VCDesign（Value Continuity Design）と境界指向アーキテクチャ（BOA）：生成AI自動開発時代における価値と責任の構造化に関する包括的リサーチレポート

1. 序論：Vibe Coding時代の到来とソフトウェア工学の転換点

2025年、Andrej Karpathyによって提唱された「Vibe Coding」という概念は、ソフトウェア開発の現場に不可逆的なパラダイムシフトをもたらしました。自然言語による高レベルな指示（プロンプト）のみで、AIが機能的なコードを生成し、人間が実装の詳細を管理しないこのスタイルは、開発の民主化と爆発的な生産性向上を約束する一方で、従来のソフトウェアアーキテクチャが前提としてきた「秩序」を根底から揺るがしています[1]。

かつて、コードは「資産」であり、その品質、保守性、可読性が長期的な価値の源泉でした。しかし、生成AIの限界費用がゼロに近づくにつれ、コード自体は使い捨て可能な「流動的なマテリアル」へと変貌しつつあります。この環境下において、システムが崩壊せずに長期的な価値を提供し続けるためには、何が必要なのでしょうか。その答えとして浮上しているのが、「VCDesign（Value Continuity Design）」という思想体系と、それを具現化する「BOA（Boundary-Oriented Architecture）」です。

本レポートは、VCDesignの中核をなす構成要素であるvcdesign-core、boa-core、responsibility-closure、value-continuity-protocolsを徹底的に分析し、その理論的背景、既存アーキテクチャとの比較、そしてAI時代における定量的評価手法について、15,000語規模の深度で論じるものです。

---

2. VCDesignの上流思想：憲法（Constitution）と時間軸の設計

2.1 価値連続性（Value Continuity）の哲学

VCDesign、すなわち「価値連続性設計」は、ソフトウェアを静的な構造物としてではなく、時間の経過と共に変化し続ける「物語」として捉えます。Vibe Codingが一般化すると、ビジネスロジックの実装（How）はAIによって瞬時に、かつ頻繁に書き換えられるようになります。この流動性の中で、「何を作るか」という問い以上に、「何を変え続け、何を守り抜くか」とい

う問いが重要になります[3]。

ここで導入されるのが「憲法（Constitution）」という概念です。これは、Anthropicが提唱する「Constitutional AI」がAIモデルの出力に対して倫理的なガイドラインを設けるのと同様に、ソフトウェアシステムの「振る舞い」と「責任」に対して、不変の原則を定義するものです[4]。

2.1.1 憲法による保護領域とVibe領域の分離

VCDesignにおける憲法は、システムを以下の二つの領域に厳格に区分します。

1. **Vibe領域（流動領域）**: AIが自由にコードを生成し、リファクタリングし、最適化することが許される領域。ここは「Meaning（解釈）」の生成が行われる場所であり、実装の詳細は問われません。
2. **憲法領域（保護領域）**: ビジネスの核となる価値、データの整合性、法的責任、そしてユーザーへの約束が担保される領域。ここは「Fact（事実）」と「Responsibility（責任）」が支配し、AIによる恣意的な改変が禁止されます。

この区分により、開発者はAIの創造性を最大限に活用しつつ、システムが制御不能なカオスに陥るリスク（Change Propagationの暴走）を抑制することができます[6]。

2.2 時間軸の構造化：Chapters（章）という概念

VCDesignのもう一つの特徴は、開発プロセスを連続的なフローではなく、「Chapters（章）」という離散的な時間の単位で管理する点にあります[3]。

- **Chapterの定義**: システムの前提条件、主要なビジネスルール、あるいは使用するAIモデルの世代が切り替わるタイミングを一つの区切りとします。
- **価値のハンドオーバー**: Chapter NからChapter N+1へ移行する際、コードベースはAIによって刷新される可能性がありますが、憲法で定義された「価値」と「責任」は、厳格なプロトコル（Value Continuity Protocols）を通じて確実に継承されなければなりません。

これは、従来のバージョン管理システムが「ファイルの差分」を管理していたのに対し、VCDesignは「意味の差分」と「責任の所在」を管理しようとする試みと言えます。環境法規制における「Closure（閉鎖・完了）」の概念が、施設のライフサイクル終了後も続く長期的な管理責任を規定するように、ソフトウェアにおけるChapterの移行もまた、前のChapterで発生した責任（データの永続化、トランザクションの完了）を次のChapterが正しく引き継ぐことを保証する必要があります[7]。

