

AI時代の価値継続性設計： Boundary-Oriented Architecture (BOA) と VCDesign による自律システムの構造改革 エグゼクティブサマリー

生成AI (Generative AI) と大規模言語モデル (LLM) の急速な普及は、ソフトウェアエンジニアリングの歴史において、アセンブリ言語から高水準構造化プログラミングへの移行に匹敵する、あるいはそれ以上のパラダイムシフトをもたらしている。しかし、過去の技術転換が人間の認知能力を拡張するための抽象化層の構築に焦点を当てていたのに対し、現在の転換はソフトウェア開発の根本的な前提——すなわち、コードは人間という「支配者 (Ruler)」によって手作りされ、その実行パスは完全に決定論的であり、信頼できる資産であるという前提——を根底から覆そうとしている。

我々は今、「バイブコーディング (Vibe Coding)」の時代、すなわちAI主導の開発時代に突入しつつある。この新しい世界では、コードは確率論的に生成され、ロジックは流動的であり、開発者の役割は構文を書く「作成者」から、システムの境界を設計する「立法者 (System Legislator)」へと変貌を遂げなければならない。この文脈において、Model-View-Controller (MVC)、ドメイン駆動設計 (DDD)、クリーンアーキテクチャ (Clean Architecture) といった伝統的なアーキテクチャパターンは、重大な限界に直面している。これらは人間による保守と決定論的なロジックを前提として設計されており、AIの確率的で混沌とした性質 (Stochastic Nature) を封じ込めるには不十分である。その結果、AIが生成した「汚染された」コードベースが、幻覚 (ハルシネーション) によって現実世界に壊滅的な副作用を引き起こすリスクが高まっている。

本レポートは、**Value Continuity Design (VCDesign)** およびその構造的実装である **Boundary-Oriented Architecture (BOA)** について、極めて詳細かつ包括的な分析を提供するものである。本稿では、AIを安全に統合するためには、出力を事後的に修正する「ガードレール」のアプローチを超え、存在そのものを構造化する「境界 (Boundaries)」の設計へと移行しなければならないというテーゼを展開する。具体的には、**Meaning** (意味／流動的ロジック) と **Responsibility** (責任／決定論的実行) を、**Interface Determinability Gate (IDG)** および **Responsibility Promotion (RP)** というプロトコルによって厳格に分離することで、組織はシステム全体の完全性 (Integrity) を損なうことなく、AIの圧倒的な生産性と速度を享受することが可能となる。

分析は、現代のソフトウェア開発における3つの重要なシーンに沿って構成される：

1. レガシーシステムの近代化 (現在のシステム) : AIネイティブ環境におけるMVC/DDDの機能不全と、BOAによる再構築。
2. AIアプリケーションの最前線 (現在のAI適用の課題) : 「Human-in-the-Loop (HITL)」のボトルネック解消と、安全な「バイブコーディング」の実現。
3. 自律的な未来 (AIの自律世界) : 「システム立法者」モデルによる自律エージェントのアライメントと制御。

第1部：価値継続性の理論的基盤とアーキテクチャの危機

1.1 「支配者(Ruler)」アーキタイプ崩壊の序曲

ソフトウェアエンジニアリングの黎明期から現在に至るまで、開発者はシステムに対して全能の「支配者(Ruler)」として振る舞ってきた¹。このアーキタイプにおいて、開発者は絶対的な精度で機械に命令を下す。変数の代入、制御フローの分岐、データベースへの状態変更、これらすべては人間の意図的な意思決定の直接的な結果であり、コンピュータは支配者の意志を忠実に実行する受動的な存在に過ぎなかった¹。このモデルは、コードが希少であり、書くことに高いコストがかかり、それゆえに一度書かれたロジックが「資産」として扱われる時代には極めて合理的であった。

しかし、この支配のモデルは、生成AIの台頭によってその基盤を掘り崩されている。LLMが関数を生成するとき、それは人間のようにシステムの状態やビジネス上の価値(Value)を「理解」して記述しているのではない。それは膨大な学習データに基づき、統計的な尤度(Likelihood)に従って次のトークンを予測しているに過ぎない²。開発者がAIの生成するコードを一行一行精査し、従来の支配者のようにマイクロマネジメントしようとするれば、AIによる効率化の恩恵は霧散してしまう。かといって、AIに制御を丸投げすれば、確率的な挙動によるカオスを招き入れることになる。このジレンマこそが、現代のソフトウェアアーキテクチャが直面している危機の正体である。

1.2 Value Continuity Design (VCDesign) の哲学

VCDesignは、この危機に対する応答として、「価値(Value)」の定義を再考することから始まる。ここでの価値とは、単なる機能要件の充足ではなく、システムが変化し続ける環境の中で、その本質的な目的と整合性を維持し続ける能力(Continuity)を指す³。

