Sonar

AIコーディングの7つの習慣

品質とセキュリティを妥協することなく、AIで開発者の生産性を高めるためのガイド

【翻訳に関するご注意】

本資料は、SonarSource S.A.が発行した英語原文 "7 Habits of Highly Effective AI Coding" を、生成AIを用いて日本語に翻訳したものです。
英語原文が正式版であり、本資料の内容と原文に齟齬がある場合は、原文を優先してください。
翻訳の性質上、専門用語・固有表現・ニュアンスの解釈に誤りが含まれる可能性があります。重要な判断や意思決定の際は、必ず原文をご参照ください。
製品名（SonarQube、SonarQube Cloud、SonarQube for IDE 等）および固有の技術用語は、原文表記を維持しています。

原文（英語）参照先：7 Habits of Highly Effective AI Coding

翻訳日：2026年4月

本翻訳資料に関するお問い合わせは、下記までご連絡ください。

“お問い合わせ先“
SonarSource Japan
奥野：yoshihiko.okuno@sonarsource.com
Chi Hoon：chihoon.moon@sonarsource.com

https://www.sonarsource.com/sem/7-habits-of-highly-effective-ai-coding-ebook/

はじめに

あなたのSDLCはもう元には戻らない

ソフトウェア開発の世界は、一世代に一度とも言える最も根本的な変革を経験しています。人工知能（AI）が牽引するこの変化は、遠い未来の話ではなく、現在進行形の現実です。AIコーディングは急速に「目新しいもの」から「必須のもの」へと移行し、開発ワークフローを根底から変え、生産性向上の前例のない機会と、同等の規模のリスクをもたらしています。これは特に、プロフェッショナルな開発現場において顕著です――大規模で複雑なレガシーコードベースでは、バグであれセキュリティ上の欠陥であれ、たった一つのミスが甚大な損害をもたらす可能性があります。

AIの急速な統合がもたらすリスクの根源は、新たな課題にあります。それが「エンジニアリング生産性のパラドックス」です。AIは膨大な量のコードを生成しています――たとえばGoogleでは、新しいコードの4分の1以上がAIによって生成されています。しかし実際のエンジニアリングベロシティの向上は、わずか10%程度にとどまっています。

> 「今日、Googleで書かれる新しいコードの4分の1以上はAIによって生成され、エンジニアがレビューして承認しています」
>
> 「しかし最も重要な指標は――私たちは慎重に測定していますが――AIによって会社のエンジニアリングベロシティがどれだけ向上したかです。測定は難しく、私たちは厳密に計測しようとしており、その推定値は現在10%です」
>
> **— Sundar Pichai（Google CEO）**

AIはコードの作成速度を加速していますが、多くのチームの実際の生産性は頑固に横ばいのままです。期待されていた恩恵は、新たな大きな下方圧力によって相殺されており、ポテンシャルと現実の間のギャップが広がっています。ボトルネックはなくなったのではなく、単にレビューフェーズへと移行しただけです。これが現代チームにとっての2つの主要な課題を生み出しています：

- 新しいコードの作成速度：AI生成コードのベロシティは重大な新たな課題をもたらします。バグやセキュリティ脆弱性がシステムに侵入するスケールと速度が増大しているのです。これにより技術的負債とメンテナンスが増加し、生産性を引き下げます。AIツールを使っている開発者のほぼ60%がデプロイメントの問題を経験しており、初期の速度上の優位性は、大規模なテストと修正の必要性によってすぐに侵食されています。
- エンジニアのレビュー・検証負荷の増大：AI生成コードの膨大な量は、人間のレビュープロセスに前例のない直接的な負荷をかけます。これが「レビュー疲れ」をもたらし、品質を確保するための重要な監視が提案の洪水の中で薄れていきます。この「根拠のない自信」の状態では、開発者がAIの提案を95%以上承認してしまうケースも見られ、説明責任の崩壊とコードオーナーシップの危機を示しています。品質のゲートになるべきプロセス自体がリスクの源泉となり、ツールのメリットをさらに打ち消しています。

開発者は難しい立場に置かれています。完全には信頼できないツールを使わざるを得ず、2つのネガティブな結果のどちらかに陥ってしまいます。レビュー疲れに屈して欠陥のあるコードを盲目的に受け入れるか、AIのすべての提案を手動で入念に検証するか——後者は「開発者のトイル（雑務）」とでも呼ぶべき行為で、生産性向上のメリットを台無しにします。

