B-003

オープンソースソフトウェア開発における コミュニティ活動の課題分析とその解決方法の提案
Analysis of issues on the community activities for open source software development and a proposal of their solutions

夏目 貴史† 市原 裕史† 市川 俊一† 坂井 博†
Takashi Natsume Hirofumi Ichihara Toshikazu Ichikawa Hiroshi Sakai

1. はじめに

1.1 オープンソースソフトウェアの普及と浸透

価値観の多様化と製品の細分化が進む時代を背景に，市場のニーズを重視した製品の開発と提供が重要になってきている．市場の変化に対応できるスピーディで効率的な製品の開発が求められており，そうした中でソフトウェア開発においてオープンソースソフトウェア（以下「OSS」と呼ぶ）を活用する動きが拡大している．多くの企業が OSS を導入しており，例えば，ある携帯電話のソフトウェアの 80%は OSS がそのまま活用され，残りの 20%はその製品を作るために開発されたソフトウェアである，という事例がある[1]．ソフトウェア開発において OSS を組み込んで活用することは極めて一般的となっている．

オープンソースというと，一般的にはソースコードがオープンなライセンスで公開されていることを意味する．そのライセンスのもとで，誰でも公開されているコードを評価したり，改造を加えたものを別の製品として扱ったり配布したりすることが可能となる．また，OSS を評価するうえでもう一つ重要な側面は，その OSS に対するコミュニティの存在である．誰でも参加することができ，設計が公の場で議論によって決められ，開発のプロセスが公開されており，参加者が協調して作業を行うオープンなコミュニティは，ソフトウェアに様々な作り手と使い手からのアイデアと提案によって改善される機会を与える．また，様々な作り手と使い手が参加することで，そこに新たな市場が生まれ，規模と価値が拡大することが期待できる．

1.2 OSS 活用とコミュニティ参加の利点

筆者らはビジネスパートナと共に新しい世界を切り拓いていくことを目標に，外部とのオープンイノベーションを推進するための研究開発に取り組んでいる．具体的には，グローバル・クラウドサービスを提供するために OpenStack[2]というオープンコミュニティによって支えられた IaaS（Infrastructure as a Service）を構築するための OSS を活用する形で，キャリアの強みを活かせるクラウドサービスの基盤技術の研究開発に取り組んでいる．

筆者らにとって，OpenStack を活用する利点は以下のとおりである．

- 効果的な投資

クラウドサービスの市場の共通的なニーズがある部分の開発について全てを自前で抱え込むことなく，自らのビジネスで重要性が高い部分にリソースを集中することができる．

- ビジネスパートナとのコラボレーションの可能性の拡大

共通的な部分がオープンとなっていることで，ビジネスパートナとのコラボレーションの可能性が拡大し，その実現も容易となる．

- 維持コストの低減

オープンなコミュニティに参加し，ソフトウェアの開発に貢献していくことで，自前で抱え込む部分をできるだけ小さくし，それを維持していくためのコストを下げることができる．特に市場のニーズや技術の変化が大きい昨今において，それに対応していくコストを下げ，スピーディに追従できることは重要である．

本稿では，筆者らが OpenStack を活用したプロダクトの研究・開発において，オープンなコミュニティに参加して貢献していく過程で経験した試行錯誤について振り返り，どのようにすればコミュニティとのやり取りを円滑に進め，効果的な貢献を行い，ソフトウェア製品の研究開発を進めることができるかについて考察を行う．

2. コミュニティにおける開発

OpenStack のような OSS コミュニティでは，多種多様なバックグラウンドを持った開発者が協力して，1 つのソフトウェアを作りあげていかなければならない．特に世界中に開発者や利用者がいる OpenStack のような巨大なコミュニティでは，時差の問題やどのようにオープン性を確保するのか，プロジェクトの方向性に関する開発者や利用者の合意の取り方等，開発に関する様々な問題が存在する．OpenStack コミュニティでは，開発を円滑に進めるために様々な取り決めやツールを用いて開発者同士や開発者と利用者を結びつけて，これらの問題を解決している．本章では OpenStack コミュニティでの開発の進め方について説明する．