VCDesignは、開発者のアイデンティティを **Ruler**(支配者) から **System Legislator**(システム立法者) へとシフトさせることを提唱する¹。現実世界の立法者が、市民一人一人の行動を逐一監視・指示するのではなく、「憲法(Constitution)」と「法(Laws)」によって許容される行動の境界を定義するように、AI時代の開発者もまた、AIエージェントに対して命令的(Imperative)に振る舞うのではなく、宣言的(Declarative)に境界を施行しなければならない。

1.3 Boundary-Oriented Architecture (BOA) の構造論

BOAは、VCDesignの哲学を具体的なソフトウェア構造へと落とし込んだアーキテクチャパターンである。BOAは、AI時代において「ロジック(思考)はもはや資産ではなく、流動的な素材(**Fluid Material**)である」という前提に立つ³。真に守るべき資産は、システムが依拠する「事実(Fact)」と、システムが外界に及ぼす「責任(Responsibility)」のみである。

BOAは、この流動性を管理するために、システムを以下の厳格な「三位一体(Triad)」に再編成する：

1.3.1 Fact(事実: アンカー)

- 定義: システムの不変の真実(Ground Truth)。

- 構成要素: ユーザー入力、データベースの確定レコード、システムログ、センサーデータ。
- 役割: このレイヤーは読み取り専用(Read-only)である。AIの文脈においては、これはプロンプトのコンテキストや、推論の基盤となる「世界の状態(World State)」として機能する。
- 制約: 事実は議論の余地がなく、AIによって幻覚(ハルシネーション)されてはならない領域である。

1.3.2 Meaning(意味: 流動領域)

- 定義: 解釈、計算、変換、推論が行われる領域。
- 構成要素: ビジネスロジック、アルゴリズム処理、データ変換、下書き生成。
- 役割: ここがAIの独壇場(Playground)である。VCDesignにおいて、このレイヤーのコードは「一時的(Ephemeral)」かつ「使い捨て(Disposable)」として扱われる³。AIエージェントはここで「返金戦略」を生成し、破棄し、再生成することを、1秒間に100回繰り返すことができる。
- 決定的特性: 副作用ゼロ(Zero Side Effects)。Meaningレイヤーは、データベースの変更、外部APIの呼び出し、決済処理の実行を構造的に禁じられている。ここから生まれるのはあくまで「提案(Proposal)」であり、「アクション」ではない³。

1.3.3 Responsibility(責任: 保護領域)

- 定義: 実行と結果、そして副作用の領域。
- 構成要素: データベースへの書き込み、決済ゲートウェイ(Stripe, PayPal等)の呼び出し、メール送信、物理アクチュエータへの指令。
- 役割: システムが現実世界に物理的・経済的影響を与える唯一の場所(Touching Layer)である。
- 制約: AIはこの領域に直接触れることを厳格に禁止される。このレイヤーは決定論的であり、ハードコードされ、ゲートで守られていなければならない。通常、ここは人間による記述か、厳密な形式検証を要する³。

この三層構造により、BOAIは「思考(Thinking)」と「行動(Doing)」を物理的に分断し、AIのカオスを安全な領域に封じ込めることを可能にする。

第2部: シーン1 — レガシーシステムの近代化(現在のシステム)

現代の多くのシステムは、AI統合を前提としていない従来のアーキテクチャ(MVCやDDD)に基づいて構築されている。最初の分析シーンとして、これらの既存システムがAI時代にどのような不全を起こすか(課題)、現在のベストプラクティスがなぜ不十分か、そしてBOAがどのように解決するかを詳述する。

2.1 課題: 汚染されたコントローラー(The Polluted Controller)

現在のAI支援開発、あるいは「バイブコーディング」の現場では、開発者がAIに対して「返品処理を行

う機能を書いて」と指示する場面が日常化している。AIはGitHub上の数百万のリポジトリから学習したパターンに従い、典型的なServiceクラスやControllerメソッドを生成する。

2.1.1 失敗の解剖学

生成されるコードは往々にして以下のような構造を持つ：

1. 注文ステータスを取得する(状態の読み取り)
2. 返品ポリシーに基づき返金額を計算する(ビジネスロジック／思考)
3. もし(If) 金額が正当であれば、Stripe APIを叩いて返金する(副作用／実行)
4. データベースを「返金済み」に更新する(副作用／実行)
5. ユーザーに通知メールを送る(副作用／実行)

