



生成AI時代の大規模ソフトウェア開発を支える トレーサビリティ自動解析技術

生成AIを大規模なソフトウェア開発で活用するためには、変更に関係する要件・設計・実装・テストの依存関係を正確に把握することが重要です。本稿では、開発成果物を自動的に分割・関連付けし、トレーサビリティを構築することで、これらの依存関係を正確に把握する技術を紹介します。この技術を活用して生成AIに必要な情報を適切に与えることで、要件変更に伴う影響分析や修正、ドキュメントの最新化などを効率化する取り組みについて説明します。

キーワード：#ソフトウェア開発、#生成AI、#トレーサビリティ

くらばやし としゆき

倉林 利行

NTTコンピュータ&データサイエンス研究所

特集

生成AIによるソフトウェア 開発の高速化と、大規模開発 への壁

生成AIの発展により、自然言語で実現したいことを伝えながら、ソースコードの生成や修正、テストを短いサイクルで繰り返す開発が可能になっています。特に比較的小規模なソフトウェアでは、生成AIを活用することで、アイデアを素早く実装し、試行錯誤しながらソフトウェアを改善していく開発スタイル（パイプコーディング*など）が広がっています。

一方で、このような生成AIを活用した開発を、そのまま大規模なソフトウェアへ適用することには課題があります。

大規模なソフトウェアには、数多くの要件、設計書、ソースコード、テストケースが存在し、それらが複雑に依存しています。そのため、1

つの要件を変更する場合でも、関連する設計や実装、テストを特定し、それらの整合性を保ちながら変更する必要があります。

しかし、生成AIが一度に扱える情報量には限界があります。大規模なソフトウェアに関する要件・設計・実装・テストの情報をすべて同時に与え、その依存関係まで正確に理解させることは現実的ではありません^{(1),(2)}。

その結果、要件変更への対応に生成AIを活用しようとしても、設計・実装・テストのどこを修正すべきかを正確に特定できなかったり、影響範囲を見誤ることで既存機能の不具合を引き起こしたりする可能性があります。

大規模なソフトウェアで生成AIを活用するためには、すべての情報を一度に与えるのではなく、変更対象に関連する要件・設計・実装・テストを特定し、必要な情報を辿りながら生成AIに与える仕組みが必要です⁽³⁾。その鍵となるのが、ソフトウェア開発で従来から品質確保のた

めに用いられてきた「トレーサビリティ」です。

大規模ソフトウェア開発に おけるトレーサビリティの 重要性

トレーサビリティとは、要件・設計・実装・テストの対応関係を明確にし、それらの一貫性や網羅性を確認したり、変更による影響を追跡したりできる状態にすることです⁽⁴⁾。

大規模なソフトウェア開発では、生成AIの登場以前から、このトレーサビリティが品質を確保するための重要な仕組みとして活用されてきました⁽⁵⁾。

例えば、ある要件が設計や実装へ正しく反映されているか、必要なテストが用意されているかを確認するためには、それぞれの成果物の対応関係を把握する必要があります。また、要件や設計を変更した際には、その変更がどの実装やテストに影響するのかを把握しなければ、修正漏れや予期しない不具合につな

* パイプコーディング：生成AIと自然言語でやり取りしながら、コード生成・修正を繰り返してソフトウェアをつくる開発スタイル。

がります。

そして生成AI時代には、このトレーサビリティが新たな役割を持ちます。変更対象から関連する成果物を辿ることで、大規模なソフトウェア全体を生成AIに与えることなく、その作業に必要な情報だけを選択して与えられるようになるためです。

つまりトレーサビリティは、従来の大規模開発において品質を守るための仕組みであると同時に、生成AIを大規模なソフトウェア開発へ適用するための基盤にもなります。

大規模なソフトウェアを生成AIによって継続的かつ安全に改善していくため、トレーサビリティの重要性はこれまで以上に高まっています。

めざす世界

私たちがめざすのは、要件・設計・実装・テストを自動的に解析してトレーサビリティを構築・維持し、さらに生成AIがその情報を活用することで、大規模なソフトウェアであっても、低コストかつ安全に継続的な

改善を実現できる世界です（図1）。

人手でトレーサビリティを管理する場合、ソフトウェアが変更されるたびに、その対応関係を確認し、更新する必要があります。一方、要件書や設計書、ソースコード、テストケースなどを継続的に解析し、それらの関係性を自動的に抽出・更新できれば、開発者が意識的に管理しなくても、最新のトレーサビリティ情報を維持することが可能になります。

さらに、このトレーサビリティ情報を生成AIが活用することで、大規模なソフトウェア全体の情報を一度に生成AIへ入力する必要もなくなります。例えば、ある要件を変更する際には、その要件に関連する設計、実装、テストをトレーサビリティから辿り、その作業に必要な情報だけを生成AIへ与えることができます。

