EUC ツールの近代化における生成 AI 活用手法の検討

米山 基 [1)], 柳 利佳子 [1)], 佐藤 亙人 [1)], 山田 晃久 [1)], 神馬 豊彦 [1)2)]

1) 株式会社早稲田大学アカデミックソリューション

2) 香川大学大学院 創発科学研究科

m.yoneyama@w-as.jp

An Examination of Methods for Applying Generative AI to Modernize EUC Tools

Motoi Yoneyama[1)], Rikako Yanagi[1)], Arito Sato[1)], Teruhisa Yamada[1)], Toyohiko Jimma[1)2)]

1) Waseda University Academic Solutions Corporation

2)Graduate School of Science for Creative Emergence, Kagawa University

概要

多くの組織で技術的負債となっている Microsoft Access 等で作られた業務ツールは、バイナリ形式で生成 AI が直接解釈できない点がモダナイゼーション（近代化）の障壁であった。本稿では、ツール内オブジェクトを解析可能なテキストへ変換し、生成 AI に入力することで Python コードを半自動生成するモダナイゼーションの手法を提案する 。生成されたコードは、専門家が AI と対話しながら修正するアプローチで品質を担保する。実務ツールを用いた移植では、手動での移行に比べ約 90% の工数削減を達成し、本手法が実用的なアプローチであることを定量的に実証した。

1 はじめに

大学をはじめとする多くの組織では、かつて End-User Computing (EUC) 推進の一環として、Microsoft Access（以下、Access）に代表されるローコードツールが広く導入された。これらは、情報システム部門に依存することなく、業務に精通した現場担当者が自らの手で迅速に業務ツールを開発できるという利便性をもたらした。しかし、長年にわたる運用と改修の結果、これらの業務ツールは深刻な技術的負債を抱えており、その対応は喫緊の課題である。

課題は大きく二つに大別される。第一に、技術的負債の蓄積である。業務ツールは度重なる制度変更への対応を経てロジックが複雑化し、仕様を記録したドキュメントも不十分、あるいは存在しない場合が多い。また、開発者の異動や退職により現場での維持管理が困難となり、結果として情報システム部門が Access 等の特定スキルを持つ人材を確保し、メンテナンスを継続せざるを得ないという組織的な課題も顕在化している [1]。第二に、プラットフォームとしての将来性である。例えば Access により作られたツールはその性質上、Git[2] 等を用いたバージョン管理やチーム開発といった現代的な開発手法をとることができず、また昨今注目を受ける生成 AI による開発支援の恩恵を受けにくい。

この状況を解決するアプローチとして既存ツールを Python 等の現代的なプログラミング言語へ移植する「モダナイゼーション」が考えられる。しかし組織内に存在する多数の業務ツールをすべて手作業で移植することは、膨大な工数を要するため現実的ではない。

この課題に対しレガシーシステムのモダナイゼーション、特に COBOL 等で構築された基幹システムを Java 等へ移行する文脈で生成 AI を活用する研究が近年注目を集めている。Solovyeva らによる PL/SQL から Java への変換に関する研究 [3] や、Lachaux らが開発した言語間翻訳モデル TransCoder[4] など、その基礎技術は確立されつつある。

ただし、これらの先行研究の多くは、ソースコードがテキストファイルとしてアクセス可能であることを前提としている。 Access で作られた業務ツールは、VBA コード、 SQL クエリ、 GUI 定義といった複数の要素を内包するバイナリ形式のコンテナである。このため、生成 AI が直接その内容を解析することはできない。このソースコードの非可読性が、業務ツールをモダナイゼーションするという段階において大きな障壁となっている。

そこで本稿では、Access で作られた業務ツール（以下「Access 業務ツール」）内の全オブジェクトを、それらの関係性を維持したまま生成 AI が解釈可能なテキスト形式へ変換し、それを用いて Python コードへと半自動的に生成する新たなモダナイゼーションの手法を提案する。2 章では提案手法で使用する開発環境について述べる。3 章では提案手法の手順について述べる。4 章では提案手法による移植作業の結果について述べる。5 章は考察を述べる。6 章はまとめを述べる。

