システム/制御/情報，Vol. 66， No. 5， pp. 1-7， 2022

解　　説

機械学習を中心としたAI活用によるソフトウェアの品質保証

鷲崎　弘宜*

1. はじめに

社会の基盤としてソフトウェアシステムが大規模化および複雑化し，繋がり，さらに極端に不確実な時代においては，探索的な開発や頻繁な変更，拡張，環境適応が求められる．そうした中では，品質の作り込みや検証・評価・改善・管理の活動も自ずと探索的および適応的なものとなる．そこでは，汎化性能を備えた機械学習による新たな対象や状況へのデータ駆動の適応や，メタヒューリスティクスである遺伝的アルゴリズム（Genetic Algorithm; GA）による探索などの取組みが，最適化および効率化の面で有用であり，要求から保守に至るまで，品質の活動に対するAI活用が研究実践されつつある．

本稿では，AIの種類を主に機械学習およびGAを中心として，最初にソフトウェアの品質全般に関連するAI活用の全体を概観する．続いて，特に要求，設計・実装，テスト，デバッグ，他の品質管理および保守のプロセス別に，広く品質にかかるAI活用の取組みを解説する．

2. AI活用による品質活動の全体

[1][2]を参考に，プロセスおよび目的の観点から，プロダクトの広く品質に関わる活動に対する主要なAI活用をまとめた結果を第1表に示す．AI活用の目的について，[1]における分類に基づき次の三つとした: 対象（リソース，プロセス，プロダクトあるいはデータ）の属性の予測や見積り，属性の特定や抽出，対象の変換や生成．

プロダクトの品質保証を直接に扱うプロセスとしてはテストやデバッグおよび品質管理などが該当し，AI活用による検証や妥当性確認のためのテストの効率化を通じたプロダクトの品質改善や，プロダクトの欠陥予測，さらには品質評価および欠陥報告管理の取組みがある．

加えて，要求や設計・実装，保守のプロセスにおいても，AI活用を通じた品質の効率的な作り込みや改善の取組みがある．

以降において筆者の成果も交えて，プロセス別に主要なAI活用の取組みを解説する．

* 早稲田大学／国立情報学研究所／システム情報／エクスモーション

Key words: ソフトウェア品質保証，ソフトウェア品質評価，ソフトウェアテスト，機械学習，人工知能

3. 要求における品質へのAI活用

複雑な要求の文書が多く得られ続け，かつ変更されやすい状況で，特に機械学習に基づく自然言語処理を通じた効率的で適応的な処理が有用である．[3]では，要求工学全般におけるAI活用の取組みが要求獲得，要求分析，要求優先順位付けについて整理されている．特に要求分析では非機能要求の分類や検証，要約などにより要求の一貫性や無あいまい性といった品質を高めるとともに，以降において要求を正しく効率よく組み入れて検証可能とし，プロダクト品質の向上に寄与する．以降では要求の品質評価と分類の取組みを解説する．

3.1 要求の品質評価

要求仕様の代表的な品質として，IEEE/ISO/IEC 29148-2018では次の品質特性があげられている: 各要求個別の必要性，実装独立性，非あいまい性，一貫性，完全性，単独性，実現可能性，追跡可能性，検証可能性，および，要求の集合の完全性，一貫性，満足可能性，定義範囲の明確性．特に非あいまい性は，要求文書単独で自然言語処理により評価可能である．

例えば[4]では，長・短期記憶（Long Short-Term Memory; LSTM）をベースとした深層学習モデルの転移学習と対象領域におけるファインチューニングにより，文法や問題領域上のあいまいさを自動判定している．こ

第1表　プロダクト品質保証上のAI応用が活発な活動

目的	要求	設計・実装	テスト・品質管理	保守
予測	要求品質評価		テスト工数予測 欠陥予測 品質評価	保守工数予測
特定	要求分類 要求検証	アーキテクチャ自己適応 再利用	テスト優先順位 欠陥限局 脆弱性検出 欠陥報告管理	デザインパターン検出 追跡関係特定
変換	要求要約	自動プログラミング	テスト生成 自動プログラム修正	リファクタリング

こで，深層学習ではデータの規模が重要なため，機械的な逆翻訳（例えば英語→スペイン語→英語）を通じて訓練データを拡張している [4].

