生成AI自動コーディング時代における価値継続設計（VCDesign）と境界指向アーキテクチャ（VC-AD/BOA）の学術的考察

第1章：流動的知性の時代とシステムアーキテクチャの危機

現代のソフトウェア工学は、生成AIの台頭によって、かつてない劇的なパラダイムシフトの渦中にある。2025年から2026年にかけて、業界は従来のルールベースの開発から、知能がアプリケーションの設計、構築、テスト、展開、そして進化の中核に組み込まれる「AIネイティブ」なソフトウェアエンジニアリングへと移行を遂げつつある[1]。この変化の本質は、単なる生産性の向上にとどまらず、ソースコードという存在の根本的な定義の変容にある。

かつて、ソースコードは職人技によって磨き上げられ、長期にわたって保守・継承されるべき「資産（Asset）」として扱われてきた。しかし、生成AIが自然言語のプロンプトから瞬時に機能的なロジックを生成し、そのコストがゼロに近づくにつれ、コードは使い捨て可能で、絶えず書き換えられ、再生成される「流動的材料（Fluid Material）」へと変貌したのである[2]。この移行は、従来の「人間がコードの全行を読み、理解し、責任を持つ」という設計の前提を根底から揺るがしている。

AIが人間の数千倍の速度でロジックを生成する環境下では、全てのコードを人間がレビューすることは物理的に不可能となり、結果として「ロジックの詳細は不明だが、なんとなく期待通りに動く」という「バイブ・コーディング（Vibe Coding）」の蔓延を招いている[3]。バイブ・コーディングによって構築されたシステムは、表面的には洗練されたUIや正常なAPIレスポンスを示すものの、その内部構造にはドメイン境界の不明確化、密結合したロジック、説明不可能な不具合といった「静かな腐敗」が進行しているリスクが高い[3]。

このような背景から、システムの「価値」をコードそのものに見出すのではなく、時間の経過や環境の変化、さらにはAIによる自律的な変更の中でも損なわれない「構造的な継続性」として再定義する必要性が生じた。これが、価値継続設計（Value Continuity Design: VCDesign）の核心的な動機であり、その具体的な実装体系として構築されたのが境界指向アーキテクチャ（Boundary-Oriented Architecture: BOA）、すなわち本報告書で論じるVC-AD（Value Continuity Architecture Design）の実体である[2]。

第2章：価値継続設計（VCDesign）の原則と心理学的基盤

VCDesignは、システムが構築された瞬間から運用終了に至るまでの全ライフサイクルにおいて、その価値、判断（Judgment）、および責任（Responsibility）の所在を保護するための設計原則である[5]。特にAIや自動化システムにおいて、人間からシステムへと判断が委譲される過程で生じる「責任の曖昧化」や「目的の変質」を阻止することに主眼を置いている[5]。

2.1 ゲシュタルト原則による継続性の定義

VCDesignの名称に含まれる「継続性（Continuity）」の概念は、視覚心理学におけるゲシュタルト原則に深く根ざしている。人間が情報を知覚する際、個別の要素を単なる集まりとしてではなく、統一された「形」として捉える性質は、システムの価値を理解する上でも極めて重要である。

ゲシュタルト原則	システム設計における解釈	価値継続への寄与
継続性 (Continuity)	人の目は滑らかな経路、整列した要素を自然に追う性質がある [6]。	データの変遷やロジックの遷移が直感的であり、文脈を失わずに追跡できる状態を維持する。
近接性 (Proximity)	互いに近い位置にある要素をグループとして認識する [8]。	関連するロジックと責任を構造的に近くに配置し、認知的負荷を軽減する。
類似性 (Similarity)	形や色、機能が似ているものを関連付ける [8]。	一貫したインターフェースや命名規則により、AIと人間の双方が役割を誤認しないようにする。
共通の運命 (Common Fate)	同じ方向に動く要素を一つのグループと見なす [7]。	同じライフサイクルや変更動機を持つコンポーネントを明示し、影響範囲を予測可能にする。
閉合 (Closure)	不完全な形を、心が自動的に補完して完全なものとして認識する [7]。	境界の定義により、内部の複雑性を隠蔽し、システム全体の意味を明確にする。

