AI開発ツールの
リスクと安全な活用

～「使わない」から「安全に使う」へ～

SIOS Technology, Inc.

2026.02

セクション１

イントロダクション

本日のゴール

AIは危険だから
使わない

→

AIは正しく使えば
安全に活用できる

自己紹介

名前	田中 龍之介
所属	SIOS Technology, Inc. / PS事業部
役割	アプリケーションエンジニア

経験・実績

- SCANOSSの技術検証・アプリケーション開発を担当
- 自社YouTubeチャンネルでセミナー・技術解説を配信
- 外部登壇実績 10件以上

※ 本セミナーでは SCANOSS を紹介しますが、SIOSはSCANOSSの正規代理店です。この点を踏まえてお聞きください

本日持ち帰っていただくこと

1. **リスクを理解し、適切に判断できる**
    - AI開発ツールの3つのリスク（ライセンス・脆弱性・機密漏洩）
2. **「安全に使う」体制を設計できる**
    - 多層防御の考え方と各層の役割
3. **明日から実践できる具体策を知る**
    - AIツールの推奨設定、SCAツールのプロセス統合

セクション2

AI開発ツールの現状

AI開発ツールの生産性向上効果

調査元	対象	タスク完了時間
GitHub公式研究	95名の開発者	**55.8%短縮**
Google内部研究	96名の正社員	**21%短縮**
Microsoft/Accenture	5,000名以上	PR **26%増加**

現実: **84%** の開発者がAIツールを使用[1]、**41%** のコードがAI支援で生成[2]

「使わない」という選択肢は**競争力低下**につながる

[1] Stack Overflow Developer Survey 2025 | [2] DX AI-assisted engineering Q4 impact report 2025

しかし、リスクも存在する

リスク種別	内容	具体例
ライセンスリスク	OSS混入・著作権侵害	GPL混入、帰属表示漏れ
脆弱性リスク	既知の脆弱性を持つコード混入	**45%** に脆弱性含有
機密漏洩リスク	社内コードが学習データに	設定ミスによる情報流出

> **参考**: 品質リスク（コード量増加による検証負荷）も存在するが、「何をもって品質とするか」の定義が必要なため、本日のスコープ外とします

※ これらのリスクは手動コードにも存在する。AI生成コードで問題が拡大する理由は「スピードと量の増加」（41%がAI生成）にある

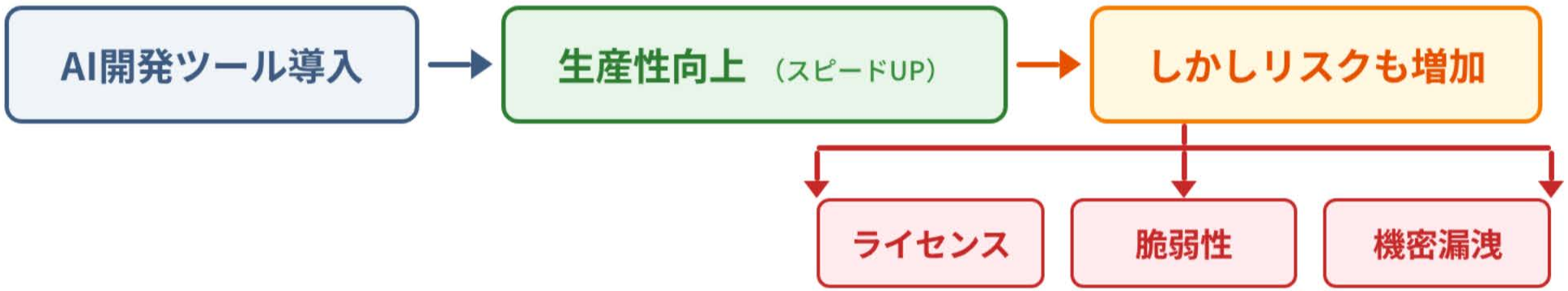
AI関連の法的リスクの顕在化

訴訟・事例	状況	金額
Doe v. GitHub	係争中（第九巡回控訴裁判所）	10億ドル+
Anthropic和解	2025年9月和解	**15億ドル**
読売新聞 vs Perplexity	提訴（2025年8月）	21.7億円

日本での動向: 著作権法第30条の4で学習は許容、**生成物利用はリスクあり**[3]

