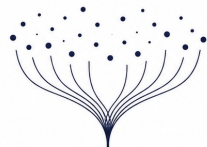


～Bplats 様に AI 駆動開発を全面導入したプロジェクトの舞台裏～

AI 駆動開発支援基盤 AMANEX のご紹介



AMANEX

あまねく、たどりつく。

Everywhere, everything, within reach.

2026年8月 Koozyt, Inc

クウジット (Koozyt) について

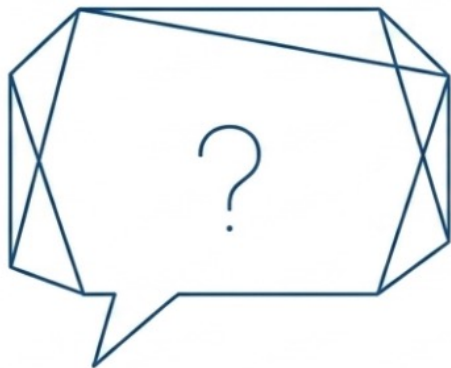


未来のライフスタイルからあるべき形を想像する
共創型テクノロジー企業へ

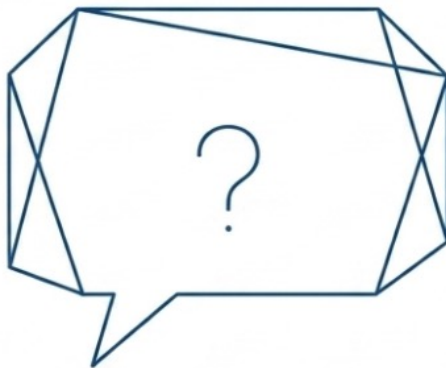
空と実をつなぐクウジットです



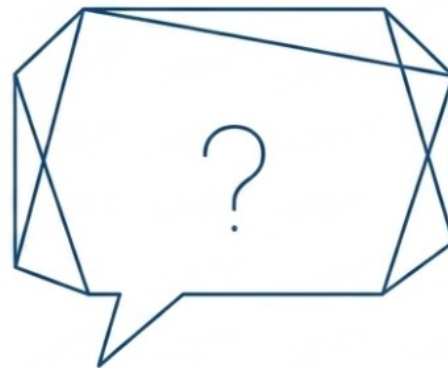
AI がコーディングしてくれる時代における “誤解”



「AIがコードを書くな
ら、ドキュメントはもう
なくても困らない」



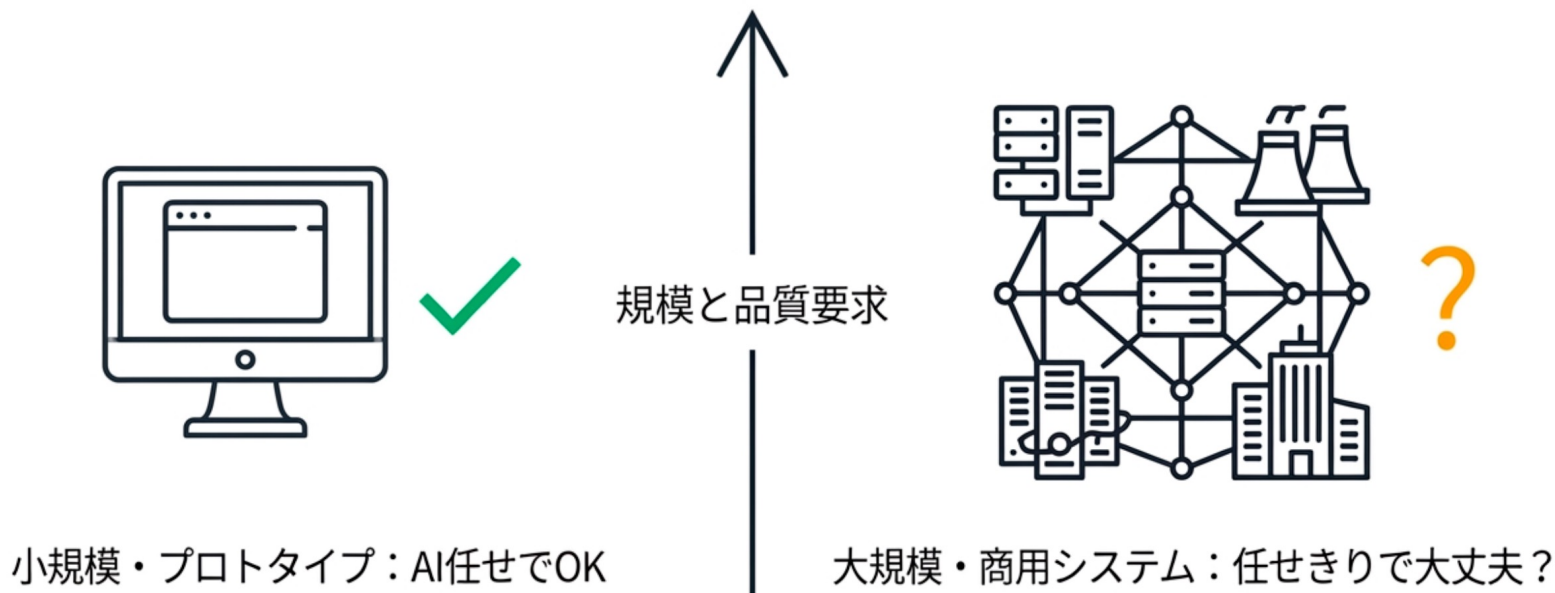
「分からないことは、
その都度 AI に聞けば
答えてくれる」



「必要ならドキュメント
も AI が作ってくれるか
ら問題ない」

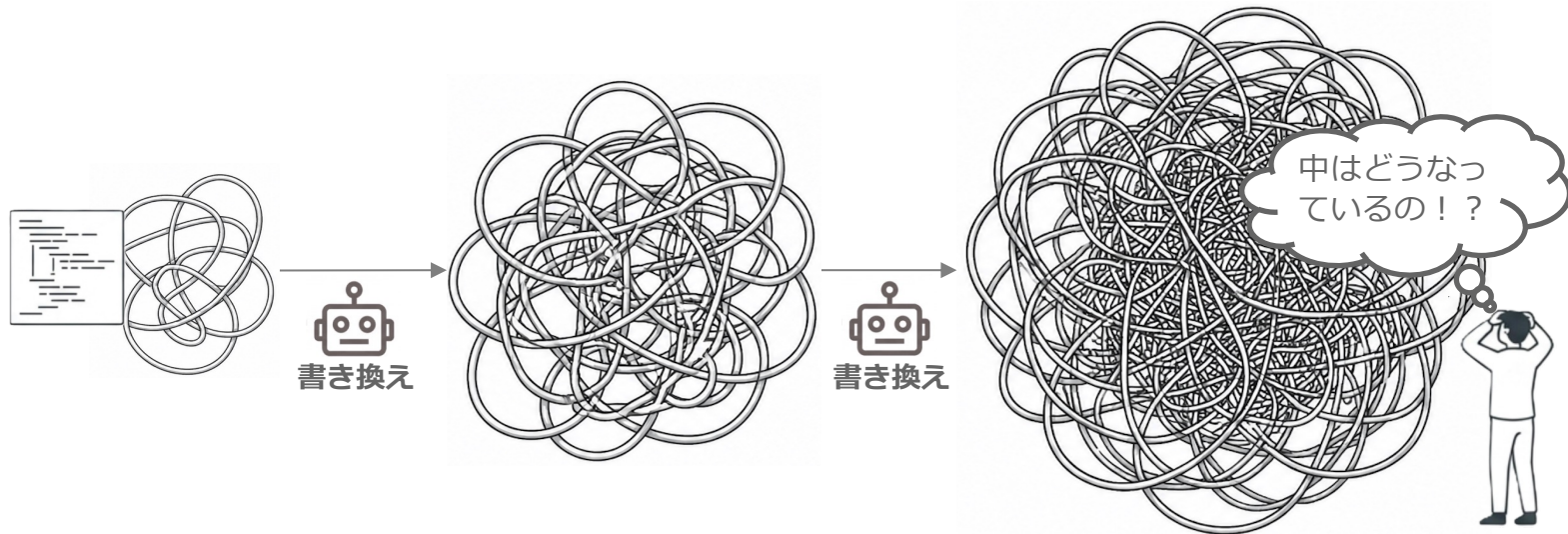
はたして本当にそうなのでしょうか？

大規模な商用システムでは話が変わる



- 小規模プロトタイプ／社内ツールなら、AI 任せで十分速く作れます
- 顧客に対する信頼と継続運用が求められる大規模商用システムを AI だけに任せると問題が顕在化します