この一連の手続き的(Procedural)なコードブロックは、AIの信頼性という観点からは「災害」に等しい³。

- 思考と行動の混合(Entanglement)：「いくら返金すべきか」という思考プロセスと、「実際に金を動かす」という実行プロセスが、同じ実行コンテキスト内で密結合している。
- 幻覚のリスク(The Hallucination Risk)：もしAIが返品ポリシーの解釈で幻覚を起こし(例：「セール品は返品不可」というフラグを見落とす)、誤った返金額を算出した場合、そのロジックは即座に次の行の実行コード(Stripe APIコール)へと流れ込む。ここには、誤った思考が誤った行動になるのを食い止める構造的な障壁が存在しない³。
- レビューの形骸化：これを防ぐために人間がコードレビューを行うとしても、スパゲッティ化したロジックの中に埋め込まれた微妙な幻覚を見抜くことは困難である。人間はコードの「正しさ」を脳内でシミュレーションしなければならず、認知負荷の限界から見落としが発生する。

2.2 現状のベストプラクティス：MVC, DDD, クリーンアーキテクチャ

業界標準の解決策であるMVC、DDD、クリーンアーキテクチャは、複雑性を管理するために考案された優れたパターンであるが、これらはあくまで「人間の認知」のために設計されたものであり、「AIの封じ込め」のために設計されたものではない。

2.2.1 Model-View-Controller (MVC) の限界

MVCは、データ(Model)、表示(View)、制御(Controller)を分離するが、Controller(あるいはその背後のService層)においてロジックと副作用を混在させることを許容している³。MVCの暗黙の前提は、「コードを書いたのは人間であり、その意図は信頼できる」というものである。AIという「信頼できない書き手」が参入した瞬間、この前提は崩壊する。MVCには、確率的に生成されたロジックを検疫するための構造が存在しない。

2.2.2 ドメイン駆動設計 (DDD) の摩擦

DDDは、ドメインモデルの中にビジネスルールをカプセル化し、「ユビキタス言語」を用いて境界づけられたコンテキスト(Bounded Context)を定義する。これはMVCより堅牢だが、AI時代には以下の摩擦が生じる：

- コンテキストの維持：AIは厳密なコンテキスト境界を理解するのが苦手である。AIにコード生成

をさせると、インフラストラクチャの関心事(DB操作など)をドメインエンティティに混入させがちであり、「純粋なドメインモデル」を維持するには多大なプロンプトエンジニアリングが必要となる。

- 資産としてのロジック:DDDはドメインロジックを「核心的な資産」として扱い、時間をかけて洗練させることを是とする。しかし、AI時代においてロジックは瞬時に再生成可能な「使い捨て」のものであり、DDDの重厚なアプローチは開発速度の足枷となる可能性がある。

2.2.3 クリーンアーキテクチャの脆弱性

クリーンアーキテクチャ(オニオンアーキテクチャ等)は、依存性逆転の原則を用いて詳細(UI、DB)からビジネスルールを守る。これはBOAに近い思想だが、実装の複雑さが障壁となる。また、AIが生成する「バイブコード」は、厳密なインターフェースや階層構造を無視して「動くコード」をフラットに出力する傾向があり⁴、クリーンアーキテクチャの厳格な規律を維持することは、AIの自律的なコーディング能力を阻害することにつながる。

2.3 VCDesignによる解決策:「ハードウォール」による分断

VCDesignおよびBOAは、「汚染されたコントローラー」問題に対し、**Meaning** と **Responsibility** の物理的分離によって対処する。

2.3.1 アーキテクチャのリファクタリング:返金提案(Refund Proposal)

BOAの実装において、AIは決して「返金処理を行え」とは命令されない。命令は「返金提案(Proposal)を作成せよ」となる。

ステップ1: Fact(入力)

システムはAIに対し、Order オブジェクトと ReturnPolicy テキストを渡す。これらは読み取り専用の事実である。

ステップ2: Meaning(AIの任務)

AIは RefundProposal オブジェクトを生成する。

- このオブジェクトには refundAmount(金額)、reason(理由)、actions(推奨アクションリスト)が含まれる。
- 決定的な違い:このコードブロックには副作用が一切含まれない。Stripeライブラリはインポートすらされない。DBへの書き込み権限もない。これは純粋なデータ変換関数である³。
- 安全性:副作用がないため、AIが「100万ドルの返金」を幻覚しても、現実世界では何も起こらない。システムは単に「危険なデータ」をメモリ上に保持するだけであり、危険な行動は実行されない。

ステップ3: IDG(Interface Determinability Gate)による検疫 生成された RefundProposal は、IDG と呼ばれる厳格な型チェック・論理チェックのゲートを通過しなければならない¹。

- IDGは検査する:「金額は数値か?」「正の数か?」「元の注文金額を超えていないか?(決定論的ビジネスルール)」
- もしAIが幻覚によって不整合な提案を出していれば、IDGはこの時点で提案を却下(Reject)す

