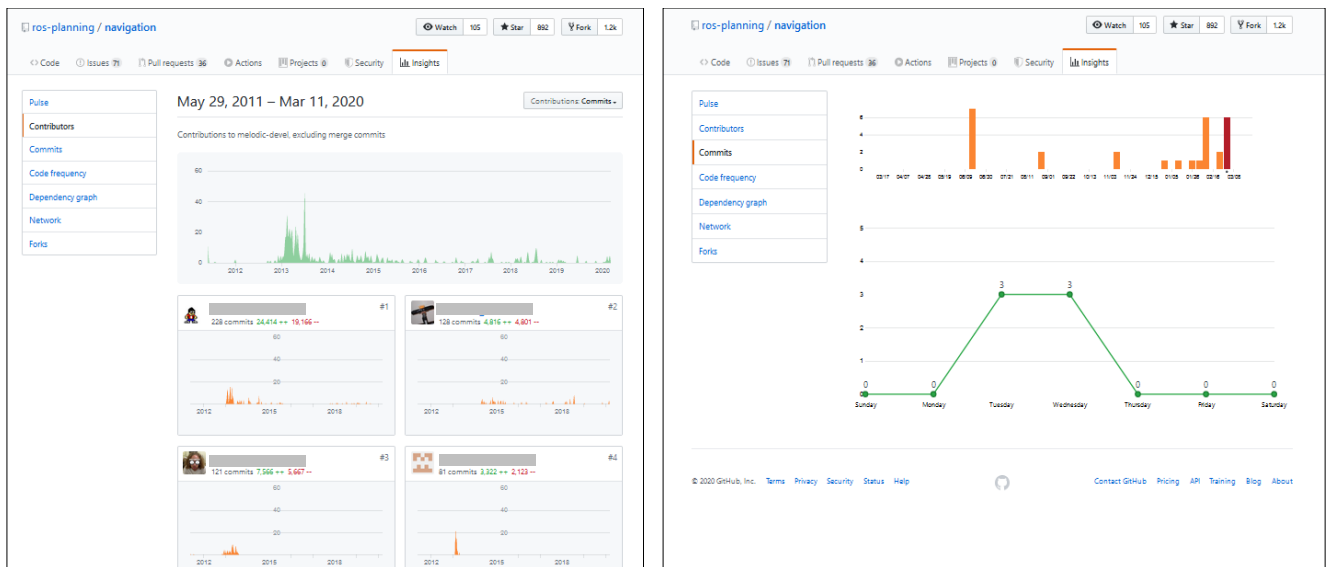
OSS は商用製品と比べてサポート体制が整備されていないことが多いため、サポート状況に関する情報収取を実施する。

- 有償サポート

広く使用されている OSS については有償サポートを提供する会社が存在する場合がある。そのような場合、費用、サポート内容、問い合わせ窓口、対応の早さなどに関する情報を収集する。対応に関しては問い合わせの際に使用する言語（日本語か英語か）、問い合わせ手段（メール、オンコール対応）など、必要としているサポートが迅速に得られるかを重点的に確認する。

- 無償で利用可能なサポート

有償サポートを提供する会社がない場合、サポートは基本的には OSS の開発コミュニティにおけるサポートを利用することになる。OSS 開発コミュニティは有志による運営が基本であるため、対応はコミュニティでの質問が中心となる。このため、質問の方法（メール、質問サイトでの受付）、回答の早さ、回答の質について、コミュニティの Web サイトを中心に調査することになる。有償サイトとは異なり、OSS 開発コミュニティには質問者に対して要求を 100% 満たす回答をする義務はない。このため、開発コミュニティ以外のサポートが得られない場合、自社の開発リソースで自力でサポートをする体制を構築できるかどうかを検討する必要がある。開発コミュニティの活性度を調査することも有効な手段であり、OSS 開発コミュニティが提供しているポータルサイトの状況を調査するほか、GitHub 等のリポジトリ開発管理サイトが提供する活動状況の可視化ツールを用いることも効果的である。



● 図 4 コミュニティ活動状況可視化の例 (GitHub の Activity)

#### ● バグ報告窓口の対応状況

OSS の使用において重大なバグが見つかった場合、バグ対策を OSS 開発コミュニティに依頼する必要がある。この場合、バグ報告用の窓口が整備されているか、バグを報告した場合、どの程度対応が見込めるか、対応が見込める場合、どのくらいの早さで対応がなされるかを可能な範囲で調査する。バグ報告窓口については、例えばバグ修正コードを合わせて報告すると対応が速やかに行われるなどのケースが考えられるため、報告内容に関する調査も有用である。