AI コーディングは短期間で巨大なブラックボックスができあがる



別セッションのAIが継ぎ足すたび、構造の一貫性が崩れ、より複雑でスパゲッティ化していく

- AI は大量のコードを素早く生成しますが、設計や依存関係は誰の頭にも残らないまま積み上がります
- 大きなシステムでは、かえってレビューや保守コストが膨らみます

「分からなければ都度AIに聞く」にも限界がある

探索の限界

巨大コードの全体を毎回読ませることはできず、**浅い探索**で答えてしまう

ハルシネーション

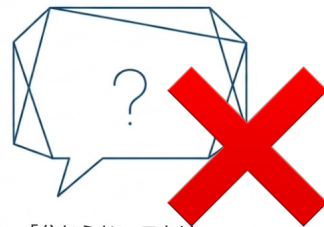
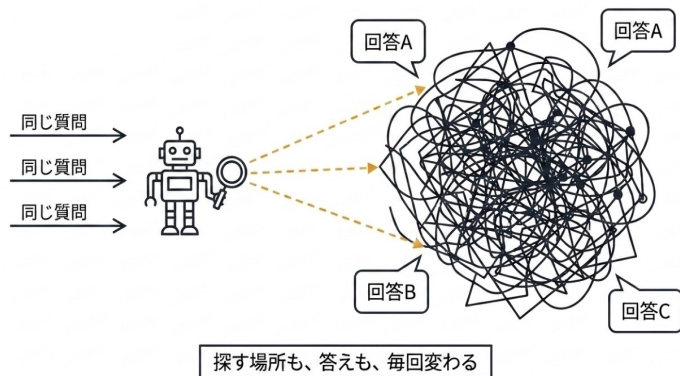
誤りが混じっても、どこにどのような誤りがあるかの**検出が難しい**

非決定性

同じ質問でも、毎回探す場所も**答えも変わ**りうる

コスト

問い合わせのたびに巨大コードを読み直すのは、**時間もトークンも高くつく**

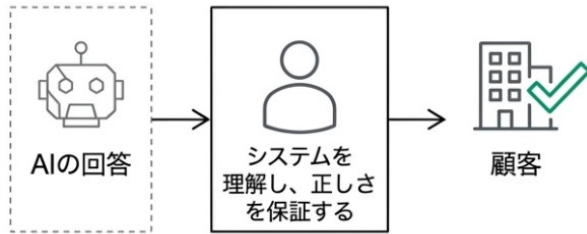


「分からないことは、
その都度 AI に聞けば
答えてくれる」

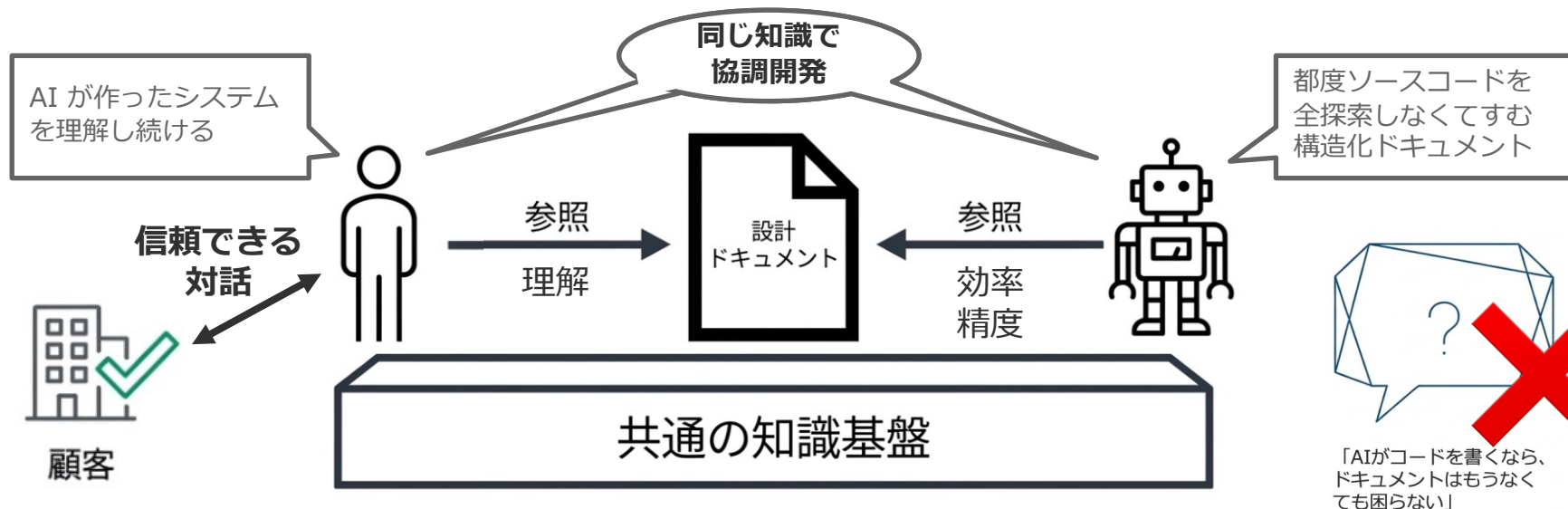
その回答を誰が保証するのか — 信頼は精度だけでは決まらない

顧客からの問い合わせに、誰が回答内容を保証するのでしょうか？ 興味深い論文があります

- 医療の実験では診断精度が同じでも、被験者はAIでなく人間の医師を選びました
(Longoni et al., 2019) — 人にはこの心理的バイアスがあります
- 「AIを使った」と明かすだけで信頼が下がる、という研究報告もあります
(Schilke & Reimann, 2025) — AIの回答を“AIのまま”顧客に返すと信頼低下を招く恐れ
- だから、間に人が立って回答を保証すべき — そのためには、人がAIの回答を理解し、正しさを確かめられなければならない

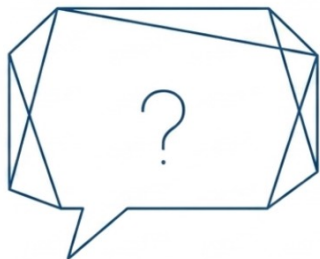


人と AI が共有する“共通の知識基盤”、それが設計ドキュメント



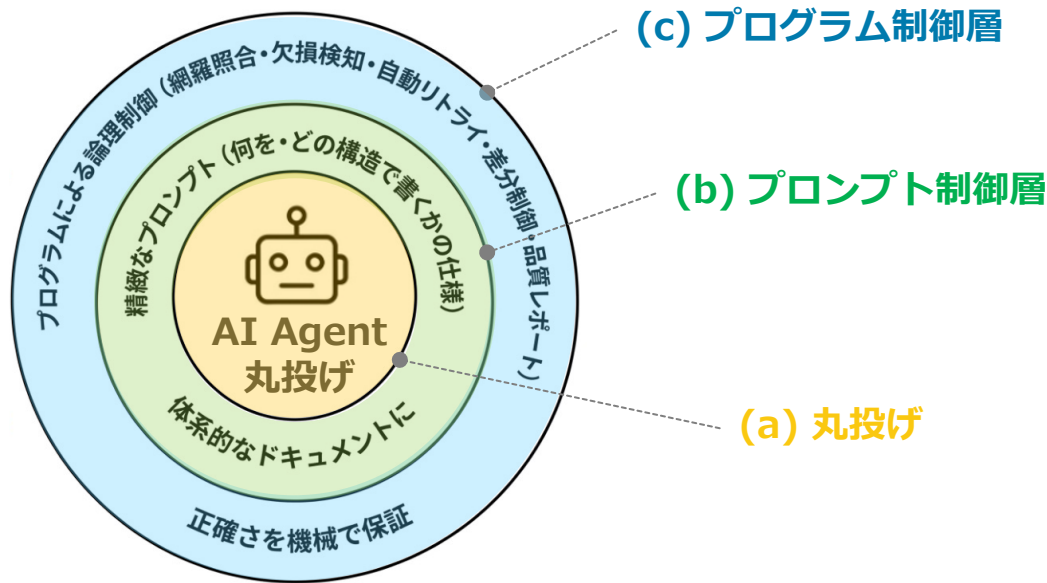
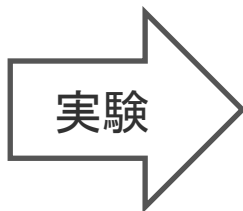
- 人と AI が同じ知識を参照し、同じ知識ベースで対話し、役割分担しながら協調して開発を進めます
- AI コーディング時代だからこそ
「コード → ドキュメント化 → 知識の共有 → AI との協調開発」が重要になります

