

循環型ソフトウェアエコシステムの実現に向けた ソフトウェア 3R の提案

齋藤 忍^{1,a)}

概要: 企業・組織の IT 投資のトレンドとしては、ERP 等の非差別化領域のソフトウェアは個社での独自開発からパッケージ製品を利用する時代になる一方、差別化領域のソフトウェアは個社の狙い・ニーズに特化した戦略的な投資がますます重要となってきた。しかしながら、企業・組織の IT コストの約 80 % は保守・運用（つくった IT を使い続ける）が占めている。つくる→つかう（DevOps）を繰り返すだけでは、IT は無秩序に増殖する。結果として、企業内のリソースも逼迫し戦略的な投資は困難となる。本稿では、従来のソフトウェア社会の前提（つくる→つかう）を覆し、循環型ソフトウェアエコシステム（つくる→つかう→すてる→もどすのサイクル）の実現に向けたソフトウェア 3R（Reduce, Reuse, and Recycle）のアプローチを提案する。ムダなソフトウェアを「すてる」ことにより、維持・運用のコストの大幅な低減を実現し、戦略的な投資へのリソースシフトを可能にする。また、既存ソフトウェアを資源（再利用可能なプログラム部品）に「もどす」ことにより、ソフトウェア開発タスクの極小化・非属人化を実現する。これにより、現状は様々な制約でプロジェクトへの参画が難しかった人達（育児者・介護者等）の社会復帰にも貢献する。

Software 3R (Reduce, Reuse, and Recycle) for Promoting Circular Software Ecosystem

1. はじめに

ソフトウェアはその性能、利便性等により生み出される価値のみではなく、それ以外の尺度で価値を捉え直す時期に来ていると考える。ハードウェア（ネットワークや CPU の速度）の進化は今までも、そしてこれからも続き、将来の世界では今まで以上に大量のソフトウェアに人間は囲まれることになる。このような世界では、ハードウェアが生み出す大量の計算資源、およびソフトウェアが扱う大量のデータにより、人間が処理できる容量をはるかに超えた情報が、日々提供されるようになる。ソフトウェアの性能や利便性は飛躍的に向上する一方、自身の処理能力を大幅に超えた大量の情報にさらされた人間は、精神的なストレスが増大することが予想される。大量の計算資源や大量のデータがもたらすソフトウェアの性能や利便性の向上は、必ずしも人間の生活を豊かにするわけではない。デジタル

起因の精神疾患は既に何十年以上も前から指摘されており、そもそも人間（の脳）はテクノロジーの進化に順応できない、という話もある [12]。筆者の周りでは「人間はやがて（デジタル起因の）精神疾患で絶滅する」という話も存在する。

超高速・大容量のコンピュータリソースが出現する未来では、人間はそのリソースを全て使い切ることはできない。例えば、10Y(ヨタ = 10^{24}) bps のネットワークがあっても人間には動画を綺麗に見れば十分（数 Gbps）であり、それ以上の帯域は人間には不要である。この領域は主にコンピュータによって利用されると考える。このような世界においては、コンピュータ自身が利用するコンピュータの世界と、人間が利用するコンピュータの世界とを、それぞれ別々に考える必要がでてくる。大量・高速を追い求めるコンピュータが利用する前者の世界では、利便性を追求し、大容量・超高速化を追い求める。一方で、人間が利用する後者の世界では、大量・高速とは異なる、人間を幸福にするという新たな価値の定義が必要になる。本稿では、後者の世界におけるソフトウェアの未来像を考えていく。

¹ 日本電信電話 (株)
NTT Corporation, Chiyoda, Tokyo 100-8116, Japan
a) shinobu.saitou.cm@hco.ntt.co.jp

新型コロナのような世界規模の環境変化が起こり、これからはじまろうとしている新しいビジネスや生活では、これまで以上にソフトウェア、そしてソフトウェア工学の重要性は高まってくると考える。従来のソフトウェア工学では（そして筆者自身も）、ソフトウェアを「つくる（開発する）」ことに重きを置いていた。そのため「つくる」のアプローチに革新をもたらすことに注力する研究が大勢を占めていたと言える。一方、（多分に筆者の私見も混じってはいるが）ソフトウェアの開発や運用の現場では、不要・不急なサービスや機能がいつまでも残り、新規に作る必要がないソフトウェアをフルスクラッチで作っている状況から脱却できていない。実際、企業の業務システムにおいては、ソフトウェアが提供する機能の45%が「全く使われていない（Never Used）」、19%が「めったに使われていない（Rarely used）」という調査報告もされている [7]。