これにより、私たちは、大規模なソフトウェアにおいても、変更すべき箇所や影響範囲を低コストかつ正確に特定し、生成AIを活用した継続的で安全なソフトウェア改善を実現することをめざします。

トレーサビリティ自動解析技術

前述した世界を実現するため、私たちは、要件書・設計書・ソースコード・テストケースなどの成果物から、トレーサビリティ情報を自動的に構築する技術に取り組んでいます（図2）。簡単にいえば、大規模な成果物を「機能ごとのまとまり」に分け、異なる成果物の間で、どのまとまりどうしが対応しているのかを見つける技術です。本技術の中核となるのは、大規模な成果物から機能などの意味のある単位を抽出する自動分割手法と、異なる開発工程の成果物間にある依存関係を推定する自動関連付け手法です。これら2つの技術を組み合わせることで、成果物の自動分割と自動関連付けを行い、要件・設計・実装・テストを横断したトレーサビリティを構築します。

成果物の自動分割

トレーサビリティを構築するため

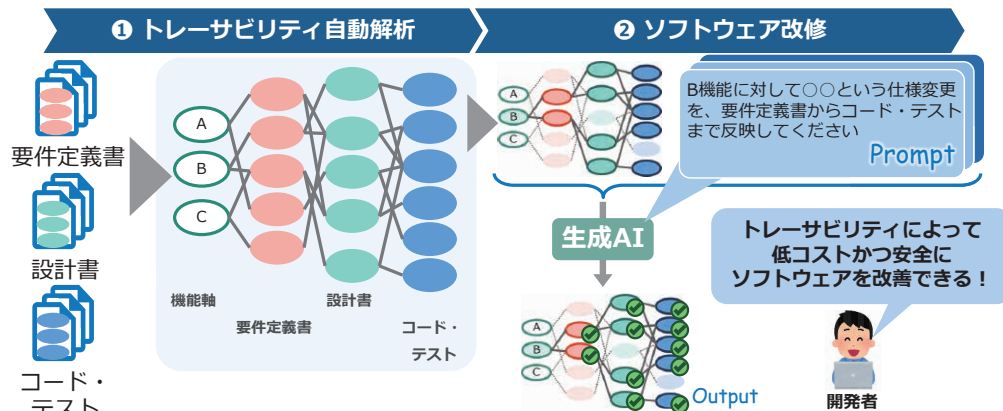


図1 めざす世界

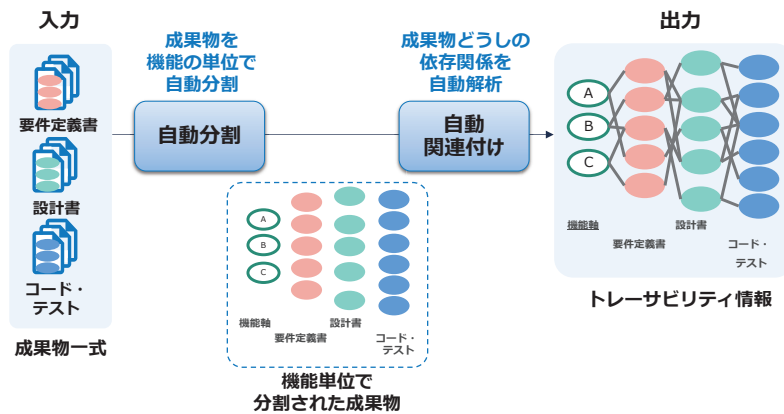


図2 トレーサビリティ自動解析技術

には、まず大規模な成果物を、要件や機能などの意味のある単位に分割する必要があります。例えば、1つの設計書に「ログイン」「ユーザ登録」「パスワード変更」といった複数の機能の説明が含まれている場合、それぞれを別の機能のまとまりとして取り出します。しかし、既存のテキスト分割⁽⁶⁾では、文章中の単語の出現頻度や類似度などに基づいて分割位置を決定するため、文書ごとに異なる情報量や構造に応じてしきい値を調整する必要があります。特に、1つの文書に複数の機能が混在する大規模な成果物では、固定的な基準によって適切な粒度（分割する単位のこと）で分割することは困難です。

そこで、文章が「1つの機能だけに関連している度合い」を表す独立スコアを用いて、粒度を自動的に調整します。

まず文章を一定の単位に分割し、それぞれの文章が各機能とどの程度関連するかを算出します。そのうえで、特定の機能との関連度が他の機能と比較して際立って高い文章ほど、

1つの機能を明確に表していると思われる、高い独立スコアを与えます。

文章をさらに細かく分割した際に独立スコアが向上する場合は、より明確な機能単位に分離できたと判断して分割を続けます。一方、独立スコアが低下する場合は、必要以上に細分化されたと判断して分割を停止します。