では、そのドキュメントは AIに丸投げで作れないのか



「必要ならドキュメントも AI が作ってくれるから問題ない」

本当？

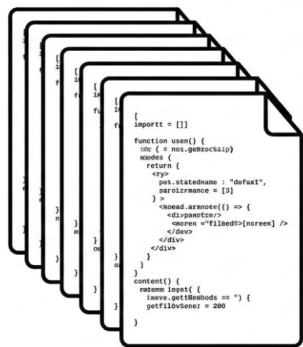


次の3レベルで実験

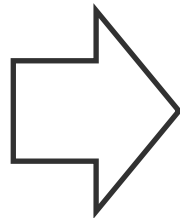
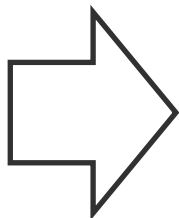
※(a)(b)は同一の AI Agent ・ 同一モデル (Claude Code / Opus 4.8) で実施

- (a) AI Agent に丸投げする
- (b) 精緻なプロンプトを与える (プロンプト設計)
- (c) プログラム制御によるハーンズ設計を行って AI Agent の外側から不具合検知とリトライ制御を行う

実験 (a): AI 丸投げは 全体“概要”をいくつか作成して終了



大規模ソースコード群
(PHP 8,000ファイル規模)



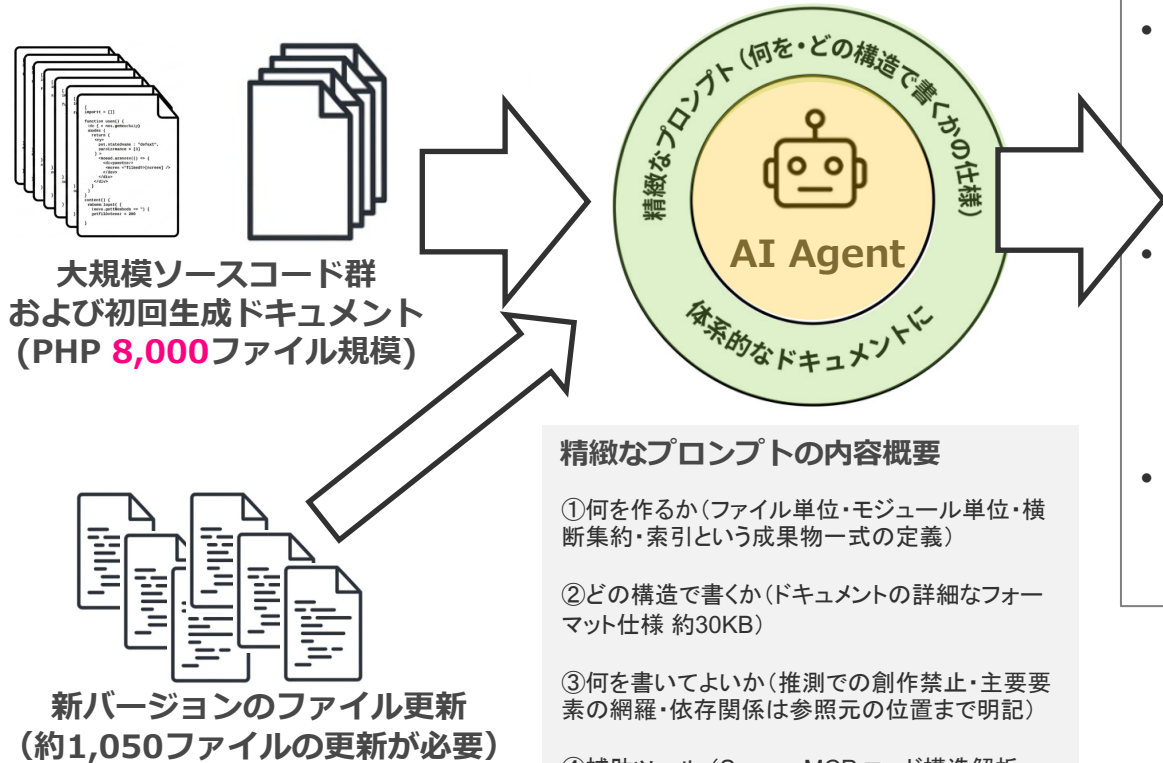
実際のプロンプト

このフォルダのソースコードを読んで、開発者向けの技術ドキュメントを作成してください。
 ソースルート: src
 出力先: docs
 各ソースファイルと各フォルダの説明、および全体の目次(README)を作ってください。

- ファイル単位の網羅ゼロ・相互参照ゼロ、ファイル横断的記載ゼロ
- 概要ドキュメントが6つでしかできませんでした

AI Agent に丸投げでは実用的な設計仕様書は作成できませんでした

実験 (b): 体系的な生成には成功 — しかし差分更新で致命的な失敗



精緻なプロンプトの内容概要

- ①何を作るか(ファイル単位・モジュール単位・横断集約・索引という成果物一式の定義)
- ②どの構造で書くか(ドキュメントの詳細なフォーマット仕様 約30KB)
- ③何を書いてよいか(推測での創作禁止・主要要素の網羅・依存関係は参照元の位置まで明記)
- ④補助ツール (Serena MCP コード構造解析ツールの活用)

- 精緻なプロンプトを与えると、初回は体系的なドキュメント一式の生成に成功 (内容もソースと突き合わせて監査し、専用ツールと互角の品質)
- しかし差分更新では：約1,050ファイルの更新が必要な新バージョンを投入したところ、244ファイル(約4件に1件)が“黙って”未反映でした
- 変更部分に対応したドキュメントを漏れなく更新するのは、精緻なプロンプトでも困難なようです