この道を進むには、これらのリスクに対抗するための新しい実践のセット、つまりAIの力を責任を持って活用することでエンジニアリング生産性のパラドックスを解決するフレームワークが必要です。本eBookでは、そのフレームワークを構成する7つの習慣を提示します。これにより、組織は品質やセキュリティを妥協することなく生産性を最大化できます。

習慣 1

開発者は（依然として）説明責任を負う

黄金律：承認したら、あなたのものになる

開発者の説明責任という原則は新しいものではありません。「壊したら、自分で直す（You break it, you own it）」は、ソフトウェアエンジニアリングにおいて数十年にわたって暗黙のルールとして機能してきました。しかし、3秒で100行の「もっともらしいが欠陥のある」コードを生成できるAIツールが新たなペアプログラマーとなった今、「説明責任」はより緊急性の高い意味を帯びています。それは受動的なルールから、積極的なプロフェッショナルとしての姿勢へと進化しています。この習慣は、タイピングした内容を所有するだけではなく、承認するすべてのコード行に対して懐疑的なシニアレビュアーになることです。このマインドセットは、現代の開発チームを脅かす説明責任の危機に対する主要な防御であり、プロセスの「レビューと承認」フェーズの有効性を確保するための鍵です。

黄金律：承認したら、あなたのものになる

開発者がAIアシスタントにデータベースからユーザーIDのリストを処理する関数を書くよう依頼するシナリオを考えてみましょう。AIは一見完全に合理的なコードを生成するかもしれません。構文的に正しく、プロンプトに論理的に従っています。ジュニア開発者や、レビュー疲れを起こしているシニア開発者はそれを承認するかもしれません。しかし説明責任のある専門家は、重大なパフォーマンスボトルネックを発見します。forEachループはデータベースに対して逐次的・ブロッキングな呼び出しを行います。100件のユーザーIDに対して、これは100回の直列ネットワークリクエストを意味し、将来的な本番環境の障害を引き起こす時限爆弾となります。AI時代における真の説明責任とは、このような微妙だが重大な欠陥を発見する懐疑心を持つことを意味します。

この問題を修正するには、非同期処理についてのより深い理解が必要です。並列でデータベースリクエストを実行するバージョンは、パフォーマンスを劇的に改善します。これが説明責任のギャップです：単に動くコードと、堅牢でパフォーマンスが高く、本番環境に耐えられるコードとの違いです。

SonarQube アドバンテージ

AIのすべての提案に対してこれほど懐疑的であることは、長期的に開発者にとって精神的に疲弊し、持続不可能です。ここで自動コードレビューが重要なパートナーとなります。SonarQubeのAIコードアシュアランスは、AIによって生成されたコードに「鋭い焦点」を当て、徹底的な分析でAIモデルがしばしば導入する問題（技術的負債の増加、セキュリティ脆弱性の導入、ロジックの欠陥といった複雑なバグなど）を積極的に特定し、人間のレビュアーが自ら発見する負担を軽減します。これにより説明責任は、煩雑な手動タスクから、自動化されたバックストップが確保する自信あるプロセスへと変わります。

習慣 2

プロジェクトのコンテキストを（過剰なくらい）文書化する

コンテキストは王様：より良いドキュメントでAIコーディングを強化する

AIコーディングアシスタントには本質的な理解がありません。プロジェクトの歴史、アーキテクチャの哲学、技術的な意思決定の背景にある「なぜ」について何も知りません。本当に役立つコードを生成するために、AIは与えられたコンテキストに完全に依存しています。AI時代において、このコンテキストを提供することは2段階のゲームになります：AIに高レベルのアーキテクチャブループリントとドキュメントを与えて長期的な理解を促すことと、明確で実行可能なプロンプトを作成することです。

レベル1：AIにドキュメントを与える

AIコーディングアシスタントは、オンボーディングが必要な新しいチームメンバーとして扱うべきです。システムの構造を理解するために、人間の開発者と同じ高レベルのドキュメントにアクセスする必要があります。つまり、ドキュメントを生きたコードとして扱い、AIツールに重要なコンテキストを提供できるよう、主要な成果物を最新の状態に保つことを意味します。効果的なアーキテクチャ文書には次のものが含まれます：