## 4. ライセンス等の確認

OSS は開発者が使用方法に関するライセンスを規定している。OSS の自社システムへの適用に当たり、ライセンスに関する情報収集を行うことは必要不可欠な作業である。OSS の選定においてはデュアルライセンスの有無にも留意しながら、下記の項目について検討する。

- 商用利用の可否と条件

OSS のライセンス条項から、商用利用の可否を判定する。商用利用が不可の OSS はシステム開発に適用することは基本的にはできない。商用利用が可の場合、利用に関する条件を確認する。例えば商用利用に関してライセンス料が発生するケースや、OSS の適用を Web サイト、あるいは製品のドキュメント等で明示する必要があるケース等が考えられる。

- 改変部分の開示義務

性能や機能面の要件から OSS を改変する必要がある場合、改変部分の取り扱いについてライセンス条項を確認する。OSS のライセンスは OSS の無条件の改変を許容するものから、改変部分の公開を義務づけるものまでさまざまな条件があるため、改変部分の取り扱いが自社の開発方針と合致するかを詳細に検討する。

- ソフトウェアの再配布

自社システムに組み込んだ OSS の再配布に関する条項を確認する。OSS の再配布が許容されない場合、OSS については製品の納入先に独自に入手・インストール等の作業を依頼する必要がある

- 関連特許情報

ライセンス条項のほかに、OSS 開発コミュニティで関連する特許等に関する情報が公開されていないかどうかを確認する。OSS が特許等で保護されているケースは多くはないが、不用意に他社の特許を侵害しないように留意する必要がある。

## 4.2. プロトタイプ開発段階

プロトタイプ開発段階の主な目的は、一般的には仮想的にシステム要件の一部を可視化し、顧客と将来の製品イメージを共有することであり、以下のステップに分割される。

1. プロトタイプのシステム要件を策定し、評価指標を定める。

2. プロトタイプを開発し、1で定めた評価指標に従って、評価する。

3. 2で得られた評価結果に対し、妥当性を検証する。

4. プロトタイプ開発段階の可否を判定する。

短期間でプロトタイプを開発するには、OSS の活用は有効な手段であるため、選定段階にて選定した OSS をベースに不足機能を自社で開発してプロトタイプシステムを構築するケースが多い。OSS を活用したプロトタイプ開発段階において、留意すべきことを以下に述べる。

ステップ1において、プロトタイプシステム内の機能ブロックを明確にした上で、どの機能ブロックでどの OSS を使用するかを明記する。続いて、評価指標と機能ブロックの関係を明記し、評価指標と使用する OSS の関係を明確にしておく。

評価指標の考え方については、ロボットイノベーション WG の移動ロボット評価指標検討委員会でも議論されているが、近年、機械学習アルゴリズムを実装した OSS も登場しており、OSS はますます高度化している。ロボットシステムは従来のティーチングプレーバックのようにユーザが教示した作業を繰り返すタイプから、自律的な動作を実現するタイプへ進化しており、AI プロダクトといっても過言ではない。

AI プロダクトの品質保証については、2018 年 4 月 1 日設立された AI プロダクト品質保証コンソーシアム ( <http://www.qa4ai.jp> ) にてガイドライン [8]策定が進んでいる。品質保証に対する過度の期待を防ぎ、開発側と顧客の双方が安心できることを目的とし、AI プロダクトが抱える、帰納的開発、精度が 100%にならない、性能や不具合の因果関係の解明が困難といった課題に対し、Data Integrity(DI)、Model Robustness(MR)、System Quality(SQ)、Process Agility(PA)、Customer Expectation(CE)の 5 つの軸とバランスで品質保証を評価するとしている。ロボットシステムのすべてが上記ガイドラインで規定されているケースに適用できるわけではないが、考え方を参考にする。

ロボットシステムのプロトタイプ開発における DI/MR/SQ/PA/CE の 5 つの軸を具体的に解釈して、下記に留意して評価指標を定める。

#### DI: ロボットシステムに対する入力データの妥当性

- 入力データに偏りがないか
- 入力データの量は十分か
- 入力データの質は適切か
- 入力データの整合性は取れているか

#### MR: ロボットシステムのモデル

- 適用しようとしているアルゴリズムがロボットシステムに適合するか
- 外乱に対して頑健か
- どのようなデータに対しても処理できるか
- 精度は十分か

#### SQ: ロボットシステムの価値

- ロボットシステムの機能は適切か
- 性能は十分か
- 動作に対して合理的な説明が可能か
- 安全機能は明確になっているか
- 発生する可能性のある事故は許容されるものか
- 事故発生の頻度は十分低い
- 事故発生時の復旧方法は明確になっているか
- 将来的な処理量増加に対してシステム拡張可能か