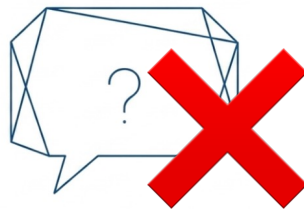
AI まかせの弱点は“失敗が見えない”こと

できました！といいながら黙って漏れ落ちが起きる



ファイル欠落だけでなく、ファイル内の項目が欠落することもあります

どこが失敗しているかわからないため
結果としてかえってレビューコスト
を増大させてしまいます

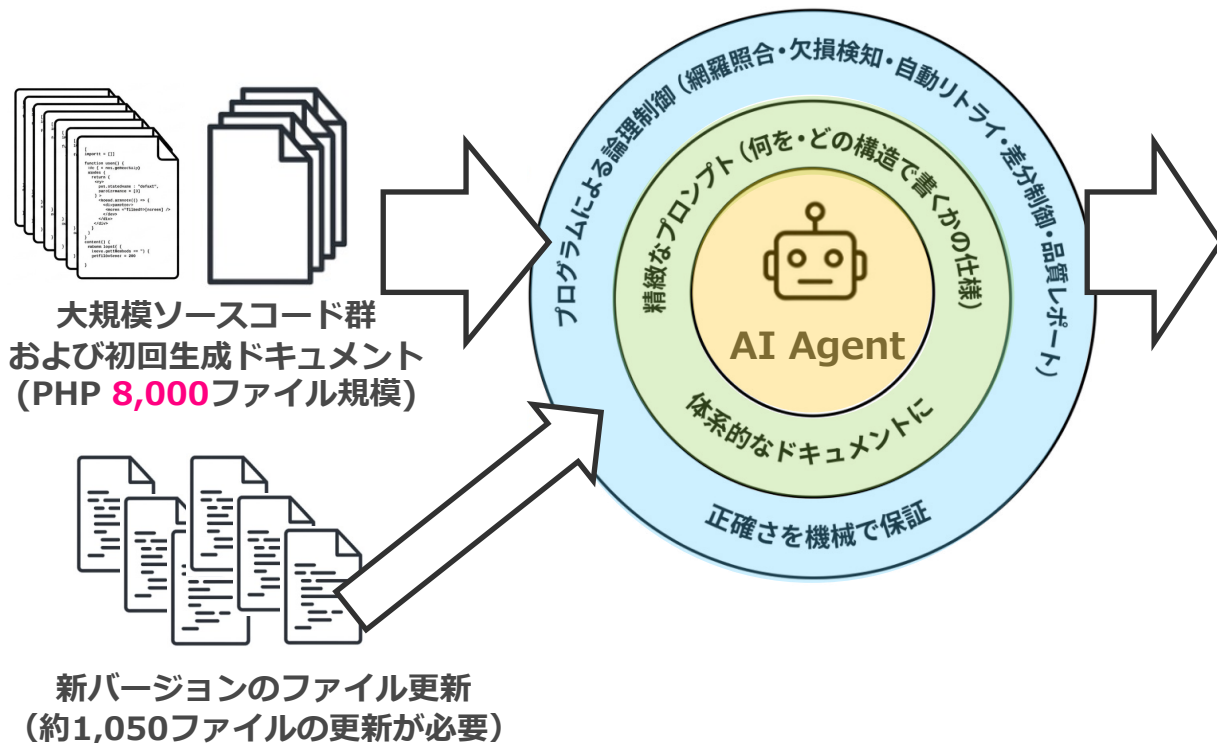


「必要ならドキュメント
も AI が作ってくれるから
問題ない」

- 処理しなかったファイルがあっても**黙って完了**になります
- 同じ入力でも毎回結果が変わり得ます (run1 ≠ run2)
- そのため機械的な検査をしないかぎり誰も気づきません — ドキュメントが**静かに劣化**していきます

実験 (c): 正しいものができるまでリトライして完全なものを生成

網羅性のチェック / 欠損検知 / 自動リトライ / 未生成箇所の明示

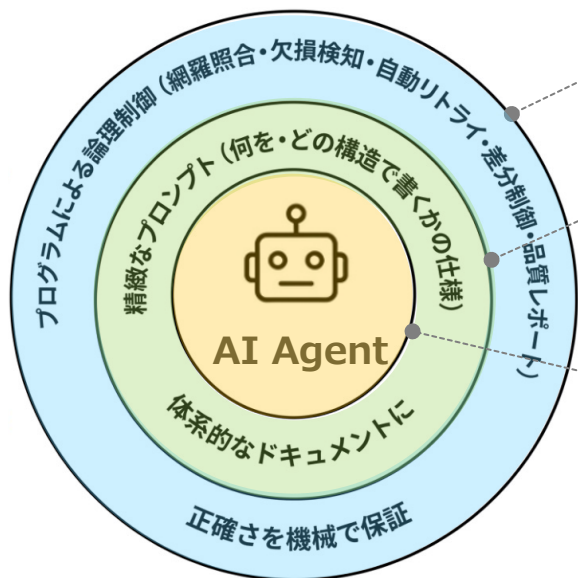


- 更新が必要な約1,050ファイルを漏れなく反映し、据置の約7,000ファイルは*いっさい触らない* — 対象すべてで抜け落ちなしでした
- 各ファイルの*内容も抜け落ちなし*でした
- 万ーリトライ上限まで失敗しても、そのファイルをログに記録 ("*失敗の見える化*") します

AI + 機械的なチェック&リトライの重要さを実測値で示しました

AMANEX docgen とは

この3層設計をまとめてパッケージ化したもの



(c) プログラム制御層

(b) プロンプト制御層

(a) AI Agent (現時点では Claude Code)

AMANEX docgen

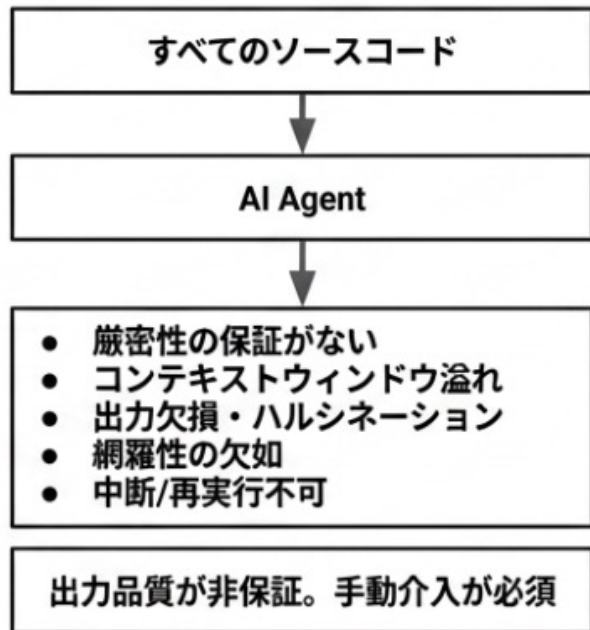


ドキュメント生成対応言語

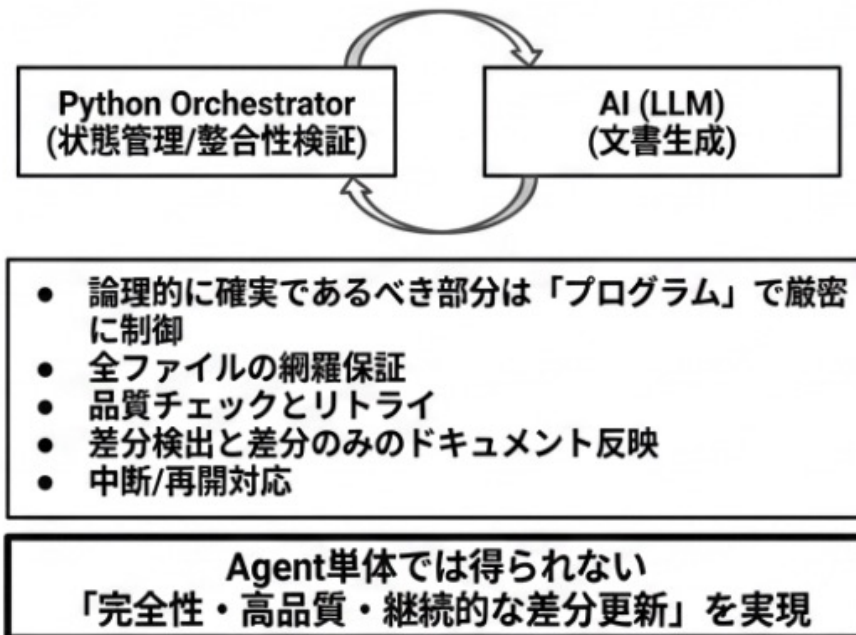
JavaScript, TypeScript, Python, PHP, C/C++, C#, Ruby, Go, Rust など
ビジネスでの使用率調査で上位に出てくる言語を中心に対応済み
今後も拡充予定

