

なぜオープンソースソフトウェアに コントリビュートすべきなのか

須田 瑛大 (NTT ソフトウェアイノベーションセンタ)


akihiro.suda.cz@hco.ntt.co.jp

自己紹介

- 2014年 日本電信電話株式会社 入社
以来 OSSコミュニティ活動に従事
- Docker (Moby)、BuildKit、containerd、runc、OCI Runtime Spec 等、コンテナに関するものを中心に多数のOSSのコミッタ
 - Docker : Moby = Chrome : Chromium
- nerdctl (containerd サブプロジェクト) や Lima (CNCf プロジェクト) などの創設者
- 2023年 CNCf Top Committer



なぜOSSにコントリビュートすべきなのか

- 結論から言うとOSSの持続可能性のため
- OSSは「タダ飯」のように消費されがち 
- 物理的な「タダ飯」と異なり、遠慮なく好きなだけ食べても他の人の迷惑にはならない
- しかし、「タダ飯」を作る側のことを誰かが気にかけないと「タダ飯」が出てこなくなったり、毒入りののが紛れ込んだりする

- 「無料で使えるソフトウェア」のことでは無い
- Open Source Initiative は OSS の定義 (v1.9)として10個の条件を挙げている [1]
 - 再配布の自由
 - ソースコードでの配布
 - 派生ソフトウェアの作成・配布の許可
 - 利用目的制限の禁止 など
- 無料で使えても、ソースコードが公開されていなかったり、利用目的に制限があったりするソフトウェアはOSSではない

- Richard M. Stallman らが唱える Free Software と類似する。実践上は区別がつかないことが多い。
 - Free は「無料」ではなく「自由」
 - 日本語の「フリーソフト」は単なる無料ソフトを指すことが多い
- Free Software は 「自由と正義のための運動」 [1]
- **OSS は 理念より実利を重視**
- 両者を合わせて FOSS や FLOSS とも言う (L = Libre)

[1] <https://www.gnu.org/philosophy/open-source-misses-the-point.en.html>

- OSS には 様々なライセンスがある
 - MIT License, Apache License v2.0, GPL v2, GPL v3, ...
- OSS に似て非なるライセンスもあり、OSS と混同されがち
- ライセンスが異なるソフトウェアは混ぜて良いこともあるし良くないこともある
 - Apache License v2.0 のプロジェクトに MIT License のコードを混ぜても良いが、GPL のコードは混ぜてはいけない
 - GitHub Copilot などに勝手に混ぜられることが問題になりつつある
- ライセンスについてある程度の知識を得るまでは
OSSにコントリビュートしてはいけない

- MIT License
 - 制約が緩い。プロプライエタリな派生物を作成してもよい。
 - **採用例:** Ruby on Rails, Node.js など多数
 - GitHub上で最もよく使われているライセンス [1] (44.69%, 2021年)
 - 類似ライセンスに X11 License, BSD License (2-Clause), ISC License など
- Apache License v2.0
 - MIT License に似ているが、特許ライセンスを明示的に付与する。ユーザが特許訴訟を起こすと、ユーザ自身に付与された特許ライセンスは終了する。
 - **採用例:** Apache HTTP Server, Docker, Kubernetes など

[1] <https://github.blog/open-source/open-source-license-usage-on-github-com/>

- GPL (GNU General Public License) v2
 - バイナリを受け取ったユーザへのソースコード再配布義務有り
 - **採用例:** Linux, git など
- GPL v3
 - 改変したソフトウェアをインストールすることの妨害 (“Tivoization”) の禁止などの条項を追加
 - **採用例:** GCC, Emacs など大半のGNUプロジェクト
 - Linuxはv3への移行予定無し。Tivoizationを禁止したくないため。

主要なOSSライセンス

- AGPL (Affero GPL) v3
 - バイナリを受け取っていないSaaS ユーザに対してもソースコード再配布義務有り
 - **採用例:** Mastodon, OnlyOffice など
- LGPL (Lesser GPL) v2.1, v3
 - 動的ライブラリとして呼び出す場合にはライセンスが「感染」しない
 - **採用例:** glibc, glib など

