6A-03

ソフトウェア開発における秘匿コードを保護するセキュアな開発環境のユーザビリティ評価

吉田 貴裕 † 鈴木 源吾 ‡ 堀川 桂太郎 § 飯村 結香子 ¶
開志専門職大学 † 開志専門職大学 ‡ 開志専門職大学 § 日本電信電話株式会社 ¶
斎藤 忍 ‖
日本電信電話株式会社 ‖

1. はじめに

フリーランスを含む人材確保の流動化とリモートワーク普及に伴いソフトウェア開発スタイルが多様化している．複雑なビジネスニーズ, セキュリティ要件の厳密化, 秘匿情報が増大する状況において, ソフトウェアサプライチェーンにおけるソースコード漏えいは重大なインシデントに直結する．秘匿漏えいは, 開発者の意図しない不注意に起因するケースが多く, 故意の漏えいはむしろ少ない [1]．そこで, 高い確率でヒューマンエラーが発生する前提で, 意図しない情報漏えいを防ぎ, 健全な開発プロセスを確保する二重三重の対策が不可欠である．筆者らが検討を進める MORDEn (Micro Organized Remote Development Environment, モルデン) は, 開発中のソースコードから「秘匿したいコード」を分離し, 秘匿コードを読み書きできない状態で, プログラムを作成／テスト／実行可能とする開発環境である [2]．これにより意図しない機密漏えいを防ぎ, 安心してプログラミングに集中する効果が期待できる．本稿は, MORDEn が提供するコード秘匿機能の確認とともに, 全コードに自由に触れることのできない状況が, 実際の開発者に何等かの不便さや非効率さをないかの検証を目的とする．実際にプログラマが MORDEn 上で作業に取り組み, 開発の操作性への影響を確認する検証実験を行った．§2 で評価対象である MORDEn について, §3 でユーザビリティ評価で用いた事例の概要, §4 でその結果と分析を述べ, §5 でまとめる．

2. セキュアド開発環境 MORDEn

MORDEn は, ソースコードを 3 つに分類する：(1) 公開可能コード (2) 内部隠ぺいコード (3) 完全隠ぺいコードである．(1) は通常の読み書き可能なコード, (2) は読み書きできないが存在は認識できるコード, (3) は存在自体も隠されたコードである．開発者が自由にアクセスできるのは (1) 公開可能コードのみである．図 1 に, MORDEn の構成を示す．クライアント－サーバ構成で, クライアント側でリモート開発を前提とする．クライアントリポジトリには, (1) 公開可能コードのみが含まれる．管理サーバ側のリポジトリには全てのコードが管理されている．開発者は MORDEn クライアントがインストールされたエディタを使って公開コードの編集やデバッグを行う．管理サーバは MORDEn クライアントにコーディング支援やデバッグ支援などの機能を提供する．MORDEn LSP プロキシはオープンスタンダードプロトコルである Language Server Protocol（LSP）[3] を使用する．開発者が公開コードを編集すると MORDEn LSP プロキシ→言語サーバ経由で, 編集内容をサーバリポジトリ内の公開コードに反映する．この際, 秘匿情報を漏えいするあらゆるアクションをブロックし, 秘匿コードを隠ぺいする．MORDEn DAP プロキシは Debug Adapter Protocol（DAP）[4] を用いて構成されていて, 作成したコードのテストやデバッグ操作において, コード内のエラー検出や修正を行うことを通じて秘匿コードが漏えいすることを防ぐ．MORDEn 環境により, 開発者は公開可能コードだけを編集することで, 秘匿コードにアクセスすることなく, 安全にプログラミング作業を進める．

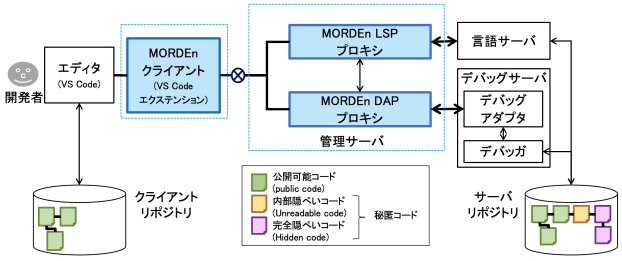

図 1. MORDEn の全体構成

3. ケーススタディの概要

MORDEn が提供するコード秘匿機能により, 全てのコードに自由に触れることのできない状況が, 実際の開発者に何等かの不便さや非効率さをもたらさないか検証する．秘匿ロジックを含む保険料金計算の Web サービス開発プロジェクトを設定し, Java 言語でロジック追加改修を要件としたコーディング作業実験を行なった．営業機密や企業間競争に影響する重要な保険料金計算ロジックは開発者にも知られたくない理由がある．4 つの完全隠ぺいコード, 1 つの内部隠ぺいコード, 5 つの公開可能コードから構成されるプロジェクトを設定し, Java 開発経験が 2 年以上のプログラマを, クラウドソーシングを活用して 14 名集めた．各被験者は MORDEn クライアントをインストール

Evaluating the usability of a secure development environment for protecting confidential code in software development
† Takahiro Yoshida, Kaishi Professional University
‡ Gengo Suzuki, Kaishi Professional University
§ Keitaro Horikawa, Kaishi Professional University
¶ Yukako Iimura, NTT
‖ Shinobu Saito, NTT

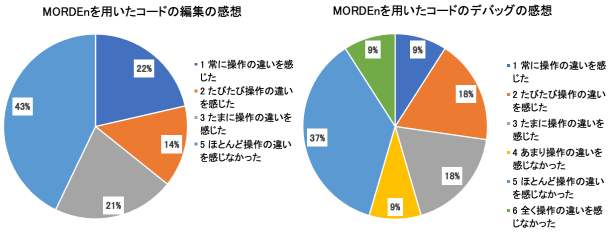

図 2. MORDEn 操作性のアンケート分析 (左: 編集時, 右: デバッグ時) .