3.2 要求の分類

要求文書の特徴を機械学習により捉えることで，非機能要求かどうか，さらにはセキュリティ要求やユーザビリティ要求といった個別の品質要求へと自動分類し，大量の要求文書の効率的で誤りの少ない処理を支援できる．

例えば [5] では，語彙や構文上の特徴量を用いてサポートベクターマシン（Support Vector Machine; SVM）により自動分類している．[6] では，SVM を含む基本的な機械学習アルゴリズムの比較評価がまとめられている．

また自然言語処理では BERT（Bidirectional Encoder Representations from Transformers）[7]の登場以来，トランスフォーマーモデルを用いた大量の一般文書による言語表現の事前学習と対象領域におけるファインチューニングの取組みが活発であり，文脈を加味した分析が期待できることから要求文書へも応用されつつある．トランスフォーマーは，エンコーダ・デコーダとセルフアテンションをあわせたニューラル機械翻訳モデルである．BERT ではトランスフォーマーモデルを用い，ラベル付けされていないテキストについて一部をマスクしてその箇所を推測するタスクと，入力文の連続性の判定タスクを訓練することを通じて，文脈や文の関係性を把握可能とする．例えば [8]では，BERT を用いて高精度な非機能要求文書の分類を実現している．

4. 設計・実装における品質への AI 活用

品質要求が複雑，大規模あるいは変更されやすい状況では，設計・実装上の取りうる無数の選択肢の中で適切なものを人手で選択し続けることは困難であり，AI の支援が有用である．以降では特に，自己適応，再利用，自動プログラミングを通じた品質の確保や改善を解説する．

4.1 アーキテクチャの自己適応

組込みや IoT ソフトウェアシステムにおいて固定された設計および実装では，しばしば環境や状況の変化に応じて実行効率性やエネルギー効率性といった品質を著しく損なう．そこで AI により，望ましい品質を確保し続けるようにアーキテクチャを自己適応させる取組みがある．

例えば [9]では，IoT システムの特にエネルギー効率とデータトラフィックについて許容値を超えると予測できる場合に，強化学習により最適なアーキテクチャ適応パターンを選択し適応させている．こうした自己適応の取組みは，適応保守の一環とみなすこともできる．

4.2 コンポーネント再利用

ソフトウェアの設計および実装における実証済みのデザインパターンやコンポーネント，ライブラリ，フレームワークの再利用は，開発プロセスを効率化させるのみならず，信頼性をはじめとする品質の向上にも寄与する．ここで機械学習の応用は特に，無数に存在するコンポーネント群のうちで要求や文脈に合致するものの検索や，再利用性の評価などに有用である [10].

4.3 自動プログラミング（生成および合成）

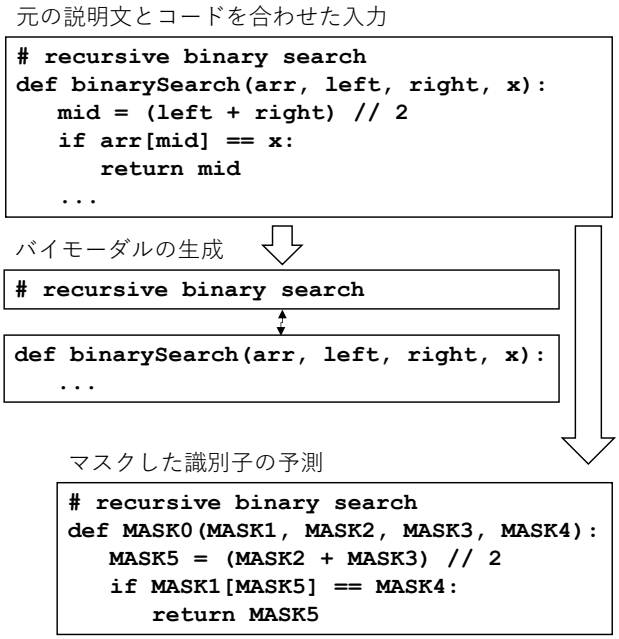
再利用と並び生成もまた，効率的かつ高品質な設計・実装に有用である．生成や合成を通じた自動プログラミングの取組みとして，GA による探索的な生成や，機械翻訳を通じた生成などがある．

GA は，個々の解の候補を遺伝子として表現し，生物の進化に着想を得て適応度関数に基づき選択した遺伝子について他の遺伝子との交叉や，突然変異などの操作を繰り返していくことで近似解を探索する．遺伝的プログラミング（Genetic Programming; GP）は，GA を拡張してプログラムの木構造による表現を遺伝子として扱い，解となるプログラムを進化的に生成させていく手法である．例えば [11]では，GA について対象プログラミング言語固有の遺伝子の構築や削除の仕組みを組み込み，さらに，プログラミング言語上で利用可能な命令を制限することで探索空間を狭めて，簡単な要求仕様を満足するコードを生成可能としている．GA に基づく取組みは，プログラミング言語の特徴を正確に組み入れられるが準備コストがかかり，また，制限された規模や内容を扱う．