[3] 文化庁「AIと著作権に関する考え方について」

スピードとリスクのジレンマ



問い: スピードとリスク、どうバランスを取るか？

セクション３

DevSecOpsと多層防御

スピードを落とさずリスクに対処するには？

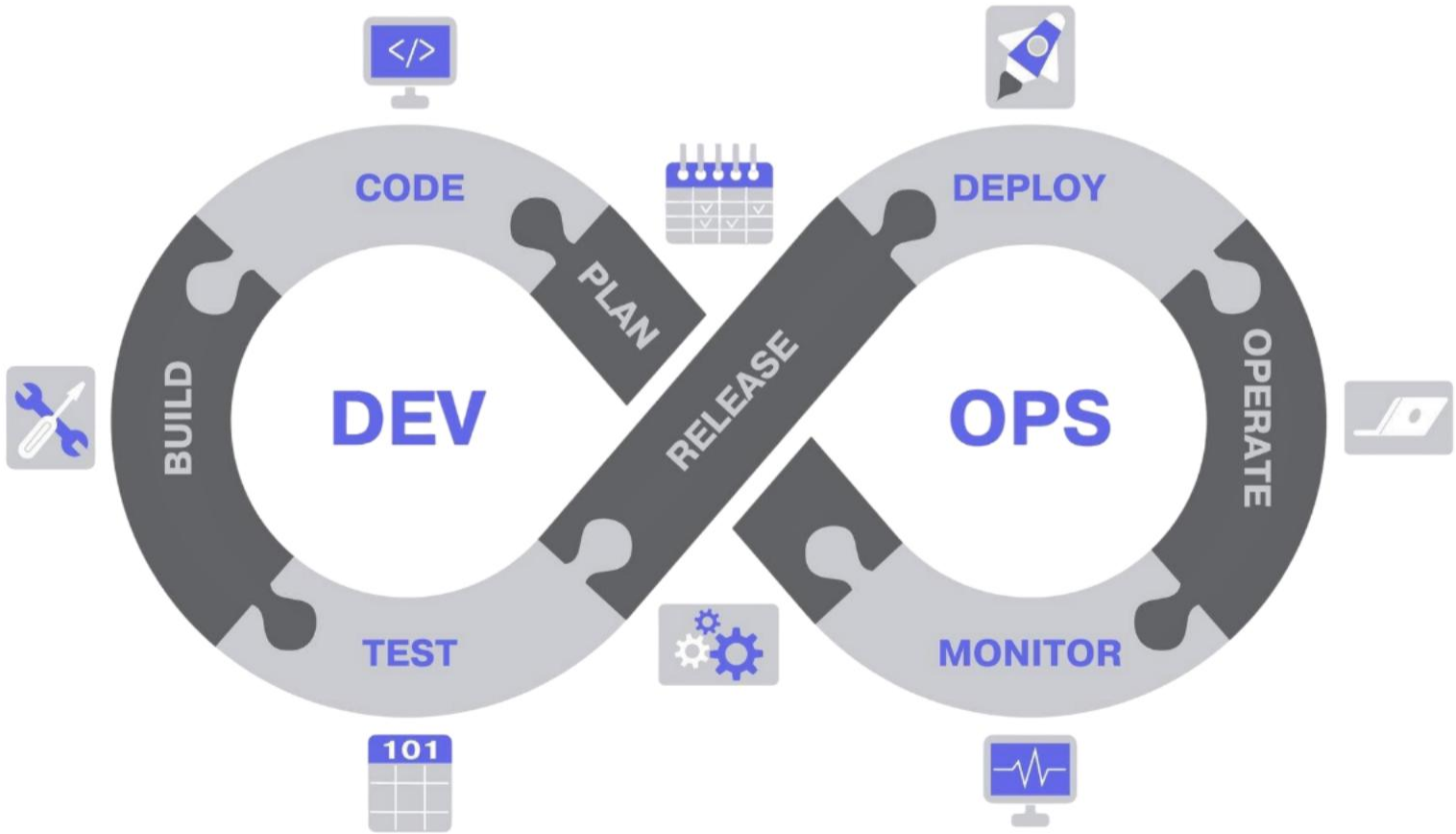
セキュリティチェックをどこに入れるべきか？

アプローチ	結果	
開発後にセキュリティチェック	手戻りが発生し、スピードが落ちる	✗
チェックを省略	潜在的リスクを内包する	✗
自動化して開発プロセスに組み込む	**スピードを維持しつつリスクに対処**	✓