OSSに似た非OSSライセンス

- Business Source License
 - 商用利用を許可しない
 - 一定期間後に別のライセンス (大抵OSS) が適用される
 - **採用例:** Terraform, Vagrant など
- Server Side Public License
 - AGPLv3に類似するが、当該のソフトウェアのみならず、提供するサービス全体のソース開示を求める
 - **採用例:** MongoDB など
- Llama 3 Community License
 - ユーザ数や使用目的に制約有り
 - **採用例:** Llama 3

- OSS とは何かを理解するためには、OSS定義までの歴史を辿る必要がある
- (OSSの文脈での) "open source" という表記は 1998年まで出てこない
 - 1996年の用例 "Caldera Announces Open-Source Code Model for DOS" [1] もあるが、1998年以降の用法とは意味が異なる
- 電子計算機黎明期のソフトウェアはOSSに近いものが多かったが1960年代末からプロプライエタリ化が進んだ
- **1974:** UNIX (5th Edition) の AT&T 外への提供開始。提供先はごく少数。無料ではあったが、利用は学術・教育目的に限られた。[2] [3]
- **1975:** UNIX (6th Edition) 有料化 [3]

AT&T: American Telephone & Telegraph

[1] <https://groups.google.com/g/no.linux/c/1UZo-3iv0tM>

[2] <https://www.bell-labs.com/usr/dmr/www/licenses.html>

[3] BYTE OCTOBER 1983 Vol. 8 No. 10, 132頁

- **1983:** Richard M. Stallman が GNU (Gnu's Not Unix) を創設
 - “*Free Unix!*”
 - The Golden Rule: “*if I like a program I must share it with other people who like it.*” [1]
 - カーネルを含む完全なOS開発を目指したが、実際に普及したのは bash などのユーザ空間のみ
 - 最初のカーネル (c. 1985) は開発失敗。GNU Hurd (c. 1990) は2024年現在でも一応は開発が続いている。[3]
- **1988:** 初のfreeなBSDである(ことを目指した)4.3BSD Net/1 リリース [2]
 - 「free な BSD」 ≠ FreeBSD
 - AT&T 関係の著作物を除去したことにより再配布が可能になったとされたが、除去が不完全で訴訟に至った (1992)。4.4BSD Lite (1994)にて解決。
 - Net/1→Net/2 (1991)→386BSD (1992) から NetBSD や FreeBSD (1993) が派生した。後に4.4BSD Liteのコードにリベース。

[1] <https://www.gnu.org/gnu/initial-announcement.en.html> [2] <https://diswww.mit.edu/MENELAUS.MIT.EDU/4bsd-bugs/140>

[3] <https://www.gnu.org/software/hurd/history.html>

- **1991:** Linux v0.01 リリース

- 有償再配布を禁じており、「無料」ではあっても「自由」ではなかった [1]。
Linux v0.12 (1992) にてGPL v2を採用し、晴れて「自由」ソフトウェアになった。
- Linus Torvalds の趣味で、“Just for Fun” [2] として開発された。
“just a hobby, won't be big and professional like gnu”
- “Freax” という名前になるはずだった [2]
- 初のfreeなUNIX互換OSというわけではないが、GNUは技術的に、BSDは法的に難航している間に、漁夫の利を得て勢力を築いた

[1] <https://cdn.kernel.org/pub/linux/kernel/Historic/old-versions/RELNOTES-0.01>

[2] “Just for Fun: The Story of an Accidental Revolutionary” (Linus Torvalds and David Diamond, 2001)

- **1997:** Eric S. Raymond による講演 「伽藍とバザール」 (The Cathedral and the Bazaar)
 - 閉鎖的な「伽藍」型 (GNU Emacs等) と 開放的な「バザール」型 (Linux等) の開発モデルの比較
 - 「伽藍」=プロプライエタリ、「バザール」=OSS では**ない** (混同されがち)
- **1998:** Netscape社が次期製品のソースコードを公開すると声明 [1]
 - 「伽藍とバザール」の影響とされる [2]
 - “*free source distribution with a license which allows source code modification and redistribution*”
 - “open source” の表記はこの時点では使われていない
 - 紆余曲折を経てFirefoxに繋がる