機械翻訳による取組みは，トランスフォーマーを用いて大規模なプログラムソースコードの集合からコード表現の言語モデルを事前学習し，ファインチューニングを経て，要求文書からのコードの生成や補完，あるプログラミング言語から他の言語へのコードの翻訳，さらにはプログラム修正，要約・理解支援，類似コード片特定などを実現する．先駆的な取組みとして CodeBERT [12]および CuBERT [13] が知られている．例えば CodeBERT では，ソースコードとその内容を説明する自然言語記述のバイモーダル学習により，要求に基づくコードの検索やコードの生成，さらには逆方向にコードから説明文書の生成などを試みている．以降も短期間での発展が目覚ましい．例えば Code T5 [14]では，第 1 図に示すように，事前訓練において自然言語記述とコードの対応を学習するのみならず，識別子のマスクとその予測などの事前訓練を通じて，識別子を考慮した高精度なコード翻訳を実現している．また手法を比較評価するベンチマークデータセットも得られてきている [15]．こうした取組みは人手による特徴設計を要さず汎用のプログラミング言語を扱えるが，大規模かつ高品質なコードコーパスがカギとなる．例えば [16]では，GitHub 上の 5400 万リポジトリから得た計 159GB のコードにより言語モデルを獲得し，それを用いて関数名とコメントから関数の中身を自動補完するサービス GitHub Copilot を実現している．

5. テストにおける AI 活用

複雑，大規模あるいは変更されやすい状況においてテ



第 1 図　事前訓練タスクの一部の様子 [14]

ストケースを用いたプログラムの動的な検証にあたり，以下のすべての活動について AI 活用による最適化や効率化が取り組まれている．またセキュリティテストの一環としての脆弱性検出についても AI 活用が活発である．

- テストの計画: テスト工数や欠陥の高精度な予測
- テストの設計と実施:テストケース生成およびそれを組み入れた探索的なサーチベーストテスト
- テストの繰り返し: テストケース優先順位付けによる効率的なリグレッションテスト

5.1　テスト工数予測

テストに費やされる工数や労力が，プロダクト品質を最も決定づける因子であるという報告がある [17]．そこでテストの計画にあたり，プロジェクトの種別や要求，規模に応じた適切な工数の予測は重大な研究課題である．一般に，一定量の過去のプロジェクトデータがある状況において，機械学習に基づく予測は機械学習に基づかない予測（例えば単回帰分析）よりも優れていることが知られている．こうしたテスト工数予測は，特にプロジェクト全体のコストやスケジュールを計画する際や，テストチームを構成するにあたり有用である．

例えば [18] では公開データセットに対するテスト工数予測にあたり機械学習アルゴリズムとして事例ベース推論（Case-Based Reasoning; CBR），多層パーセプトロン（Multilayer Perceptron; MLP），サポートベクター回帰（Support Vector Regression; SVR），GP，決定木（Decision Tree; DT）を比較し，特に SVR，GP，DT が優れていることを報告している．

5.2　欠陥予測（バグ予測）

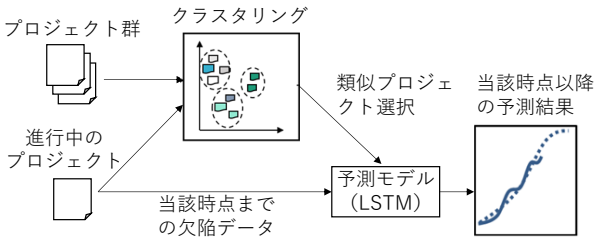
テストプロセスの実行中に必要な労力や活動を適切に管理するうえでは，対象プロダクトに対する適時かつ高精度な潜在欠陥（バグ）の予測が有用である．予測により重点的にテストすべき箇所の計画につながり，さらには，テスト計画全体の見直しや設計や実装の再検討，機能の絞り込みといった対応も可能となる．欠陥予測のモデルは，欠陥密度モデル（Defect density model）と信頼度成長モデル（Software reliability growth model; SRGM）の二つに大別できる [19]．