この「自動化」と「プロセス統合」を実現する土台が → DevOps

自動化・プロセス統合の土台: DevOpsとは？

開発（Dev）と運用（Ops）を統合し、**自動化**して継続的に改善する考え方



- 自動化
- 継続的改善
- プロセス統合

DevSecOps: DevOpsにセキュリティを組み込む

DevOpsの自動化・プロセス統合に**セキュリティ（Sec）**を加えた考え方

DevOpsの要素	セキュリティへの活用
自動化	セキュリティチェックを自動実行
継続的改善	フィードバックで品質を向上
プロセス統合	開発フローにセキュリティを組み込む

開発の各段階にセキュリティを自動で組み込む

早期発見・早期対応

リスクごとに適切な対応が異なる

リスク	必要な対応
ライセンスリスク	混入防止、スキャン、法的備え
脆弱性リスク	セキュアな記述、脆弱性検出
機密漏洩リスク	流出経路の遮断、ルール化

すべてを包括的にカバーするツールは少なく、得意分野もそれぞれ異なる
→ 「一つの対策で全部解決」はできない → **リスクごとに対応を組み合わせる**

大前提：完全な防御は存在しない

ツールの限界

公開コード対策は100%ではない

SCA検出に誤検出・見落としあり

設定ミス・設定漏れのリスク

レビューの限界

OSS混入の人力検出は非現実的

AI生成コード増加でレビュー負荷増大

検出品質が担当者スキルに依存

ツールも人間も完全ではない → 層を重ねてカバーし合う → **多層防御**

提案: 多層防御でリスクに対処する

層 / 対策	機密漏洩	ライセンス	脆弱性
1 第1層 AIツール設定	✓	▲	▲
2 第2層 SCAツール	—	✓	✓
3 第3層 既存レビュー	▲	▲	✓
+α ＋α IP補償	—	◎	—

✓ 主に対応　◎ リスク移転　▲ 限定的　— 対応しない

第1層・第2層で問題を減らし、第3層（人間）が最終判断する

セクション４

【第1層・予防】ツール選定と使い方でリスクを減らす

予防の2つのアプローチ

ツール選定（組織）→ **使い方**（個人）の順でリスクを軽減する

AIツールの選定

- 学習データ利用設定
- 公開コード対策機能
- IP補償の有無

ユーザーの使い方

- セキュアプロンプト

🔍 まずは組織としての対策

ツール選定と設定

機密漏洩を防ぐ: 学習データ利用設定

サービス	設定	推奨
GitHub Copilot	Business/Enterpriseはデフォルト無効	✓
Amazon Q	Pro Tierで無効化可能	✓
Claude Code	Team/Enterprise/APIはデフォルト無効	✓
ChatGPT	設定で無効化可能	要確認

> **推奨**: Business/Enterprise/Proプランを使用し、学習データ利用を無効化する

ライセンスリスクを軽減: 公開コード対策機能

GitHub Copilot（ブロック方式）

設定手順:

1. GitHubページ → プロフィール → **Copilot settings**
2. **「Suggestions matching public code」** を探す
3. **「Block」** を選択

- 検出対象: 65 lexemes（プログラムの語句単位）以上（約150文字）[4]
- マッチ頻度: 全提案の1%未満[4]

[4] GitHub Docs - Finding public code that matches

ライセンスリスクを軽減: 公開コード対策機能

Amazon Q Developer（参照表示方式）

設定手順:

1. IDE設定 → Amazon Q → Open Settings
2. **「Show Code With References」** をOFF

- 動作方式: ブロックではなく「参照情報表示」
- コード参照ログで出所確認可能

公開コード対策機能がないAIツールの場合

注意: すべてのAIツールが公開コード対策機能を持つわけではない

ツール例	ブロック機能	参照表示機能
Claude Code	なし	なし
その他一部ツール	なし	なし

> 公開コード対策がない場合 → 後続で話す **IP補償の活用**と**SCAツールでの検出**がより重要になる

万が一に備える: IP補償

多層防御の **+α** として、企業がAIツールを導入する際の**リスク移転手段**

> **IP補償とは**
>
> 第三者から著作権侵害の訴訟を起こされた場合、AIツールベンダーが**法的防御費用や損害賠償を負担**する仕組み

主要AIツールのIP補償比較

サービス	対象プラン	補償条件
GitHub Copilot	Business / Enterprise	Block設定必須
Amazon Q	Pro Tier	-
ChatGPT	Enterprise / API	Copyright Shield
Claude / Claude Code	API / Team / Enterprise	商用利用時に適用
Gemini	Enterprise	Google Cloud経由