#### PA: ロボットシステム運用の機敏性

- 顧客からのフィードバックを受けてロボットシステムのモデル変更は容易か
- 継続的なフィードバックを受けることは可能か
- システム更新時に不具合が見つかった場合、ロールバックは可能か
- 類似システムの実績がある場合、経験が反映されているか

CE: ロボットシステムに対する顧客満足度

タスク成功率

動作

異常時の通知方法

見た目の動作に対して顧客は受容可能か

各ステークホルダーはロボットシステムにおいてどのように関与しているか

法規制、コンプライアンスに適合しているか

なお、システム要件および評価指標は顧客と協議して選定するのが望ましい。

ステップ2において、ステップ1で定めたすべての評価指標に対してプロトタイプを使って評価するのが望ましい。費用、期間の観点で全項目の実施が困難な場合は、顧客と協議の上、シミュレーション評価で代替してもよい。

ステップ3において、評価指標に対して目標未達であった場合は原因を明確にする。特に、原因が OSS に起因するか否かを明確にする。

ステップ4において、判定結果が不合格だった場合は、ステップ3において目標未達であった評価指標に対するアクションを検討する。目標未達の原因が OSS にあった場合は OSS 選定段階に戻ってやり直す。改善しても目標達成の見込みがない場合は、目標自体を見直して再評価する。

本ステップにおいても、顧客と協議して合否判定や次のアクションを決定することが望ましい。

## 4.3. 製品開発段階

製品開発段階の目的は、企業が定める品質基準に達していることを確認することである。具体的には、4.2 のプロトタイプ開発段階の残件対応、最終システムでの性能検証、およびライセンスへの対応を実施する。

## プロトタイプ開発段階の残件対応

4.2 のプロトタイプ開発段階で実施した下記の4ステップに残件があれば、対応を完了し、顧客と製品イメージを共有する。

- (1) プロトタイプのシステム要件を策定し、評価指標を定める。
- (2) プロトタイプを開発し、(1)で定めた評価指標に従って、評価する。
- (3) (2)で得られた評価結果に対し、妥当性を検証する。
- (4) プロトタイプ開発段階の可否を判定する。

## 最終システムでの性能検証

ロボットシステムの開発において、OSSを適用可能なソフトウェアとして、MoveIt!、Navigation Stack 等の実績のあるアプリケーション機能のソフトウェアが挙げられる。一方、制御や安全機能に関しては、信頼性、セキュリティ等を考慮すると、OSSを適用するのが難しいため、各社がソフトウェアを内製する必要があると考えられる。適用した OSS が ROS 上で動作する形式の場合、内製したソフトウェアを ROS に対応する必要がある。

OSS を活用するメリットとして、3章で述べた以下が挙げられる。

- ・ 自社専用プログラムを使用する場合と比べて導入コストを削減できる
- ・ 1 社では開発が困難な多種多様かつ高度なソフトウェアを導入できる

しかし、一般的に OSS は企業が定める品質基準に達していないことが多く、特に異常系やテストコードなどで作りこみが必要となることが多い。そのため、企業が OSS を製品に導入するためには、事前に、ユーザへ提供する機能の網羅的な検証が必要である。

OSS の品質や脆弱性に関する技術的な問題が発生する前に対応するため、OSS の開発は下記の Request、Development、および Review の3つのステップにより実施されているので、それらの経過を確認する必要がある。

#### ・Request

ユーザおよび OSS 開発者からバグ報告や新機能の提案などが課題管理システム上に Issue として投稿される。この Issue を OSS へ取り入れるかどうかは、その Issue に対して OSS 開発者がコメントを投稿することで議論し、決定する。OSS へ該当 Issue が取り込まれる場合には、実装の優先順位や実装方法、担当者などについても議論することがある。

#### ・Development

上記 Request における議論結果をもとに実装する。小規模な変更であれば、Request プロセスを経ずに実装されることもある。

#### ・Review

開発者が実装した変更とその変更に対する説明を版管理ホスティングサービスに Pull Request として投稿する。その後、Reviewer がその変更を Review する。Review の結果、変更が承認されなかった場合、開発者は Review コメントに基づいて修正を繰り返す。また、変更が承認された場合、その変更は OSS に反映される。このように Review プロセスを通して開発者の変更を OSS へ反映しても問題無いか議論され、問題なければその変更が OSS へ取り込まれる。



全てのソフトウェアの開発完了後、OSSを含むソフトウェアの正常系と異常系の動作確認を最終システムにて実施し、ソフトウェア構成を確定する。具体的には、以下の単体テスト、結合テスト、全体システムテストを実施する。