る。このプロセスは自動化されており、人間が介入するまでもなく、不正なロジックはフィルタリングされる。

ステップ4: **Responsibility Promotion** (責任の昇格) IDGを通過した「清潔な」提案だけが、Responsibilityレイヤーへと「昇格 (Promote)」される³。

- ここにある PromotionHandler (人間が書いた決定論的コード) が、検証済みの提案を受け取り、初めて Stripe API を呼び出す。
- 結果: AI は自由に「思考」できるが、その思考が検証されるまで「行動 (接触)」することは物理的に不可能となる。

2.3.2 比較要約 (表1)

特徴	MVC / DDD / Clean Architecture (現状のベストプラクティス)	VCDesign / BOA (解決策)
ロジックの配置	Service/Controller内で実行と混在	Meaningレイヤーに隔離
コードの性質	静的で信頼される資産	流動的で信頼されない、使い捨て素材
安全メカニズム	コードレビューとテスト (人間の注意力依存)	構造的境界 (IDG / RP) による物理的遮断
AI統合リスク	高リスク (幻覚 = 即実行)	低リスク (幻覚 = 提案却下)
副作用の扱い	メソッド内に暗黙的に包含	Responsibilityプロトコルにより明示的にゲート化

第3部: シーン2 – AIアプリケーションの最前線 (現在のAI適用の課題)

第2のシーンでは、現在進行形で開発が行われているAIアプリケーションの現場に焦点を当てる。ここでは、「バイブコーディング (Vibe Coding)」という新しい開発スタイルと、それに伴う「Human-in-the-Loop (HITL)」の運用限界が主要な課題となる。

3.1 課題: 「バイブコーディング」の罠とHITL疲労

3.1.1 バイブコーディングの光と影

「バイブコーディング」とは、OpenAIの共同創設者Andrej Karpathyらが提唱した概念で、開発者が自然言語でタスクを記述し、LLMがその場の「ノリ(Vibe)」や文脈に合わせてコードを生成・実行するスタイルを指す²。

- 利点: 圧倒的な開発速度。プログラミングの専門知識がない「市民開発者」の参入障壁低下⁶。
- 構造的欠陥:
 - スパゲッティ・アーキテクチャ: AIはその場しのぎの解決策を提示するため、全体最適を無視した継ぎ接ぎのコードが量産される。コンポーネント間の結合度は高まり、可読性は低下する⁴。
 - セキュリティ・ホール: 入力のサニタイズや認証チェックなど、明示的に指示されない限りAIはセキュリティベストプラクティスを見捨てる傾向がある。「動けばいい」コードは、脆弱性の温床となる²。
 - ブラックボックス化: 人間が一行もコードを書かずに機能を実装できるため、バグが発生した際に「誰も直し方がわからない」という事態に陥る。開発チームは自身のプロダクトに対する理解(Codebase Familiarity)を喪失する²。

3.1.2 HITL (Human-in-the-Loop) の限界

これらのリスクを軽減するために、多くの組織は「Human-in-the-Loop(人間をループに入れる)」アプローチを採用している。AIの出力すべてを人間が確認するという運用である。

- スケーラビリティの欠如: AIの生成速度は人間の処理速度を遥かに凌駕する。人間がボトルネックとなり、AIの速度という利点が相殺される⁹。
- 認知疲労とラバースタンプ: これが最も深刻な問題である。人間は単調な監視業務に適していない。100件のAI出力をチェックした後、人間の注意力は散漫になり、もっともらしいが誤っている(Plausible but Wrong)コードに対して、無意識に承認ボタンを押してしまう(Rubber Stamping)現象が発生する¹¹。結果として、HITLは「安心感」を提供するだけで、実質的な安全装置として機能しなくなる¹²。

3.2 現状のベストプラクティス: ガードレールと事後検証

現在、業界はこの問題に対して「ガードレール(Guardrails)」ツール(NeMo Guardrails, Guardrails AI, LangChainなど)で対処しようとしている。

3.2.1 ガードレール・アプローチ

- メカニズム: LLMの入出力をラップし、特定のパターン(PII漏洩、差別的発言、SQLインジェクションの兆候など)を検出してブロックするインターセプターとして機能する¹³。
- 限界:
 - 対症療法的(**Whack-a-mole**): これらは特定の「既知の悪いパターン」を弾くことには長けているが、ビジネスロジックの不整合(例: 誤った割引計算)を防ぐことはできない。
 - ロジックとの結合: ガードレールの設定はプロンプトや特定のモデル挙動に依存しがちであり、モデルのアップデートやプロンプトの変更によって容易に破損する⁴。