> ⚠️ **各サービスの利用規約を必ず確認してください**
> — 条件や範囲はサービスにより異なります

ツール選定のまとめ

リスク	対策	カバー状況
機密漏洩	学習データ利用設定	✓ 防止できる
ライセンス	公開コード対策 + IP補償	✓ 軽減・リスク移転できる
脆弱性	–	✗ ツール選定だけではカバーできない

脆弱性リスクにはツール選定だけでは不十分
--------ユーザーの使い方（セキュアプロンプト）で補う

次に個人としての対策

ユーザーの使い方

脆弱性リスクを軽減: セキュアプロンプトとは

AIにコード生成を依頼する際、**セキュリティ要件を明示的に指示**すること

具体例

× 「ログイン機能を作って」
○ 「ログイン機能を作って。SQLインジェクション対策、
パスワードのハッシュ化、CSRF対策を含めること」

セキュアプロンプトの効果

セキュリティ研究より

- プロンプトなし: **56%** が安全なコード
- セキュリティ指示あり: **66%** が安全なコード（+10pt改善）[5]

セキュリティ要件を明示するだけで **+10ポイント改善**

[5] Veracode GenAI Code Security Report 2025

第1層（予防）のまとめ：それぞれに限界がある

対策（第1層）	限界
公開コード対策	100%ではない（マッチ頻度1%未満だがゼロではない）
学習データ設定	設定ミス・設定漏れのリスク
セキュアプロンプト	56%→66%に改善するが、30-34%は脆弱なまま

予防だけでは防げないリスク:

- **脆弱性リスク** → 軽減はできるが効果は限定的
- **ライセンスリスク** → 対策をすり抜けるケースがある

予防だけでは不十分

ソースコードを解析して検出する仕組みが必要

セクション５

【第2層・検出】SCAツールとSCANOSS

なぜ機械的検出が必要か

予防の限界

Block設定は100%ではない

脆弱性は予防で軽減できるが限界あり

依存関係の問題

検出で対応

OSS混入を検出

既知脆弱性を検出

SBOM生成で可視化

ライセンスリスク検出：人間には困難な作業

OSS混入の検出は人間には事実上不可能

困難な理由	説明
膨大なOSSの存在	数億のOSSリポジトリと照合が必要
断片的な混入	数行のコードスニペットでも著作権は発生
コード量の増加	41%がAI生成[2]、レビュー負荷は増大