2　提案手法で使用する開発環境

本稿の提案手法では、Access 業務ツール内のオブジェクトを生成 AI が解析可能なテキスト情報に変換・出力をおこない、出力されたテキスト情報と Access 業務ツールの利用手順書をもとに Python のコードを生成する。生成された Python コードは、生成 AI との対話によりデバッグと修正がおこなわれ、業務で利用可能なツールとして開発される。図 1 は、提案手法を構成するツール、開発環境のコンポーネント図を示す。提案手法は、Access 業務ツール内オブジェクトを

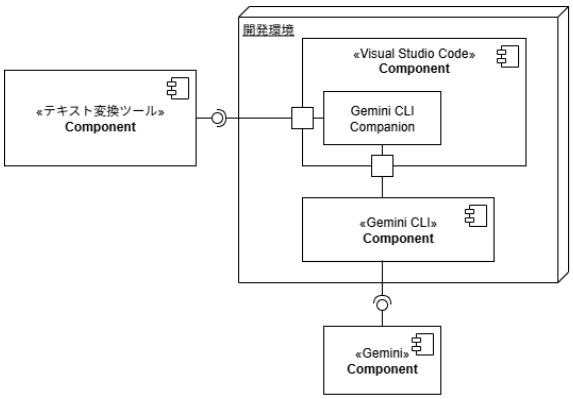

図 1　提案手法のコンポーネント図

生成 AI が解析可能なテキスト情報に変換・出力をおこなう「テキスト変換ツール」、出力されたテキスト情報と Access 業務ツールの利用手順書をもとに Python コードを生成し、生成 AI との対話によるデバッグと修正をおこなうための開発環境から構成される。テキスト変換ツールは、Access の VBA（Visual Basic for Applications）、開発環境は Microsoft 社の高機能コードエディタである Visual Studio Code（以下，VS Code）[5]、Google 社の生成 AI モデル「Gemini」をコマンドラインから利用可能にする「Gemini CLI」[6] と，VS Code 上で Gemini CLI を円滑に操作するための拡張機能「Gemini CLI Companion」[7] により構成される。

テキスト変換ツールは、Access 業務ツールを構成する全てのオブジェクト（クエリ、テーブル、フォームなど）を保ったままプレーンテキスト形式のファイル群として出力する VBA プログラムである。オブジェクトの種類ごとに、以下の手法でテキスト化を行う。

- テーブル
  データを格納する表形式のオブジェクト。Access 業務ツール内のテーブルの設計情報（TableDef オブジェクト）からフィールド名やデータ型などを取得し、SQL 文の CREATE TABLE 文の形式に再構成してテキストファイルに出力する。
- クエリ
  テーブルからデータを操作するための命令文。Access データベース内のクエリ定義（QueryDef オブジェクト）を参照し、そこに格納されている定義（SQL 文）をテキストファイルに出力する。
- フォームおよびレポート
  フォームは、データ入出力用の GUI。レポートは印刷用の出力レイアウトを定義するオブジェクト。Access の標準機能である Application.SaveAsText メソッドを利用し、レイアウト情報や配置されたコントロールのプロパティをテキストファイルに出力する。
- モジュール
  複雑な処理を記述する VBA コードの集合体。Access が持つプログラム開発環境（VBIDE）の機能を通じて直接コードを取得し、テキストファイルに出力する。
- マクロ
  一連の操作を自動化する簡易的な命令群。一度 Access の機能で VBA コードに変換した上で、モジュールと同様の方法でコードとして抽出し、テキストファイルに出力する。