欠陥密度モデルは，コード行数や入出力といった主としてプロダクトの属性を用いて，機械学習アルゴリズムなどによりプロダクトのモジュールや部分ごとに潜在欠陥の有無や多さを予測する．障害を含みやすいモジュール予測 (Fault-prone module prediction) とも呼ばれる．例えば [19]では，不均衡データを扱う上で有用なコスト考慮型学習（Cost-Sensitive Learning; CSL）に基づいて拡張したラージマージン分布マシン（Large margin Distribution Machine; LDM）を用いて，欠陥を持つモジュールが全体の 7%～20%程度という不均衡なデータセットにおいて高精度な予測に成功している．

一方の信頼度成長モデルは，時系列上の欠陥検出データを用いて，プログラム全体において主に未来の特定の時刻までに発見される累積の欠陥数を予測する．代表的な予測モデルとして，欠陥検出過程の非同次ポアソン過程（Non-Homogeneous Poisson Process; NHPP）によるモデル化やその拡張がある．検出欠陥数に加えてプロダクトやプロセスの属性を入力とする予測モデルや，個々のモジュール別に予測するモデルもある [20]．さらに時系列データ上で，より複雑な固有のパターンを特定する取組みはデータ駆動 SRGM と呼ばれ，SVM や GA，ニューラルネットワーク（Artificial Neural Network; ANN）をはじめとする AI 活用が活発である [21]．例えば筆者らは第 2 図に示すように，クラスタリングで特定した過去の類似プロジェクト群の時系列の欠陥検出データを用いて LSTM モデルを訓練することで，新たな進行中のプロジェクトの将来の欠陥数を高精度に予測している [22]．

5.3　テストケース生成

無限に近い実行領域から，欠陥検出に役立つテストケースの有限の集合を適切に選択するにあたり，ランダムテスト，記号実行，サーチベーストテスト (Search-Based Testing; SBT)，組み合わせテスト（Combinatorial



第 2 図　類似プロジェクトによる欠陥数予測 [22]

Testing; CT）などの入力値生成手法がある［23］．特にSBTについて，有用な入力値の評価や探索にあたりAIの応用が活発である．

SBTとは，達成したい要件に対する達成度合いを定量的に評価できるように設計した評価関数に基づいて，ヒューリスティック探索アルゴリズムを用いて達成したい要件を満足するテストケースの集合を生成する方法である（第3図参照）．主に以下の手順A〜Dを踏む［23］．

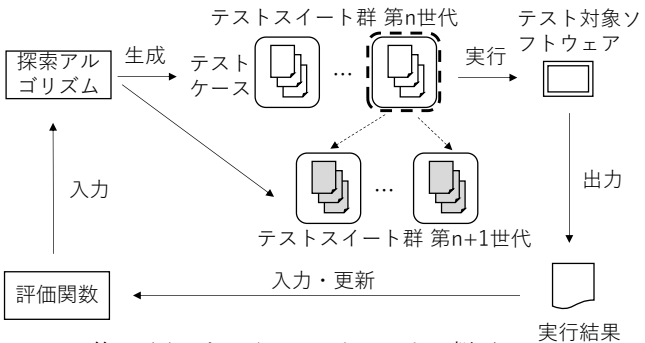
A) 要件の達成度合いを定量的に評価する評価関数を設計（例: プログラム経路網羅率，実行時間）
B) 用意したテストケース集合としてのテストスイートを入力実行し，評価関数の値を取得
C) 評価関数の値が優れているテストスイートを元に，ヒューリスティック探索アルゴリズムによって新規にテストスイートを生成（アルゴリズムの例: GA/GP，粒子群最適化，機械学習など）
D) 探索打ち切り条件を満たすまで，B，Cを繰り返し実行（例: 網羅率 ≧ 90%）

SBTは連続系の制御システムのテストについて活発に応用されている．例えば［24］では第4図に示すように，遺伝子を信号の振幅，時間幅，変化（例えばステップ関数やsin関数など）により構成し，それらの連結により信号列を生成している．そしてGPの適用により，遺伝子を組み替えていくことで様々な信号列を自動生成する．

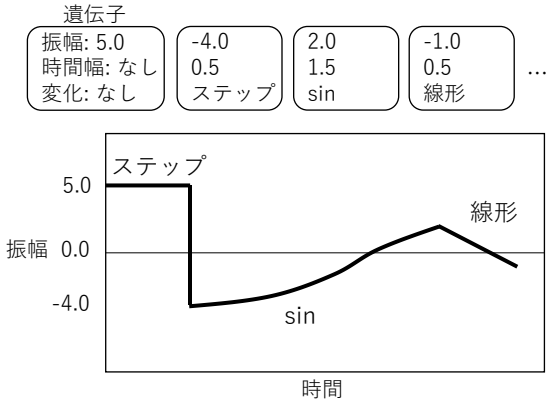
Simulinkのようなモデルベース開発の仕組みとの連動も活発である．[25]では，Simulinkモデルテストの多様性を考慮したSBTによるテスト入力値の自動生成を実現している．具体的には，Simulinkモデルの構造上の網羅性では，欠陥発見にあたり不十分であるため，出力信号間の距離や特徴の違いに基づき出力が多様となるように生成している．