2.1 開発ツール

OpenStack コミュニティで主に用いられている開発ツールは，コミュニケーションツールと開発補助ツールの大きく 2 つの種類に分けることができる．

A) コミュニケーションツール

開発者や利用者がオンライン上でコミュニケーションを取る手段である．OpenStack コミュニティでは主に以下のツールが用いられている．

- インターネットリレーチャット（IRC）

チャット形式でリアルタイムにコミュニケーションを取ることが可能なため，即時性が強く，毎週行われるプロジェクトごとのミーティングにも用いられている．しかし，時差の問題により決まった人としかやり取りが出

†日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ，NTT Software Innovation Center, NTT Corporation

来ないという問題や議論の決定事項を整理しにくいといった問題がある．

- メーリングリスト（ML）

コミュニティの意見を広く求める際には ML が用いられる．ML は OpenStack コミュニティでも最も活発に議論が行われるコミュニケーションの場であり，開発グループの ML だけでも 1 ヶ月で 2000 件以上のメールのやり取りが行われる．

B) 開発補助ツール

ソースコードの管理，バグや新機能提案の管理など，開発時に開発者を補助するツールである．OpenStack コミュニティでは以下のツールが使用される．

- Launchpad[3]

バグや新機能提案を管理するツールである．Launchpad ではプロジェクトごとにバグの報告と新機能の提案が可能であり，次項の Gerrit[4]と同期して動作し，これらのバグや新機能が承認されて実際にソースコードへ取り込まれるまで継続的に管理が可能となる．また必要に応じて，OpenStack の開発者および利用者は Launchpad 上のコメント機能を用いて他の開発者および利用者と議論ができる仕組みになっている．

- Gerrit

ソースコードのレビューシステムである．Reviewer は Web ブラウザを利用してレビューを行なう．

- Github[5]

OpenStack のソースコード管理，パッチの管理を行う．Gerrit と組み合せて，開発時における利便性を高めている．

2.2 開発メンバー

OpenStack コミュニティの管理するプロジェクトへ直近の 2 リリースに直接貢献した開発者は ATC（Active Technical Contributor）と呼ばれている．また，OpenStack のプロジェクトごとに Blueprint と spec（後述）やパッチをソースコードへ取り込む権限を与えられた者を Core Reviewer（あるいは Core Developer）と呼ぶ．OpenStack コミュニティではプロジェクトごとに Core Reviewer がチームを組んで，そのプロジェクトの方針やどの新機能・パッチを取り込むかなどを決定する．このチーム運営のリーダーとなる存在を PTL（Project Team Lead）と呼び，OpenStack プロジェクト内では大きな権限を持っている．OpenStack コミュニティでは ATC，ATC の中から選ばれる Core Reviewer，Core Reviewer の中から選ばれる PTL というピラミッド構造となった開発メンバーで開発を行っている．

2.3 開発スケジュール

OpenStack は 6 ヶ月ごとに新しいバージョンをリリースする．OpenStack では開発中のソースコードを master バージョンと呼び，リリースされる際のバージョンを stable バージョンと呼ぶ．stable であるとは一般的に OpenStack コミュニティがその安定性を利用者に保証することを意味する．OpenStack の stable バージョンのリリースは最初の Austin バージョンより 11 回行われており，2015 年 6 月現在の最新のバージョンは Kilo となっている．OpenStack では最新のリリースを含め 2 つ前のバージョンまではコミュニティとしてメンテナンスを行っているが，それ以前のバージョンはメンテナンス対象外となる[6].