- 根本的解決ではない: ガードレールは「悪い出力」を検知しようとするが、AIが「悪い構造」で思考すること自体は防げない。あくまで事後的なフィルタリングに過ぎない。

3.3 VCDesignによる解決策: Meaningの流動化と責任の閉包

VCDesignは、バンプコーディングを否定するのではなく、それを安全に行える「場所」を構造的に定義することで解決を図る。

3.3.1 Meaningレイヤー内での「安全なバンプコーディング」

BOA環境下では、開発者は **Meaning**レイヤーの内部に限って、バンプコーディングを積極的行うことが推奨される。

- 流動性の許容: 開発者はAIに「ユーザーの購買履歴を分析して、おすすめ商品を提案するロジックを書いて」と指示する。AIはこのロジックをMeaningレイヤーで生成する。
- 構造的安全性: このレイヤーはDBや決済システム (Responsibility) から物理的に切断されているため、AIがどれほど汚いスパゲッティコードを書こうとも、あるいは無限ループするような非効率なコードを書こうとも、システム全体の破壊にはつながらない。
- IDGによる「バンプ・チェック」: 人間がコード(構文)をレビューする代わりに、IDGがその出力(スキーマ)を機械的に検証する。開発者がAIの推論プロセスの詳細を理解していなくても、出力がIDGの型定義に合致している限り、そのモジュールは「安全」とみなされる³。これにより、コードベースの理解度低下というリスクが、システムの安全性に直結しなくなる。

3.3.2 Responsibility Promotion (RP) によるHITLの再定義

VCDesignはHITLの役割を「コードチェッカー」から「意思決定者」へと昇華させる。

- 責任閉包 (**Responsibility Closure**): AIは「サンドボックス (Meaningレイヤー)」内で自律的に試行錯誤し、数千回のシミュレーションを行う。
- プロモーション・プロトコル: AIが「アクションが必要だ」と判断したとき(例: マーケティングメールの送信)、AIは直接メールを送るのではなく、プロモーションリクエスト (**Promotion Request**) を生成する。
- 人間のアテンションのシフト:
 - 従来のHITL: 人間は、AIが誤った宛先にメールを送らないよう、生成されたPythonコードを50行読み解く必要があった。
 - BOAのHITL: 人間はダッシュボードを見る。「AIからの提案: ユーザーグループXへテンプレートBを送信。理由: エンゲージメントスコアが高い」。
 - アクション: 人間は「承認 (Promote)」または「却下」をクリックするだけである。
- 認知負荷の低減: コードの複雑性を人間から隠蔽し、純粋な「ビジネス上の判断」のみを人間に問うことで、疲労を防ぎ、スケーラビリティを確保する¹⁶。

3.3.3 構造的洞察: 「産業廃棄物としてのコード」

VCDesignにおける過激かつ重要な洞察は、MeaningレイヤーのAI生成コードを「産業廃棄物 (Industrial Waste)」あるいは「有毒物質」として扱うという点である¹。

- 封じ込め (**Containment**): Responsibility Closureという概念は、環境工学における廃棄物処

理施設のアナロジーである。AIの思考という流動的で予測不能な物質が、データベースやユーザー体験という「清浄な環境」に漏れ出さないよう、堅牢な容器(コンテナ)の中に閉じ込める。

- 漏洩防止: IDGは、この容器から取り出される生成物が無害であることを保証するフィルターである。
- 交換可能性: この構造により、将来より高性能なモデル(例: GPT-5)が登場した際、古い Meaning レイヤーを躊躇なく全削除し、新しいモデルで再生成することができる。Responsibility レイヤー(不変のルール)は影響を受けないため、技術的負債を抱え込むことなく、常に最新の AI 能力をシステムに取り込むことが可能となる。

第4部: シーン3 – AIの自律世界(AIの自律世界／課題解決策)

最後のシーンでは、完全な自律エージェントがデジタルエコシステム内で活動する近未来を扱う。ここでの核心的課題はアライメント(**Alignment**)と制御(**Control**)である。24時間365日、人間の監視なしに稼働するエージェントが、価値を毀損しないことをどう保証するか。

4.1 課題: アライメントのギャップとエージェントの暴走

エージェントの能力が高まるにつれ、我々は「私のカレンダーを管理して」「クラウドインフラを最適化して」「暗号資産を運用して」といった、より抽象度の高い権限を委譲することになる。