[1] <https://web.archive.org/web/19980127155653/http://home.netscape.com/newsref/pr/newsrelease558.html>

[2] <http://www.catb.org/esr/writings/homesteading/cathedral-bazaar/ar01s13.html>

- **1998:** Netscape の声明を受けた有識者会合にて、**”Open Source”** の表現が Christine Peterson らにより提案される [1]
 - “Free Software” が「自由」ではなく「無料」ソフトウェアのように聞こえることが問題視された
 - Open Source Initiative (OSI) の創設、**The Open Source Definition** の発表に至る
- **1999:** SourceForge サービス開始
- **1990年代末:** Linuxの商用導入が進む
- **2007:** Open Source Development Labs (OSDL) と Free Standards Group (FSG) が合併し、Linux Foundation を結成
- **2008:** GitHub サービス開始
- **2000年代末:** MicrosoftがLinux等のOSSへの敵対を中止 [2]
- **2023:** OSI が “Open Source AI” の定義の策定を開始 [3]

[1] <https://opensource.org/history> [2] <https://www.networkworld.com/article/735210/windows-microsoft-we-love-open-source.html>

[3] <https://opensource.org/deepdive/drafts>

OSSへの依存は不可避

- 96%の商用コードはOSSを含んでいる (2023年) [1]
- OSSを直接使っている認識がなくても、開発ツールやOSなどのことを考えると、誰しものが少なくとも間接的にはOSSに依存しているといえる
- OSS の停滞や脆弱性が、ビジネスや社会の脅威につながる
 - 特に、OSS プロジェクトが乗っ取られて悪意のあるコードを仕込まれると、多大な損害が発生する可能性がある
- OSSに自ら積極的に関与することで、脅威を抑えられる
 - コーディングというよりはガバナンスの話 (ではあるが、どのみちコーディングで貢献しないとガバナンスには携われないことが多い)

OSSは誰が開発・維持しているのか

- 1991年のLinuxは 趣味 (“Just for Fun”) で開発されたが、今日の主要なOSSは業務で開発されていることも多い
 - Linus Torvalds は 2003年よりOSDL (現 Linux Foundation) に雇用されている
- OSS メンテナの36%はOSS活動で収入を得ている (2023年) [1]
 - 13% (収入の大半がOSS) + 23% (収入の一部がOSS) = 36%
 - 非メンテナの開発者は収入を得ている割合が下がりそう
 - 大規模で活発なOSSに絞れば収入を得ている割合は上がりそう (個人的な感覚)
- OSS を個人の趣味とみなして「やりがい搾取」するのは持続可能ではない

メンテナ: 管理権限を持つ開発者のこと。「コミッタ」とは同義だったり同義でなかったりする (プロジェクトに依る)。

[1] <https://tidelift.com/open-source-maintainer-survey-2023>

- 営利企業は経済的に合理的な行動を選択する（はず）
- **合成の誤謬** (fallacy of composition): ミクロな視点で合理性を追求すると、マクロではかえって非合理的になる
- OSSは無料で使えるし、他の誰かが勝手に開発・維持してくれるので、自らはコントリビュートしないのが合理的と思われるがち
 - 皆が「合理的」に振る舞うと誰もコントリビュートしなくなる
 - 自らコントリビュートし、他者と分かち合うのがが実は合理的

- OSS文化の理解のため、Eric S. Raymond は社会地位を築く方法を3つに分類した (1998年) [1]
 - **Command hierarchy**: 軍事力・強制力に依る
 - **Exchange economy**: 使用・交換するモノに対する支配力に依る
 - 自由市場経済
 - **Gift culture**: 何を贈与するかに依る
 - 北米先住民のポトラッチ (競覇的贈与) が典型例
 - **関連**: 「贈与論 (Essai sur le don)」 (Marcel Mauss, 1925)
贈与には(時に過大な)返礼の義務が伴う [2] (→贈与が地位の上下を生み出す)
- Raymond は OSS を gift culture に分類し、身内での**評判** (*reputation among one's peers*) が競争の成功を測る唯一の指標となる状況が生まれると指摘した