SBTの実運用にあたり，探索効率の向上が課題となり様々な工夫が試みられている．例えば［26］では自動車制御システムのSBTにあたり，現実的ではない入力信号の組み合わせ（シナリオ）に基づいたテストを防ぐため，実走行データより得られる実走行条件分布に基づいてシナリオを評価し，探索空間を重点化することで探索効率の向上を図っている．

また，複数のパラメータに割り当てる値の組み合せを



第 3 図　サーチベーステストの概要



第 4 図　遺伝子に基づく入力信号列の例［24］

テストするCTについても，組み合わせを網羅かつ数を抑えたテストケース群を得るうえでAI応用の取組みがある．例えば［27］ではGAを用い，［28］では他のメタヒューリスティクスアルゴリズムと強化学習を併用している．

5.4　テスト優先順位付け

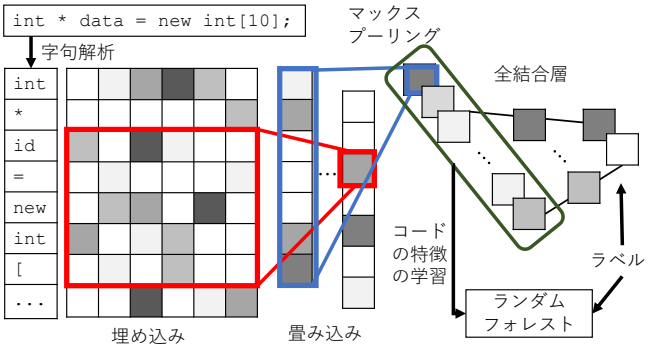
リグレッションテストは，プログラムの変更が既存機能に悪影響を与えていないことを，変更前に成功していた既存テストケース群の実行により確認することである．開発の大規模化や，アジャイル開発の中で頻繁なビルドを繰り返す継続的インテグレーション（Continuous Integration; CI）が採用されるにつれ，一度のビルドとリグレッションテストにかけられる工数や時間は限られつつある．そうした中でテストケース群を優先順位付けして順位の高いものから優先テスト実行する取り組みが活発化しつつある．最適解の探索はしばしばNP困難な問題であるなかで，機械学習による優先順位付けと選択支援の取り組みが活発である［29］．

5.4　脆弱性検出

セキュリティ上の既知の脆弱性を機械学習により大規模なコード上で自動的に検出する取り組みが活発である．例えば［30］では第5図に示すように，畳み込みニューラルネットワーク（Convolutional Neural Network; CNN）および回帰型ニューラルネットワーク（Recurrent Neural Network）を用いてコードの分散表現上で特徴を特定し，共通脆弱性タイプ一覧（Common Weakness Enumeration; CWE）の脆弱性をラベル付けし，ランダムフォレストにより検出している．ここで機械学習では訓練データの質と量が課題となるため，［30］では，ルールベースの静的解析ツールにより得られる警告や問題を個々のCWEと紐づけておき，同ツールの適用結果に基づいて脆弱性を含む箇所の正解データを大規模に得ている．

6.　デバッグにおけるAI活用

主としてテスト結果を活用してコード上で欠陥の原因箇所を特定する欠陥限局や，その結果を応用した自動プログラム修正についてAIの応用が活発である．



第 5 図　コード上の特徴の学習と脆弱性検出 [30]

6.1　欠陥限局（バグ局所化）

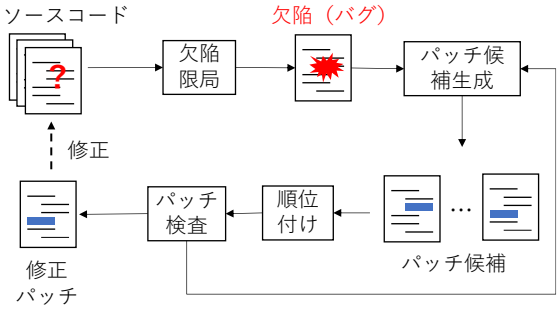
欠陥限局（Fault Localization; FL）では，動的なテスト実行結果や静的なコード解析結果，欠陥報告に対するコードの類似度などを用いて，ステートメントや関数といったコード中の各プログラム要素の単位で欠陥の原因としての疑わしさ（Suspicious）を計算する．例えばテスト実行結果に基づくと，失敗したテストケースにより実行された要素ほど疑わしい．また静的な解析結果に基づくと，複雑な要素ほど疑わしい．こうした疑わしさによる順位付けに基づき，人手あるいは後述の自動的なプログラム修正を効率よく進めることを目的する．