#### (1) 単体テスト

全てのモジュールに関して、パラメータの正常系と異常系に関する動作確認を行う。

#### (2) 結合テスト

モジュール間で定義したインタフェースで使用する全てのパラメータの正常系と異常系に関する動作確認を行う。

#### (3) 全体システムテスト

全体システムの動作確認を行うため、ブラックボックステストで正常系と異常系に関する動作確認を行う。

その後、確定したソフトウェア構成の外部設計、内部設計、マニュアル等を準備する。なお、OSSに関してはドキュメントが不足しているため、可能な範囲でそれらを準備する。

### ライセンスへの対応

製品出荷前には、4.1 の選定段階で述べた商用利用の可否と条件、改変部分の開示義務、ソフトウェアの再配布、および関連特許情報を十分に確認した後、下記の構成管理記録の作成、ライセンスの要求の確認、およびライセンス・著作権表記等の準備が必要である。

#### (1) 構成管理記録の作成

開発したソフトウェアの中でどのような OSS がどのように使われているかの情報を構成管理記録に記述する。構成管理記録に書かれた OSS の利用状況が正確であることを確かめるために、この

タイミングでソースコードスキャンツールを積極的に使用し、OSS の正確なリストを作る。そのリストには、各 OSS がどのようなライセンスで利用許諾されたものかも正確に記述する。

## (2) ライセンスの要求の確認

(1) で作成したリストをもとに、それぞれのライセンスが何を求めているか確認する。法務あるいは知的財産権の専門家の助けが得られるのであれば支援を求める。

## (3) ライセンス・著作権表記等の準備

ライセンス・著作権表記等に関して表記方法を決定し、必要な確認を実施する。

また、OSS の内部に特許権を取得しているアルゴリズムを使用している場合があり、特許権侵害の確認も必要である。これらの確認、準備完了後、開発が完了した製品に関して、品質保証部門からの出荷許可を得る必要がある。

## 4.4. 保守段階

保守段階の目的は、製品ライフサイクルに見合ったソフトウェア保守が、規定された品質(4.3 章までの活動)を満たしながら、持続的に実施されること、とする。

保守段階において OSS が自社開発のソフトウェアと大きく異なるのは、製品のリリース以降もコミュニティによる更新が継続される点である。リリース後のロボットシステムにおいては、利用している OSS を頻繁に更新することは考えにくい、致命的な不具合の修正やセキュリティパッチなどで、OSS を選択的に更新する必要性が考えられる。従って自社開発のソフトウェアに対する保守と並行して、製品に搭載した OSS の保守をどのような方針で行っていくかという戦略が必要である。例として、以下のようなものが挙げられる。

- ・ 自社で利用する OSS と元の OSS の差分をできるだけ少なくする
- ・ OSS に対する継続的な品質評価の手段を確保する
- ・ OSS のコミュニティとの連携手段を確立する

- 自社で利用する OSS と元の OSS の差分をできるだけ少なくする

製品のリリース後にコミュニティによって加えられる変更点を自社のシステムに再度取り込む場合、元の OSS と自社で利用している OSS との間に実装面で大きな乖離が生じていると、取り込みの際の作業コストや取り込みによるデグレード発生のリスクが大きくなる。従って製品開発段階までに自社が加えた変更で、汎用的なものはなるべく元の OSS にフィードバックしておくことが望ましい。また同じ理由で、OSS の最新バージョンと自社で利用しているバージョンが乖離し過ぎることも望ましくないため、OSS のコミュニティが公開しているロードマップ等を参照しながら、OSS のバージョンアップを予め計画しておくことも有用である。

- OSS に対する継続的な品質評価の手段を確保する

製品のリリース後に OSS を更新するかどうかの判断材料として、製品開発段階で利用した時点と比較して OSS の品質にどのような変化が生じているのかを確認する事は重要である。OSS のコミュニティで CI 等による品質評価結果が公開されているのであれば、その結果を保守段階でも参照できるようにしておくが良い。コミュニティでそのような結果が公開されていない場合や、自社のシステムに導入した後で品質評価結果が異なる場合は、任意のタイミングで品質評価結果を参照できるように、自社で OSS に対する品質評価環境を構築しておく事も有用である。

- OSS のコミュニティとの連携手段を確立する

製品のリリース後に緊急の問い合わせや不具合への対応が求められる場合もある。一般的にコミュニティによる OSS の保守に緊急性の高い対応を求めることはできないが、連携の仕方によってはコミュニティに参加する多くの技術者の協力を得ることで、自社だけで対応するよりも迅速な問題解決に至る場合もある。そのためには保守段階において、コミュニティの参加者と良好な関係を構築しておく、コミュニティ内の議論や問題解決のフローを理解しておく、といった活動も重要である。自社でそのためのリソースを割り当てられない場合は、OSS のコミュニティに参加しながら保守を行う事業者を見つけておくのも有効である。OSS のコミュニティとの連携手段の例としては、付録の OSS を使用している時の保守の例を参照のこと。