3　提案手法の手順

図 2 は、本稿で提案するモダナイゼーション手法のユースケース図を示す。3 章では、提案手法の具体的な手順について、詳述する。

3.1　ステップ 1: Access 業務ツール内オブジェクトのテキストファイルへの変換

移植対象となる Access 業務ツールのオブジェクトを、テキスト変換ツールを使用しテキストファイルに

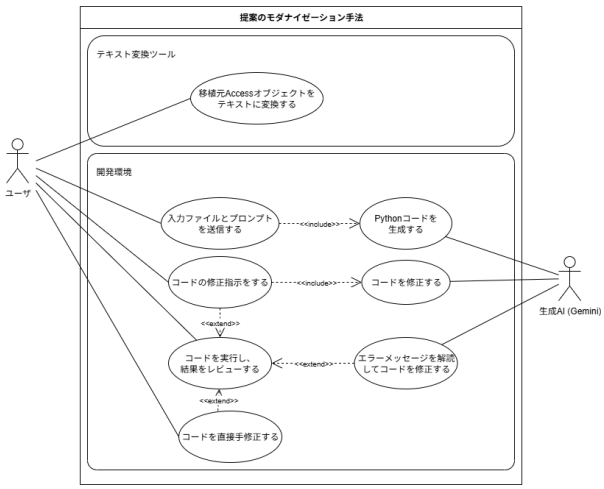

図 2　提案するモダナイゼーション手法のユースケース図

出力する。開発したツールのコードを VBA モジュールとして対象 Access 業務ツール内に埋め込み、実行することでそれぞれのオブジェクトの種類ごとにテキストファイルを出力する。

3.2　ステップ 2: テキスト情報と利用手順書からの Python コード生成

ローカル PC 上に特定の作業用フォルダを作成し、そこにステップ 1 で作成したテキストファイル群と、Acceess 業務ツールを使用する際の「利用手順書」をテキストファイル化したものを格納する。ステップ 1 で得られるテキストファイルは、あくまで Access 業務ツールの静的な構造を記述したものである。これに加え、実際の業務における操作手順やデータの流れが記載された利用手順書を Python コード生成のインプット情報とすることで、Access 業務ツールの振る舞いや業務ロジックを生成 AI が深く理解し、より精度の高い Python コードを生成することを促す。

次に、VS Code でこの作業用フォルダを開き、ターミナルから Gemini CLI に対して、Access 業務ツールのオブジェクトの種類ごとのテキストファイル、利用手順書のテキストファイルを読み込ませ、Python コード生成のプロンプトを送信する。これにより Python コードが生成される。

3.3　ステップ 3: 生成 AI との対話によるデバッグと修正

最終ステップでは、生成 AI が出力した Python コードを、生成 AI との対話によりデバッグと修正をおこなう。生成 AI には、事実に基づかない情報を生成する「ハルシネーション」や、同じ入力に対しても異なる出力が生成される「出力の非決定性」という技術的限界が存在し、一度の指示で完璧かつ正確なコードが生成されるとは限らない。そのため、生成されたコードを生成時と同様のローカル環境上で実際に動作させながら、その挙動を確認し、構文エラーやロジックの誤り、仕様との齟齬が発見された場合は、ステップ 2 と同様に VS Code 上のターミナル上から生成 AI との対話により修正を行うか、あるいは直接コードを修正する。このアプローチにより、ツールの動作を確認しながら、実用可能なアプリケーションを完成させる。

4　提案手法による移植作業の結果

4.1　移植作業の概要

提案手法での移植作業が、手動での場合と比較してどの程度効率的であるかを定量的に検証するために、実際に業務で使用されている Access 業務ツールをロジックや構造の複雑さ別に 3 本選び、それぞれを提案手法を用いて Python に移植した。対象とした Access 業務ツールは以下の通り。

- ツール A: 定型的なデータ抽出と集計が主体の比較的単純なツール。
- ツール B: 複数のクエリが連携する、やや複雑なデータ加工ロジックを含むツール。
- ツール C: ユーザー入力フォームや VBA ロジック、レポート出力を含むツール。

ステップ 2 における生成 AI へのプロンプト設計では、生成コードの品質担保と後続作業の効率化という二つの目標を掲げた。品質面では、AI の解釈精度を上げるための入力情報の定義、業務ルールやセキュリティ要件を遵守させる禁止事項の明記、そしてアーキテクチャの統一を図るためのライブラリ指定を行った（表 1 参照）。一方、効率化の観点からは、デバッグを容易にする詳細な日本語コメントや具体的なエラーメッセージの生成を指示するとともに、プロセスの一貫性を確保すべく実装手順を段階的に示した。

ステップ 3 における修正作業では、問題の種類に応じて二つのアプローチを段階的に適用した。まず、Python コード生成の初期段階で頻発する単純な構文エラーや変数名の誤りなどに対しては、プロンプトを工夫することで生成 AI 自身による自律的なエラー修正を促した。具体的には、「発生したエラーメッセージやログはすべてコンソール上に出力せよ」と指示することで、生成 AI が出力内容を自ら読み取り、エラーの原因を解釈してコードを修正するよう仕向けた。次に、プログラムはエラーなく動作するものの、元のツールの仕様と挙動が異なるロジック上の不具合が確認さ