---

3. BOA（Boundary-Oriented Architecture）の構造解析

BOA（Boundary-Oriented Architecture）は、VCDesignの思想を実装レベルに落とし込むための具体的なアーキテクチャパターンです。BOAは、システムを「Fact（事実）」、「Meaning（意味）」、「Responsibility（責任）」という三つの要素（Triad）に分解し、それらの間に強力な境界（Boundary）を設けることで、AIによる自動生成コードの不確実性を封じ込めます[3]

。

3.1 BOAの三要素（The Triad）の詳細定義

要素	定義	AI時代における役割	不変性
Fact (事実)	観測された事象、ユーザー入力、永続化されたデータ。解釈を含まない純粋な情報。	AIが生成する推論の「入力」となるアンカー。AIが幻覚（Hallucination）を起こしても、Fact自体は揺るがない。	高 (Immutable)
Meaning (意味)	Factに対する解釈、計算、変換ロジック。文脈（Context）に依存して変化する。	AIが最も能力を発揮する領域（Vibe Codingの主戦場）。同じFactでも、文脈によって異なるMeaningが生成される。	低 (Ephemeral)
Responsibility (責任)	外部世界への作用（Side Effects）、状態の確定、合意形成。	AIに委ねてはならない領域。最終的なコミットメント（DB書き込み、決済、メール送信）を行うゲートキーパー。	中 (Auditable)

3.2 境界（Boundary）の機能と実装

BOAにおける境界は、単なるモジュール分割線ではありません。それは、AIという「確率的で信頼できないエージェント」と、現実世界という「確定的で責任を伴う環境」を隔離する防壁です。

3.2.1 MeaningとResponsibilityの分離

従来のMVCアーキテクチャやドメイン駆動設計（DDD）では、ドメインモデルの中に「計算ロジック」と「状態変更メソッド」が混在することが一般的でした。しかし、Vibe CodingにおいてAIがロジック（Meaning）を書き換える際、誤って状態変更（Responsibility）のトリガーまで書き換えてしまうリスクがあります。

BOAでは、これらを明確に分離します。例えば、「在庫が不足しているか判断する（Meaning）」コードと、「発注を行う（Responsibility）」コードは、物理的にも論理的にも別の空間（Closure）に配置されます。AIは「在庫不足」というMeaningを生成することはできますが、「発注」というResponsibilityを直接実行することはできず、必ず人間または決定論的なプロトコルによる承認を経由しなければなりません。

この構造は、セキュリティ分野における「サンドボックス」や、プログラミング言語理論における「純粋関数と副作用の分離」を、アーキテクチャレベルで強制するものです[9]。

---

4. Responsibility Closure（責任閉包）とValue Continuity Protocols

4.1 Responsibility Closureの深層分析

responsibility-closureは、VCDesignにおいて最も独創的かつ重要な概念の一つです。これは、ある一連の処理において、責任が「どこで発生し、どこで完全に解消（Close）されるか」を定義する境界線です[11]。

4.1.1 環境法規制からのメタファー

「Closure」という用語は、RCRA（資源保全回復法）などの環境規制において、廃棄物処理施設の閉鎖に伴う一連の財政的・法的責任の履行完了を指す言葉として使われます[7]。施設が物理的に閉鎖されても、土壌汚染のリスクや監視義務は残ります。同様に、AIエージェントがあるタスク（コード生成やデータ処理）を完了しても、その出力がシステムに与える影響（バグ、セキュリティホール、データの不整合）に対する責任は残り続けます。

4.1.2 ソフトウェアにおける実装

Responsibility Closureは、AIが生成した一連の処理（Vibe）をカプセル化し、その副作用が外部に漏れ出すことを防ぐ「封じ込め容器」として機能します。

- **Leakageの防止**: AIが生成したコードが、Closureの境界を越えて勝手にグローバル変数を書き換えたり、許可されていないAPIを叩いたりすることを防ぎます。
- **Atomic Commitment**: Closure内の処理が全て正常に完了し、かつプロトコルによる検証を通過した場合にのみ、その結果（Meaning）が一括してResponsibility（実行）へと昇格されます。