ポイント: 脆弱性は経験豊富なエンジニアなら気付ける可能性があるが、
OSS混入はどれだけ優秀でも人間には検出できない → 機械的な検出が必須

だから自動化が必要

機械がやること

- OSS混入の**検出**
- 脆弱性パターンの**スキャン**

人間がやること

- 検出結果の**判断**
- ビジネス影響の**評価**

機械的にできることは機械に任せ、人間は判断に集中する

→ **SCAツールの出番**

SCA（Software Composition Analysis）とは

ソフトウェア構成分析ツール

コードベースに含まれるOSSコンポーネントを**自動検出**し、ライセンス・脆弱性・コンプライアンスリスクを**可視化**するツール

依存関係

package.json
requirements.txt等

コードスニペット

ソースコードに
混入したOSS断片

バイナリ

コンパイル済み
コンポーネント

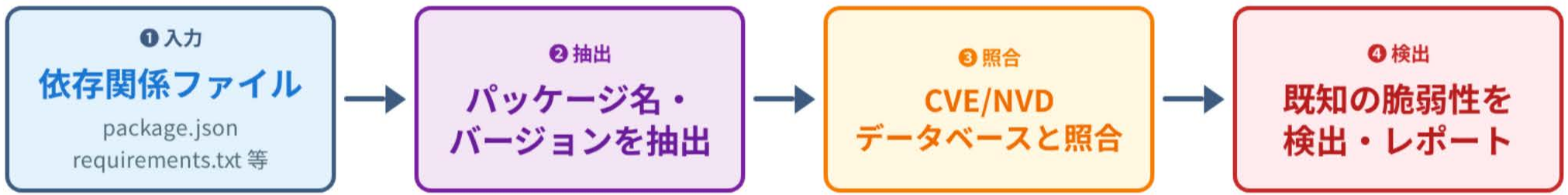
※ スニペット検出の精度・対応範囲はツールにより大きく異なる

一般的なSCAツールの機能

機能	内容
依存関係分析	package.json等のマニフェストファイル解析
脆弱性検出	CVE（公開脆弱性の識別番号）データベースとの照合
ライセンス検出	依存パッケージのライセンス確認
SBOM生成	SBOM（ソフトウェア部品表）による構成管理の可視化

> 一般的なSCAは「依存関係」ベースの分析 → ソースコードに直接混入したOSSの検出は得意分野による

SCAツールによる脆弱性検出の仕組み



ポイント 依存関係に記録されたパッケージの脆弱性は、SCAツールで**自動検出**できる

AI生成コードの課題：スニペット検出の必要性

依存パッケージ

npm install等
→ 依存関係に記録される
→ 一般的なSCAで検出可能

AI生成コード

ソースに直接混入
→ 依存関係に現れない
→ **スニペットレベルの検出が必要**

AI時代のSCAツール選定では**スニペット検出機能の有無**が重要になる

スニペット検出対応SCA

検出対象	一般的なSCA	スニペット検出対応SCA
npm/pip等の依存パッケージ	✓	✓
依存パッケージの脆弱性	✓	✓
AI生成コードに混入したOSS	✗	✓

SCANOSSはスニペット検出対応SCAの一つ:

- ソースコードレベルでOSS断片を検出（2億以上のURLナレッジベース）
- 依存関係分析や脆弱性検出に加え、スニペットのライセンスリスクもカバー

※ スニペット検出対応のSCAは他にも存在します（Black Duck等）。プロジェクト要件に応じて選定が必要です。

検出から第3層（レビュー）へ

従来	SCAツール導入後
全コードを人間がレビュー	検出された箇所に集中してレビュー
見落としリスクが高い	機械が検出、人間が判断
レビュー負荷が大きい	対応アクションから開始できる

予防（第1層） で減らし、**検出（第2層）** で見つけ、

人間（第3層） が判断する

層を重ねて**リスクを最小化**する

セクション６

【第3層・判断】プロセス統合とDevSecOpsへの組み込み

第3層: 人間によるレビュー

多層防御の最終層

第1層	予防	AIツール設定
第2層	検出	SCAツール
第3層	**判断**	**人間（既存のレビュープロセス）**

第3層は**新規導入ではなく、既存のレビュープロセスを活用**
第1層・第2層で問題を減らし、人間は**判断に集中**する

プロセス統合の全体像

セクション4-5で説明した対策を開発プロセスに組み込む:

1. **事前検討** → リスク評価・ツール選定
2. **予防の組み込み** → AIツール設定
3. **検出の組み込み** → SCAツール統合
4. **段階的導入** → フェーズを分けて定着させる

事前検討: リスク評価とツール選定

リスクの整理

検討項目	質問
対応したいリスク	機密漏洩？ライセンス？脆弱性？優先順位は？
許容するリスク	どこまでを許容範囲とするか？
現状の把握	既存コードにリスクはあるか？

事前検討: AIサービスの選定（ツール選定の観点）

観点	検討内容
機密漏洩対策	学習データ利用の設定は可能か？
ライセンス対策	公開コード対策機能はあるか？
IP補償	補償条件は？プランは？
コスト	ライセンス費用と導入効果のバランス

> ⚠️ **すべてのリスクに対応できるツールはない**
>
> → 対応したいリスクに合わせてツールを選定 → 足りない部分はSCAツールで補完

事前検討: SCAツールの選定（ツール選定の観点）

観点	検討内容
スニペット検出	AI生成コードのOSS混入を検出できるか？
脆弱性検出	CVE/NVDとの照合機能はあるか？
CI/CD統合	既存の開発パイプラインに組み込めるか？
コスト	無料枠の有無、ライセンス費用