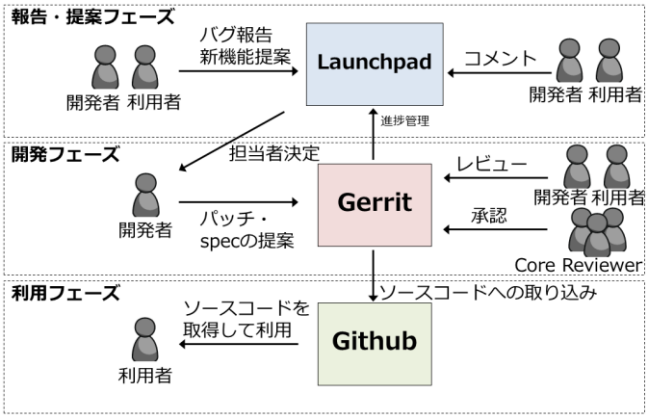

図 1. OpenStack の開発フロー

OpenStack の開発スケジュールの中でも重要な要素としてサミットの存在がある．サミットとは，新バージョンのリリース直後に行われる会合のことで，OpenStack の利用者および開発者が 6 ヶ月に 1 回，世界中から集まり議論を行う場である．このサミット期間中に，新バージョンの機能紹介や開発中の課題の洗い出し，また次のバージョンに向けたマイルストーンの設定が行われる．プロジェクトごとにオフラインで重要な機能や注力すべき機能が話し合われる機会はサミット以外には少ないため，開発者にとっては非常に重要度が高い．

2.4 開発フロー

OpenStack における開発フローはバグ修正と新機能提案で大きく 2 つに分けることができる．フローの全体像は図 1 に示す．

OpenStack でバグ修正が必要となった場合，そのバグの発見者が Launchpad にバグレポートを行う．このバグレポートを開発者が選定し，バグを実際に修正する開発者を選択する．これは他薦自薦のどちらでも可能だが，一般的には自薦となることが多い．次にバグ修正のためのパッチを開発者が作成するが，この際に IRC や ML，また Launchpad のコメント欄等で修正の方針が議論される．作成したパッチは Gerrit 上に投稿され，パッチのレビューが行われる．パッチが Core Reviewer に承認された後，自動化されたテストに合格するとソースコードへの取り込みが行われる．

新機能提案は，提案者が Launchpad 上の Blueprint という項目上で申請を行う．この Blueprint は一般に PTL により管理されており，提案が承認されるには Core Reviewer および PTL の承認が必要となる．この承認を取る際に，多く用いられるのが spec と呼ばれる仕様書である．spec は新機能の機能詳細やソースコードの変更点，テストの必要性，ユーザへの影響などの詳細が記載されている．この spec のレビューを通して，コミュニティで議論を行ない，Blueprint が承認された後は，バグ修正時と同じように新機能のパッチの投稿とレビューが行われ，Core Reviewer に承認された後，自動化されたテストに合格するとソースコードへの取り込みが行われる．