- ダイアグラム：Mermaidなどのツールを使って、システムのアーキテクチャ、データフロー、サービスのインタラクションを示す明確でバージョン管理されたダイアグラムを作成する。
- 設計ドキュメントとADR：主要な技術的選択の背後にある理由を説明する簡潔なアーキテクチャ・デシジョン・レコード（ADR）を維持する。
- プロジェクト構造ファイル：適切に書かれたREADME.mdと明確なフォルダ構造でプロジェクトレイアウトを定義し、人間とAIエージェント両方にとってのマップとする。

プラットフォームおよびDevExリーダーにとっての課題は、コンテキストを運用化することです。開発者がAIにドキュメントを手動で渡すことを期待するのは、スケーラブルな戦略ではありません。代わりに、4つの主要なアクションに焦点を当てた体系的で安全な「コンテキストパイプライン」を構築しましょう：

- ドキュメントの標準化：ADR、READMEなどの主要なドキュメントをバージョン管理されたリポジトリに集約する。これにより、人間とAI両方にとっての唯一の正しい情報源が生まれます。
- インジェスチョンの自動化：ドキュメントを解析、インデックス化し、ベクトル埋め込みに変換する自動パイプラインを構築する。これにより、AIツール用のクエリ可能なデータベースが作られ、コンテキストを手動で提供する手間が省けます。
- セキュリティとアクセス制御の強制：コンテキストパイプラインが既存の開発者権限を厳密に継承することを確認する。AIは、開発者が閲覧権限を持つドキュメントにのみアクセスできなければなりません。
- ツールチェーンへの統合：コンテキストパイプラインをデベロッパープラットフォームのシームレスなコンポーネントにする。CI/CDパイプライン、IDE、および選択したAIアシスタントとネイティブに統合し、摩擦なく豊富なコンテキストを提供する。

このプラットフォームレベルのソリューションを構築することで、組織はアドホックなプロンプト作成から、信頼性の高いガバナンスされたシステムへと移行し、AIを使用するすべての開発者の生産性と有効性を直接向上させます。

レベル2：実行可能なプロンプトを作成する

AIが高レベルのマップを持ったら、次は手元のタスクのための明確な詳細な指示が必要です。ここでプロンプトエンジニアリングがコアな開発者スキルになります。以下の違いを考えてみましょう：

> 【役に立たないプロンプト（曖昧な指示）】「ユーザー入力を検証する関数を書いてください。」このプロンプトは、プロジェクトの既存パターンとはおそらく互換性のない、汎用的でコンテキストを考慮しないコードを生成します。
>
> 【実行可能なプロンプト（具体的な指示）】「Zodバリデーションライブラリを使用しています。既存のエラー処理パターンはsrc/errors/ApiError.tsにあります。新しいユーザーサインアップフォーム用のバリデーション関数を書いてください。ユーザースキーマには以下が必要です：email（有効なメールアドレスであること）、password（文字列、最低10文字、大文字1字・数字1字・特殊文字1字）、age（18以上の整数）。バリデーション失敗時はApiErrorをスローしてください。」

最初のプロンプトはコードを生成します。2番目のプロンプトはアーキテクチャに合ったコードを生成します。この新しいパラダイムでは、開発者のプロンプトが新しい設計ドキュメントです。具体的で、コンテキストを考慮し、実行可能でなければなりません。

SonarQube アドバンテージ

開発チームがドキュメントとプロンプトを通じて豊富なコンテキストを提供することに集中している間、Sonarはバックグラウンドで生成されたコードが確立されたコード品質基準やアーキテクチャのブループリントに実際に準拠していることを確認します。SonarQubeのIDEおよびサーバーサイド分析は、新しいコード（人間が書いたものであれ、AIが生成したものであれ）がプロジェクトのコーディング標準、既存のパターン、コンポーネント、およびロジックと同期しているかどうかを評価します。これにより、コードベースの長期的な健全性にとって重要な高レベルの設計決定を強制するガードレールとして機能します。

習慣 3

本当にシンプルに保つ

シンプルさこそが究極の洗練

「シンプルに保つ」というアドバイスは、常に優れたエンジニアリングの象徴でした。AI時代においては、良いアドバイスから交渉の余地のない技術的な前提条件へと昇格します。積極的にメンテナンスされないコードベースは無秩序に向かう傾向があり、これは「コードエントロピー」と呼ばれる現象です。適切に誘導されなければ、AIアシスタントはこのエントロピーを、過度に複雑または難解なコードを生成することで加速させる可能性があります。