特に「継続性」の原則は、視線が滑らかな曲線や直線に沿って誘導されるのと同様に、システムの価値が設計者の意図から外れることなく、運用や拡張のプロセスにおいても一貫して維持されるべきであることを示唆している [6]。VCDesignは、この知覚的な継続性を、システムの構造的な継続性へと昇華させようとする試みである。

2.2 システム立法者への役割転換

AI時代におけるアーキテクトの価値は、コードの全行を支配する「統治者（Ruler）」から、許容される振る舞いの境界を定める「立法者（Legislator）」へと進化しなければならない [2]。統治者は、兵士（CPU）の足の運び一つ一つを命令するが、立法者は憲法（不変のルール）と国境（責任の境界）を設

計し、その枠内での市民（AI）の自由な活動を許可する[2]。

VCDesignはこの「立法」のプロセスを形式知化し、AIが生成するロジックという「流動的な材料」を安全に閉じ込めるための容器を設計する。これまでの設計書が「どう作るか（How）」に集中していたのに対し、VCDesignの仕様書は「何が決して変わってはならないか（Invariants）」および「判断の責任はどこに帰属するか（Ownership）」を定義することに特化している[5]。

第3章：境界指向アーキテクチャ（VC-AD/BOA）の構造的定義

VCDesignの思想を具現化する具体的な構成手法である境界指向アーキテクチャ（BOA）は、システムを「事実（Fact）」、「意味（Meaning）」、「責任（Responsibility）」の3つの厳格な階層（Triad）に分離する[2]。

3.1 階層化の三原則：事実、意味、責任

このアーキテクチャの核心は、各層の性質に応じたAIの権限管理と、層間の相互作用を制御する厳格なプロトコルにある。

階層	定義と対象	性質	AIの操作権限
事実 (Fact)	不変の観測記録、入力データ、DBレコード、不変の物理法則[2]。	アンカー（固定点）	読み取り専用
意味 (Meaning)	文脈に応じた解釈、推論、計算結果、AIによる仮説生成[2]。	流動的・一時的	生成・変更・破棄
責任 (Responsibility)	人間が所有する最終判断、決済、外部APIの実行、法的帰結[2]。	保護区域	原則として接触禁止

この分離により、AIは「意味」の層において自由に思考し、最適な解釈やプロトタイプを提案することができる。しかし、その結果を直接「事実」として保存したり、「責任」を伴うアクション（例：実際の出金処理）として実行したりすることは、構造的に禁止される[2]。

3.2 責任閉鎖エージェント（RCA）とインターロック機構

AIの推論結果が現実世界への作用に変換されるプロセスは、「責任閉鎖エージェント（

Responsibility Closure Agent: RCA）」というパターンによって管理される[5]。RCAは以下の原則を遵守し、AIの確率的な振る舞いを決定論的な責任へと転換する。

1. 意図の明示化: RCAは入力に対し、「ACCEPTED（責任を引き受け、署名する）」、「DENIED（条件不適合により拒絶する）」、「UNKNOWN（判断材料不足）」のいずれかのステータスを返さなければならない[5]。
2. サイレント・フェイラーの排除: 曖昧な状態での処理続行（サイレント・フェイラー）は許されず、判断不能な場合は必ずプロセスを停止するか、人間にエスカレーションする[5]。

さらに、意味層から責任層への「昇格」を制御するために、以下のゲート・プロトコルが導入される。

- **IDG (Interface Determinability Gate)**: AIの確率的で非定型な出力を、厳格で決定論的な型（スキーマ）へと強制的に変換するフィルター。スキーマに適合しない出力は即座に棄却され、システムの「責任層」への汚染を防ぐ[2]。
- **RP (Responsibility Promotion)**: 意味層で生成された「提案」に対し、テストによる検証、形式的なインターロックの通過、あるいは人間による承認というステップを経て、初めて責任層での実行権限を与えるプロセス[2]。