また製品開発段階までに利用した OSS とは別に、自社開発したソフトウェアを新たな OSS として公開する戦略も考えられる。ソフトウェアを OSS として公開する際のメリットは品質や保守だけの面に限らないが、公開する OSS のユーザが増えてコミュニティが形成されれば、自社だけでなくコミュニティによる保守を期待できる点がある。自社開発したソフトウェアを OSS として公開する際の例としては、付録の自社で公開する OSS 保守の例を参照のこと。

## 5. 終わりに

本書では、ロボットシステムの開発に関わる企業の現場で参照されることを念頭に、ロボットシステムの開発プロセスに OSS を導入する際に必要な品質保証活動を紹介した。

2 章で述べたようにロボットシステムに対する要求が多様化する中で、開発現場においてはコンセプト検証と現場実証を繰り返しながらシステム要求を明確化する活動が行われており、システム要求を確定して業務実装に向けた開発を始めるよりかなり前から、ロボットシステムの開発は始まっている。その中で誰もが自由に入手して使える OSS の価値が高まっているが、製品化までの道のりを考えると、3 章に述べたような課題が多いのも事実である。そこで 4 章で述べたように、OSS の選定段階とプロトタイプ開発段階で、自社の利用目的に合致するかどうかをしっかりと検証しておく活動が重要で、後の段階で生じるリスクを低減しておく必要がある。

開発プロセス・品質管理調査検討委員会では、委員の経験を持ち寄って OSS の品質保証のあるべき姿を議論しながら、本書の執筆を分担して進めた。執筆にあたっては全章を通じて統一的な基準値を示すことよりも、各委員の経験値を盛り込むことを重視した。このため、「本書に書かれている基準さえ守れば品質を担保できる」というガイドラインにはなっていないが、各章を読んで頂く中で、自社の品質保証活動に反映できるヒントを得て頂ければ幸いである。また必要に応じて参考文献にある他の調査検討委員会の成果物も参照して頂きたい。

## 6. 参考文献

- [1] IPA, “OSS ライセンス遵守活動のソフトウェアライフサイクルプロセスへの組み込み,” [オンライン]. Available: <https://www.ipa.go.jp/files/000029079.pdf>.
- [2] IPA, “共通フレーム2013の概説,” [オンライン]. Available: <https://www.ipa.go.jp/files/000027415.pdf>.
- [3] IPA, “ESPR Ver.2.0,” [オンライン]. Available: <https://www.ipa.go.jp/files/000005126.pdf>.
- [4] Carnegie Mellon University / Software Engineering Institute, “CMMI for Development, Version 1.3,” [オンライン]. Available: [https://resources.sei.cmu.edu/asset\\_files/TechnicalReport/2010\\_005\\_001\\_15287.pdf](https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf).
- [5] 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 ロボット安全設計開発調査検討委員会, “自律型生活支援ロボット安全ガイド,” [オンライン]. Available: <https://www.jmfrri.gr.jp/followup/1121.html>.
- [6] 安藤 慶昭、琴坂 信哉、岡田 慧、増田 昌庸, “ロボット活用型市場化適用技術開発プロジェクト— ロボットのプラットフォーム化を目指す研究開発(第3報)—,” 計測自動制御学会 システムインテグレーション部門 講演会 2019 (SI2019), 香川県高松市, 2019.
- [7] 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 ロボットソフトウェアライセンス・特許調査検討委員会, “オープンソースを活用したロボット開発のためのライセンス・特許ガイド

ライン,” [オンライン]. Available: <https://www.jmfrri.gr.jp/followup/1132/>.

[8] A. プロダクト品質保証コンソーシアム, “AI プロダクト品質保証ガイドライン,” [オンライン]. Available: <http://www.qa4ai.jp/download>.