上述の調査報告 [7] の数字（45%や19%）の妥当性に関する議論 [4] は存在するとはいえ、筆者はシステムにおけるムダなソフトウェアは減らす、そして、今あるソフトウェアは活かすことが、これまで以上に必要になると考える。そのためには、ソフトウェアを「つくる」だけの研究ではなく、更に俯瞰的な視点が求められる。ソフトウェアを循環させる、即ち、つくる→つかう→する→もどす、のサイクルが繰り返される新しいソフトウェアの社会環境（エコシステム）を実現する研究が、これからのソフトウェア工学が（そして筆者自身も）目指すべき1つの方向性であると考えている。ソフトウェアを「つくる」側と「つかう」側の双方に価値をもたらす研究であると考えている。

本稿の構成は以下のとおりである。2章では、ソフトウェアを取り巻く状況を概観し、3章では、新しいソフトウェアの世界のコンセプトと新たなアプローチを提案する。4章では、アプローチを遂行する上で解決すべき問題やソリューションの方向性の提示を試みる。その上で、今後議論すべき論点を4章で示し、5章でまとめを述べる。

2. ソフトウェアを取り巻く状況

2.1 増え続けるコード (Big Code)

企業・組織のシステムは、改修や追加を繰り返してきた結果、その規模は膨大になり、複雑化・肥大化している。北米（アメリカ合衆国とカナダ）のプログラマを対象とした調査によれば、プログラマの扱うコード量は10年前と比較して膨大になっており、調査対象の半数以上のプログラマが「10年前と比較して100倍以上に増えている」と回答をしている [2]。このようなトレンドはBig codeとも呼ばれている。一方、ソフトウェアの保守における作業効率の研究では、保守プロセスを標準化することにより、技術者あたりの保守量が概ね8倍（少なく見積もると2倍、多く見積もると35倍）程度改善することが期待されると報告されている [13]。しかしながら、このままのペースでソ

フトウェアのコード量が増え続ければ、（たとえ保守の技術者の作業効率を35倍に改善できたとしても）いずれはコード量がヒトの能力でカバーできる範囲をはるかに超えてしまう状況に至ってしまう。

増え続けるコード (Big code) がもたらす状況は、ソフトウェアをつくる技術者のみだけでなく、ソフトウェアをつかう側にも問題を引き起こす。1980年代にBohemが始めた、ソフトウェアの規模（例：LoC）とソフトウェアの保守・運用コストの関係の研究では、ソフトウェアの規模が膨大化すれば、ソフトウェア保守コストも劇的に向上することが明確に指摘されている [1]。これは現代においても変わらない。

企業・組織のシステム投資のトレンドとしては、ERP等の非差別化領域のソフトウェアは個社での独自開発からパッケージ製品を利用する時代になっている。一方、差別化領域のソフトウェアは、個社の狙い・ニーズに特化し、戦略的な投資がますます重要となってきている。しかしながら、ITコストの80%は保守・運用（つくったITを使い続ける）が占めている。上述のようにソフトウェアが無秩序に増え続ける結果として、企業・組織のリソース（予算や人員）は更に逼迫し、彼らの戦略的な投資はますます困難となってしまう。

2.2 消費リソースの肥大化 (Software Bloat)

プログラムのコード量が増え続けることの結果として、増大したコードで規定されたソフトウェアは、ハードウェアのリソース（CPU能力、メモリ）をより多く使う傾向が顕著になってくる。このような状況はSoftware bloatと呼ばれている。ソフトウェアそのものは電力を消費はしない。一方、ソフトウェアがハードウェアのリソースを大量に使い続けるようになれば大量の電力を消費することになる。実際、ビットコインネットワークを維持するために必要な電力は、スイス一国の電力消費量の2倍に達する。これは2020年の世界の国別の電力消費量で比較すると、スウェーデンを上回っており、エジプト、マレーシアに迫る消費量であると試算されている [3], [8]。