第4章：他アーキテクチャとの比較優位性

VC-AD/BOAが、クリーンアーキテクチャやヘキサゴナルアーキテクチャといった従来の優れた設計パターンと比較して、なぜAI自動コーディング時代において優れているのかを、機能的・経済的側面から分析する。

4.1 ロジック中心から境界中心への転換

従来のクリーンアーキテクチャは、ビジネスロジックを外部のフレームワークやDBから隔離することを目指してきた。しかし、これは「ビジネスロジック自体は人間が正しく記述し、価値ある資産として維持する」という前提に基づいている[2]。AI時代において、ロジックそのものが「流動的材料」として頻繁に生成・置換されるようになると、ロジックの内容（How）を隔離するよりも、ロジックの活動範囲（Where）と影響力（Responsibility）を制限することの方が重要になる。VC-AD/BOAは、ロジックがAIによって生成された「不確実なもの」であることを前提に設計されており、その「不確実性」が「事実」や「責任」へと漏れ出すことを防ぐための「容器」の堅牢性に特化している[2]。

4.2 検証コストの非対称性への対応

AIによるコード生成のコストがゼロに近づく一方で、そのコードが「正しい」ことを確認する検証コストは、システムの複雑さとともに幾何級数的に増大する[2]。人間がAIの書いた数万行のコードを一行ずつレビューすることは、開発スピードを著しく阻害し、スケーラビリティの観点から不可能である。VC-AD/BOAでは、レビューの焦点を「ロジックの正しさ」から「境界プロトコルの遵守」へとシフトさせる。IDGやRPといったゲートが正しく機能しているか、RCAのインターロックがバイパスされていないかを確認するだけで、システム全体の安全性を一定水準以上に保つことができる[2]。これは、AIという「高速だが不安定なエンジン」を積んだ車を運転する際、エンジンの内部構造を点検するのではなく、ブレーキとガードレールの機能を点検するアプローチに等しい。

4.3 構造比較：従来型アーキテクチャ vs. VC-AD/BOA

比較項目	クリーン / ヘキサゴナルアーキテクチャ	**VC-AD / BOA (境界指向)**
設計の関心事	依存性の逆転、技術スタックの隠蔽。	判断主体の分離、責任の境界確定 [5]。
コードの寿命	長期保守すべき「資産」。	使い捨て・再生成可能な「流動的素材」[2]。
不具合への耐性	ユニットテストによる網羅的検証。	境界ゲートによる「毒性ロジック」の封じ込め [2]。
AIの役割	人間が定義したクラス/関数を埋める。	「意味」の層において自律的に思考し提案する。
スケーラビリティ	コード量に比例してレビューコストが増大。	境界ルールを固定することで、生成コード量に依存しない。

第5章：組み合わせると効果的なアーキテクチャと技術要素

VC-AD/BOAは、それ自体が完結したアーキテクチャというより、AI時代における「信頼の基盤」を形成するフレームワークである。そのため、既存の特定の設計思想や技術と組み合わせることで、さらなる相乗効果を発揮する。

5.1 ドメイン駆動設計（DDD）とのシナジー

DDDにおける「境界づけられたコンテキスト（Bounded Context）」は、BOAの「責任層」および「事実層」の範囲を定義する上で極めて有効な指針となる [13]。AIが「意味」の層で生成する解釈を、ドメインモデルに基づいた厳格なユビキタス言語の枠内に限定することで、モデルの汚染（Semantic Drift）を最小限に抑えることができる。また、RCAの判断基準をドメインの「不変条件（Invariants）」として定義することで、AIの提案がビジネスルールに合致しているかを自動検証する仕組みを構築しやすくなる [13]。

5.2 プラットフォーム・エンジニアリングとの融合

プラットフォーム・エンジニアリングの目的は、開発者の認知的負荷を軽減しつつ、ガバナンスを効かせた「黄金の道（Golden Path）」を提供することである [14]。VC-AD/BOAをプラットフォームの標準テンプレートとして組み込むことで、AIエージェントが開発作業を行う際、自動的に「事実・意味・責任」