AMANEX docgen のアプローチ ～ AI Agent に丸投げしない

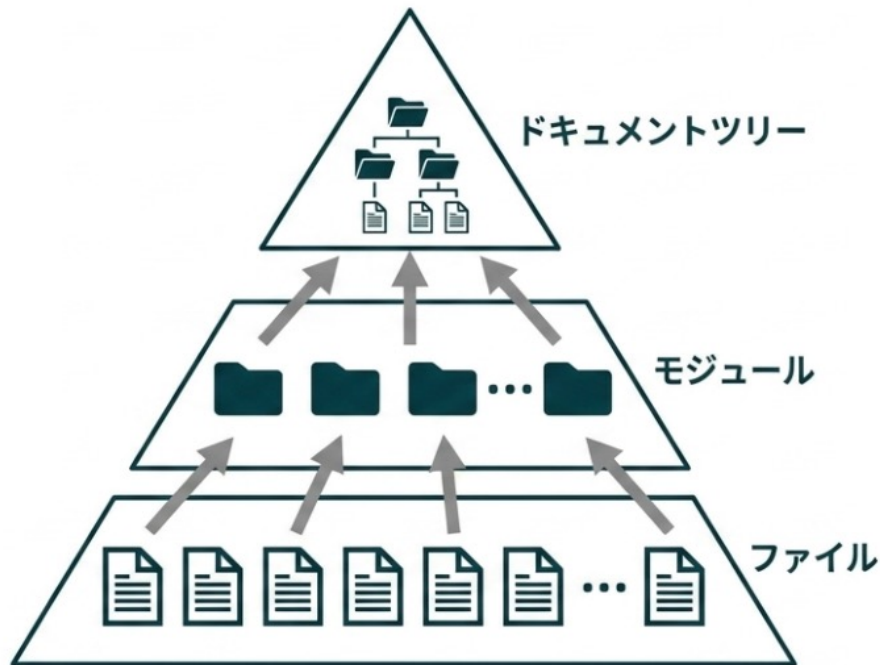
従来のアプローチ (AI単体)



AMANEXのアプローチ(Python + AI Agent)

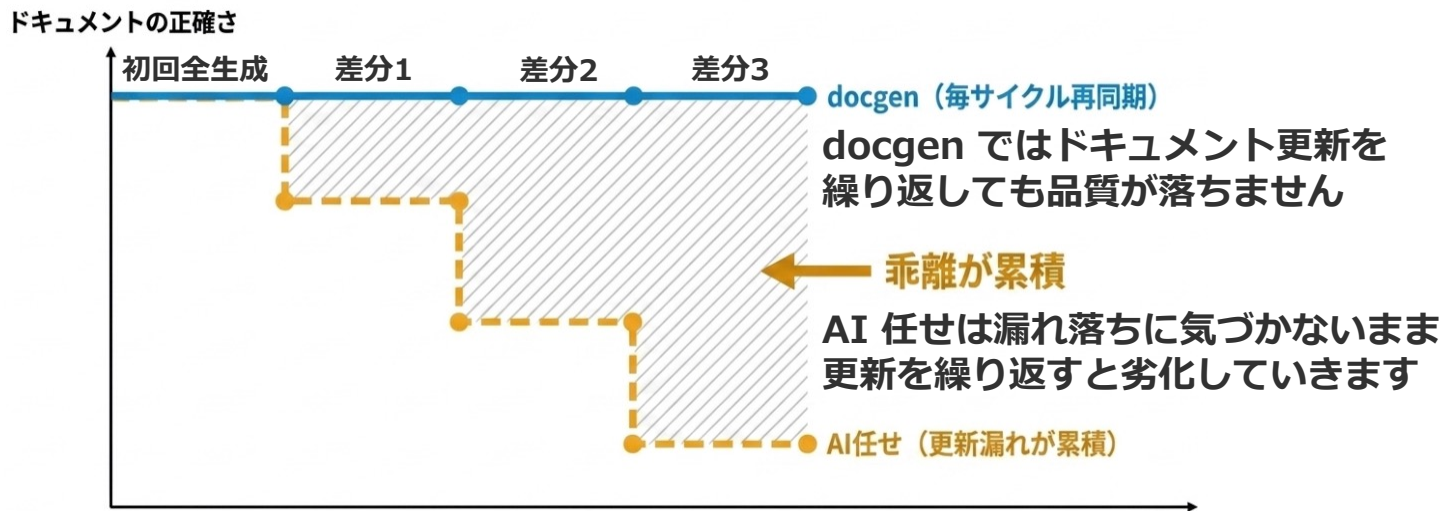


AMANEX docgen のアプローチ ～ ボトムアップビルド



- 最下層ファイル → 上位モジュールへ段階的に解析・集約 — 巨大コードでも漏れ落ちなく完遂し、完成後はトップダウンで辿れるドキュメントツリーになります
- ER図・DB定義書・モジュール依存関係・セキュリティ/不具合懸念など、エンジニアが必要とする図解や一覧も出力します（自動でHTML化しブラウザ閲覧）
- コード構造は Serena（定義・参照・継承の解析: LSP）で正確に把握します

AMANEX docgen のアプローチ ～ ドキュメントの差分更新



- 変更箇所に対応する部分だけを置き換え、変わらない部分は触らない — git 管理でも差分が汚れません
- 読む範囲・書く範囲が最小なので、更新コストも最小
- 毎サイクルでチェック&リトライが動くため、更新を繰り返しても品質が落ちません — AI 任せは漏れ落ちに気づかないまま劣化していきます

実例紹介: ～ ビープラッツ様のケース ～

対象リポジトリ : 7950ファイル + 1000モジュール
生成ドキュメント数 : 9,083件 (75MB)

記載される内容例

ファイル/モジュールドキュメント

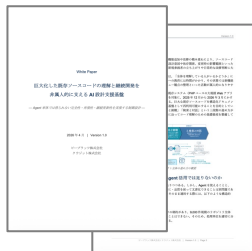
概要, 設計パターン, クラス/関数一覧, 依存関係, 定数一覧, データ構造, セキュリティ考慮, 技術的負債・既知の問題, 拡張性・保守性, など

ファイル横断/全体ドキュメント

定数・Enum 値 逆引き辞書, DB 定義書, 依存パッケージ一覧, ER 図, エラーコード一覧, 共通コンポーネント・ヘルパー関数一覧, モジュール依存図, ユニットテスト一覧・機能対応表, など



出力例 (※ 実際のビープラッツ様のドキュメントでは有りません)



AI 導入支援コンソーシアム会員ページにて
ホワイトペーパー公開中

【デモ①】 ブラウザによるドキュメント閲覧例

某個人向け資産管理アプリ
PHP, JavaScript, Vue 合計1,800ファイル (約30万行)

TOP ファイル・モジュール 定義詳細 その他の最新ドキュメント

ファイル名で検索...

- monolog
 - monolog
 - Attribute
 - Formatter
 - Chrono.phpFormatter.php
 - ElasticFormatter.php
 - ElasticsearchFormatter.php
 - FlowdockFormatter.php
 - FluentFormatter.php
 - FormatterInterface.php
 - GetMessageFormatter.php
 - GoogleCloudLoggingFormatter.php
 - HtmlFormatter.php
 - JsonFormatter.php
 - LineFormatter.php
 - LogglyFormatter.php
 - LogmaticFormatter.php
 - LogstashFormatter.php
 - MonologFormatter.php
 - NormalizerFormatter.php
 - ScalarFormatter.php
 - SyslogFormatter.php
 - WildfireFormatter.php
 - Handler
 - Processor
 - Test
 - DateTimeImmutable.php
 - ErrorHandler.php
 - JsonSerializableDateTimeImmutable.php
 - Level.php
 - Logger.php
 - LogRecord.php
 - Registry.php
 - ResettableInterface.php
 - SignalHandler.php
 - Utils.php

FormatterInterface.php

基本情報

項目	内容
ファイルパス	src/Monolog/Formatter/FormatterInterface.php
言語	PHP
種別	Interface
行数	38行

概要

Monolog のフォーマッター層の基底インターフェースを定義するファイル。すべてのフォーマッタークラスはこのインターフェースを実装する必要がある。ログレコードの種別 (単一レコードおよびバッチ) の契約を定義し、Handler がフォーマッターを差し替え可能なための抽象化を提供する。システム全体のフォーマッター一括リネームの中間として機能する。

設計パターン

パターン名	適用箇所	説明
Strategy	インターフェース全体	Handler がフォーマッターを自由に差し替え可能な契約を定義

主要な構成要素

クラス / 関数一覧

名前	種別	可読性	シグナチャ	概要
FormatterInterface	Interface	public	---	フォーマッターの契約を定義するインターフェース
format	method	public	format(LogRecord \$record): mixed	単一のログレコードをフォーマットする
formatBatch	method	public	formatBatch(array \$records): mixed	複数のログレコードを一括フォーマットする

重要なメソッド・関数の詳細

format(LogRecord \$record): mixed

- 目的:** 単一のログレコードを指定された形式にフォーマットする
- 引数:**
 - `$record` (LogRecord): フォーマット対象のログレコード
- 戻り値:** mixed - フォーマットされたレコード (変数依存で string, array 等)

TOP ファイル・モジュール 定義詳細 その他の最新ドキュメント

ファイル名で検索...