ハードウェアレベルの省リソース化の取り組みと比較して、ソフトウェアレベルの取り組みはこれからであると言われて久しい [14]。特にアプリケーションレベルの取り組みは、コンパイラ技術に比べて変更の自由度が大きいことから、電力削減効果が非常に大きいと言われていた [14]。近年、データセンタのアプリケーションプラットフォーム（デバイスとソフトウェアの組み合わせ）としてのエネルギー効率指標の国際標準規格ISO/IEC 23544 Application Platform Energy Effectiveness (APEE) も発行されている [6]。このように、ITサービスの省リソース化には、ハードウェアと同様にソフトウェアでの取り組みが重要であるとの認識が高まってきている。

目指す世界（循環型ソフトウェアエコシステム）のイメージ

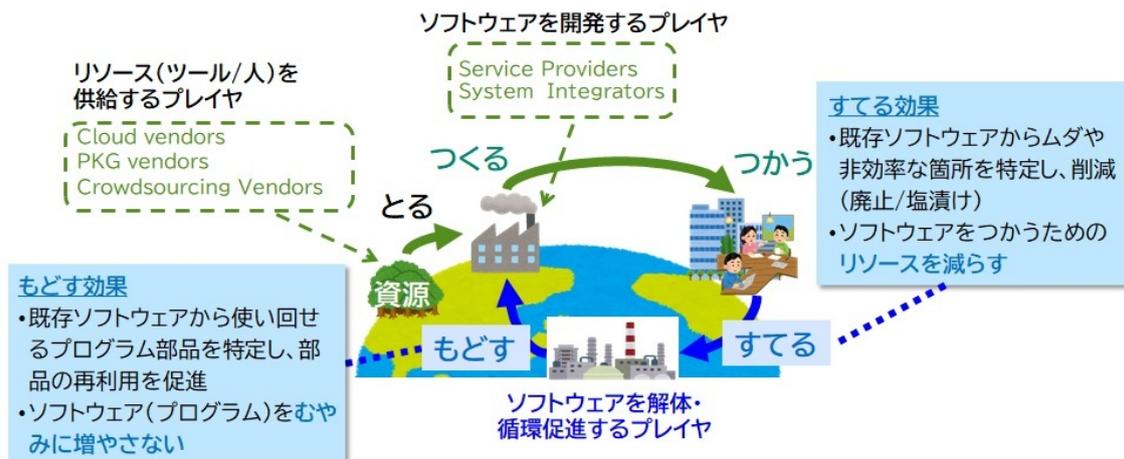


図 1 循環型ソフトウェアエコシステム
 Fig. 1 Circular Software Ecosystem

3. コンセプトとアプローチ

3.1 これまでとこれからのソフトウェア工学

ソフトウェア工学の一般的な定義は諸説あるが、筆者は一文で表す場合には「目的とするソフトウェアを時間やコスト等の現実的な制約の中で開発・運用を行うための理論や技術の体系」としている。更に平易な表現としては「ソフトウェアの開発をラクにしてくれる、特にプログラムが早く・たくさん作れるようにしてくれる技法やツール」としている。これまでのソフトウェア工学の技法・ツールの取り組みの変遷としては、2000年代には標準化（例：モデル作成の標準言語としてのUML [10]）、そして、2010年代には自動化（例：自動プログラムの修正手法であるGenProg [11]）に研究のトレンドが推移してきている。しかしながら、これらの取り組みは、プログラムが早く・たくさん作れるようにしてくれること、即ち「つくる」に主に焦点を当てている。そのため、前章で述べたようなソフトウェアを取り巻く状況を必ずしも解決はしてくれない。筆者は、これからのソフトウェア工学は、ソフトウェアの開発（Software Development）をラクにしてくれる研究だけでなく、ソフトウェアの解体（Software Demolition）をラクにしてくれる研究にももっと目を向けるべきであると考えている。

3.2 コンセプト：循環型ソフトウェアエコシステム

ソフトウェアの前提（つくる→つかう）を覆し、ソフトウェアを循環させる（つくる→つかう→すてる→もどす、のサイクルが繰り返される）新しい社会環境（エコシステム）のコンセプトを提案する。