これにより、文書ごとにしきい値を細かく調整することなく、内容に応じた機能単位の分割を可能にします。

成果物の自動関連付け

次に、分割された成果物どうしの依存関係を自動的に解析します。例えば、「ユーザはパスワードを変更できる」という要件に対して、パスワード変更画面の設計や、その処理を実装するソースコードが対応します。自動関連付けでは、こうした異なる成果物の対応関係を自動的に見つけます。ここでいう依存関係とは、例えば、ある要件を変更した際に、

それに対応する設計やコードも変更する必要がある、といった関係です。

単純に文章どうしの類似度を比較するだけでは、異なる工程の成果物を正確に関連付けることは困難です⁽⁷⁾。一般に、要件から設計、実装へと開発が進むにつれて情報が具体化・詳細化し、工程間で情報量に大きな差が生じるためです。

そこで本技術では、下流工程の成果物から上流工程に相当する情報を復元し、比較する情報の粒度をそろえたうえで依存関係を判定します。

例えば、要件と設計を関連付ける場合、まず要件から比較に必要な要点を抽出します。一方、設計からは、その設計がどのような要件を実現するためのものなのかに相当する情報を復元します。そして、上流工程に相当する粒度へそろえた情報どうしを比較することで、要件と設計の依存関係を評価します。

同様に、設計とコードなどについても、下流工程の詳細な情報を上流工程に相当する情報へ変換して比較することで、工程間の情報量や具体

度の違いによる影響を抑えます。

この自動分割手法と自動関連付け手法を組み合わせることで、大規模な成果物から機能単位の依存関係を継続的に抽出し、要件・設計・実装・テストを横断したトレーサビリティを低コストで構築・維持することが可能になります。

■活用事例1：要件変更に伴う影響

分析・修正支援

トレーサビリティを活用することで、要件変更に伴う影響範囲の特定から、設計の修正、レビューまでを一連の流れとして支援できます（図3）。

例えば、ある要件が変更された場合、従来は担当者が関連する設計書を確認し、どの個所を修正すべきかを人手で特定する必要がありました。大規模なソフトウェアでは、関連する設計や機能が複数の文書にまたがることも多く、影響個所の見落としや確認コストの増大につながります。

そこで、要件と設計の間に構築したトレーサビリティを利用し、変更

された要件から関連する設計個所を自動的に抽出します。さらに、その情報を生成AIに与えることで、変更内容に応じた設計の修正案を作成できます。修正後は、変更された要件と設計の整合性や、関連個所への反映漏れがないかを生成AIが確認し、レビューまで支援します。

これにより、要件変更のたびに大規模な成果物全体を確認する必要がなくなり、影響分析から修正、レビューまでにかかる時間とコストを削減しながら、安全なソフトウェア改善を実現できます。

■活用事例2：ドキュメント最新化

長期間運用されているソフトウェアでは、ソースコードは継続的に修正されている一方で、設計書や仕様書などのドキュメントが十分に更新されず、実際のシステムと記載内容が乖離していることがあります。その結果、ドキュメントが形骸化し、当時の設計や実装の経緯を知る一部の担当者に仕様理解が依存するなど、システムのブラックボックス化につ

ながります。

この課題に対して、ソースコードと設計書・仕様書の間にトレーサビリティを構築します。これにより、ソースコードの各機能がドキュメントのどの記述に対応しているのかを把握できるようになります。さらに、実際のソースコードの内容と対応するドキュメントを比較し、記載が古くなっている個所や不足している個所を自動的に特定します（図4）。

特定した差分を基に生成AIがドキュメントの修正案を作成することで、現在のソフトウェアの実態に合わせて仕様書や設計書を効率的に最新化できます。これにより、長年の改修によって仕様が分からなくなったシステムについても、既存のソースコードから仕様を再整理し、保守・改修を行う開発者のシステム理解を支援できます。