[1] <http://catb.org/~esr/writings/homesteading/homesteading/ar01s06.html> [2] <https://dl.ndl.go.jp/pid/1902440/1/75>

- 今日のOSSは**評判**のみがモチベーションであるとは言い難い
 - 個人レベル
 - 給与
 - 自己研鑽
 - 企業レベル
 - プロジェクト維持によるセキュリティの担保
 - 社外開発者との協力による新技術創出
 - 社内forkを維持する負担の軽減
- **評判**も主要なモチベーションではあるが、**承認欲求(感情論)**
ではなく経済的実益として解釈されるべき
 - 個人レベル: 昇進、転職
 - 企業レベル: 売上、人材獲得

- 結局、古典的贈与モデルではOSSコミュニティのダイナミクスを説明しきれない
 - 元々開発に参加していないユーザは、「贈与」に対する「返礼」を怠っても、開発者コミュニティ内での地位を失わない(失いようがない)
- 自由市場経済での合理的行動として解釈できる
 - 純粋公共財 (pure public goods) が 非公共部門 (non-public sectors) により効率的・持続的に供給されうる稀有な例

純粋公共財: 非排除性 (対価を支払わない消費者を排除しない) と 非競合性 (消費が他者による消費を妨げない) の両方を満たす財。ただ乗りされがちなので、自由市場では供給されにくい。国防や警察など、公共部門により供給されるものが多い。

OSSにただ乗りし続けると何が起こるか

- ただ乗りが続くとOSSコミュニティは停滞する
- 新機能が追加されなくなったり、バグが修正されなくなったりするのは大した問題ではない
 - 他のソフトウェアに乗り換えるとか、自分でforkするとかの選択肢がある
- 悪意を持った開発者からの *Gift* が問題
 - **gift** (英): 贈与物
 - **Gift** (独): 毒 (< 投与物 < 贈与物)

xz 乗っ取り事件

- 2024年3月、xz・liblzmaにバックドアが仕込まれていることが発覚 [1]
 - sshd が (libsystemd 経由で) liblzma を呼び出すと、sshdの公開鍵認証部分が不正コードに置き換えられるようになっていた
 - 幸い、主要なディストリビューションにパッケージングされる前に見つかったが、氷山の一角かもしれない
- 悪意を持ったメンテナ(中途参加)によって仕込まれた
 - コミュニティが停滞する中、有益・無害(と思われる)貢献に2年を費やし、信頼を築いていた [2]
 - 本名や所属機関は不明。個人のいたずらにしては時間をかけすぎているとも言われる。他のOSSに対しても組織的に乗っ取りを試みている可能性あり。

[1] <https://tukaani.org/xz-backdoor/> [2] <https://research.swtch.com/xz-timeline>

- **2022年1月:** colors.js および faker.js が開発者自身によって意図的に破壊された。意味不明な文字列やアスキーアートを無限回表示するコードが加えられた。[1]
 - *"Respectfully, I am no longer going to support Fortune 500s (and other smaller sized companies) with my free work."*
- **2022年3月:** node-ipc (Node.js用IPCライブラリ)が開発者自身によって意図的に破壊された。ロシアやベラルーシで実行されている場合にファイルを破壊するコードが加えられた。[2]
- **2024年2月:** polyfill.io (ブラウザの差異を吸収するJavaScriptライブラリ) のドメイン及びGitHubアカウントが売却された。悪性サイトへリダイレクトするコードが加えられた。[3]

[1] <https://www.sonatype.com/blog/npm-libraries-colors-and-faker-sabotaged-in-protest-by-their-maintainer-what-to-do-now>

[2] <https://orca.security/resources/blog/cve-2022-23812-protestware-malicious-code-node-ipc-npm-package/>

[3] <https://blog.qualys.com/vulnerabilities-threat-research/2024/06/28/polyfill-io-supply-chain-attack>