第 3 章で分析を行なう新機能提案においては，Core Reviewer による Blueprint と spec の承認およびパッチ（ソ

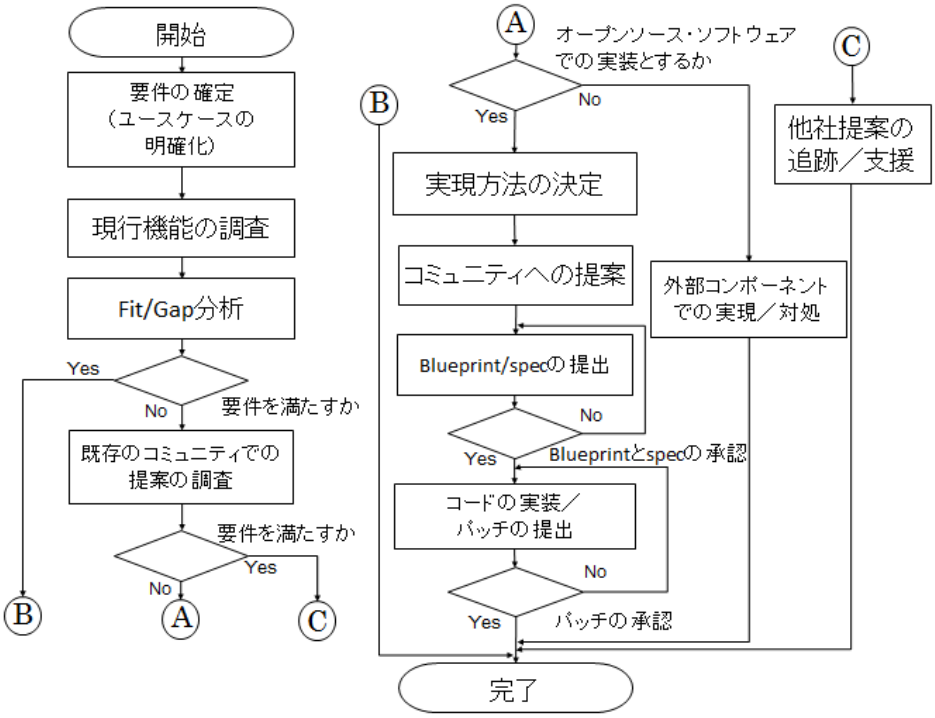

図 2. 新機能提案時の作業フロー

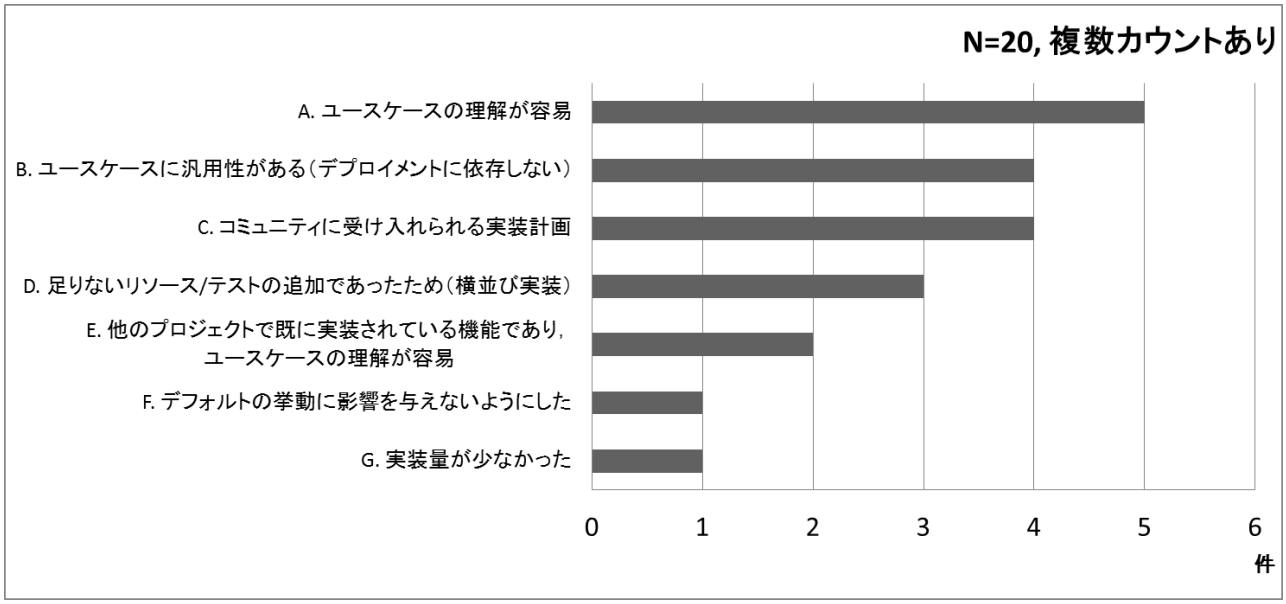

図 3. 成功要因の分析

ースコード）の承認が得られないとコミュニティのソースコードには取り込まれないということになる．そのため，この承認をどのように取り付けるかという点が重要である．

3. コミュニティ活動の分析

3.1 コミュニティ活動

一般的なパッケージ・ソフトウェアを利用したシステムの開発フローおよび 2.4 節で記述したコミュニティでの開発フローを基にした新機能の提案・実装の作業の流れを図 2.に示す．

筆者ら（およびその協力会社）は 2013 年 9 月から 2015 年 5 月の期間に 37 件の Blueprint をコミュニティに提出し，新機能の実装を提案した（一部機能追加を目的としたバグレポートを含む）．そのうち，コミュニティのソースコードに実装を完了させることができたものは 10 件であり，実装を完了させることができなかったもの（活動を保留したものを含む）は 22 件であり，また，現在活動中のものは 5 件である．次節以降において実装完了の成功要因および失敗要因について分析する．

3.2 新機能実装の成功要因の分析

筆者らのチームがコミュニティに提出した Blueprint のうち，実装まで至り，コミュニティのソースコードに取り込まれた Blueprint について，その成功要因を分析した