図 1 に目指す世界（循環型ソフトウェアエコシステム）のイメージを示す。従来、ソフトウェアのプロセスは、「つ

くる（Development）」の後に「つかう（Operation）」、即ち、DevOps が一般的なプロセスとして表現されている。提案する循環型ソフトウェアエコシステムにおいては、この前半のプロセス（DevOps）に対して、「すてる（Removal）」と「もどす（Return）」の後半のプロセスを加えることで循環サイクルを実現する。あくまで筆者の呼称であるが、この後半のプロセスを「RemRet（レムレット）」と呼ぶ。加えて、これら4つのプロセス全体を「DevOps2RemRet（デブオプス・トゥ・レムレット）」としている。

循環型ソフトウェアエコシステムは、以下に記す4つのプレイヤーで構成される。

- **ソフトウェアを開発するプレイヤー**
 ソフトウェアを開発し、IT サービスを提供する実行主体である。広義にはソフトウェアの保守・運用を担う企業・組織も含まれる。例えば、予め用意したIT サービスを提供するサービスプロバイダや、顧客（ユーザー企業等）からの要望を受けてソフトウェアを開発するシステムインテグレータが、このプレイヤーに相当する。
- **リソース（ツール/人）を供給するプレイヤー**
 ソフトウェアを開発するプレイヤーに対して、開発に必要なリソース（ツールや人）を供給する実行主体である。例えば、クラウド環境やパッケージ製品を供給するベンダや、クラウドソーシングにより開発者を供給するベンダが、このプレイヤーに相当する。
- **ソフトウェアをつかうプレイヤー**
 自身のビジネスを遂行するためにソフトウェアを利用する実行主体、いわゆるユーザー企業である。
- **ソフトウェアを解体・循環促進するプレイヤー**
 循環型ソフトウェアエコシステムを実現するために新たに本稿で創出した実行主体である。本プレイヤーは、既存のソフトウェアからムダや非効率な箇所を特

定し、それらのソフトウェアの削除（すてる）の支援を実施する。更に、既存ソフトウェアから使い回せるプログラム部品を特定する等により、ソフトウェアの循環（もどす）の促進も実施する。

3.3 アプローチ：ソフトウェア 3R (Reduce, Reuse, and Recycle)

2004年に開催されたG8サミット（シーアイランドサミット）において日本政府は、物的資源（もの）の有効利用を通じて環境と経済の両立を図るため、「3R（スリーアール）」を提唱した[9]。3Rは、廃棄物の発生抑制（Reduce）、再使用（Reuse）、再生利用（Recycle）の取り組みから構成される。更に、小学生に向けた環境リサイクル学習では、3つの言葉（R）について、心がける順に並べて説明をしている[16]。無駄な「もの」をできるだけ少なくすること（Reduce）、一度つかった「もの」を何度も使うこと（Reuse）、使い終わった「もの」をもう一度資源にもどして製品をつくること（Recycle）、の順番である。

上述の3Rの考え方において「もの」をソフトウェアに読み替えることは可能であると考え。そこで本稿では、3Rの対象（＝もの）をソフトウェアに置き換え、前節のコンセプトを実現するアプローチとして「ソフトウェア 3R (Reduce, Reuse, and Recycle)」を提案する（図2参照）。

既存のソフトウェアを「すてる」ためには、ムダや非効率な箇所（サービスや機能）を減らす（Software Reduce）必要がある。また、ソフトウェアを「もどす」ためには、既存のソフトウェアをプログラム部品に再生すること（Software Recycle）、および（再生された部品か否かに関わらず）プログラム部品を新たなソフトウェアの開発に使い回すようにする（Software Reuse）が必要となる。

図2では3Rの概念的イメージを示している。左側の既存ITのソフトウェアAは、削減可能（Reducible）であることから、「すてる（廃棄・塩漬け）」プロセスが実行される。一方、ソフトウェアBは、一部が再資源化可能（Recyclable）であり、一部は再利用可能（Reusable）である。前者は、再資源化されてプログラム部品となり倉庫（Software Inventory）に格納された後、新規ITのソフトウェアCの開発に使われる。後者は、再利用可能な個所が、新規ITのソフトウェアCに組み込まれる。こうして「もどす」プロセスが実行される。