4.2 Value Continuity Protocols（価値連続性プロトコル）

value-continuity-protocolsは、Meaning（AIの解釈）をResponsibility（現実への作用）に変換する際の厳格な審査手続きです[12]。

4.2.1 IDG (Interface Determinability Gate)

IDGは、AIの確率的な出力を、決定論的なインターフェースに変換するゲートです。Vibe Codingでは、AIは自然言語や曖昧な構造で思考しますが、システムの中核（Fact/Responsibility）は厳密な型とスキーマを要求します。
IDGは、AIの出力に対して「Constrained Decoding（制約付きデコーディング）」や「Type Constraints（型制約）」を適用し、不適合な出力を即座に棄却します9。これにより、下流のプロセスに「解釈の揺らぎ」が伝播することを防ぎます。

4.2.2 RP (Responsibility Promotion Protocol)

RPは、IDGを通過した「妥当なMeaning」を、実際に「実行可能なResponsibility」へと昇格させるための承認プロトコルです。

- **人間による承認 (Human-in-the-loop)**: クリティカルな操作（高額決済など）の場合。
- **自動テストによる承認**: ユニットテスト、回帰テスト、Constitutional AIによる倫理チェックを通過した場合。
- **ポリシーベースの承認 (Policy as Code)**: 事前に定義されたガバナンスルール（アクセス権限、時間帯制限など）に合致する場合[14]。

---

5. 類似アーキテクチャとの比較と独創性

VCDesignおよびBOAは、既存の優れたアーキテクチャパターンから多くの影響を受けつつも、AI時代特有の課題解決に向けて独自の進化を遂げています。

5.1 vs. DCI (Data, Context, Interaction)

DCIアーキテクチャは、データ（Data）と振る舞い（Interaction）を文脈（Context）によって動的に結合するという点で、BOAと非常に近い概念を持っています[16]。

比較項目	DCI (Data, Context, Interaction)	BOA (Fact, Meaning, Responsibility)
データ層	Data: ドメイン知識（What the system is）。振る舞いを持たない。	Fact: 不変の事実。解釈を含まない。より厳格にImmutable。
ロジック層	Context/Interaction: ロール（Role）を通じて動的に振る舞いを注入する。	Meaning: 文脈依存の解釈・計算。AIが生成・廃棄する対象。
実行主体	ユーザーのメンタルモデルに基づいたRoleの相互作用。	Responsibility: 副作用の隔離と執行。権限管理と監査に重点。

主な目的	ユーザーのメンタルモデルとコードの乖離を解消する（可読性）。	AIの不確実性とシステムの安全性を分離する（堅牢性）。

独創性: DCIが「人間の認知モデル」をコードに反映させることを目指したのに対し、BOAは「AIという異質な知性」をシステムに統合するための安全弁として機能します。特に「Interaction」を「Meaning（純粋計算）」と「Responsibility（副作用）」に分割した点が決定的です。これにより、AIに計算はさせても、引き金は引かせないという制御が可能になります。

5.2 vs. ドメイン駆動設計 (DDD)

DDDは、複雑なドメインロジックを「Bounded Context（境界づけられたコンテキスト）」によって管理する手法です[19]。

比較項目	Domain-Driven Design (DDD)	VCDesign / BOA
境界の定義	言語（Ubiquitous Language）の適用範囲による意味的な境界。	責任（Responsibility）と価値（Value）の連続性による機能的な境界。
変更への対応	モデルの洗練とリファクタリング（人間主導）。	Chapterの移行とVibe Codingによる再生成（AI主導）。
コンテキスト	空間的な分割（Eコマース文脈 vs 物流文脈）。	時間的な分割（Chapter N vs Chapter N+1）も重視。

独創性: DDDが「空間的」な境界設定に優れているのに対し、VCDesignは「時間的」な境界（Chapters）と「責任」の境界（Closure）を導入しています。AIによる高速なイテレーション（Vibe Coding）において、DDDのドメインモデルは急速に陳腐化または変質するリスクがありますが、BOAのFact/Responsibilityは不変のアンカーとして機能し、モデル（Meaning）の流動性を許容します。