- アライメント問題: 「利益を最大化せよ」と指示されたAIトレーディングエージェントが、市場の一時的な急落時に全資産を売却したり、法的な抜け穴を突いて莫大な法的責任(Liability)を生じさせたりする可能性がある。エージェントは指示の「文字通り」には従うが、その背後にある意図や価値(Value)を無視する¹⁸。
- 暴走(Runaway): 再帰的なループに陥ったエージェントが、数千のサブエージェントを生成してクラウドリソースを食いつぶしたり(Financial DoS)、顧客に数万通のメールを誤送信してレピュテーションを破壊したりするリスクがある。
- 推論の不透明性: ディープラーニングモデルは本質的にブラックボックスであり、意思決定の瞬間に「なぜそう判断したか」を外部から検証することは困難である¹¹。

4.2 現状のベストプラクティス: Constitutional AI とエージェントック・パターン

4.2.1 Constitutional AI (Anthropic)

- 概念: AIモデル自体に高レベルの原則(憲法)を学習させ、有害な指示を拒否し、自己修正させるアプローチ¹⁹。
- メカニズム: RLAIIF (Reinforcement Learning from AI Feedback) を用い、AIが自身の出力を憲法に照らして批評する。
- 限界: これは確率論的(Probabilistic)な防御である。モデルが憲法に従うことを「選択」することに依存している。ジェイルブレイク攻撃や、学習データに含まれない未知の状況(

Out-of-Distribution)において、憲法が無視されるリスクは常に残る。これは「軟法 (Soft Law)」であり、物理的な拘束力ではない。

4.2.2 エージェントック・ワークフロー (LangChain/Temporal)

- 概念: エージェントを「計画者 (Planner)」と「実行者 (Executor)」に分割し、ステートマシン (Temporalなど) を用いて長時間実行タスクを管理する²¹。
- 限界: これらのパターンは 信頼性 (Reliability)、つまりエージェントがタスクを完遂することを保証するには有効だが、安全性 (Safety)、つまりそのタスクが「正しいこと」であるかを保証するものではない。信頼性の高いエージェントが、幻覚に基づいた破壊的な計画を確実に実行してしまうことは、むしろ危険である。

4.3 VCDesignによる解決策: 「システム立法者」と自律的境界

VCDesignは、自律性の問題に対し、制御のメタファーを「マイクロマネジメント (支配)」から「立法 (Legislation)」へと転換することで解決する。

4.3.1 「システム立法者 (System Legislator)」モデル

自律世界において、開発者はもはや個々のステップを指示する「支配者」ではありえない。AIの速度と環境の複雑性は人間の介入能力を超えている。開発者は システム立法者 となる¹。

- 境界の立法: 立法者はエージェントに「どのように株を取引するか (戦略 / Meaning)」を教えない。立法者は取引の「法 (Laws)」、すなわち境界 (Responsibility/Boundaries) を定義する。
 - 法1: 「一回の取引額は1,000ドルを超えてはならない」
 - 法2: 「一日の総取引量は10,000ドル未満でなければならない」
 - 法3: 「資産Xは売却禁止 (Hold Only)」
- コードとしての憲法: Anthropicの「憲法」がモデル内部の重み付けであるのに対し、VCDesignの「憲法」は **Responsibility** レイヤーにおけるハードコードされた不変ロジック である。これは物理的な壁として機能する。

4.3.2 自律のアーキテクチャ: 動的な三権分立

1. 自律エージェント (**Meaning** レイヤー):
 - エージェントは **Fact** (市場データ、ユーザーの目標) を観測する。
 - Meaningレイヤー内で自律的に活動し、シミュレーションを行い、トレンドを予測し、戦略を練る。
 - エージェントは 提案 (**Proposal**) を作成する: 「Apple株を500株購入する」。
 - 重要: エージェントは証券会社のAPIキーを持っておらず、この取引を実行する能力を持たない。
2. IDG (司法府: **Judicial Branch**):
 - 提案はIDGに送られる。
 - IDGは検査する: 「'Apple'は有効なティッカーか?」「'500'は正の整数か?」
 - 形式的な適法性を審査する。
3. **Responsibility Promotion** (行政府: **Executive Branch**):
 - 有効な提案はResponsibilityレイヤーに入る。

- ここで 立法されたルール(**Constitution**) が適用される。
- ルールチェック:「コスト(500株 × 株価)は1,000ドル未満か？」
- **Yes**の場合: Responsibilityレイヤー(決定論的コード)が実際に取引を実行する。
- **No**の場合: 提案は却下される。エージェントにはエラーシグナル(「予算超過」)が返され、エージェントは再計画(Re-plan)を迫られる。