- monolog
 - monolog
 - Attribute
 - Formatter
 - Chrono.phpFormatter.php
 - ElasticFormatter.php
 - ElasticsearchFormatter.php
 - FlowdockFormatter.php
 - FluentFormatter.php
 - FormatterInterface.php
 - GetMessageFormatter.php
 - GoogleCloudLoggingFormatter.php
 - HtmlFormatter.php
 - JsonFormatter.php
 - LineFormatter.php
 - LogglyFormatter.php
 - LogmaticFormatter.php
 - LogstashFormatter.php
 - MonologFormatter.php
 - NormalizerFormatter.php
 - ScalarFormatter.php
 - SyslogFormatter.php
 - WildfireFormatter.php
 - Handler
 - Processor
 - Test
 - DateTimeImmutable.php
 - ErrorHandler.php
 - JsonSerializableDateTimeImmutable.php
 - Level.php
 - Logger.php
 - LogRecord.php
 - Registry.php
 - ResettableInterface.php
 - SignalHandler.php
 - Utils.php

FormatterInterface.php

基本情報

項目	内容
ファイルパス	src/Monolog/Formatter/FormatterInterface.php
言語	PHP
種別	Interface
行数	38行

概要

Monolog のフォーマッター層の基底インターフェースを定義するファイル。すべてのフォーマッタークラスはこのインターフェースを実装する必要がある。ログレコードの種別 (単一レコードおよびバッチ) の契約を定義し、Handler がフォーマッターを差し替え可能なための抽象化を提供する。システム全体のフォーマッター一括リネームの中間として機能する。

設計パターン

パターン名	適用箇所	説明
Strategy	インターフェース全体	Handler がフォーマッターを自由に差し替え可能な契約を定義

主要な構成要素

クラス / 関数一覧

名前	種別	可読性	シグナチャ	概要
FormatterInterface	Interface	public	---	フォーマッターの契約を定義するインターフェース
format	method	public	format(LogRecord \$record): mixed	単一のログレコードをフォーマットする
formatBatch	method	public	formatBatch(array \$records): mixed	複数のログレコードを一括フォーマットする

重要なメソッド・関数の詳細

format(LogRecord \$record): mixed

- 目的:** 単一のログレコードを指定された形式にフォーマットする
- 引数:**
 - `$record` (LogRecord): フォーマット対象のログレコード
- 戻り値:** mixed - フォーマットされたレコード (変数依存で string, array 等)

外部依存 (このモジュールが依存する内部モジュール)

依存先モジュール	主要クラス	用途
src/Monolog	Logger, LogRecord, Level	ログレベル変換、ログレコード参照

内部依存 (このモジュールに依存するもの)

依存先モジュール	主要ファイル	用途
src/MonologHandler	FingersCrossedHandler.php, SyslogHandler.php, ...	アクティベーション戦略の使用、デフォルト戦略の生成

データフロー

Mermaid形式

```

graph LR
    LogRecord --> FingersCrossedHandler
    FingersCrossedHandler --> ActivationStrategy
    ActivationStrategy -- true --> BufferFlush[バッファフラッシュ]
    ActivationStrategy -- false --> AsyncFlush[非同期バッファフラッシュ]
  
```

セキュリティ考慮

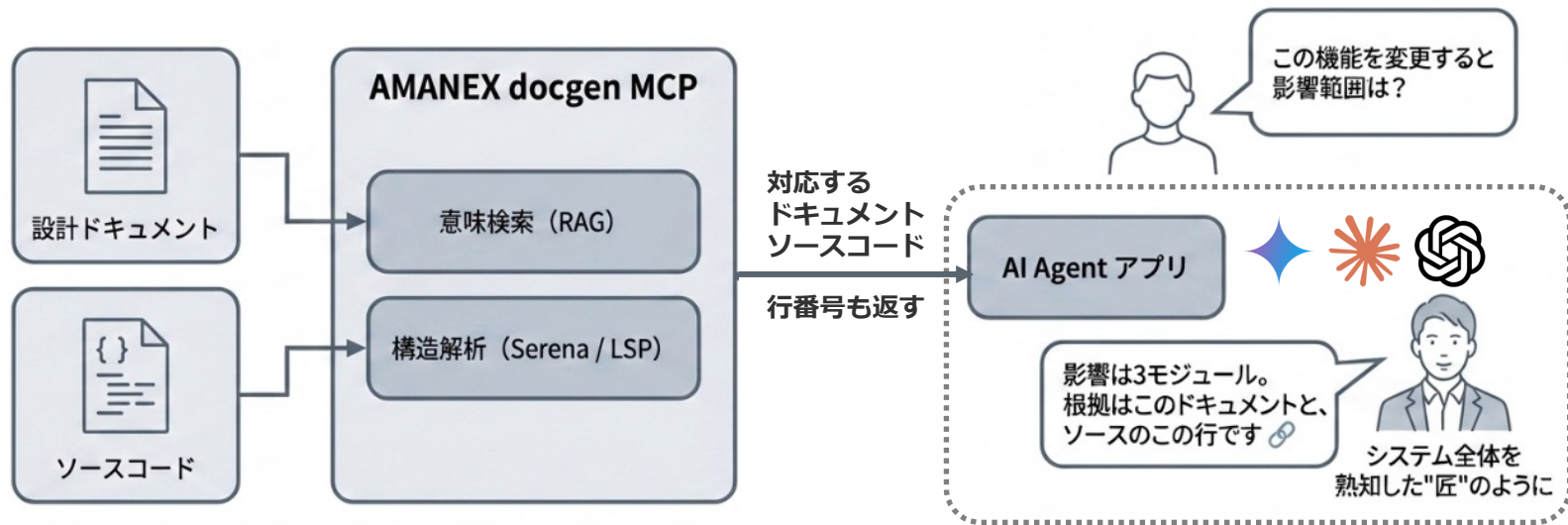
観点	対策状況	詳細
入力検証	該当なし	ログレコードの検証のみ
認証/認可	該当なし	アクセス制御なし
機密データ	該当なし	機密データの処理なし

技術的負債・既知の問題

特に問題は見出されませんでした。

※画面イメージは今回のデモとは別の Monolog の分析結果です

生成ドキュメントとソースコードが、“対話できる匠”になる



- 生成ドキュメントは、人が読むだけでなく **AI の“入力”として活用できます**（浅い探索を防ぎ、仕様に従った安全なコード変更を支える）
- 設計ドキュメント：意味検索、ソースコード：構造解析で索引化し、AMANEX docgen MCP が束ねます。MCP なので**既存の AI Agent プログラムと連携**できます。
- 手持ちの AI Agent から接続して質問すると、関連ドキュメントと該当ソースの場所を**根拠として返す** — システムを熟知した匠・ベテランPMと対話するように使えます

【デモ②】 ドキュメントとソースコードを 把握した“匠 AI”と対話して設計する デモ①の個人向け資産管理アプリ PHP, JavaScript, Vue 合計1,800ファイル (約30万行) 生成ドキュメント 2,340ファイル (約38万行)

AMANEX docgen chat

monolog / 1907 churks

🔍 検索

📄 新しい会話

AMANEX docgen chat

monolog の設計について何でも聞いてください。

機密情報がログに漏れるリスクや対策はどこと？

機密情報の漏洩リスクと対策は、設計ドキュメントの各コンポーネントの「セキュリティ考慮」節と `LogRecord` の実装に分散して記述されています。結論から言うと、`Monolog` はログ内書きのファイルタリングを行わず、機密データの管理は基本的には利用者の責務とされています [1]。