# 付録

以下では、製品開発で OSS を使用したり、自社で開発したソフトウェアを OSS として公開する場合の例を参考に紹介する。

## OSS を使用している時の保守の例

該当パッケージが公開されている場合はパッケージ管理者に問い合わせる、または Issue に問題をあげ、修正依頼を出す。

また関係する OSS のウォッチをすることで、OSS の不具合が常に監視できる体制をとる。

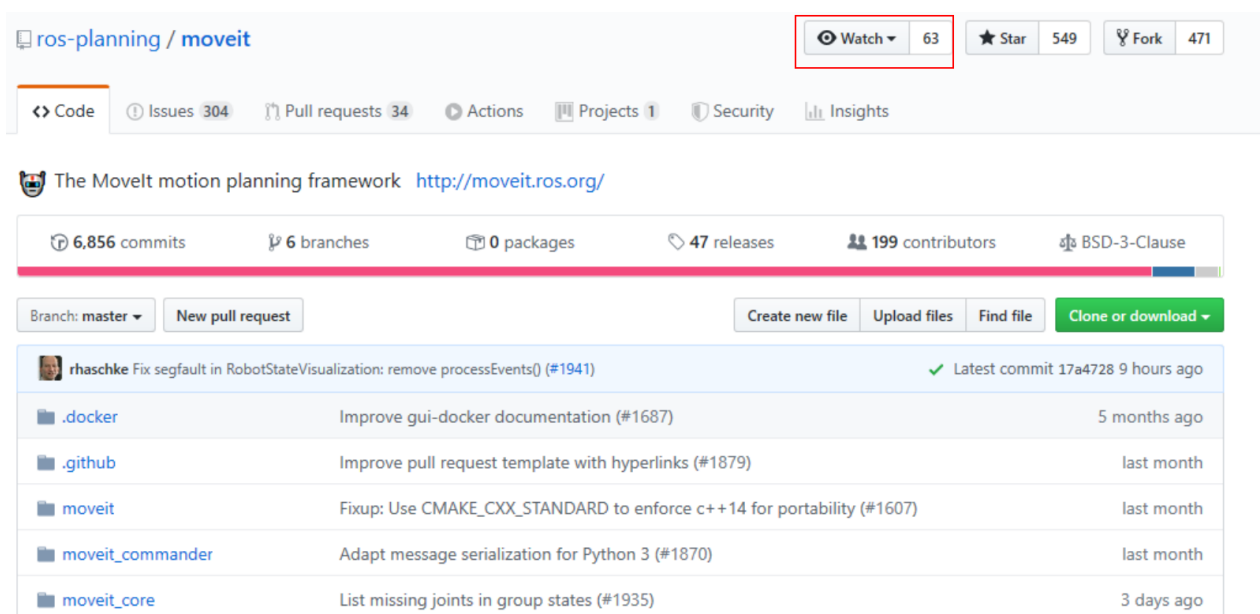


図 5 MoveIt!リポジトリ(<https://github.com/ros-planning/moveit>)

上記赤枠のように GitHub では watch にステータスを変更するとそのリポジトリへの push や Issues の作成、Pull Request の作成などの情報が登録したメールアドレス宛に流れるようになる。通知が多い場合はリリース情報だけ受け取ることも可能。



ros-planning / moveit		Watch 63	Star 549	Fork 471
<> Code		Issues 304	Pull requests 34	Actions
Projects 1		Security	Insights	
Filters	is:issue is:open		Labels 32	Milestones 1
New issue				
304 Open 454 Closed		Author	Label	Projects
		Milestones	Assignee	Sort
Goal state in Rviz should not be set to "current" if execution fails bug		#1939	opened 2 days ago by felixvd	1
To use collision_detection_bullet in kinetic branch bug		#1938	opened 2 days ago by Hubert51	5
distinguish ros_control controllers from Controller interfaces for moveit_simple_controller_manager bug		#1937	opened 3 days ago by v4hn	
Implementation of parameter "TranslationXY2D" IKFast bug		#1931	opened 4 days ago by edetleion	4
Moving MoveIt Msgs into ros-planning/MoveIt enhancement		#1928	opened 4 days ago by mlautman	4
jog_server continues to execute jog command even when it is not being published bug		#1925	opened 5 days ago by m-talha	
approximate solutions from OMPL should be handled as failure bug		#1920	opened 8 days ago by simonschmeisser	6
Connect Moveit with KUKA enhancement		#1913	opened 13 days ago by CesMak	5
'Unable to locate package ros-melodic-moveit' bug		#1910	opened 17 days ago by xu-ye	3

図 6 MoveIt! Issue 一覧

また上記のように修正依頼を出す前に他ユーザーが同じようなバグ情報を提示していないか確認することで余計な手間を省くことができる可能性がある。

自社で修正した OSS を公開しながら保守を行う場合は、自社のリポジトリに Fork して必要な修正を加える。自社で加えた修正が元の OSS にも有効なものであれば、Fork 元のリポジトリに対して Pull Request を作成して修正をフィードバックする。