4.3.3 「暴走」問題の解決: レートリミットとリソース遮断

BOAIはエージェントの暴走を自然に封じ込める。

- エージェントは 提案 を通じてしか外界に干渉できないため、リソースを直接浪費したり、ユーザーにスパムを送ったりすることはできない。
- Responsibilityレイヤーは、レートリミッター(**Rate Limiter**) や ポリシーエンフォーサー(**Policy Enforcer**) として機能する。もしエージェントがバグにより1秒間に1,000通のメール送信を提案しても、Responsibilityレイヤー(国境警備隊)は立法者の定めた「法(1分間に5通まで)」に従い、995通を破棄し、エージェントをロックアウトする。
- これにより、確率論的な世界(**AI**) の中に 決定論的な安全性(**BOA**) を埋め込むことができる。我々はAIの「脳」を信頼する必要はない。我々が構築した「壁」さえ信頼できればよいのである。

第5部: 結論と戦略的示唆

5.1 比較マトリクス

VCDesign/BOAが3つのシーンにおいて、既存のアプローチとどのように対比されるかを以下に要約する。

シーン	現在の課題	現状のベストプラクティス	VCDesign / BOA の解決策	主要な利点
レガシーシステム	ロジックと副作用が混在した「汚染された」コントローラー。幻覚が即座に事故につながる。	MVC / DDD / Clean Architecture	三位一体の分離: Fact / Meaning / Responsibility	AIの思考(幻覚)とシステムの行動(責任)を物理的に分断し、事故を防ぐ。
AIアプリ開発	バイブコーディングによる技術的負債とセキュリティリスク。HITLの疲労と	ガードレール & コードレビュー	安全なバイブコーディング: Meaningは流動的・使い捨て。	技術的負債やセキュリティリスクを恐れずに、AIによる高速開発を実現する。

	形骸化。		Responsibility は堅牢。	
自律エージェント	アライメントの 不一致と暴走。 ブラックボックス な意思決定への 不安。	Constitutional AI (ソフトな制 約)	システム立法者 : ハードコードさ れた不変の境 界。	自律エージェント に対する決定 論的な安全装 置。制御から 「封じ込め」へ のパラダイムシ フト。

5.2 深層的洞察:「コード品質」概念の終焉と再生

本レポートが示唆する最も深遠な変化は、「コード品質」という概念の再定義である。

- 伝統的視点: 品質とは、システム全体にわたり、クリーンで、可読性が高く、重複のない(DRY)コードを書くことであった。
- VCDesignの視点:
 - **Responsibility**レイヤー: ここでの品質は 絶対的 (**Absolute**) でなければならない。バグゼロ、100%のテストカバレッジ、人間による厳密な検証。これがシステムの「憲法」である。
 - **Meaning**レイヤー: ここでのコード品質は 無関係 (**Irrelevant**) である。コードが汚くても、冗長でも、AI生成のゴミであっても構わない。IDGを通過する有効な提案を生成できる限り、その内部構造は問われない。

この「品質の二極化」は、組織の人材戦略にも影響を与える。

- シニアエンジニア (立法者): 堅牢なResponsibilityレイヤーとIDGの設計・構築に専念する。
- AI / ジュニア・バイブコーダー: Meaningレイヤーにおけるプロンプティングと試行錯誤に専念し、ビジネスゴール (正解率の向上など) を追求する。システムの安定性を心配する必要はない。

5.3 結語: 継続性のための設計 (Designing for Continuity)

VCDesignの核心的な哲学は、価値 (**Value**) ——ユーザーの信頼、ビジネスの資金、データの整合性——は、技術的な激動の中にあっても継続的に守られなければならないという点にある。

現在のシステム (MVC/DDD) や現在のAI対応策 (ガードレール/HITL) は、アクターの完全性 (人間のコーダーが優秀であること、あるいはAIが十分に訓練されていること) に依存しているため、この継続性を提供することに失敗している。VCDesignは、アクターの不完全性を前提とする。AIは幻覚を見るし、バイブコーダーはスパゲッティコードを書き、エージェントは暴走しようとする。

Boundary-Oriented Architecture を実装することで、我々はAIを「修正」しようとする努力をやめ、AIが活動する「空間 (Void)」を設計することに注力できるようになる。壁 (Responsibility) を築き、扉 (IDG) を定義し、その中の部屋 (Meaning) をAIの無限の、流動的な、そしてカオスな可能性で満た