（図 3.）．（複数の要因が考えられるものは，それぞれ 1 件として計上した．グラフのアルファベットは以下の箇条書きのアルファベットに相当する．N=20．）

A) ユースケースの理解が容易
多くのユーザ，多くの運用者の利用シナリオに当てはまり，理解が容易である．

B) ユースケースに汎用性がある（デプロイメントに依存しない）
特定の環境や特定の製品を使用したケースに当てはまらず，多くのユーザ，多くの運用者の利用シナリオに当てはまる．

C) コミュニティに受け入れられる実装計画

D) 足りないリソース/テストの追加であったため（横並び実装）

E) 他のプロジェクトで既に実装されている機能であり，ユースケースの理解が容易
提案・実装対象のコンポーネント以外のコンポーネントで既に実装されている機能であり，対象コンポーネントへの実装について理解が容易である．

F) デフォルトの挙動に影響を与えないようにした
デフォルトの挙動が変更された場合，ユーザや運用者に影響が出ることが考えられ，理解が得られない場合がある．そのような影響を与えないようにした．

G) 実装量が少なかった
他社の機能実装により機能実現のための実装の量が少なくて済んだ．

これらの要因を分析すると，ユースケースに関する要因（A，B，D，E）と実装に関する要因（C，F，G）に分かれ，この 2 つの要因が成功のために重要な要因であることが分かった．

3.3 新機能実装における課題分析

筆者らが提出した Blueprint のうち，最終的に実装まで至らなかったもの，現在提案活動中で 6 ヶ月以上承認をもらうことができていないものの要因を分析した（図 4.）．（複数の要因が考えられるものは，それぞれ 1 件として計上した．グラフのアルファベットは以下の箇条書きのアルファベットに相当する．N=44．）．それぞれの要因について以下に説明する．

A) 前提となる機能が未実装
筆者らが提案する機能の前提となる機能が他社・他者から提案されており，それが実装されないと筆者らが提案する機能が実装できないため，実装まで至らなかったということである．

B) 過度な品質である
ユースケース的に高品質である必要性が理解されない．

C) API の仕様そのものや変更が決まらないと実装できなかった
API の仕様がコミュニティにおいて議論されており，その結論が出るまで，提案した機能が実装できない．

D) プロジェクトの機運が盛り上がらない
他にプロジェクトで優先度の高い取り組みがある．

E) メリットが見えにくい（リファクタリング）
新規機能を入れたとしても，既存機能のリファクタリングのように見え，改修量の割にメリットが少ないと見做されてしまった．