5.3 vs. Constitutional AI (Anthropic)

AnthropicのConstitutional AIは、AIモデルのトレーニングプロセスに「憲法」を組み込むアプローチです[4]。

比較:

- **Anthropicのアプローチ**: AIモデルそのものを「善きもの」にする（Alignment）。入力プロンプトと出力レスポンスの間のブラックボックス内部を調整する。
- **VCDesignのアプローチ**: AIモデルを「信頼できないもの」と仮定し、その外側に「善きシステム」を構築する（Architecture）。AIの出力がシステムに与える影響を構造的に制限する。

独創性: VCDesignは、AIモデルの内部（Weights）ではなく、AIとシステムとのインターフェース（Architecture）に憲法を適用します。これにより、モデルが差し替わっても（例：GPT-4からClaude 3へ）、システムの「憲法」は維持されます。

---

6. 生成AI自動開発時代におけるVCDesignの価値

6.1 「作るコストの消失」がもたらすカオスへの対抗

Vibe Codingにより、コードを書くコストはほぼゼロになります。しかし、それは「検証するコスト」「理解するコスト」「責任を負うコスト」が指数関数的に増大することを意味します。VCDesignは、これらのコストを管理可能なレベルに抑え込むための唯一の解です。

- **コンテキストスイッチコストの低減**: 開発者は、AIが生成した大量のコード（Meaning）の細部を理解する必要がなくなります。Factが正しく定義され、Responsibilityのゲートが機能している限り、中間のロジックはブラックボックスのままでも安全だからです。これにより、平均23分かかるとされるコンテキストスイッチの認知的負荷を大幅に削減できます[22]。

6.2 幻覚（Hallucination）のリスクヘッジ

AIの幻覚は避けられません。しかし、BOAにおいては幻覚は「誤ったMeaningの生成」に過ぎません。IDG（Interface Determinability Gate）がその出力とFactとの矛盾を検知し、RP（Responsibility Promotion Protocol）が実行を阻止することで、幻覚が現実のデータベース汚染や誤発注といった実害（Responsibilityの行使）に繋がることを防ぎます。これは、AIを活用しつつも、AIに「全権委任」しないための現実的な解です。

---

7. 定量評価とメトリクス

VCDesignおよびBOAの導入効果を測定するために、以下の定量的指標を提案します。これらは、従来のソフトウェアメトリクスをAI時代に合わせて再解釈したものです。

7.1 責任境界透過性 (Responsibility Boundary Permeability - RBP)

Change Propagation Probability (CPP) [6] を応用した指標です。Meaning層（AI生成コード）の変更が、どれだけの確率でResponsibility層（手動管理/厳格管理コード）の変更を要求するか

を測定します。

- **定義**: $RBP = \frac{\Delta Responsibility}{\Delta Meaning}$
- **評価**: RBPが0に近いほど、BOAが機能していることを示します。AIがロジックを自由に書き換えても、外部インターフェースや副作用の管理機構は安定している状態です。
- **計測方法**: Gitのコミットログを分析し、boa-core/meaningディレクトリの変更とboa-core/responsibilityディレクトリの変更の相関を算出します。

7.2 認知複雑度対数 (Logarithmic Cognitive Complexity - LCC)

従来のCognitive Complexity [25] は、コードのネストや分岐の深さを測定しますが、Vibe CodingではAIが生成するコードの複雑さは人間にとって制御不能な場合があります。LCCは、システム全体を理解するために人間が読まなければならない「必須コード行数」の割合を測定します。

- **定義**: $LCC = \log(\text{Total LOC}) - \log(\text{AI Generated Meaning LOC})$
- **評価**: BOAにおいては、Meaning層のコード量が増えても、人間がレビューすべきResponsibility層とFact層がシンプルに保たれていれば、LCCは低く抑えられます。これは「AIに任せることで人間が楽になった度合い」を示します。

7.3 憲法遵守率 (Constitutional Adherence Rate - CAR)