表 1　使用ライブラリ一覧

ライブラリ名	機能
PyQt6	Access のフォーム機能を再現する対話型インターフェースを提供する。
Pandas SQLAlchemy	Access のクエリ機能に相当するデータ処理の中核を担う。
psycopg2	PostgreSQL データベースへの接続を確立し、SQL クエリの実行を仲介する。
fpdf	Access のレポート機能に相当し、PDF ドキュメントを生成する。

表 2　評価指標

評価指標	説明
移植元ツール規模	移植元 Access 業務ツールの規模を FP（ファンクションポイント）法で測定した値
移植元ツールオブジェクト数	移植元 Access 業務ツールに含まれるオブジェクトの数
手動移植想定制作工数	専門家がゼロから手動で移植した場合の想定作業工数（人日）。手動移植の想定製造工数の算定にあたり、情報処理推進機構（IPA）が示す再開発の標準生産性（FP/人月の中央値）[8] を用いて、移植元ツールの FP 値から開発全工程の総工数を算出し、算出された総工数に、同じく IPA が示す工程別工数割合の中央値から「製作工程」の比率 [9] を乗じることで、制作工程に限定した工数を求めた。
提案手法工数	提案手法で移植作業を行った際の実際の作業工数（人日）
レビュー・修正の割合	提案手法での工数のうち、レビュー・修正が占める割合
工数削減率	(手動移植想定制作工数 - 提案手法工数) / 手動移植想定制作工数 × 100 (%)

れた場合は、開発者が介在する対話ベースでの動作修正を進めた。このプロセスでは、開発者がローカル環境で生成されたツールを実行し、動作を確認する。問題点を発見した際には、その内容、すなわち「期待する動作」と「実際の動作」の差異を自然言語で生成 AI に伝え、修正を指示する。生成 AI が提示した修正案は、VS Code の差分表示機能によって変更箇所が明確に可視化されるため、開発者はその内容をレビューし、問題がなければ適用する。生成 AI の提案に問題がある際は、開発者が直接コードを修正することもあった。このような「実行・確認・指示・適用」のサイクルを繰り返すことにより、コードの品質を段階的に高めていった。結果として、対象とした 3 本のツールとも、元の Access ツールと同等の機能を持つ Python アプリケーション（フォームやレポートを含むものはそれも忠実に再現したもの）を完成させることができた。

4.2　移植作業結果の評価

移植作業の評価指標は表 2 の通り設定し、手動で移植作業を行った場合の想定工数と、提案手法での工数との比較を行った。

4.3　評価結果

各ツールに対する移植作業の結果を表 3 に示す。提案手法は全ての対象ツールにおいて手動移植に要する工数を大幅に削減、特に、フォームとレポートを備えた比較的複雑なツール C においても、88.1% という高い工数削減率を達成した。一方で、 AI 支援工数の内訳を見ると、レビュー・修正工程が全体の 50% 以上を占めている。

5　考察

移植作業の結果から、提案手法が移植元となる Access 業務ツールの複雑性に依らず、一貫して高い工数削減効果を発揮しており、その有効性を定量的に示しているといえる。特筆すべきは、単純なデータ処理が中心のツール A において、生成 AI との対話的修正がわずか 2 回で完了した点である。これは、定型的な構造を持つツールであれば、本手法によりほぼ自動的に移植が可能であることを示唆している。一方、多数のオブジェクトを含有する複雑なツール C では、17 回という修正作業が発生した。しかし、それでもなお手動開発と比較して大幅な工数削減を実現できた背景には、本稿で構築した開発環境がレビューおよびデバッグ作業の効率化に大きく貢献したことが挙げられる。この開発環境がもたらした利点は、主に以下の 3 点に集約される。第一に、修正から動作検証までのサイクルの高速化である。生成 AI がローカル環境の Python ファイルを直接編集するため、開発者は VS Code 上で生成 AI による変更点を即座に確認し、そのまま実行できる。これにより、シームレスな試行錯誤が可能となった。第二に、生成 AI による自律的なエラー修正機能の実現である。プロンプトの工夫により、生成 AI が Python の実行エラーを自ら解釈し修正案を提示するよう促した。この機能は、開発初期段階