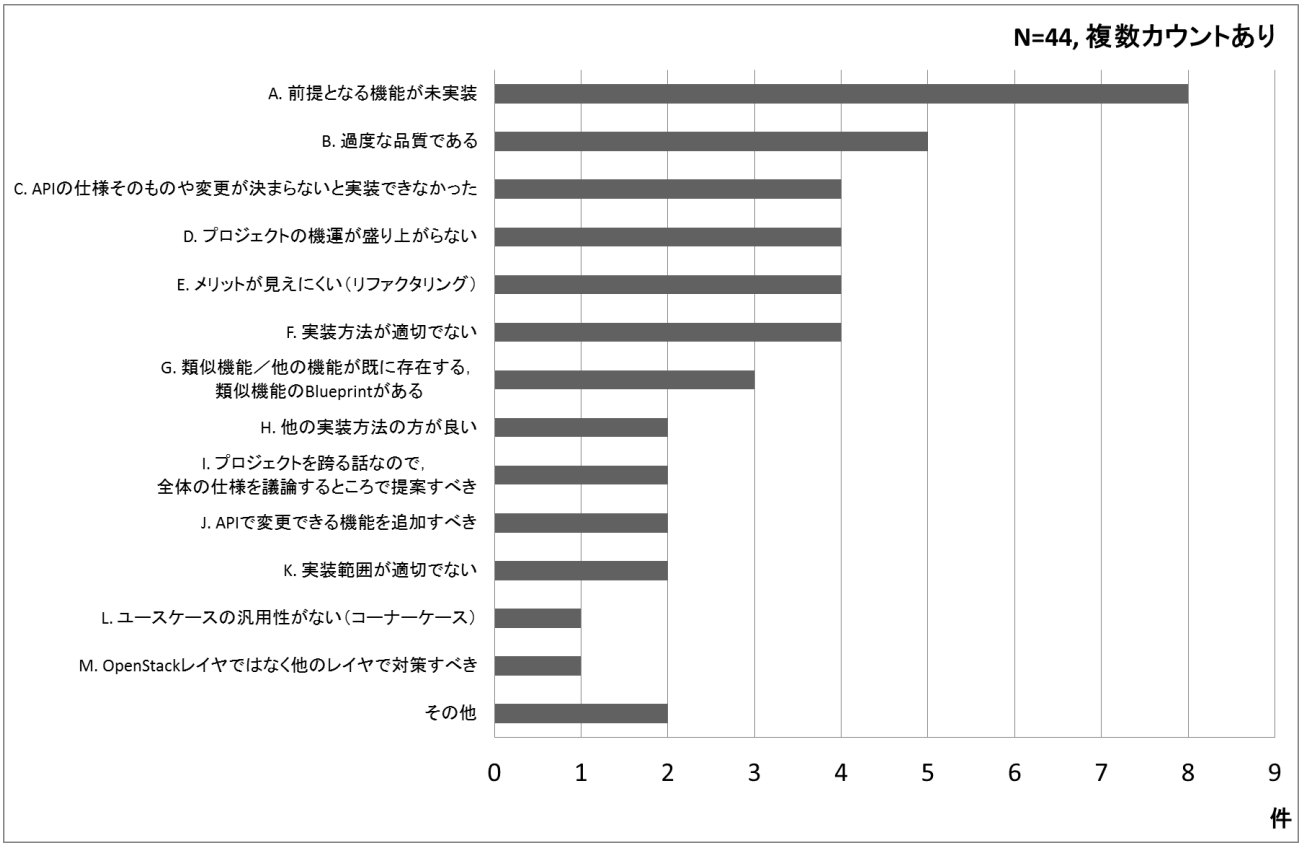

図 4. 実装にまで至らなかった要因の分析

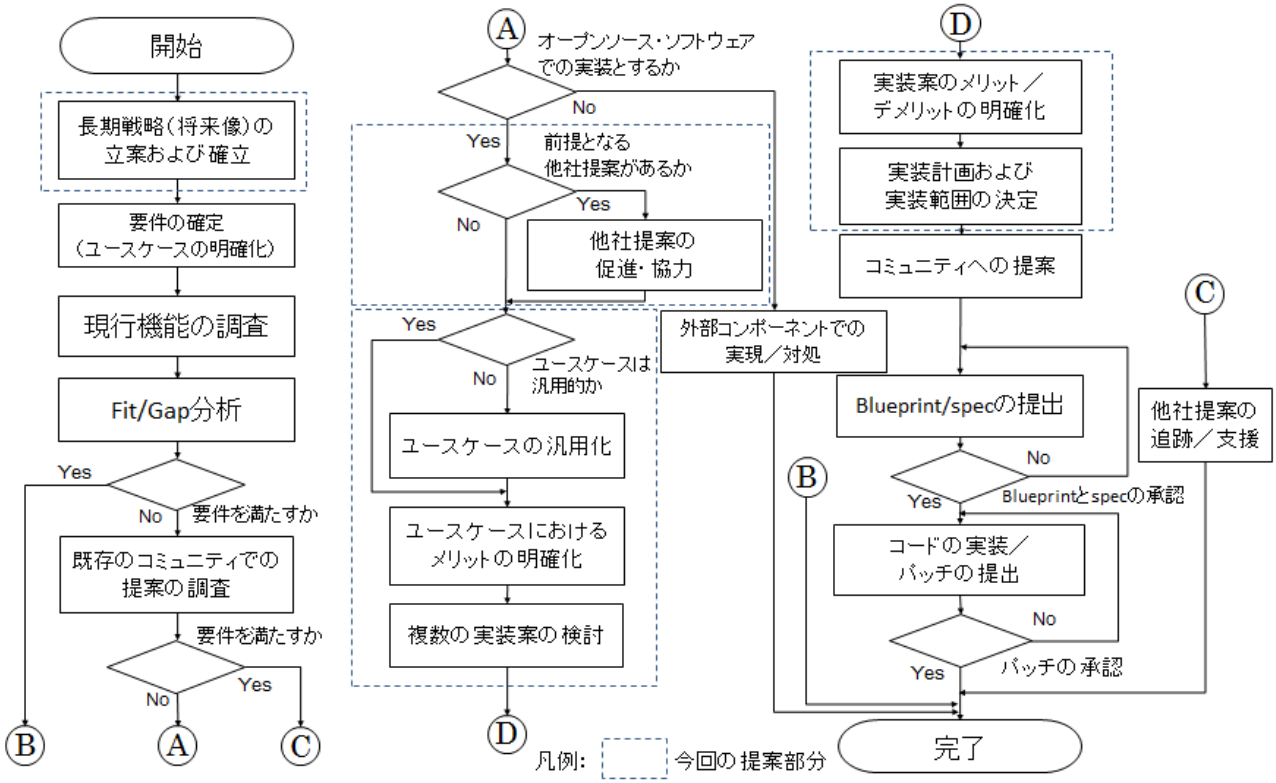

図 5. 提案する新機能提案時の作業フロー

F) 実装方法が適切でない

セキュリティの観点から実装方法が適切でなかったり，ユーザにとって使用しにくくなったりする．また，その修正を行うとデメリットが発生する，コードの修正量が多いといった理由である．

G) 類似機能／他の機能が既に存在する，類似機能の Blueprint が存在する

他社・他者により既に類似の機能が実装されている，あるいは，類似機能の提案がされており，その機能を使用すれば，提案した機能の実装は必要ないと判断された．

H) 他の実装方法の方が良い

提案した実装方法ではなく，他の実装方法を勧められた．

I) プロジェクトを跨る話なので，全体の仕様を議論するところで提案すべき

あるプロジェクト（コンポーネント）へ新機能を提案したところ，多くのプロジェクトに関係する提案であり，OpenStack 全体の仕様として提案すべきだと指摘を受けた．

J) API で変更できる機能を追加すべき

設定ファイルにより設定を行う方式ではなく，ユーザあるいは管理者が API で設定を変更できるようにすべきとの意見が付いた．

K) 実装範囲が適切でない

提案した改修する範囲が広すぎる，または狭すぎるなど．

L) ユースケースの汎用性がない（コーナーケース）

ユースケースが一般的ではなく，筆者らの特殊な使用方法（シナリオ）に当てはまると判断された．

M) OpenStack レイヤではなく他のレイヤで対策すべき