3.4 解決すべき問題とソリューションの方向性

ソフトウェア3Rのそれぞれのアプローチを遂行するためには、どのようなことが問題となり得るのか（解決すべき問題）、そして問題の解決にはどのような方針（ソリューションの方向性）があるのかについて、本節では筆者の考えを提示していく。

3.4.1 Software Reduce

- 解決すべき問題

ソフトウェアをつかう実行主体（企業・組織）それぞれにおいて、価値のない（廃止対象の）ソフトウェアは異なってくる。そのため、廃止対象のソフトウェア（図2のReducibleの箇所）の発見には、企業・組織内のソフトウェア全体の利用・運用状況を俯瞰的に分析した上で、ソフトウェアの改廃を精査する必要がある。しかしながら、これらの調査作業を人手で実行することは極めて困難である。

- ソリューションの方向性

企業・組織内で運用される全てのソフトウェアに蓄積されるログデータ等を収集・分析し、各ソフトウェアの利用・運用状況を把握・学習することで、当該企業・組織において不要・不急なソフトウェアの発見・抽出を自動化する。これにより、調査作業の効率化・高精度化を実現する。

3.4.2 Software Recycle

- 解決すべき問題

ソフトウェアを構成するプログラムのなかから、再生利用が可能なプログラム部品の候補を抽出することになる。そのためには企業・組織で運用されている全ソフトウェアを調査し、ソフトウェアの構成上の類似箇所（図2のRecyclableの箇所）の発見と分類を行う必要がある。しかしながら、これらの調査作業を人の目視や手作業で実施することは極めて困難である。

- ソリューションの方向性

企業・組織が運用するソフトウェアの構成情報（例：UIの画像データ）を収集・学習（例：画像認識やクラスタリング）し、当該企業・組織にとって、共通化や標準化の価値があるプログラム部品の発見・レコメンドを自動化する。これにより、調査作業の効率化・高精度化を実現する。

3.4.3 Software Reuse

- 解決すべき問題

ソフトウェアを構成するプログラムのなかから、そのままの状態での別のソフトウェアに使い回し可能な個所（図2のReusableの箇所）の切り出しをすることになる。一方、切り出されたプログラムには様々な機密情報が含まれる場合もある。このような機密性の高い情報を含む資産（プログラム）を使い回す場合には、漏えいや悪用リスクを減らす狙いより、たとえ同じ企業・組織内でもできるだけ機密情報にアクセスできる人を減らすことが重要である。しかしながら、あまりに厳格にアクセス制限をしてしまうと、再利用可能な個所が限定的になる、限られた開発者しか再利用ができなくなる、といった弊害も生まれる。

- ソリューションの方向性



図 2 ソフトウェア 3R (Reduce (削減), Reuse (再利用), Recycle (再資源化)).

Fig. 2 Software 3R (Reduce, Reuse, and Recycle).

企業・組織の内外の技術者が、再利用可能なソフトウェアで開発をすることは担保しつつ、ソフトウェアを構成する各プログラム単位の（細い粒度の）アクセス管理を実現する。これにより、たとえ機密性の高いプログラムを含む再利用可能なソフトウェアであっても安心・安全な再利用の促進を実現できる。

4. 議論

4.1 V 字モデルの両翼の連結

システム開発のプロセスモデルとして V 字モデルがある。米国連邦政府のシステム開発のハンドブックでは「両翼 (The wings)」という表現を用いて、V 字モデルの前後のプロセスを重要な追加要素 (a key addition) と呼び、これらを考慮すべきであると主張している [5]。左翼 (V 字モデルの前プロセス) には、「System Architecture(s) (アーキテクチャ定義)」が定義され、右翼 (V 字モデルの後プロセス) には「Retirement/Replacement (リタイアメント/置き換え)」が定義されている。

一方、上述のハンドブックでは、単純に古くなったシステムはリタイアか置き換えを実施することを述べているに留まっている。本稿で提案する循環型ソフトウェアエコシステムは、ハンドブックにおける両翼を連結するイメージと捉えることもできる。これにより、単なる追加要素ではなく、V 字モデルへの好影響 (開発・運用の効率化) をもたらすことも可能となる。