何をコントリビュートすべきか

- コミュニティの持続可能性
 - 他の開発者がやりたがらない作業
(バグ修正、テスト、リファクタ、ドキュメント更新、質問対応、…)
 - これらの作業を地道にやってくれる人の善意を疑わないといけなくなったのがxz事件の最も悲しいところ 😞
 - 他の開発者の支援
 - pull requests のレビュー
 - 途中で放棄された pull requests の引き継ぎ
 - 他の開発者の監視 👁️ (相互かつ友好的に)
 - 不審なコミットがないか
 - アカウントが乗っ取られていないか
 - 名前や所属に偽りがないか

何をコントリビュートすべきか

- 自分・自社の活動の持続可能性
 - 自分・自社がモチベーションを保ち続けられることをやるのが一番
 - 大きい新機能を入れると、その機能のメンテナンスで数年以上活動を続けられる
 - 縛られるとも言える
 - 自己研鑽や趣味として凝った機能を作ってみるのもいいが、他の開発者がメンテナンスできるかへの配慮も必要
 - 配慮したくない場合は、既存プロジェクトにマージさせるのではなく、forkしたり新規プロジェクトを立ち上げることも検討
 - typo fix など簡単なことから始めても良いが、typo fix ばかりやっていると荒らし認定されることもある

何をコントリビュートすべきか

- コードを書くだけがコントリビューションではない
 - マネジメント面でのコントリビューションが実は一番重要
 - どうすれば (特に、自社従業員以外の) 他者の行動を促せるか?
 - コーディング、テスト、ドキュメント作成、レビュー、脆弱性対応 (他のプロジェクトとの折衝を含む)、リリース、メンテナ人事…
 - レビューされず放置されている pull request を動かしてくれる人はありがたい
 - 技術より人脈
 - リジェクトは放置よりありがたい (次の行動に進める)
 - foundation 等への資金拠出もコントリビューション
- とは言っても、コードを書かない「口だけ番長」は発言力を得られない → 結局はコードを書くのが基本

- コミュニティ内での評価（メンテナの選定）
 - 大きい新機能を追加した開発者は、当該モジュールのメンテナンスを長期的に任されることがある
 - バグ管理、リリース管理、他のプロジェクトとの折衝などができるプロジェクト全体の管理を任せやすい
- OSS活動に従事する従業員の社内評価
 - OSS活動が自社製品の売上に直結していると理想的ではあるが、それだけの評価指標にするとコミュニティの持続可能性を損なう
 - コミュニティ内での評価も考慮すべき
 - コミット件数とか行数にこだわりすぎると迷惑行為を助長するのでよくない
 - 数値目標を満たすために typo fix 等、些細な pull request を大量に投稿する組織が散見される

NTTのOSSコントリビューション事例

- NTTグループは Linux, Docker, containerd, Kubernetes, PostgreSQL, OpenStack, Hadoop など多くのOSSにコントリビューートしてきた
- 日本電信電話株式会社 ソフトウェアイノベーションセンタ オペレーティングシステムグループでは 特に Linux, Docker, containerd, Kubernetes などに注力

自身のOSSコントリビューション事例

- 主要コンテナOSSのメンテナ
 - **Docker/Moby** (2016-): 最も有名なコンテナ型仮想化エンジン
 - Docker : Moby = Chrome : Chromium
 - **BuildKit** (2017-): Docker イメージビルダ
 - **containerd** (2017-): Docker や Kubernetes の下にいるランタイム
 - **runc** (2020-): containerd の下にいるランタイム
 - **OCI Runtime Spec** (2022-): runc の仕様書
- 2015年末、Docker のファイルシステム関連の問題に遭遇したのがきっかけで、pull request を投稿したり、課題整理 [1] などのコントリビューションを行なったりするようになった