今後の展望

現在、トレーサビリティの自動構

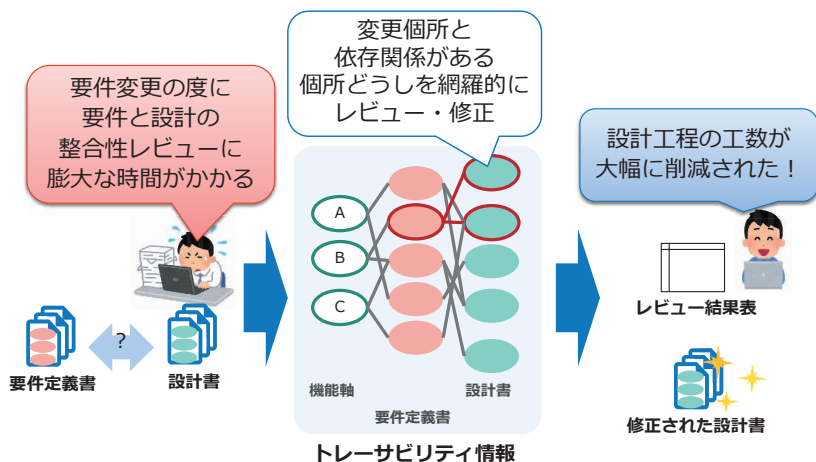


図3 活用事例1

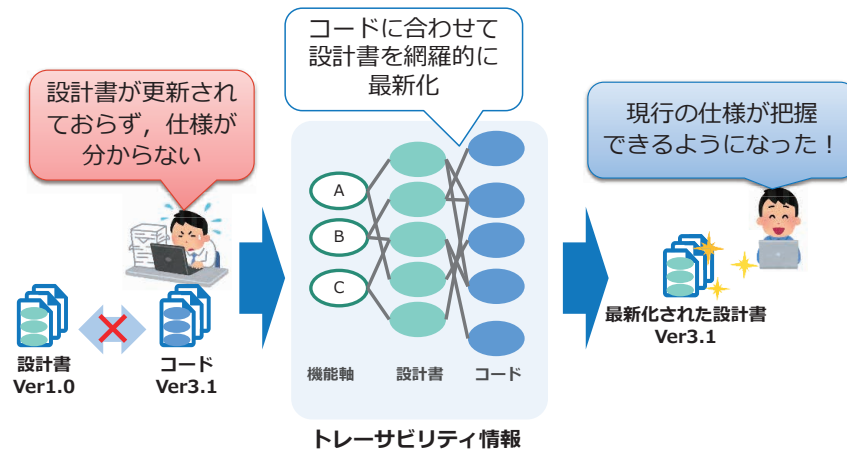


図4 活用事例2

策については、要件・設計・ソースコードまでを対象として、その実現可能性を確認しています。これにより、要件変更に伴う設計・実装への影響範囲の特定や、生成AIを用いた修正支援が可能になりつつあります。

今後は、この対象をテストまで広げていきます。要件・設計・実装・テストを一貫して関連付けることができれば、要件を変更した際に、修正すべき設計やソースコードだけでなく、追加・修正すべきテストケースまで自動的に特定できるようになります。さらに、変更内容に応じたテストの生成や実行、テスト結果の確認まで生成AIが支援することで、設計・実装の修正にとどまらず、その変更が要件どおりに正しく実現されているかを確認する品質保証まで含めた自動化が可能になります。

そして最終的には、要件書や設計書、ソースコードといった成果物に明示的に記録された情報だけでなく、なぜその設計を選択したのか、なぜ

その変更を行ったのかといった設計判断や変更理由、開発者の意図まで、成果物と関連付けて蓄積できるようにします。こうした情報を活用することで、開発者が要件を変更すると、トレサビリティから影響範囲を辿り、過去の設計判断や変更理由、開発者の意図を踏まえながら、設計・実装・テストの更新と検証を連動して行えるようになります。これにより、大規模なソフトウェアにおいても、設計の背景や意図を継承・活用しながら品質を維持し、迅速かつ継続的にサービスを改善できる開発環境を実現します。

■参考文献

- (1) C. E. Jimenez, et al.: "SWE-bench: Can Language Models Resolve Real-world Github Issues?," The Twelfth International Conference on Learning Representations, 2024.
- (2) <https://aclanthology.org/2024.tacl-1.9/>
- (3) <https://aclanthology.org/2023.emnlp-main.151/>
- (4) <https://www.iso.org/standard/72089.html>
- (5) J. Cleland-Huang, O. Gotel, and A. Zisman: "Software and Systems Traceability," Springer London, 2012.

- (6) M. A. Hearst: "TextTiling: Segmenting Text into Multi-paragraph Subtopic Passages," Computational Linguistics, Vol. 23, No. 1, pp. 33-64, 1997.
- (7) A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora: "Recovering traceability links in software artifact management systems using information retrieval methods," ACM Transactions on Software Engineering and Methodology, Vol. 16, No. 4, Article 13, 2007.



倉林 利行

NTTコンピュータ&データサイエンス研究所では、生成AIによる開発の高速化だけでなく、品質も大切にしています。大規模なソフトウェアでも安心して生成AIを活用し、継続的に改善できる世界をめざして研究開発を進めています。

◆問い合わせ先

NTTコンピュータ&データサイエンス研究所
革新的コンピューティングアーキテクチャ
研究プロジェクト