さらに、AIツール自体も複雑さに苦労します。認知的複雑度が高く、関数が長く、ネストが深いコードによって混乱しやすくなります。これは、大きなコンテキストウィンドウと複雑な制御フローがモデルの推論能力を低下させるためで、「複雑さの崖（complexity cliff）」として知られる問題です。シンプルさの強制は、もはや人間の同僚のためだけではありません——AIツールがコードベースを効果的に分析、リファクタリング、貢献できるようにするための要件でもあります。

具体的なガードレールでシンプルさを強制する

コードベースを「AI対応」にするためには、漠然とした原則を超えて、明確で強制可能なシンプルさのガードレールを確立する必要があります。これらのルールはLLMにプロンプトとして与え、パイプラインで自動的にチェックされるべきです。一般的で効果的なガードレールには次のものがあります：

- 関数の長さ：関数を50〜100行以下に保つ。
- 認知的/循環的複雑度：複雑度スコアを15などの定義された閾値以下に保つ。
- コードの重複：重複を可能な限り0%に近づける。
- ネストの深さ：ステートメントのネストを最大3〜4レベルに制限する。
- マジック値の禁止：ロジックで使用されるすべての数値や文字列を名前付き定数として定義することを要求する。これにより文脈のない値（例：if (user.role === 3)）を防ぎ、コードを自己文書化し、リファクタリングを容易にする。

SonarQube アドバンテージ

シンプルさのルールのリストを暗記して手動で強制することは非効率で、エラーが発生しやすいです。Sonarを使えば、チームはこれらの基準をコード化し、自動的に適用できます。SonarQubeの分析エンジンには、高い複雑度、重複、その他のコードスメルなど、コード品質と保守性に影響する問題をチェックするための包括的なルールセットが含まれています。さらに、カスタムルールを使用することで、組織固有の要件を定義して適用できます。これにより、コードの出所に関わらず、すべてのコードが同じ、相互に合意された高水準のシンプルさと品質に準拠することが保証されます。

習慣 4

不要なコードは絶対に残さない

散らかりのない環境：未使用コードを排除する

これはAIコーディングにより新たな緊急性を持つ、もう一つの基本的なルールです。AIアシスタントは、役に立ち明示的であるよう最適化されていることが多く、冗長になる傾向があります。ボイラープレート、冗長なコメント、「念のため」のロジックを生成し、散乱、認知的オーバーヘッド、大幅なメンテナンス負担を追加します。

このような不要なコードは単に整理されていないだけではなく、真のセキュリティ上の脅威を表しています。未使用の関数、変数、依存関係はアプリケーションの攻撃対象領域を拡大します。より巧妙なのは、「バックドア」攻撃や「スリーパーエージェント・インジェクション」などの新しい攻撃ベクトルへの入口を作ることです。悪意のある行為者がLLMを騙して、一見無害だが後で悪用される可能性のある未使用の依存関係を含めさせます。したがって、開発者の役割は、冗長なAIの出力をその本質的なコアに削ぎ落とす、冷徹な編集者としての役割をも含むよう進化しなければなりません。

冷徹な編集者としての開発者

gitのdiffは、このタスクのための開発者の最良のツールです。AIがユーザー権限をチェックする関数を過度に明示的に生成するとします。この習慣は、そのような出力をクリーンでイディオマティックなコードに継続的にリファクタリングすることです。明確で不可欠な目的を果たさないコードのすべての行を積極的に削除し、「ミニマリズムの原則」に従うことが目標です。

SonarQube アドバンテージ

未使用の要素についてコードのすべての行を手動で精査することは、人間の開発者には不向きな、退屈でエラーが発生しやすい作業です。Sonarはこのプロセスを自動化します。SonarQubeの静的解析エンジンは、変数、関数、パラメータなど、デッドコードや未使用のコードを自動的に検出して通知します。この自動チェックはCI/CDパイプラインにおける重要なバックストップとして機能し、チームが不要な要素がメンテナンスの悪夢やセキュリティ上のリスクになる前に排除するのを支援します。