しかしあらゆる状況に適した単一の FL 手法は得られていない．そこで，異なる特徴量に基づく様々な疑わしさの値を組み合わせて，教師あり学習により効果的な順位付けを学習するランク学習（Learning-to-Rank）による FL の取組みが研究されている．例えば [31]では，拡張した MLP を用いて，テスト実行結果や複雑さ，類似度などの様々な特徴量により高精度な FL を実現している．

6.2　自動プログラム修正

自動プログラム修正（Automated Program Repair; APR）は，テストなどで検出された欠陥を自動修正する技法である．典型的には第 6 図に示すように，コードに対して FL 等により欠陥の位置を特定したうえで，修正パッチ候補群を生成し，機械学習などにより順位付けし，テストにより正しさを検査の上で最終的に修正パッチを得る．

修正パッチ候補の生成は，過去の修正傾向に基づいて人手で準備した変換パターン・ルールやテンプレート，過去の修正実績への機械学習適用により自動的に学習した変換，GA/GP による探索的な変換，およびそれらの組合せなどにより実現される．例えば [32]では，変数の型の拡大変更や条件文の変形，メソッド実行の変更・挿入といった種々の変換ルールを用いてパッチ候補群を生成したうえで，事前に訓練済みのロジスティック回帰モデルにより順位付けている．[33]では，エンコーダ・デコーダの組合せにより，過去の欠陥修正時の修正前後のプログラム抽象構文木の対応関係を周囲の文脈を含めて学習しておき，その適用により新たなコードについてパッチ候補を生成し，CNN により順位付けている．



第 6 図　自動プログラム修正の概要（[31]より改変）

現状の APR 手法の多くは，原因箇所が限定された単純な欠陥の修正にとどまるが，十分な量と質のテストケースを準備することで，単純な欠陥の修正を AI に委ね，開発者は複雑な問題に取り組む形態を期待できる．[34]では，オープンソースソフトウェアの開発において AI による自動修正案を提出して，他の開発者が AI 由来と気付かずに受け入れられる場合があることを報告している．これは，AI と人間による価値共創の姿といえる [35]．

7.　その他の品質管理における AI 活用

品質管理においてテストやデバッグ以外では，品質特性の評価や欠陥報告管理に AI 活用の取組みがある．

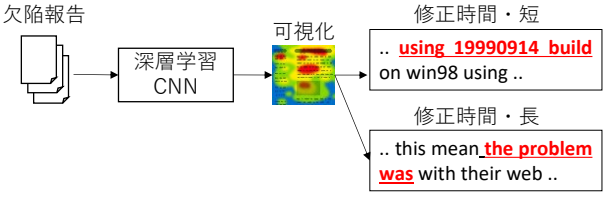
7.1　品質評価および予測

欠陥の扱いに基づく機能適合性や信頼性以外にも，機械学習による様々なプロダクト品質特性の評価や将来の予測が研究実践されている [36]．機械学習は，品質の評価基準の調整や適応にも有用である．筆者らは保守性評価にあたり，開発者によるレビュー結果に基づく DT による教師あり学習により，問題領域やプラットフォーム固有の特徴を基準へ組み入れる手法を実現している [37]．

7.2　欠陥報告管理

欠陥報告（バグレポートや不具合票）が数多く得られ続ける状況において，要求文書の扱いと同様に，機械学習に基づく自然言語処理を通じた処理が有用である．例えば筆者らは BERT の応用により，重複する欠陥報告を特定する手法を実現している [38]．

また，特に深層学習の活用において説明性が意思決定のカギを握る．筆者らは第 7 図に示すように，CNN により欠陥報告群を短時間で修正されるものと長時間かかるものへと分類するにあたり，可視化手法を適用して分類モデルにおいて欠陥報告の表現の具体さを根拠としており，具体的であると修正時間が短く（例: “using 19990914 build”），抽象的であると長い傾向を特定している [39]．



第 7 図　欠陥報告の分類における判断根拠の可視化 [39]