解決すべき課題については理解されたが，OpenStack において解決するのではなく，別のレイヤ（OpenStack の外部）で対策すべきとの指摘を受けた．

実装にまで至らなかった要因として一番多かったのは他社の活動（A，C，G）であり，約 34 パーセントであった．また，ユースケースを原因としたもの（B，D，E，L）は，全体の約 32 パーセントであり，実装を原因としたもの（F，H，J，K）は全体の約 23 パーセントであった．

4. 課題の解決方法の提案

4.1 コミュニティ活動への提案

第 3 章で成功のケースについて要因分析を行うとともに，実装に至らなかったケースについて要因分析を行なった結果，新機能の提案活動については他社活動への対応，ユースケースの説明，実装計画が重要であるということが分かった．そのため，図 2.のプロセスを以下のように変更することを提案する（図 5.）．

A) 他社の活動

- 長期的な戦略を持って他社に先んじて提案を行なう
- 提案・実装をプロジェクトのロードマップに合わせる．
- 他社提案と自社提案のメリット／デメリットを明確にし，コミュニティ（サミット，ML，IRC ミーティングなど）で議論する．
- 他社の担当者に連絡を取り，他社提案の実装を促す，あるいは加速させる活動を行なう．場合によっては担当を自分たちに変更してもらう．

B) ユースケース

- ユースケースの汎用性を確保する．できる限りどのユーザにも当てはまるようにユースケースを設定する．

- 特定の業種に関するユースケースであれば，その業種の企業と連携する．（例えば通信業界）
- 実装しやすいプロジェクト（コンポーネント）で先行して実装して有用性を示す.
- メリットを明確にし，コミュニティ（サミット，ML，IRC ミーティングなど）で議論する.
- 上記に併せて，プロジェクトのロードマップに入れる活動を行なう.