習慣 5

あらゆるものを分析する

信頼しつつ検証する：包括的な分析の必要性

AIによって生み出されるコードの量は圧倒的です。AIが生み出す可能性のある問題は、しばしば隠れていて、手動レビューだけでは発見が困難です。「あらゆるものを分析する」というのは疲弊しそうに聞こえますが、開発者がすべてを手動で分析すべきということではありません。新しい現実を受け入れることを意味します：AIによる開発速度には、高速で正確な分析が必要だということです。複雑なセキュリティバグを見つけ、サードパーティの依存関係が適切にライセンスされてメンテナンスされていることを確認し、微妙なパフォーマンス問題をチェックするために手動レビューに頼ることは、開発者のトイルとバーンアウトの処方箋です。唯一のスケーラブルなソリューションは、レビューフェーズが超えられないボトルネックになることを防ぎ、エンジニアリング生産性のパラドックスを解決するために、積極的な自動化の多層戦略を採用することです。

現代の自動化分析ワークフロー

実践において、「あらゆるものを分析する」は、開発ワークフローのあらゆる段階で分析を統合する継続的な自動化プロセスであるべきです：

- 初期分析：既存のコードベースを分析して、コード全体の品質とセキュリティに関するインサイトを得る。
- コーディング中のリアルタイム：AIがコードを生成する際、IDEエクステンションが開発者がIDE内でコードを書きながらバグ、脆弱性、セキュリティホットスポット、コードスメルについて即座にフィードバックを提供する。
- コミット前：自動的に分析を実行するプリコミットフックで、トークン、パスワード、APIキーなどのシークレットなど、重要な問題がリポジトリに到達することを防ぐコントロールポイントを確保する。
- コミット/マージ時：すべてのブランチとプルリクエストのコードを徹底的に包括的に分析し、セキュリティ、信頼性、保守性に関する会社の基準を満たさないコミットやマージをブロックする。

SonarQube アドバンテージ

SonarQubeは、この現代的な開発者ワークフローを実現する、最も信頼されている統合コード品質・コードセキュリティプラットフォームです。最も広く使われているDevOpsプラットフォームとの深い統合、そしてCursorやWindsurfといった新たに普及しているAIエージェント型IDEを含む最も人気のあるIDEへのエクステンションにより、SonarQubeはSDLCの各段階ですべてのタイプのコードを自動的にレビューします。さらに、SonarQubeの分析結果は、プルリクエストのコメント内など重要な段階での明確な合否結果としてクオリティゲートに表示され、標準以下のコードがコミットやマージされることを防ぎ、常にコードが本番環境に対応できる状態を確保します。

習慣 6

ユニットテストを義務化する

テストされていなければ、壊れている

AI時代において交渉の余地がなくなる、もう一つの基本的なソフトウェア開発の実践がユニットテストです。ユニットテストは常に重要でしたが、AIとともにその役割は進化します。ボイラープレートテストを書くという退屈な作業の多くは今やAIアシスタントに委任できます。AIは「ハッピーパス」のテストケースを生成するのが得意です。この新しい習慣は単に「テストを書く」ことではなく、「AIを使ってテストを生成し、次に人間の専門知識を活用してそれを完成させる」ことです。

重要な実践の一つは、「リワードハッキング」に対して警戒することです。これは、AIが欠陥のあるコードを通過させるように設計された、誤ったテストを書く可能性がある現象です。

テストに対する規律あるアプローチ

テストとレビューを効果的にするためには、新しいレベルの規律が必要です。主要な実践には次のものがあります：

- AIがコードを生成する前にユニットテストを書き、テストが別の作者（人間またはAI）によって書かれることを確認する。これにより、AIが自分自身の欠陥のあるテストに合格するだけの欠陥コードを書く「リワードハッキング」を防ぐ。
- テストが明らかでないエッジケースと失敗モードをカバーしていることを確認する。
- 可能な場合はテスト駆動開発（TDD）を使用する。
- CI/CDパイプラインに自動化されたテスト実行を統合する。

SonarQube アドバンテージ

SonarQubeはテストカバレッジツールとシームレスに統合し、開発者がより高品質で、よりセキュアで、十分にテストされたコードを書けるよう支援します。静的コード分析結果と並んでカバレッジレポートを収集・表示するセントラルハブとして機能し、コードが品質基準を満たしていることを確認するための明確な合否指標を提供します。デフォルトでは、SonarQubeのクオリティゲートの条件は、新しいコードに対してコードカバレッジ80%以上を要求し、チームが高い基準を維持するのを支援します。また、クオリティゲートポリシーをカスタマイズして、固有のニーズに合わせることもできます。