robo-marc / ros\_comm  
forked from ros/ros\_comm

Watch 1 Star 0 Fork 659

Code Issues 3 Pull requests 1 Actions Projects 0 Security 0 Insights

Filters is:issue is:closed Labels 9 Milestones 0 New issue

Clear current search query, filters, and sorts

3 Open ✓ 17 Closed	Author	Label	Projects	Milestones	Assignee	Sort
⚠ travisでのcatkin_make run_testsでのエラー #27 by sec-matsunaga was closed on Feb 4						2
⚠ travisの設定ファイルを追加する。 #25 by sec-matsunaga was closed on Jan 30						5
⚠ 【産総研】【要対応】到達不能コード(xmlrpc_manager.cpp) #16 by sec-matsunaga was closed on Jan 30						2
⚠ 【産総研】【要対応】到達不能コード(rosout_appender.cpp) #15 by sec-matsunaga was closed on Jan 30						2
⚠ 【産総研】【対応対象外】リソース処理の指摘(transport_tcp.cpp) #14 by sec-matsunaga was closed on Feb 4						2
⚠ 【産総研】【対応対象外】リソース処理の指摘(internal_timer_manager.cpp) #13 by sec-matsunaga was closed on Feb 4						2

図 7 ロボット活用型市場化適用技術開発プロジェクトの品質改善活動による修正  
([https://github.com/robo-marc/ros\\_comm/issues?q=is%3Aissue+is%3Aclosed](https://github.com/robo-marc/ros_comm/issues?q=is%3Aissue+is%3Aclosed))

## 自社で公開する OSS 保守の例

ここではリリース版を Stable に相当するもの、メジャーリリースはされていないが最新版を含むものを開発版として各項目における作業内容を具体的に述べる。

### (1) リリース版での作業

安定版として実機動作検証でのテストを実施した状況で公開する。

構成環境としてユーザ側にこちらで指定したバージョンで使用するよう事前通知する。

例 : Ubuntu16.04 + ROS Kinetic に固定

依存ライブラリ(MoveIt!+ Navigation)も実機テストにて Pass していることを前提としている。

[https://github.com/seed-solutions/seed\\_smartactuator\\_sdk-release](https://github.com/seed-solutions/seed_smartactuator_sdk-release)

## ・テスト項目

Travis CI または Jenkins による結合テスト

- ① 作った launch ファイルに必要なファイルやパッケージがインストールされているかチェック
- ② サンプルプログラムをインクルードして、必要な topic がでているか確認

・公開手順(RRI ロボット次世代実装手法検討委員会 岡田先生コメント一部抜粋)

bloom を使用し debian パッケージとしてリリースする

手順は <http://wiki.ros.org/ja/bloom> に順ずる

- ① リリースするバージョンが CI に pass することを確認
- ② catkin\_generate\_changelog により commit 履歴を反映させる
- ③ catkin\_prepare\_release によりバージョン情報を更新
- ④ bloom によりリリース情報を rosdistro に PR する
- ⑤ ROS 管理者による merge 後、roswiki にパッケージ・バージョン情報が公開される。

[http://wiki.ros.org/seed\\_smartactuator\\_sdk](http://wiki.ros.org/seed_smartactuator_sdk)

- ⑥ shadow 公開

ROS 運営側が定期的に shadow リポジトリを正式リリースする。

以降、apt-get install でとれるようになる

<https://discourse.ros.org/tags/release>

- ⑦ アナウンス

ROS Discourse で全体アナウンスされる

## (2) 開発版(Develop 版)での作業

GitHub にソースコードを公開し、1 パッケージにつき管理者を1人つける。

GitHub にて Issue またはメールにて不具合情報やバグの報告を随時受け付ける。

Issue またはメールにて上がってきた報告を深刻度、重大さによって分けし致命的なバグであれば早急な対応が必要となる。

仮に利用している依存 OSS パッケージに問題がある場合は、保守を担当している組織またはメンテナへの報告、改善依頼を行う。

潜在的なバグが発見された場合はいつまでに改善するか(次期バージョンアップデートで対応するか長期的な調査が必要となる場合はその旨を Issue に書き込むなど)の対応をする。

報告があった場合、不具合のあったユーザ環境にできる限り近い環境を構築し検証する。

不具合修正はソフトウェアテスト(CI)に通じ次第リリース版にマージする。



図 8 Jenkins Buildfarm 画面

seed\_r7\_ros\_pkg

seed-noid meta package

Continuous Integration Status

service	Kinetic	Melodic
Travis	build passing	build passing

How to install

1. From Debian

```
sudo apt-get update
sudo apt-get install ros-{distro}-seed-r7-ros-pkg
```

2. From Source

In your catkin\_ws/src,

```
sudo apt-get install ros-{distro}-seed-smartactuator-sdk
source /opt/ros/{distro}/setup.bash
git clone https://github.com/seed-solutions/seed_r7_ros_pkg.git
catkin build seed_r7_ros_pkg
source {your-workspace}/devel/setup.bash
```

How to run

Bring up robot\_control(with real robot and Simulation on rviz)

```
roslaunch seed_r7_bringup seed_r7_bringup.launch
```