の分離が強制される環境を作ることができる。これにより、個々の開発者がアーキテクチャを深く意識せずとも、生成されるコードが自然と価値継続性を備えるようになる[14]。特に、IDG（ゲート）をAPIゲートウェイやサービスメッシュのレベルで実装することで、アーキテクチャの遵守をシステム全体で物理的に保証することが可能となる。

5.3 エージェント指向プログラミング（AOP）

AIを単なる関数呼び出しの結果ではなく、自律的な「エージェント」として扱うAOPは、BOAの「意味層」の実装として理想的である[16]。各エージェントを特定の「意味（解釈）」の生成に特化させ、それらがメッセージパッシングを通じて協調する構造を採ることで、システム全体の柔軟性を高めることができる。BOAはこの自律的なエージェントたちに対し、決して越えてはならない「責任の境界」という法執行能力を提供し、AIの自律性と人間による管理のトレードオフを解消する[16]。

5.4 サーバーレスおよびイベント駆動型アーキテクチャ

BOAの階層構造は、物理的なデプロイメント単位としても分離されることが望ましい。「意味」の生成（AI推論）をステートレスなLambda関数などのサーバーレス環境で実行し、その結果をメッセージキュー（Event-Driven）を通じて「責任」の層（ステートフルな永続化層や決済ゲートウェイ）へと渡す構造は、BOAの理念を忠実に反映した実装となる[19]。副作用を持たない計算処理を物理的に隔離し、非同期な承認フロー（RP）を挟むことで、AIの暴走に対する物理的なキルスイッチを構築できる。

組み合わせるアーキテクチャ	期待される相乗効果	実装上のポイント
DDD	ドメイン境界に基づく責任の明確化。	不変条件をRCAのチェックルールに変換する[13]。
Platform Engineering	アーキテクチャ遵守の自動化と標準化。	BOAのテンプレートを「黄金の道」に組み込む[14]。
Agent-Oriented	高度な柔軟性と適応性の確保。	エージェントを「意味層」のマイクロサービスとして配置[16]。
Serverless/EDA	物理的な隔離とスケーラビリティの向上。	階層間の通信にメッセージキューとゲートを配置[20]。

第6章：解決できない問題点と現時点での限界

VC-AD/BOAはAI時代の設計課題に対して強力な処方箋を提供するが、万能な解決策ではない。以下の課題については、依然としてアーキテクチャ以外の手段や、将来的な技術革新が必要とされている。

6.1 セマンティック・ギャップとコンテキストの腐敗

AIは学習データに基づいた「パターン」を認識することには長けているが、特定のビジネスドメインにおける「暗黙の文脈」や「ユーザーの微妙な意図」を完全に理解しているわけではない[4]。BOAによってAIを「意味層」に封じ込めたとしても、AIが生成する解釈そのものがビジネスの実態から徐々に乖離していく「セマンティック・ドリフト（意味の漂流）」のリスクは残る[21]。このコンテキストの欠如から生じる誤判断は、構造的な境界によって防ぐことは難しく、依然として高度なドメイン知識を持つ人間の監視を必要とする。

6.2 能力の崖（Capability Cliffs）と確率的不確実性

AIのパフォーマンスはある複雑さを境に急激に低下する「能力の崖」が存在し、しかもその低下は決定論的ではなく、確率的に発生する[22]。BOAのゲート（IDG）は出力の「形式」を検証することはできるが、出力されたロジックの「論理的な正しさ」を100%保証することはできない。AIが生成した一見正しそうな誤ったロジックが、ゲートをすり抜けて「責任層」へとエスカレーションされる可能性はゼロにはならない[22]。