> **ポイント**:
> AI生成コードのリスクを重視するなら**スニペット検出機能**を持つSCAツール（SCANOSS、Black Duck等）を選定する

検出の組み込み: いつ・どこで・どうやって

目的と手段を明確に → 開発プロセスに組み込んで**初めて効果を発揮する**

ローカルスキャンフロー

いつ: コミット前、PR作成前 ／ **効果**: 開発者が早期に気付ける

```
# インストール
pip install scanoss

# スキャン実行
scanoss-py scan . --output results.json

# SBOM生成
scanoss-py scan . --format spdx --output sbom.spdx.json
```

開発時のスキャンフロー（CI/CD）

いつ: PR作成時、コミット時 ／ **効果**: チームで見落としを防ぐ

```yaml
# .github/workflows/sca-scan.yml
name: SCA Scan
on: [pull_request]

jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: scanoss/scanoss-action@v1
        with:
          output-format: json
```

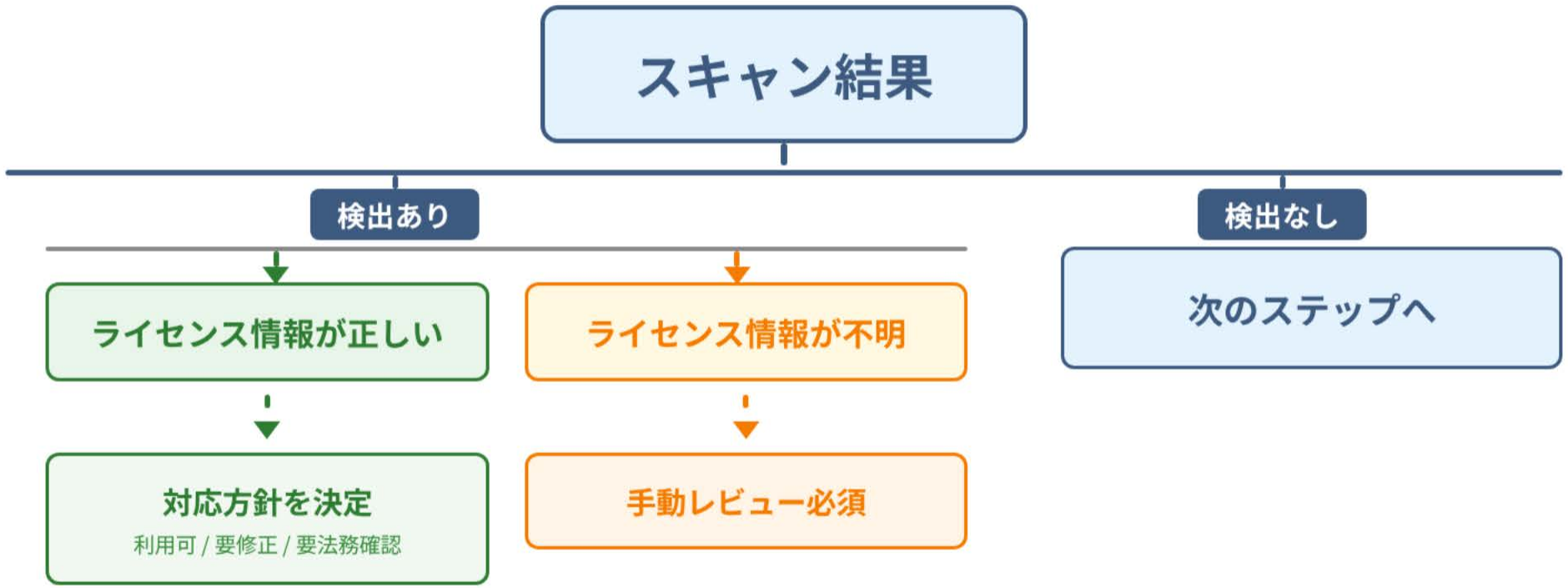
運用時のスキャンフロー

いつ: 週次/月次の定期スキャン ／ **効果**: 新規脆弱性の最終チェック

目的	内容
新規脆弱性の検出	既存コードに対する新しいCVEの確認
SBOM更新	構成情報の最新化
コンプライアンス監査	定期レポートの生成

> **開発時**: 早期発見 & **運用時**: 継続的な監視 → **自動化で人間の負荷を軽減**

検出時の対応フロー



どちらの場合も → 既存レビュープロセスで最終確認（人間が判断する）

（参考）段階的導入の例[7]