- 中核ライブラリ (src/Monolog)**：機密データ対策は「なし」で、ログ内書きのファイルタリングは利用者の責務と明記されています。入力検証としては `Logger::toMonologLevel()` によるレベル値の検証や、`Utils` での JSON UTF-8 の検証・修復は行われますが、機密情報そのもののマスキングは提供されません [1]。
- プロセス (src/Monolog/Processor)** `WebProcessor` が IP アドレス等もログに記録するため、GDPR 等の規制に注意が必要とされています。これは「機密データ・法遵」レベルの扱いで、個人情報が入力される典型的な経路です [2]。
- ハンドラー (src/Monolog/Handler)**：外部サービス連携ハンドラーが API キー・トークンによる認証を実施しており、これらの認証情報がハンドラーのプロパティに格納されます。そのためログの出力リクエスト時に API キー・トークンが漏洩しないよう注意喚起がなされています [3]。

- `ErrorHandler` の `handle()` メソッドで、ログのフォーマットやレベルを調整する際に、機密情報がログに漏れるリスクや対策はどこと？

Backにログを遡るにはどのクラス？

ログのタイムスタンプはどう生成される？

例: ログを複数のハンドラに振り分ける仕組みは？

送信

設計ドキュメント src/Monolog

TOP ファイル・モジュール 定義辞書 その他の集約ドキュメント ▼ 品質レポート一覧

ファイル名で検索...

monolog

- BBC
- Monolog
 - php-cs-fixer.php
 - _config.yml
 - phpstan-ignore-by-php-version.neon.php

下のサブモジュールは存在しません。

ファイル

- `DataImmutable.php` - `JsonSerializableDataImmutable` の非推奨エイリアス
- `ErrorHandler.php` - PHP エラー/例外/警告のイベントのログ連携
- `JsonSerializableDataImmutable.php` - `Json` シリアライズ可能な日時クラス
- `Level.php` - ログレベル定義 (PHP 8.1 backed enum)
- `LogRecord.php` - ログレコードオブジェクト
- `Logger.php` - ログチャンネルの抽象クラス
- `Registry.php` - `Logger` インスタンスのグローバルレジストリ
- `ResettableInterface.php` - リセット可能なコンポーネントのインターフェース
- `SignalHandler.php` - POSIX シグナルのログ連携
- `Utils.php` - `JSON` 文字列パース等の共通ユーティリティ

ソースコード src/Monolog/Utils.php

237 行 | バイト 132~236

```

11  ...
12  namespace Monolog;
13
14  final class Utils
15  {
16      const DEFAULT_JSON_FLAGS = JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE | JSON_PRESERVE_ZERO_FRACTION | JSON_INVALID_UTF_DETECT;
17
18      public static function getClass(object $object): string
19      {
20          $class = \get_class($object);
21
22          if (false === ($pos = strpos($class, "Anonymous("))) {
23              return $class;
24          }
25
26          if (false === ($parent = get_parent_class($class))) {
27              return substr($class, 0, $pos + 10);
28          }
29      }
30  }

```

AMANEX docgen chat

monolog / 1907 churks

🔍 検索

📄 新しい会話

AMANEX docgen chat

monolog の設計について何でも聞いてください。

機密情報がログに漏れるリスクや対策はどこと？

機密情報の漏洩リスクと対策は、設計ドキュメントの各コンポーネントの「セキュリティ考慮」節と `LogRecord` の実装に分散して記述されています。結論から言うと、`Monolog` はログ内書きのファイルタリングを行わず、機密データの管理は基本的には利用者の責務とされています [1]。

- 中核ライブラリ (src/Monolog)**：機密データ対策は「なし」で、ログ内書きのファイルタリングは利用者の責務と明記されています。入力検証としては `Logger::toMonologLevel()` によるレベル値の検証や、`Utils` での JSON UTF-8 の検証・修復は行われますが、機密情報そのもののマスキングは提供されません [1]。
- プロセス (src/Monolog/Processor)** `WebProcessor` が IP アドレス等もログに記録するため、GDPR 等の規制に注意が必要とされています。これは「機密データ・法遵」レベルの扱いで、個人情報が入力される典型的な経路です [2]。
- ハンドラー (src/Monolog/Handler)**：外部サービス連携ハンドラーが API キー・トークンによる認証を実施しており、これらの認証情報がハンドラーのプロパティに格納されます。そのためログの出力リクエスト時に API キー・トークンが漏洩しないよう注意喚起がなされています [3]。

- `ErrorHandler` の `handle()` メソッドで、ログのフォーマットやレベルを調整する際に、機密情報がログに漏れるリスクや対策はどこと？

Backにログを遡るにはどのクラス？

ログのタイムスタンプはどう生成される？

例: ログを複数のハンドラに振り分ける仕組みは？

送信

設計ドキュメント src/Monolog/Handler/Slack/SlackRecord.php

TOP ファイル・モジュール 定義辞書 その他の集約ドキュメント ▼ 品質レポート一覧

ファイル名で検索...

monolog

- BBC
- Monolog
 - Attribute
 - Formatter
 - Handler
 - Curl
 - ErrorHandler
 - Slack
 - SlackRecord.php
 - Utils.php

下のサブモジュールは存在しません。

ファイル

- `DataImmutable.php` - `JsonSerializableDataImmutable` の非推奨エイリアス
- `ErrorHandler.php` - PHP エラー/例外/警告のイベントのログ連携
- `JsonSerializableDataImmutable.php` - `Json` シリアライズ可能な日時クラス
- `Level.php` - ログレベル定義 (PHP 8.1 backed enum)
- `LogRecord.php` - ログレコードオブジェクト
- `Logger.php` - ログチャンネルの抽象クラス
- `Registry.php` - `Logger` インスタンスのグローバルレジストリ
- `ResettableInterface.php` - リセット可能なコンポーネントのインターフェース
- `SignalHandler.php` - POSIX シグナルのログ連携
- `Utils.php` - `JSON` 文字列パース等の共通ユーティリティ

ソースコード src/Monolog/Handler/Slack/SlackRecord.php

281 行 | バイト 130~380

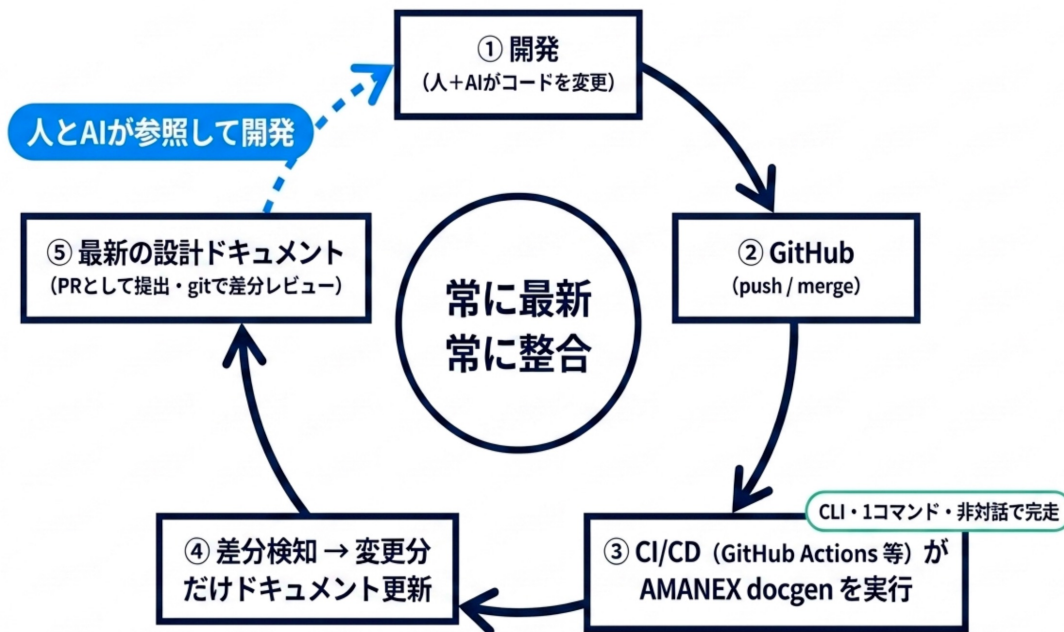
```

11  namespace Monolog\Handler\Slack;
12
13  use Monolog\Level;
14  use Monolog\Utils;
15  use Monolog\Formatter\NormalizerFormatter;
16  use Monolog\Formatter\FormatterInterface;
17  use Monolog\Handler\HandlerInterface;
18
19  /**
20   * Slack record utility helping to log to Slack webhooks or API.
21   *
22   * @author Greg Kedzierski <greg@regedzierski.com>
23   * @author Haralan Dobrev <hdobrev@gmail.com>
24   * @see https://api.slack.com/incoming-webhooks
25   * @see https://api.slack.com/docs/message-attachments
26   */
27  class SlackRecord
28  {
29      ...
30  }