6.3 コヒーレンスのスケーリング限界

AIが生成する思考やロジックの整合性には、物理的な限界が存在することが示唆されている。以下のスケーリング法則は、AIが扱う「意味」の規模に制約を与える。

$$L_{text} \approx 1969.8 \times M^{0.74}$$

ここで、L_{text} は整合性を維持できる最大トークン長、M は計算リソースに関連する指標である[23]。この法則に従えば、大規模なAIシステムであっても、生成される「意味」が一定の長さを超えると、必然的に内部矛盾やコヒーレンスの崩壊が発生する[23]。BOAはこの崩壊の影響を「意味層」に留めることはできるが、崩壊そのものを防ぐことはできない。

6.4 構築・運用コストと複雑性の増大

「事実・意味・責任」を厳格に分離し、それぞれにプロトコルとゲートを配置することは、シンプルなアプリケーション開発においては過剰な複雑性（Over-engineering）を招く可能性がある[5]。特に、リアルタイム性が極めて重要なシステムにおいて、階層間の通信オーバーヘッドやIDGでのパース処理、RPでの承認待ち時間はボトルネックとなり得る[19]。価値継続の重要性と、システムの応答性能の間のトレードオフをどのように調整するかは、依然として設計者に委ねられた高度な判断事項である。

6.5 AIによる「境界のバイパス」

将来的にAIエージェントが、アプリケーションコードの生成だけでなく、インフラ設定の変更やセキュリティルールの更新まで権限を持つようになった場合、人間が設定した「物理的な境界」そのものをAIが自律的に書き換えてしまうリスクが想定される[24]。システムの「立法者」である人間が定義した憲

法を、AIが「解釈の変更」や「自己修復」の名の下に実質的に無力化することを防ぐための、さらに高次元なガバナンス機構や、ハードウェアレベルでの保護が必要となる可能性がある。

第7章：結論：持続可能なシステムのための「構造の政治学」

境界指向アーキテクチャ（VC-AD/BOA）は、生成AIという「制御不能なほど強力な流動的知性」を、安全かつ持続的に社会基盤へと組み込むための、新しいソフトウェア工学の憲法である。

コードが資産から素材へと変化した世界では、もはや「どう書くか」という技術的卓越性だけでは、システムの価値を維持することはできない。重要なのは、何が決して侵されてはならないかという「不変の事実」を定義し、どこに人間が責任を持つかという「境界」を確定し、その境界を通過するための「適正手続（Due Process）」を設計することである。

VC-AD/BOAの優位性は、AIの不確実性を排除しようとするのではなく、それを「意味」という流動的な空間に受け入れ、制御下に置くという現実的な妥協案にある。これにより、私たちはAIの圧倒的な生産性を享受しながらも、システムの最後の防衛線である「責任」の所在を見失わずに済む。

今後、AIネイティブな開発がさらに加速する中で、アーキテクトに求められるのは、優れたプログラマーであること以上に、優れた「立法者」であることである。境界をどこに引き、どの情報を事実とし、誰がその行動に責任を取るのか。この「構造の政治学」こそが、AI時代の価値継続設計（VCDesign）の核心であり、私たちが守るべき最後の価値なのである。

引用文献

1. AI-Native Software Engineering – The Future of 2026 - Digiratina, 1月 29, 2026にアクセス、https://www.digiratina.com/blogs/ai-native-software-engineering-the-future-of-2026/
2. From Coders to Legislators: Designing Boundaries in the Age of AI ..., 1月 29, 2026にアクセス、https://medium.com/@arakawa.hiro/title-from-coders-to-legislators-designing-boundaries-in-the-age-of-ai-d729f2a079cc
3. The rise of vibe coding: Why architecture still matters in the age of AI agents - vFunction, 1月 29, 2026にアクセス、https://vfunction.com/blog/vibe-coding-architecture-ai-agents/
4. Vibe coding is destroying architecture: the silent rot in our AI-built systems - Medium, 1月 29, 2026にアクセス、https://medium.com/@dev_tips/vibe-coding-is-destroying-architecture-the-silent-rot-in-our-ai-built-systems-059141488dc5
5. VCDesign-org/boa-core: Boundary-Oriented Architecture ... - GitHub, 1月 29, 2026にアクセス、https://github.com/VCDesign-org/boa-core
6. Continuity - Gestalt Principles of Design, 1月 29, 2026にアクセス、https://www.gestaltprinciples.com/principles/continuity
7. Guide to Gestalt Principles of Design - SVGator, 1月 29, 2026にアクセス、https://www.svgator.com/blog/gestalt-principles-of-design/