自身のOSSコントリビューション事例

- Rootless コンテナ (root権限不要でセキュアなコンテナ)
 - 以前から存在した技術 (unprivileged LXC) ではある
 - Docker/BuildKit/containerd/Kubernetes のエコシステムに未対応だった
 - TCP/IPがroot権限に依存していた
 - 上記エコシステムへの対応と併せて、root権限不要で高速なユーザモード TCP/IPスタック (slirp4netns・bypass4netns) を実装した (2018-)
 - Docker に採用されてモチベーションが**上昇**したが、マージとリリースに時間がかかった点で上昇分が**相殺**された (次のスライドの話に繋がる)
 - slirp4netns は Podman (Red Hatが主導するDocker互換エンジン) でも採用
 - RHEL に入ったのでモチベーションが**上昇**したが、NTT のコントリビューションとしての認識が広まらなかった点で上昇分が**相殺**された
 - 「Dockerはroot権限が必要であるが、Podmanはroot権限が不要」とする誤った記事が量産された点もモチベーション**低下**につながった

自身のOSSコントリビューション事例

- **nerdctl** (contaiNERD CTL) [1]: containerdをベースにしたDocker互換エンジン (2020-)
 - OSSとしての Docker/Moby が当時停滞していた (2024年現在は活発)
 - 2019年7月から2020年12月まで、パッチレベル以外のリリース無し
 - containerd側で進んでいたセキュリティや性能面での改善をDockerに取り込みにくい状態が続いていた
 - containerd をベースにして nerdctl を立ち上げた。containerd のサブプロジェクト化として採択。
- **Lima** [2]: nerdctl入りのLinux VMをmacOSで動かすツール (2021-)
 - 昔の Docker Machine に類似
 - CNCF プロジェクトとして採択
 - SUSE の Rancher Desktop や AWS の Finch で使われている

nerdctl

Lima

- コンテナイメージの reproducible builds [1] (2023-)
 - Reproducible builds (同一ソースから同一バイナリを保証) は昔からあるが Docker・BuildKit ではできなかった
 - クライアント側ツールへの pull requests は、基本的なものはマージされたが、UXを向上させるものはマージされなかったりしている
 - Docker Hub 側への pull requests は難航
 - 実装より交渉が難しいことを改めて実感

[1] <https://github.com/docker-library/official-images/issues/16044>

- LLM界隈からの影響の受容・拒絶がテーマになる
 - GitHub Copilot などのLLM系アシスタントが生成するコードにはライセンス上の懸念があるが、生産性の観点からはLLMを禁止し難い
 - 禁止したとしても、コントリビュータは勝手にLLMを使う
 - 利用目的に制限を課すLLMをも“open source”と呼ぶ文化が、LLM以外のソフトウェアにも波及する恐れがある
 - ライセンスの解釈が難しくなる
 - ライセンスが異なるソフトウェアを混ぜてよいかの判断が難しくなる
 - **“open source” を名乗っていても信用せず**、ライセンスを確認することが従来にも増して重要になる

- 匿名・所属非公開での活動は難しくなる（特にメンテナ）
 - ハンドルネームでの活動が駄目というわけではない
 - 2024年3月のxz 事件以後、優秀でも身元が不明な人物をメンテナに登用することは難しくなった
 - Open Source Summit や FOSDEM などの会議にオフラインで顔を出せる人物は、出せない人物より信頼されやすくなる
 - 登壇者以外も頻繁に顔を出すべき
 - 旅費を勤務先に請求できない個人をどう包摂するかが課題
 - ただし、Linux カーネルの場合は、2023年2月よりsign off に用いる名前の要件が“real name”から“known identity”に弱められている
[1]

[1] <https://github.com/torvalds/linux/commit/d4563201f33a022fc0353033d9dfeb1606a88330>

- プロジェクトの持続可能性の数値化
 - “Bus factor”：開発者が何人バスに撥ねられても持続できるかを示す（物騒な）数値
- プロジェクトの発展・衰退（・復活）の数理モデル化
 - 再現性のあるやり方で、プロジェクトを発展・復活できるようにしたい

- なぜOSSにコントリビュートすべきなのか
→ 持続可能性のため (だけとは言っていない)
- OSSが放置されると悪意を持った開発者に乗っ取られることがある
- ただ乗りは合理的のようで合理的ではない