4.2 後付け (After-the-fact) の推進

従来、ソフトウェア工学の領域では、再利用を事前に見越したソフトウェアの設計手法 (ドメイン分析・ドメインアーキテクチャモデル等) や開発プロセスの研究が数多くなされてきた [15]。しかしながら、先を見越した (Before-the-fact) 分析では、必ずしも再利用は進んでいな

いというのが現状である。事前に再利用を見越して、時間や工数をかけて作ったプログラム部品であるが、結果としては複数のソフトウェアにはまらない (再利用されない) ケースもソフトウェアの現場で多々ある。

本稿で提案するアプローチ、特にソフトウェア 3R の Recycle では、この従来の発想を転換し、後付け (After-the-fact) の分析をすることを推進している。既存のソフトウェアをもう一度資源 (部品) に戻すことで、何度でも使い回すための分析を進めている。既存のソフトウェアのなかの類似部分を切り出しているため、当該ソフトウェアを含むシステムの次の更改やモダナイゼーションでは、確実に複数のソフトウェアにはまる (再利用される) 可能性が高まってくる。即ち、再利用できるソフトウェアの総数やそれに伴う削減コード量が、事前に高い精度で予測可能となる。そのため、部品化をする工数に対する投資回収も早くなることが期待できる。

4.3 エコシステムの中心的存在

IT の世界ではソフトウェアを開発するグローバルなプレイヤー (例: IBM、アクセンチュア、Infosys) は数多く存在する。これらのプレイヤーの事業の方向性は、ソフトウェアを効率的に増やす (つくる) こと、そして残し続ける (つかう) ことに注力している。これに対して、例えば国内のつくるプレイヤーの企業が、循環型ソフトウェアエコシステムにおける「すてる」と「もどす」 (= RemRet プロセス) の確立を目指すことは、上述のグローバルなプレイヤーとの差別化要素が強化できる可能性がある。当然のことながら、これらの企業はエコシステムのなかで鍵となるソフトウェアを解体・循環促進するプレイヤーも担える。更に「とる」や「つかう」のプレイヤーとも連携し、エコシステムのキャスティングボードを獲得することも可能となる。加えて、ソフトウェアを「つかう」企業に対しても価値 (維持・運

用コストの大幅な低減・戦略的な投資へのリソースシフトの実現)を提供できるようになる。

4.4 社会復帰・参画に向けた支援

ソフトウェアが大規模化すると、開発者が把握すべき対象ソフトウェア固有の知識も増大する。結果として、ソフトウェア開発・運用のタスクの属人化が極端に進むことになり、専任人材でなければタスクの遂行が極めて困難となってしまう。本稿で提案するアプローチのなかで、特にソフトウェアの Recycle や Reuse が容易にできる環境が整備されることで、上述の知識を極小化することが期待できる。その結果、属人化されたタスクを減少させることもできる。これにより、現状は様々な制約でソフトウェア開発のプロジェクトへの参画が難しかった人たち(育児者・介護者等)の社会復帰・社会参画の促進にも貢献できる。

5. おわりに

本稿では、従来のソフトウェア社会の前提(つくる→つかう)を覆し、循環型ソフトウェアエコシステム(つくる→つかう→すてる→もどすのサイクル)のコンセプト、およびコンセプトの実現に向けたソフトウェア 3R (Reduce, Reuse, and Recycle) のアプローチを提案した。循環型ソフトウェアエコシステムにおける「つくる→つかう」のプロセスと、「すてる→もどす」のプロセスは、車の両輪であると考えられる。前者のプロセスがアクセルであるならば、後者のプロセスはブレーキと言える。これからもソフトウェア開発の効率化・自動化の進展(スピードアップ)は望まれるべきである。一方、無秩序にソフトウェアが増えてしまうことの弊害への対策(スピードダウン)も見逃してはならない。

IT 人材の需給バランスの欠如は、国内にとどまらずグローバルなトレンドである。特にこのままソフトウェアの増大と就業人口の減少が進むと、世界的にも深刻な状況に発展することになる。ソフトウェアが増え続けるなかでも、アクセルとブレーキを駆使して適度なスピードにコントロールしていくことは社会的な課題である。そのなかで、エコシステムの中心的存在となるソフトウェアを解体・促進するプレイヤーは重要な役割を担うはずである。そして、そのプレイヤーを支援するためのソフトウェア 3R の研究開発(技法・ツールの創出)がこれからのソフトウェア工学の進む1つの方向性であると考えられる。