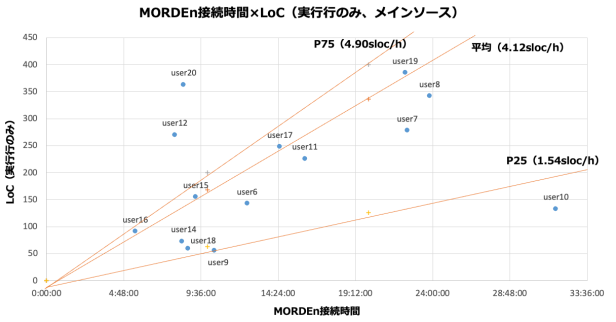

図 3. MORDENn の生産性評価. 横軸：MORDEn 接続時間. 縦軸：ソースコード追加ステップ数. 各点は生産性を示す. 実線は上からそれぞれ, 上側 25%, 平均, 下側 25% の生産性を表す [5].

した Visual Studio (VS) Code を用いて, ケーススタディプロジェクトの機能拡張やバグ改修を含む同一タスクを実施した. 前述のとおり, MORDEn はリモートの開発者が, MORDEn LSP プロキシおよび MORDEn DAP プロキシを介することで, 公開可能コードの編集, デバッグを実現する開発環境でありその都度, 編集ログを逐次保存する [2]. 評価試験では, 一人あたり 8 時間の作業時間を想定してタスクを実行し, 試験終了後に被験者にアンケート実施した. MORDEn の主機能について内部隠ぺいコードおよび完全隠ぺいコードが漏えいすることが一切なかったことを確認した. ユーザビリティ確認について, 被験者のアンケートの回答結果と編集ログを解析することで評価した.

4. ユーザビリティ評価の結果と考察

実験において, (2) 内部隠ぺいコード (3) 完全隠蔽コードが漏えいは見られなかったことから MORDEn 本来の機能が役割を果たしたことを確認できた. 被験者 14 名の MORDEn 操作ログと実験終了時のアンケート結果を分析した. アンケートでは開発環境の差異, 操作性の違いについて質問した. 図 2 は MORDEn と普段の開発環境との差異を示す. コード編集時「常に」「たびたび」操作に違いを感じた割合が 36% , デバッグ時「常に」「たびたび」操作に違いを感じた割合が 27% (「普段利用しないのでわからない」と回答した 3 名を除く) であった. 被験者は VS Code に MORDEn をインストールした環境下での開発であり, 操作性の違いなどの習熟に時間がかかることが予想されたが, アンケート結果では大きな違和感を感じていることは読み取れなかった. MORDEn のログ分析結果につ

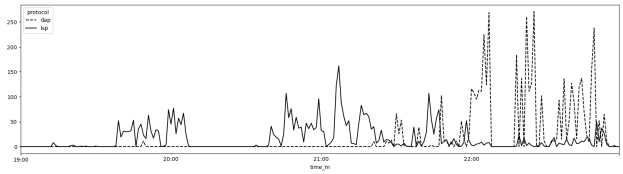

図 4. あるユーザの作業量. 横軸: 時刻, 縦軸:1 分あたりの MORDEn メソッド数. 実線/破線: LSP/DAP サーバ.

いて, 被験者の作業時間と追加したコードの行数を図 3 に示す. 上側と下側の実線に囲まれた領域が中心 50% の開発速度であることを表している. 大半の被験者がこの領域に位置し, 一般的な開発者と比べ平均速度が低い・分散が大きいといった特徴は読み取れない. ある被験者に対する時系列のメソッド発行量を図 4 に示す. メソッド数の大きなところほど被験者が活発に作業を行なったことを表している. 図から, 被験者がコード編集, デバッグと切り分け集中的に行なっていたことが読み取れる. MORDEn の使用有無でこのデータを比較すれば, 利便性低下の有無や作業への集中を妨げていないかなどの解析が可能となる.

5. まとめ

ヒューマンエラーによる秘匿ソースコード漏えいを防ぐ MORDEn を使った検証実験を行い, セキュリティと利便性の両立について評価した. 保険料金計算を秘匿する検証実験で, 秘匿コード漏えいがゼロ件, 操作性, 生産性ともに従来の開発環境と有意な差がないことが示された. 結論として, MORDEn 導入によるセキュリティとユーザビリティのトレードオフが限定的であることを意味し, MORDEn を実際の開発プロジェクトに導入する参入障壁は低いと考える.

謝辞

本研究は日本電信電話株式会社より支援を受けている (吉田, 鈴木, 堀川).

参考文献

[1] ソフトウェアベンダーが注意すべき新たな脅威「ソースコードの漏えい」とは, https://news.mynavi.jp/techplus/article/20230411-2648955/

[2] S. Saito, 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, Luxembourg, 2023, pp. 1864-1869.

[3] Language Server Protocol (LSP), https://microsoft.github.io/language-server-protocol/

[4] Debug Adapter Protocol (DAP), https://microsoft.github.io/debug-adapter-protocol/

[5] 独立行政法人情報処理推進機構 (IPA) 社会基盤センター,『ソフトウェア開発分析データ集 2022』, https://www.ipa.go.jp/digital/chousa/metrics/hjuojm000000c6it-att/000102171.pdf