8. 保守における品質への AI 活用

ソフトウェアの保守は ISO/IEC 14764:2006 に基づき是正保守，予防保守，適応保守，完全化保守，緊急保守に分類できる．上述の APR は，是正保守の支援手法とも捉えられる．また予防保守にあたっては，設計・実装上の好ましい形や好ましくない形を検出し，それを改善していくことが重要である。それらについて AI 活用がある．

8.1 デザインパターンなどの検出

頻出する設計上の問題を解決する典型的な好ましい設計の形はデザインパターンと呼ばれ，その適用結果をコード中に検出することで，設計意図や品質を把握し，以降の保守を効果的に進められる．ただし具体的な適用の形は，対象の制約や要求に応じて様々であり，しばしばルールベースでの検出は困難である．そこで筆者らは，既知のデザインパターンの複雑さや規模といった特徴量の測定値の ANN による学習を通じて多様な適用結果から高精度に自動検出する手法を実現している [40]．

類似の検出の取組みとして，保守上の問題となる複雑さおよび暫定的な好ましくない設計など（デザインスメルやコードスメル，技術的負債などと呼ばれる）を AI 活用による自動検出する取り組みも活発である [41][42]．

8.2 設計や実装の改善と追跡

スメルや技術的負債の解消に向け，外部への振る舞いを保ったまま内部を改善するコードのリファクタリングが有用であり，AI が活用されている [43]．改善にあたりしばしばコードに限らず様々な成果物を辿って扱う必要があるため，成果物間の追跡関係（トレーサビリティリンク）の特定についても AI 活用が活発である [44]．

9. おわりに

本稿では，品質保証に関する AI 活用の全体を概観し，特に応用の進展が目覚ましい品質保証活動を解説した．

データに基づく帰納的な技術的活動を進めるうえでは，データの質や量を確保する基盤，それにより訓練されるモデルの品質（特に予測性能や頑健性，説明性），さらには，得られる結果の目標への適合性が成功のカギとなる．そこで，頑健性や説明性の向上といった機械学習・AI の品質保証技術の併用も必要であり，AI による品質保証と，AI の品質保証を車の両輪として相乗効果を発揮させながら発展的に取り組む必要がある．例えば筆者らは，機械学習デザインパターンの体系化 [45]や高信頼化に向けた要求分析 [46]に取り組んでおり，そうした成果と AI 応用の取組みとの接続を展望している．

謝 辞

一部成果は次の助成を受けた： JST 未来社会 JPMJMI20B8 eAI，JSPS 二国間交流事業 120209936，科研費 国際共同研究加速基金 21KK0179，enPiT-Pro Smart SE.

参 考 文 献

[1] D. Zhang, et al., “Machine Learning Applications in Software Engineering,” Series on Software Engineering and Knowledge Engineering, 16, 2005.

[2] S. Shafiq, et al., “A Literature Review of Using Machine Learning in Software Development Life Cycle Stages,” IEEE Access, 9, 2021.

[3] K. Kaur, et al., “A Review of Artificial Intelligence Techniques for Requirement Engineering,” Computational Methods and Data Engineering, 2021.

[4] I. M. Subedi, et al., “Application of Back-translation - A Transfer Learning Approach to Identify Ambiguous Software Requirements,” ACMSE 2021.

[5] Z. Kurtanović, et al., “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning,” RE 2017.

[6] P. Talele, et al., “Classification and Prioritisation of Software Requirements using Machine Learning – A Systematic Review,” Confluence 2021.

[7] J. Devlin, et al., “BERT: Pre-training of deep bidirectional transformers for language understanding,” arXiv:1810.04805, 2018.

[8] T. Hey, et al., “NoRBERT: Transfer Learning for Requirements Classification,” RE 2020.

[9] H. Muccini, et al., “Leveraging Machine Learning Techniques for Architecting Self-Adaptive IoT Systems,” SMARTCOMP 2020.

[10] D. P. Wangoo, “Artificial Intelligence Techniques in Software Engineering for Automated Software Reuse and Design,” ICCCA 2018.

[11] K. Becker, et al., “AI Programmer: Autonomously Creating Software Programs Using Genetic Algorithms,” GECCO 2021.

[12] Z. Feng, et al., “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” arXiv:2002.08155, 2020.

[13] A. Kanade, et al., “Learning and evaluating contextual embedding of source code,” ICML 2020.

[14] Y. Wang, et al., “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation,” EMNLP 2021.

[15] S. Lu, et al., “CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation,” arXiv:2102.04664, 2021.