```

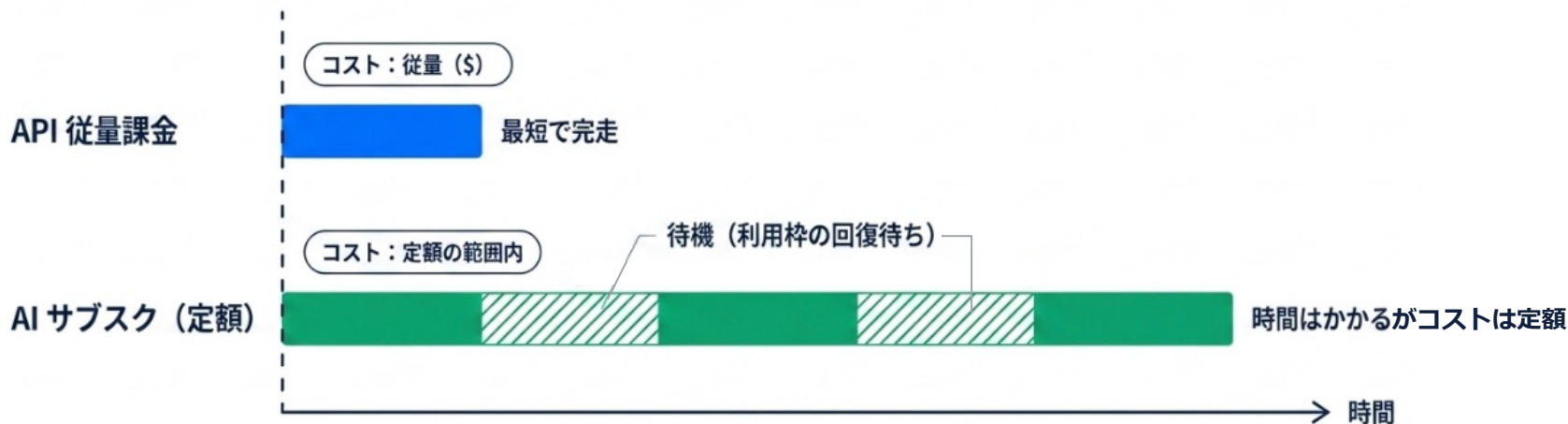
※画面イメージは今回のデモとは別の Monolog の分析結果です

AMANEX docgen を AI 駆動開発サイクルに載せる



- **GitHub 連携**：リポジトリ URL を指定するだけで初回生成・差分更新を直接実行（ブランチ/タグ指定可）
- CLI ツール（1コマンド・非対話・自動再開）なので、**既存の CI/CD や独自の開発ループに組み込みやすい**

運用時の AI コストの目安 — API 従量課金とサブスク

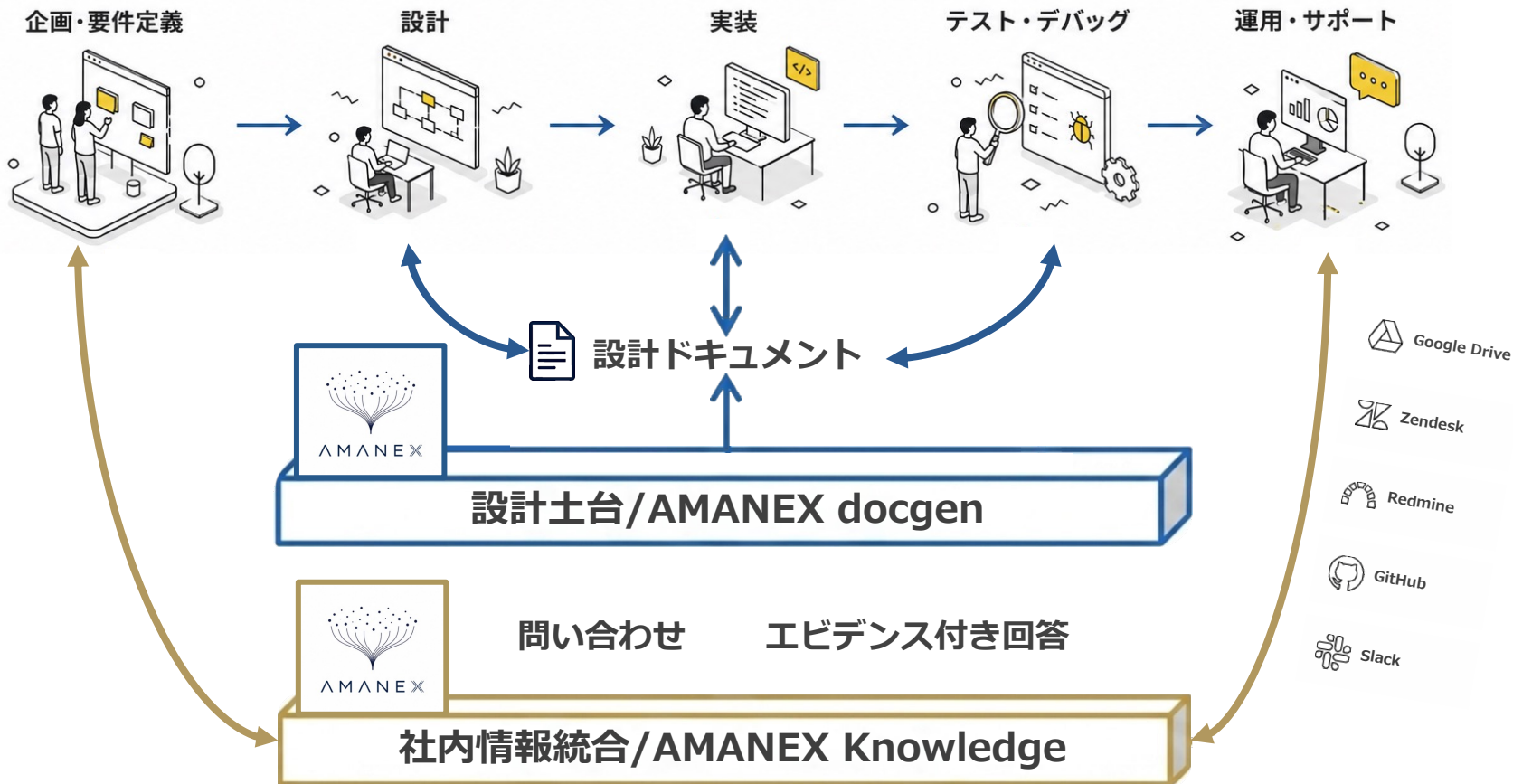


docgen は利用枠の上限を自動検知して待機・自動再開 — 人が張り付く必要はありません

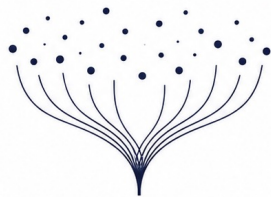
AMANEX docgen は現時点では Anthropic社の Claude Code を使用します

- 初回生成：bplats さんの 約8,000ファイルの商用システム実績で **\$1,800~\$2,500 程度(従量課金換算)**
- 差分更新：1ファイルあたり \$0.1~\$0.5 (目安 \$0.2~\$0.3) (従量課金換算) (言語・変更量で変動)
- AI サブスク利用時は**定額内でコストを抑えられる**一方、利用枠の待機で**総所要時間は伸びます**

AI駆動開発の全フェーズを支える統合インフラストラクチャの土台へ



生成は AI に、保証は仕組みに、理解は人に その役割分担を支える — AMANEX



AMANEX

あまねく、たどりつく。

Everywhere, everything, within reach.