C) 実装

- 実装計画（実装順序等）を示す.
- 実装範囲を適切な範囲に設定し，場合によっては複数回の修正で機能実装を実行する.
- 実装計画において，場合によってはコードのクリーンアップやリファクタリングから開始する.
- デフォルトの挙動に影響を与えない．他の機能への影響を最小限にする.
- できる限り少ない修正量で済むようにする.
- セキュリティの観点で問題ないかを確認する.
- ユーザの使いやすさに影響を与えないかを確認する.
- デメリットは生じないかの確認を行なう（メリット／デメリットの明確化）.
- 複数の実装案を比較する，場合によってはコミュニティ（サミット，ML，IRC ミーティングなど）で議論して，良い実装案がないか探す.
- API の追加の必要性について確認する
  （API での変更操作の必要性について確認する）.

4.2 現行の spec テンプレート記述への提案

spec にはテンプレートが用意されており，記載する必要のある項目が列挙されている．4.1 節の提案に従うと，その中でも以下の項目については，より詳細に記述するのが良い．また，各項目は以下のように記載するのが良い．（Nova[7]の Liberty リリース向けのテンプレートに基づく．）

- Problem description
  - Use Cases（ユースケース）
    ユースケースの汎用性を確保し，メリットを明確にする．（4.1 節の「B）ユースケース」に該当する．）
- Proposed change
  - Alternatives（代替案）
    考えられる実装の代替案をできる限り記載し，メリットとデメリットを論じ，提案が一番良いことを明らかにする．特に修正量，セキュリティ，ユーザの使いやすさ，デフォルトの挙動への影響，実装した場合のデメリットの観点で記載する．（4.1 節の「A）他社の活動」および「C）実装」に該当する．）
  - REST API impact（REST API への影響）
    REST API の追加を行なう必要があるかどうか検討し，追加の必要性がない場合は，影響なしとしたうえで，その理由を明記する．（4.1 節の「C）実装」に該当する．）
  - Security impact（セキュリティへの影響）
    セキュリティ上問題がないことを明記する．（4.1 節の「C）実装」に該当する．）
  - Other end user impact（エンドユーザへの影響）
    ユーザの使用性（ユーザビリティ）に影響や問題がないこと，あるいは使用性が向上することを明記する．（4.1 節の「C）実装」に該当する．）
- Implementation
  - Work Items（作業項目）
    実装範囲を適切な範囲に設定し，場合によっては複数回の修正で機能実装を実行する．（4.1 節の「C）実装」に該当する．）

5. 今後に向けて

今回提案したプロセスおよび方法を今後も適用し，提案方法が有効であることを評価することを考えている．指標としてはコミュニティのソースコードに取り込まれるまでの期間などを検討している．また，OpenStack のガバナンスも変化しているので，それに合わせた手法に変更するとともに，他の OSS への適用を行ない，一般的な OSS の開発への提案手法の有効性を評価していきたい．また，今回は新機能の提案について課題分析と解決方法の提案を行ったが，バグ修正（バグレポート）についても今後解析を行ない，課題分析と解決方法の提案および評価を行っていきたい.

謝辞

OpenStack のコミュニティ活動を筆者らとともに実施してくださった，また本論文の執筆にあたりご協力いただいた株式会社 NTT データの武田健太郎氏および飛内拓弥氏に感謝いたします.

参考文献

[1] Tim Yeaton, “Open Source Compliance – From Risk Mitigation to Competitive Advantage”, Open Compliance Summit 2013, Linux Foundation, http://events.linuxfoundation.org/sites/events/files/OCS2013_TimYeaton.pdf (2013).
[2] “OpenStack.”, https://www.openstack.org/.
[3] “Launchpad.”, https://launchpad.net.
[4] “OpenStack Gerrit.”, https://review.openstack.org/.
[5] “OpenStack Github.”, https://github.com/openstack/.
[6] “OpenStack Wiki Releases.”, https://wiki.openstack.org/wiki/Releases.
[7] “OpenStack Compute (nova) in Launchpad.”, https://launchpad.net/nova.