Value Continuity Protocolsを通過したAI生成コードの割合です。

- **定義**: $CAR = \frac{\text{Passed RP}}{\text{Total Generations}}$
- **評価**: この値が低い場合、AIプロンプトの品質が低いか、憲法（制約条件）が厳しすぎて現実的でないことを示唆します。CI/CDパイプライン上で、IDGによる拒絶回数をモニタリングすることで自動計測可能です。

7.4 トークン効率性 (Token Efficiency via Boundaries)

ある機能を修正するために、AIのコンテキストウィンドウにロードする必要があるトークン量です[27]。

- **評価**: 責任境界（Closure）が適切に設定されていれば、AIには局所的なFactとMeaningのみを与えればよく、システム全体のコードをロードする必要がなくなります。これにより、APIコストの削減と生成速度の向上（Latency削減）が期待できます。

---

8. 将来展望：アーキテクトから「憲法起草者」へ

VCDesignは、ソフトウェアエンジニアの役割を「ビルダー」から「アーキテクト（設計者）」、さらには「レジスレーター（立法者）」へと進化させます。

8.1 Governance as Codeの進化

value-continuity-protocolsは、将来的には「Governance as Code」の一部として、組織のコン

プライアンス要件やセキュリティポリシーと統合されるでしょう[14]。法律や規制が変われば、憲法（コード上のルール）が更新され、AIは新しい憲法に従って即座にシステム全体をリファクタリングする——そんな「自己修復的で遵法的なシステム」が実現します。

8.2 マルチエージェントシステムへの拡張

複数の特化型AIエージェントが協調する「Agentic AI」アーキテクチャにおいても、BOAは共通言語として機能します[29]。エージェントA（UI担当）とエージェントB（DB担当）は、互いの内部ロジック（Meaning）を知る必要はなく、共有されたFactと、合意されたResponsibilityのプロトコルを通じてのみ対話します。これにより、エージェント間の予期せぬ干渉を防ぎ、スケーラブルな自律システムの構築が可能になります。

---

9. 結論

Vibe Codingという津波のような変化の中で、VCDesignとBOAは、我々が流されることなく「価値」という足場を固めるためのアンカーです。コードは書くものではなく、AIに書かせるものになるでしょう。しかし、そのコードが「何を守るべきか（憲法）」、「どこまでやっていいのか（境界）」、「誰が責任を取るのか（閉包）」を定義するのは、依然として人間の、そして高度な設計思想を持つアーキテクトの仕事です。

vcdesign-core、boa-core、responsibility-closure、value-continuity-protocolsというツール群は、この新しい時代の秩序を構築するための不可欠な構成要素となるでしょう。我々は今、ソフトウェア工学の歴史において、構造化プログラミングやオブジェクト指向の登場に匹敵する、あるいはそれ以上の大きな転換点に立っているのです。

参照文献

3

1

引用文献

1. Vibe Coding Explained: Tools and Guides | Google Cloud, 1月 17, 2026にアクセス、 https://cloud.google.com/discover/what-is-vibe-coding
2. What is Vibe Coding? - IBM, 1月 17, 2026にアクセス、 https://www.ibm.com/think/topics/vibe-coding
3. VCDesign — Design where decisions live., 1月 17, 2026にアクセス、 https://vcdesign.org/
4. 1月 17, 2026にアクセス、 https://xenoss.io/ai-and-data-glossary/constitutional-ai#:~:text=Constitutional%20AI%20is%20an%20approach,Transparency