Without real robot (Simulation on rviz)

図 9 公開 GitHub 画面

### (3) 不具合があった場合の作業

ソースコードの修正を行い、Pull Request したものを社内で検証する。モジュール単位のテストは CI(Travis)テストを実施

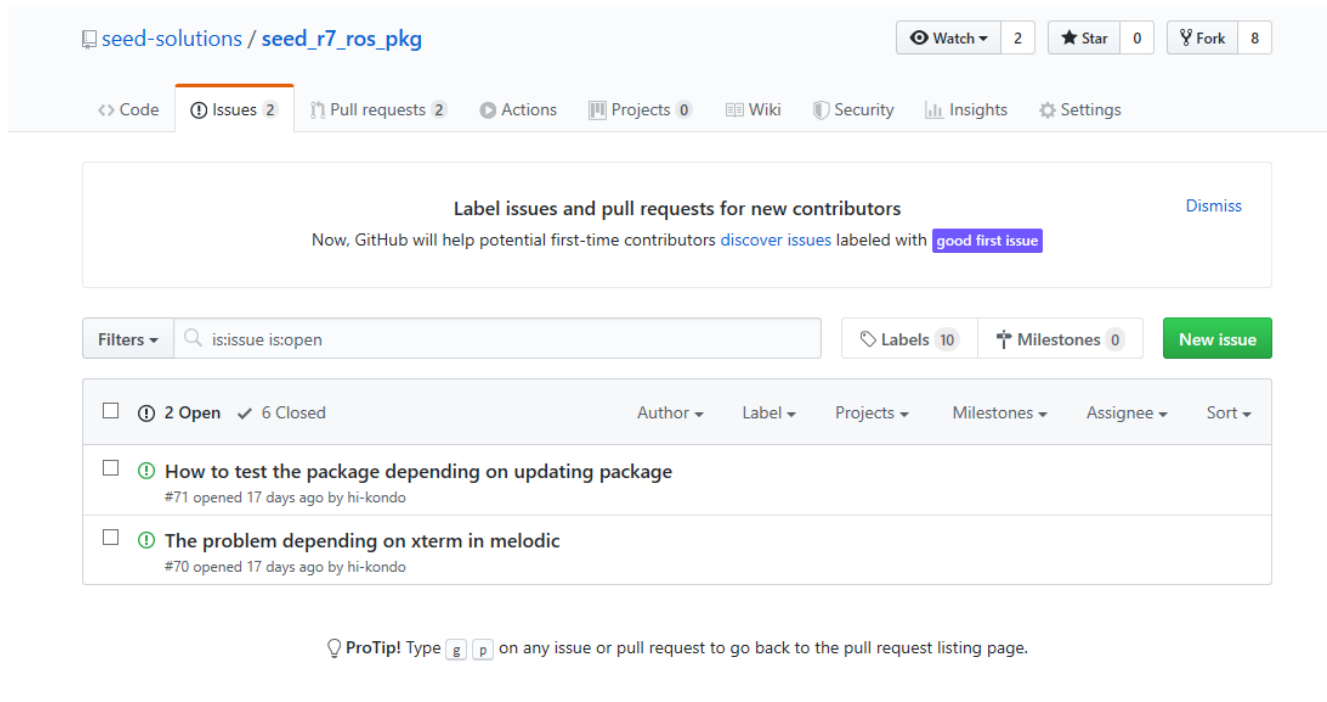


図 10 不具合対応例

## OSS に対するセキュリティ対策の例

1つのプロジェクトに対し、セキュリティ管理者を置くことが理想的ではあるが全てのOSSの脆弱性を監視することは非常に困難であるが、静的解析ツールを用いることでバッファオーバーフロー、未初期化値の仕様、解法済みリソースの仕様、ゼロ除算、デッドロック、リソースリーク等のセキュリティホールにつながりそうな問題をある程度洗い出すことができる。

このようなOSSのセキュリティホールに対応するため下記のようなサービスが提供されている。

(1) OSS 情報提供サービス(日立)

<https://www.hitachi.co.jp/products/it/appsvdiv/service/oss-information-supply/index.html>

## (2) GitHub Security Lab(GitHub)

オープンソースコードの脆弱性を見つけるための [CodeQL](#) を無償提供。CodeQL は、世界中の多くのセキュリティ研究チームが、コードのセマンティック解析に使用しているツール。

<https://github.blog/jp/2019-11-18-announcing-github-security-lab-securing-the-worlds-code-together/>

## (3) CoverityScan(Synopsys)

OSS を含めて脆弱性をチェックし管理できるツール、OSS に対しては無償利用可能

<https://www.synopsys.com/ja-jp/software-integrity/security-testing/static-analysis-sast.html>

以上



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