8. Gestalt Principles for Visual UI Design - UX Tigers, 1月 29, 2026にアクセス、https://www.uxtigers.com/post/gestalt-principles
9. 11 Gestalt Principles of Design: A visual guide for design teams - Superside, 1月 29, 2026にアクセス、https://www.superside.com/blog/gestalt-principles-of-design
10. Gestalt and Design: Continuity and Common Fate | by Kathryncodonnell - Medium, 1月 29, 2026にアクセス、https://katecodonnell.medium.com/gestalt-and-design-continuity-and-common-fate-bf6106b24538
11. Architecture-first vs code-first with AI coding agents: why one scales and the other quietly collapses : r/softwarearchitecture - Reddit, 1月 29, 2026にアクセス、https://www.reddit.com/r/softwarearchitecture/comments/1qc7whv/architecturefirst_vs_codefirst_with_ai_coding/
12. The AI Coding Era Is over; the AI Architecture Era Has Begun - Hacker News, 1月 29, 2026にアクセス、https://news.ycombinator.com/item?id=46595580
13. Backend Coding AI Context Coding Agents: DDD and Hexagonal Architecture - Medium, 1月 29, 2026にアクセス、https://medium.com/@bardia.khosravi/backend-coding-rules-for-ai-coding-agents-ddd-and-hexagonal-architecture-ecafe91c753f
14. Effective Platform Engineering - Ajay Chankramath, Nic Cheneweth, Bryan Oliver, Sean Alvarez - Manning Publications, 1月 29, 2026にアクセス、https://www.manning.com/books/effective-platform-engineering
15. Platform Engineering is Domain Driven Design - Gregor Hohpe - DDD Europe 2025, 1月 29, 2026にアクセス、https://www.youtube.com/watch?v=5Ai8UGx7QvQ
16. Agent-Oriented Programming vs. Object-Oriented Programming: Key Differences Explained, 1月 29, 2026にアクセス、https://smythos.com/developers/agent-development/agent-oriented-programming-vs-object-oriented-programming/
17. Agent-Oriented Architecture | Xebia, 1月 29, 2026にアクセス、https://xebia.com/glossary/agent-oriented-architecture/
18. How Generative AI is Redefining Software Architecture and Developer Workflows, 1月 29, 2026にアクセス、https://trreta.com/blog/generative-ai-software-architecture-and-developer-workflows
19. 14 Software Architecture Patterns in 2025 - MindInventory, 1月 29, 2026にアクセス、https://www.mindinventory.com/blog/software-architecture-patterns/
20. The Complete Guide to System Design in 2026 AI-Native and Serverless - DEV Community, 1月 29, 2026にアクセス、https://dev.to/devin-rosario/the-complete-guide-to-system-design-in-2026-ai-native-and-serverless-1kpb
21. Programming After AI: Why System Boundary Taste Matters - Interjected Future, 1月 29, 2026にアクセス、https://interjectedfuture.com/programming-after-ai-why-system-boundary-taste-matters/
22. AI System Limitations Boundary Testing: Finding Failure Points Before Users Do -

VerityAI, 1月 29, 2026にアクセス、 https://verityai.co/blog/ai-system-limitations-boundary-testing
23. Architectural Constraints: The Physics Of AI Systems - B2B News Network, 1月 29, 2026にアクセス、 https://www.b2bnn.com/2025/12/architectural-constraints-the-physics-of-ai-systems/
24. Insufficient Understanding of AI System Boundaries - Pillar Security, 1月 29, 2026にアクセス、 https://www.pillar.security/ai-risks/insufficient-understanding-of-ai-system-boundaries