フェーズ	内容
Phase 1	ローカルスキャンで現状把握
Phase 2	CI/CD統合（警告モード）
Phase 3	ポリシーに基づくブロッキング
Phase 4	継続的改善

> 組織の状況に合わせて段階的に導入することを推奨

[7] Checkmarx - 12 SCA Best Practices

既存プロセスとの連携で多層防御を完成させる

新規導入

第1層（予防）：AIツール設定

第2層（検出）：SCAツール

既にあるもの

第3層（判断）：既存のレビュー

新規導入（第1層・第2層）で問題を減らし、既存プロセス（第3層）
最終判断 ＝ 多層防御の完成

セクション 7

まとめと次のステップ

本日のポイント

1. **AI開発ツールは生産性を向上させる**
    - 使わないことは競争力低下につながる
2. **リスクは存在するが、対策は可能**
    - ライセンス・脆弱性・機密漏洩
3. **多層防御で「安全に使う」体制を構築**
    - **第1層（予防）**: AIツール設定 → 新規導入
    - **第2層（検出）**: SCAツール → 新規導入
    - **第3層（判断）**: 既存レビュープロセス：統合

AIが書くコードが増えるほど

「仕組みで守る」が重要になる

完全な防御は存在しない — だからこそ、**層を重ねて守る**

「使わない」から「安全に使う」

SCANOSS導入のご相談

SIOSはSCANOSSの正規代理店として
検証・セミナーを行っています

詳細な検証結果や導入事例にご興味のある方は
本日のセミナー後、お声がけください

お問い合わせ: **ps-info@sios.com**

付録: 参考資料

トピック	参照先
AIツール生産性データ	`docs/research/2026-01-08-ai-coding-tools-productivity-statistics.md`
AIツールセキュリティリスク	`docs/research/2026-01-21-ai-coding-tool-specific-security-risks.md`
公開コード対策設定	`docs/research/2026-01-14-github-copilot-amazon-q-public-code-block-settings.md`
SCA段階的導入	`docs/research/2026-01-14-sca-phased-implementation-best-practices.md`
日本AI著作権法	`docs/research/2026-01-21-japan-ai-copyright-law-code-generation.md`

付録: データ出典元（1/3）生産性・採用率

p.7 タスク完了時間 55.8%短縮（95名）

github.blog/news-insights/research/research-quantifying-github-copilots-impact...

p.7 タスク完了時間 21%短縮（96名）

research.google/pubs/ai-assisted-code-authoring-at-scale...

p.7 PR 26%増加（5,000名以上）

accenture.com/.../Accenture-A-New-Era-Of-Generative-AI-For-Everyone-V2.pdf

p.7 84%の開発者がAIツール使用

survey.stackoverflow.co/2025/ai

p.7 41%のコードがAI支援で生成

getdx.com/blog/ai-assisted-engineering-q4-impact-report-2025/

付録: データ出典元（2/3）セキュリティ・法的

p.8 45%に脆弱性含有

veracode.com/state-of-software-security-report

p.9 著作権法第30条の4の解釈

bunka.go.jp/.../94037901_01.pdf

p.15 65 lexemes以上、マッチ頻度1%未満

docs.github.com/copilot/using-github-copilot/finding-public-code-that-matches...

p.17 IP補償の条件（Block設定必須）

resources.github.com/.../establishing-trust-in-using-github-copilot/

付録: データ出典元（3/3）セキュリティ・法的

p.19 セキュアプロンプトで56%→66%改善

veracode.com/blog/genai-code-security-report/

p.33 SCA段階的導入ベストプラクティス

checkmarx.com/learn/sca/12-software-composition-analysis-best-practices/

付録: 訴訟・事例の出典

p.9 Doe v. GitHub — Copilotの学習・出力におけるOSSライセンス違反を主張（係争中）

theverge.com/.../microsoft-openai-github-copilot-class-action-lawsuit...

p.9 Anthropic和解 — 海賊版サイトから700万冊をDLし学習に使用、15億ドルで和解

reuters.com/legal/litigation/anthropic-settles-suit...

p.9 読売新聞 vs Perplexity — AI検索による記事無断利用で21.7億円の損害賠償請求

yomiuri.co.jp/national/20250825-OYT1T50087/