[16] M. Chen, et al., “Evaluating Large Language Models Trained on Code,” arXiv:2107.03374, 2021.

[17] M. Grover, et al., “Estimating Software Test Effort Based on Revised UCP Model Using Fuzzy

Technique," ICTIS 2017.
[18] C. López-Martín, "Machine learning techniques for software testing effort prediction," Software Quality Journal, 2021.
[19] C. Jin, "Software defect prediction model based on distance metric learning," Soft Computing, 25, 2021.
[20] 土肥正ほか, "次世代ソフトウェア信頼性評価技術の開発に向けて", SEC journal, 10(6), 2015.
[21] D. D. Hanagal, et al., "Literature Survey in Software Reliability Growth Models," Software Reliability Growth Models, 2021.
[22] K. K. San, et al., "Deep Cross-Project Software Reliability Growth Model using Project Similarity Based Clustering," Mathematics, 9(22), 2021.
[23] 丹野治門ほか, "テスト入力値生成技術の研究動向", コンピュータソフトウェア, 34(3), 2017.
[24] A. Windisch, et al., "Signal Generation for Search-Based Testing of Continuous Systems," ICSTW 2009.
[25] R. Matinnejad, et al., "Thomas Bruckmann Automated Test Suite Generation for Time-continuous Simulink Models," ICSE 2016.
[26] 西谷一平ほか, "実走行データを活用した制御システム自動検証手法の開発", 自動車技術会論文集, 49(1), 2018.
[27] T. Shiba, et al., "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," COMPSAC 2004.
[28] K. Z. Zamli, et al., "A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem," PLoS ONE, 13(5), 2018.
[29] R. Pan, et al., "Test case selection and prioritization using machine learning: a systematic literature review," Empirical Software Engineering, 27, 2022.
[30] R. Russell, et al., "Automated Vulnerability Detection in Source Code Using Deep Representation Learning," ICMLA 2018.
[31] X. Li, et al., "DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization," ISSTA 2019.
[32] R. K. Saha, et al., "ELIXIR: Effective Object-Oriented Program Repair," ASE 2017.
[33] Y. Li, et al., "DLFix: Context-based Code Transformation Learning for Automated Program Repair," ICSE 2020.
[34] M. Monperrus, et al., "Repairnator Patches Programs Automatically," ACM Ubiquity, 2019.
[35] H. Washizaki, "Towards Software Co-Engineering by AI and Developers," Handbook on Artificial Intelligence-Enhanced Software Engineering, 2022.
[36] H. A. Al-Jamimi, et al., "Machine Learning-based Software Quality Prediction Models: State of the Art," ICISA 2013.
[37] N. Tsuda, et al., "Machine Learning to Evaluate Evolvability Defects: Code Metrics Thresholds for a Given Context," QRS 2018.
[38] H. Isotani, et al., "Duplicate Bug Report Detection by Using Sentence Embedding and Fine-tuning," ICSME 2021.
[39] Y. Noyori, et al., "Extracting features related to bug fixing time of bug reports by deep learning and gradient-based visualization," ICAICA 2021.
[40] S. Uchiyama, et al., "Design Pattern Detection using Software Metrics and Machine Learning", SQM 2011.
[41] K. Alkharabsheh, et al., "A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset," Information and Software Technology, 143, 2022.
[42] E. S. Maldonado, et al., "Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt," IEEE Transactions on Software Engineering, 43(11), 2017.
[43] M. Aniche, et al., "The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring," IEEE Transactions on Software Engineering, 2020.
[44] C. Mills, et al., "Automatic Traceability Maintenance via Machine Learning Classification," ICSME 2018.
[45] H. Washizaki, et al., "Software Engineering Design Patterns for Machine Learning Applications," IEEE Computer, 55(3), 2022.
[46] J. H. Husen, et al., "Goal-Oriented Machine Learning-Based Component Development Process," MODELS-C, 2021.

著 者 略 歴

鷲崎（わしざき）　弘宜（ひろのり）

早稲田大学グローバルソフトウェアエンジニアリング研究所所長・教授，国立情報学研究所客員教授，システム情報取締役（監査等委員），エクスモーション社外取締役．IEEE Computer Society 副会長，ISO/IEC/JTC1 SC7/WG20 議長，情報処理学会ソフトウェア工学研究会主査，リカレント教育 スマートエスイー代表．著書・訳書に『機械学習デザインパターン』『ソフトウェア品質知識体系ガイド SQuBOK Guide』『演習で学ぶソフトウエアメトリクスの基礎』ほか．