習慣 7

厳格なコードレビューを行う

コードレビューを構文チェックから深い検証へと引き上げる

コードレビューは、軽微なスタイルの問題や単純なバグに焦点を当てると、しばしば遅くて退屈になります。これらの低レベルの問題の検出を自動化することで、コードレビューの「厳格さ」をついに構文から戦略へとシフトできます。これにより、機械と人間の間で作業が知的に分担される、進化したレビュープロセスが生まれます。適切なツールとプロセスに支えられた強力なコードレビュー文化こそ、AI生成コードの急増が本番環境の問題の急増を引き起こすことを防ぎます。

自動化の役割

堅牢で集中管理された自動分析プラットフォームが最初の防衛線として機能します。このシステムはすべてのコードを自動的に審査します：

- 標準的なコード品質：高い複雑度、重複、コードスメルなどの保守性の問題。
- セキュリティ脆弱性：ファーストパーティコードとサードパーティの依存関係における一般的な弱点。
- AIコードの検証：SonarのAIコードアシュアランスは、汚染されたデータフローが脆弱性を生み出す場合や、基本的な分析では見逃すような微妙な論理エラーなど、AIモデルがしばしば導入する固有の微妙な欠陥を見つけるための重要な新しい層を提供します。

人間の専門知識をどこに適用するか

コードが機能的に健全で、セキュアで、保守可能であるという確信を持って、人間のレビュアーは真の理解と戦略的思考を必要とする高レベルの問いにのみ集中できます：

- このコードはビジネス要件を実際に満たしているか？
- これは正しい長期的なアーキテクチャアプローチか？
- AIはタスクの本質的な意図を誤解していないか？

SonarQube アドバンテージ

SonarQubeはCI/CDパイプラインに直接統合することで、コードレビューワークフローを自動化して強制します。マージされる前に新しいコードのバグ、脆弱性、コードスメル、重複を自動的に分析します。これにより、問題が早期に捕捉されます——しばしば人間のレビュアーがコードを見る前に。これによりコードレビューがより効率的になり、高レベルの設計とロジックに集中できます。SonarQubeの明確で実行可能なレポートは、レビュアーが問題を素早く特定して対処するのを支援し、重大な問題が見落とされるリスクを軽減します。プラットフォームはまた、組織のコーディング標準とクオリティゲートポリシーを強制し、定義された基準を満たさないマージをブロックします。この一貫性は、微妙な問題を導入したり確立されたプラクティスから逸脱する可能性があるAI生成コードをレビューする際に特に価値があります。プラットフォームは履歴追跡とダッシュボードも提供し、継続的改善をサポートするためのコード品質トレンドの可視性を提供します。

まとめ

未来は人間が定義し、AIが支援するコード

私たちはAIを活用したソフトウェア開発の時代に突入しました。この業界の未来は、人間とマシンの戦いではなく、人間の開発者とAIエージェントの強力なシナジーにあります。AIは人間の創意工夫を増強するものであり、置き換えるものではありません。

人間は今も、そしてこれからも不可欠です。私たちの専門知識は、現在のAIには届かない領域——真のイノベーション、複雑な問題解決、戦略的なアーキテクチャ設計、ニュアンスのあるコンテキスト理解、倫理的判断の本質的な適用——において重要です。プロフェッショナルな開発者の役割は、すべてのコードを書く人から、インテリジェントなツールをオーケストレートし、その出力を深い懐疑心で検証し、最終的な製品が単に機能するだけでなく、セキュアで、保守可能で、アーキテクチャ的に健全であることを確保する人へとシフトしています。Sonarがこの新しい時代のために掲げるマントラは「バイブ、そして検証（Vibe, then verify）」です。

開発者がこれらの7つの習慣を受け入れることを可能にすることで、この新しい現実のための本質的な基盤が提供されます。組織はAIが提供する深い生産性のメリットを最大化しながら、関連するリスクを責任を持って管理できます。次世代のソフトウェアを構築するために必要な、説明責任、クリーンコード、継続的改善の文化を育みます。

コード品質とコードセキュリティがますます相互に絡み合う中で、SonarQubeはこの変革における不可欠なパートナーです。SonarQubeプラットフォームは、人間とAI双方のすべての開発者が、より優れた、よりセキュアなソフトウェアをより速く構築できるよう設計されています。