しかしながら、本稿における未来予測や問題意識、それに基づく提案や考察の内容は、技術的にも概念的にも整理・再検討が必要であることは筆者も認識している。本稿で提示した様々な点に対して読者が同意して頂くことは全く不要である。本稿の内容が、将来のソフトウェアの価値、およびソフトウェア工学の価値に関する様々な問題に関する読者の議論の出発点となり得るものであれば幸いである。

謝辞 本稿の執筆に先立ち、筆者との議論にご協力頂いた宮原 信二氏、川古谷 裕平氏に深謝します。

参考文献

- [1] Boehm, B. W.: Software Engineering Economics, *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 1, pp. 4-21 (online), DOI: 10.1109/TSE.1984.5010193 (1984).
- [2] Cardoza, C.: Report: The emergence of Big Code comes with big problems, *SD Times* (online), available from <https://sdtimes.com/softwaredev/report-the-emergence-of-big-code-comes-with-big-problems/> (accessed 2022-08-01).
- [3] CCAF: Cambridge Bitcoin Electricity Consumption Index (CBECEI), Cambridge University (online), available from <https://ccaf.io/cbeci/index> (accessed 2022-08-01).
- [4] Eveleens, J. and Verhoef, C.: The rise and fall of the Chaos report figures, *IEEE Software*, Vol. 27, No. 1, pp. 30-36 (online), DOI: 10.1109/MS.2009.154 (2010).
- [5] FHWA: Systems Engineering for ITS Handbook, Federal Highway Administration (online), available from <https://ops.fhwa.dot.gov/publications/seitsguide/index.htm> (accessed 2022-08-01).
- [6] ISO: ISO IEC 23544:2021 - Information Technology — Data centres — Application Platform Energy Effectiveness (APEE) (2021).
- [7] Johnson, J.: *ROI, It's Your Job*, 3rd International Conference on Extreme Programming (XP 2002) (2002).
- [8] Martin, K. and Nauman, B.: Bitcoin's growing energy problem: 'It's a dirty currency', *Financial Times* (オンライン), 入手先 (<https://www.ft.com/content/1aebc2db-8f61-427c-a413-3b929291c8ac>) (参照 2022-08-01).
- [9] MOE: 3R Initiative, Regional 3R Forum in Asia and the Pacific (online), available from <https://www.env.go.jp/recycle/3r/initiative/en/index.html> (accessed 2022-08-01).
- [10] OMG: Welcome To UML Web Site!, OMG (online), available from <https://www.uml.org/> (accessed 2022-08-01).
- [11] Weimer, W., Nguyen, T., Le Goues, C. and Forrest, S.: Automatically finding patches using genetic programming, *2009 IEEE 31st International Conference on Software Engineering*, pp. 364-374 (online), DOI: 10.1109/ICSE.2009.5070536 (2009).
- [12] Weise, S.: *InstaBrain: The New Rules for Marketing to Generation Z*, Independently published, english edition (2019).
- [13] 角田雅照, 門田暁人, 松本健一, 押野智樹: 受託開発ソフトウェアの保守における作業効率の要因, *コンピュータソフトウェア*, Vol. 29, No. 3, pp. 3.157-3.163 (2012).
- [14] 石原 亨: ソフトウェアに対する電力見積りと電力削減技術, *電子情報通信学会基礎・境界ソサイエティ Fundamentals Review*, Vol. 2, No. 3, pp. 3.38-3.44 (2009).
- [15] Will Tracz(原著), 畑崎 隆雄(翻訳), 鈴木博之(翻訳), 林雅弘(翻訳): ソフトウェア再利用の神話—ソフトウェア再利用の制度化に向けて, *ピアソンエデュケーション* (2001).
- [16] 資源・リサイクル促進センター: 小学生のための環境リサイクル学習ホームページ, 産業環境管理協会(オンライン), 入手先 (<https://www.cjc.or.jp/j-school/>) (参照 2022-08-01).