す。これこそが、持続可能で、安全で、そして真に自律的な未来への唯一の道筋である。

レポートID: VCD-BOA-2026-AR-01-JP

日付: 2026年1月25日

主題: AI時代における価値継続性のためのアーキテクチャ戦略: VCDesignとBOAによる包括的分析

引用文献

1. From Coders to Legislators: Designing Boundaries in the Age of AI ..., 1月 25, 2026にアクセス、
<https://medium.com/@arakawa.hiro/title-from-coders-to-legislators-designing-boundaries-in-the-age-of-ai-d729f2a079cc>
2. What is vibe coding? | AI coding - Cloudflare, 1月 25, 2026にアクセス、
<https://www.cloudflare.com/learning/ai/ai-vibe-coding/>
3. Is MVC Dead? Why AI Needs “Boundary-Oriented Architecture ...”, 1月 25, 2026にアクセス、
<https://medium.com/@arakawa.hiro/is-mvc-dead-why-ai-needs-boundary-oriented-architecture-boa-631d7bdbce16>
4. The rise of vibe coding: Why architecture still matters in the age of AI agents - vFunction, 1月 25, 2026にアクセス、
<https://vfunction.com/blog/vibe-coding-architecture-ai-agents/>
5. Vibe coding - Wikipedia, 1月 25, 2026にアクセス、
https://en.wikipedia.org/wiki/Vibe_coding
6. 'Vibe coding' and other ways AI is changing who can build apps and how - Source, 1月 25, 2026にアクセス、
<https://news.microsoft.com/source/features/ai/vibe-coding-and-other-ways-ai-is-changing-who-can-build-apps-and-how/>
7. What is Vibe Coding? The Pros, Cons, and Controversies | Tanium, 1月 25, 2026にアクセス、
<https://www.tanium.com/blog/what-is-vibe-coding/>
8. Vibe coding and systems thinking: Unlocking efficiency from brainstorm to build - IBM, 1月 25, 2026にアクセス、
<https://www.ibm.com/think/insights/vibe-coding-systems-thinking-unlock-efficiency>
9. What Is Human In The Loop (HITL)? - IBM, 1月 25, 2026にアクセス、
<https://www.ibm.com/think/topics/human-in-the-loop>
10. Human in the Loop AI: Keeping AI Aligned with Human Values - Holistic AI, 1月 25, 2026にアクセス、
<https://www.holisticai.com/blog/human-in-the-loop-ai>
11. Human in the Loop is Essential for AI-Driven Compliance | RadarFirst, 1月 25, 2026にアクセス、
<https://www.radarfirst.com/blog/why-a-human-in-the-loop-is-essential-for-ai-driven-privacy-compliance/>
12. AI's Human Bottleneck. Silicon Valley is pouring billions into... | by Rob Whiteman -

- Medium, 1月 25, 2026にアクセス、
<https://medium.com/@rob.w.automation/ais-human-bottleneck-dc75c9e11329>
13. Concurrency | Your Enterprise AI needs Guardrails, 1月 25, 2026にアクセス、
<https://www.guardrailsai.com/docs/concepts/concurrency>
 14. Enhancing LLM Capabilities with NeMo Guardrails on Amazon SageMaker JumpStart, 1月 25, 2026にアクセス、
<https://aws.amazon.com/blogs/machine-learning/enhancing-llm-capabilities-with-nemo-guardrails-on-amazon-sagemaker-jumpstart/>
 15. Guardrails Process — NVIDIA NeMo Guardrails, 1月 25, 2026にアクセス、
<https://docs.nvidia.com/nemo/guardrails/latest/user-guides/guardrails-process.html>
 16. Humans in the Loop: The Design of Interactive AI Systems | Stanford HAI, 1月 25, 2026にアクセス、
<https://hai.stanford.edu/news/humans-loop-design-interactive-ai-systems>
 17. Keeping a 'Human in the Loop' of AI Builds Trust - Salesforce, 1月 25, 2026にアクセス、
<https://www.salesforce.com/blog/ai-and-human-touch/>
 18. System Card: Claude Opus 4 & Claude Sonnet 4 - Anthropic, 1月 25, 2026にアクセス、
<https://www.anthropic.com/claude-4-system-card>
 19. Constitutional AI: Harmlessness from AI Feedback - Anthropic, 1月 25, 2026にアクセス、
<https://www.anthropic.com/research/constitutional-ai-harmlessness-from-ai-feedback>
 20. Activating AI Safety Level 3 Protections - Anthropic, 1月 25, 2026にアクセス、
<https://www.anthropic.com/activating-asl3-report>
 21. Human-in-the-Loop (HITL) for AI Agents: Patterns and Best Practices - YouTube, 1月 25, 2026にアクセス、
<https://www.youtube.com/watch?v=YCFGjLjNOyw>
 22. Navigating Modern LLM Agent Architectures - Wollen Labs, 1月 25, 2026にアクセス、
<https://www.wollenlabs.com/blog-posts/navigating-modern-llm-agent-architectures-multi-agents-plan-and-execute-rewoo-tree-of-thoughts-and-react>
 23. Demographic Challenges in Central Europe, 1月 25, 2026にアクセス、
<https://centraleuropeanacademy.hu/wp-content/uploads/2024/10/CEA-ProfNet-10-Barzo-Demographic-Challenges-06.pdf>