表 3　手動移植と提案方法との工数比較

ツール	移植元ツール規模 (FP 値)	移植元ツールオブジェクト数	手動移植想定制作工数 (人日)	提案手法工数 (人日)	レビュー・修正の割合	工数削減率 (%)
ツール A	23.1	8	4.9	0.1	50.0%	98.0%
ツール B	37.1	28	7.9	0.8	66.6%	89.9%
ツール C	39.2	66	8.4	1.0	53.3%	88.1%

で頻発する構文エラー等の解消に特に有効であった。第三に、レビュー作業の負荷軽減である。VS Code の差分表示機能により、生成 AI が提案したコードの変更箇所が可視化される。これにより、開発者は変更点のみに集中してレビューすることが可能となり、意図しないコードの破壊を防ぎつつ、検証作業の負担を大幅に軽減できた。以上の要因が複合的に作用し、修正回数が比較的多い複雑なツールにおいても、最終的な移植工数を低く抑制できたと考察される。

しかし同時に、本手法における工数の半数以上が人間によるレビュー・修正工程に費やされるという課題も明らかになった。これは、生成 AI のハルシネーションや出力の非決定性といった本質的な限界を考慮すると、人間の監督が依然として重要な役割を担うことを示唆している。その際人間が求められるスキルは対象の複雑性に依存し、単純なツールの場合は移植元 Access 業務ツールの仕様に関する知識と処理結果を検証する能力が、複雑なツールではそれらに加えて生成 AI による支援を受けて Python の読解・修正ができる能力が要求された。

6　おわりに

本研究は、Access 業務ツールが抱える技術的負債の解決を目的とし、生成 AI を活用した半自動的なモダナイゼーション手法を提案・実証した。本手法の最大の貢献は、従来、生成 AI による直接解析が困難であったバイナリ形式のツールオブジェクトを、解析可能なテキスト群へ変換し、コード生成から人間によるレビューまでの一貫したプロセスを体系化した点にある。このアプローチにより、手動での移植と比較して最大 98.0% という大幅な工数削減を達成できることを定量的に示した。

今後の展望としては、レビュー負荷をさらに軽減するため、生成コードの静的解析やテスト自動化をプロセスに組み込み、手法の効率性と信頼性を一層向上させることがあげられる。

謝辞

本研究は、日々数多くの Access 業務ツールを開発、運用している早稲田大学アカデミックソリューション IT 推進部箇所支援チームの皆様の支えがあって成立したものです。チームメンバーに心から感謝いたします。

参考文献

[1] IBM, 技術的負債とは, https://www.ibm.com/jp-ja/think/topics/technical-debt, (2025. 09. 17 確認).

[2] Git, Git –distributed-is-the-new-centralized, https://git-scm.com/, (2025. 09. 17 確認).

[3] SOLOVYEVA, Lola, et al. Leveraging LLMs for Automated Translation of Legacy Code: A Case Study on PL/SQL to Java Transformation. arXiv preprint arXiv:2508.19663, 2025.

[4] LACHAUX, Marie-Anne, et al. Unsupervised translation of programming languages. arXiv preprint arXiv:2006.03511, 2020.

[5] Microsoft, Visual Studio Code, https://azure.microsoft.com/ja-jp/products/visual-studio-code, (2025. 09. 17 確認).

[6] Google, Gemini CLI, https://cloud.google.com/blog/ja/topics/developers-practitioners/introducing-gemini-cli, (2025. 09. 17 確認).

[7] Google, Gemini CLI Companion, https://marketplace.visualstudio.com/items?itemName=Google.gemini-cli-VSCode-ide-companion, (2025. 09. 17 確認).

[8] 独立行政法人情報処理推進機構 社会基盤センター, ソフトウェア開発分析データ集 2022, 独立行政法人情報処理推進機構, p.115, 2022.

[9] 独立行政法人情報処理推進機構 社会基盤センター, ソフトウェア開発分析データ集 2022, 独立行政法人情報処理推進機構, p.171, 2022.