5. Constitutional AI aims to align AI models with human values - Ultralytics, 1月 17, 2026にアクセス、 https://www.ultralytics.com/blog/constitutional-ai-aims-to-align-ai-models-with-human-values
6. Change Propagation for Assessing Design Quality of Software Architectures, 1月 17, 2026にアクセス、 https://www.computer.org/csdl/proceedings-article/wicsa/2005/25480205/12OmNzV70xz
7. RCRA Subtitle C Closure and Post-Closure - epa nepis, 1月 17, 2026にアクセス、 https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=91021URB.TXT
8. Concise Explanatory Statement and Responsiveness Summary for Amendments to Chapter 173-303 WAC - Dangerous Waste Regulations - Washington State Department of Ecology, 1月 17, 2026にアクセス、 https://apps.ecology.wa.gov/publications/documents/0404028.pdf
9. Type-Constrained Code Generation with Language Models - Research Collection, 1月 17, 2026にアクセス、 https://www.research-collection.ethz.ch/bitstreams/822be548-b06a-4f58-9aa5-57cf64baaa1e/download
10. Type-Constrained Code Generation with Language Models - arXiv, 1月 17, 2026にアクセス、 https://arxiv.org/pdf/2504.09246
11. 1月 1, 1970にアクセス、 https://github.com/VCDesign-org/responsibility-closure
12. 1月 1, 1970にアクセス、 https://github.com/VCDesign-org/value-continuity-protocols
13. Thinking Before Constraining: A Unified Decoding Framework for Large Language Models, 1月 17, 2026にアクセス、 https://arxiv.org/html/2601.07525v1
14. Top Trends in Data Governance for 2025: Insights from Industry Leaders | Secoda, 1月 17, 2026にアクセス、 https://www.secoda.co/blog/top-trends-in-data-governance-for-2025-insights-from-industry-leaders
15. AI Safety Techniques in 2025: From Alignment to Adversarial Robustness - GoCodeo, 1月 17, 2026にアクセス、 https://www.gocodeo.com/post/ai-safety-techniques-in-2025-from-alignment-to-adversarial-robustness
16. Data, context and interaction - Wikipedia, 1月 17, 2026にアクセス、 https://en.wikipedia.org/wiki/Data,_context_and_interaction
17. Structuring Use Cases in Clean Architecture as Conversations with the User - Medium, 1月 17, 2026にアクセス、 https://medium.com/unil-ci-software-engineering/structuring-use-cases-in-clean-architecture-as-conversations-with-the-user-a91a7c517ac2
18. Data Context Interaction: The Evolution of the Object Oriented Paradigm - SitePoint, 1月 17, 2026にアクセス、 https://www.sitepoint.com/dci-the-evolution-of-the-object-oriented-paradigm/
19. Organizing Business Logic with DCI in Rails - Minthesize Company Blog, 1月 17, 2026にアクセス、 https://blog.minthesize.com/dci-rails
20. Alexej Sommer - Architectures that we can use with .NET - WeAreDevelopers, 1月

17, 2026にアクセス、 https://www.wearedevelopers.com/videos/935/architectures-that-we-can-use-with-net
21. Public Constitutional AI - Digital Commons @ University of Georgia School of Law - UGA, 1月 17, 2026にアクセス、 https://digitalcommons.law.uga.edu/cgi/viewcontent.cgi?article=1819&context=glr
22. AI Minimizes Work Context Switching_ | PDF | Artificial Intelligence - Scribd, 1月 17, 2026にアクセス、 https://www.scribd.com/document/961805850/AI-Minimizes-Work-Context-Switching
23. What Is Task Switching Cost? - Monitask, 1月 17, 2026にアクセス、 https://www.monitask.com/en/business-glossary/task-switching-cost
24. Quantifying software architectures: an analysis of change propagation probabilities - IEEE Xplore, 1月 17, 2026にアクセス、 https://ieeexplore.ieee.org/iel5/9525/30191/01387113.pdf
25. Identifying Code Complexity's Effect on Dev Productivity | Faros AI, 1月 17, 2026にアクセス、 https://www.faros.ai/blog/code-complexity-impact-on-developer-productivity
26. Why Does Cognitive Complexity Matter in Software Development? - Typo, 1月 17, 2026にアクセス、 https://typoapp.io/blog/why-does-cognitive-complexity-matter-in-software-development
27. Track: San Diego Poster Session 1 - NeurIPS, 1月 17, 2026にアクセス、 https://neurips.cc/virtual/2025/loc/san-diego/session/128331
28. The Future of AI Governance in 2025: Tools, Trends & Frameworks - Codepaper, 1月 17, 2026にアクセス、 https://codepaper.com/blog/the-future-of-ai-governance-in-2025/
29. Enterprise Agentic Architecture and Design Patterns | Salesforce Architects, 1月 17, 2026にアクセス、 https://architect.salesforce.com/fundamentals/enterprise-agentic-architecture
30. Agentic Architecture: Blueprint for Enterprise AI Architecture - Kore.ai, 1月 17, 2026にアクセス、 https://www.kore.ai/blog/agentic-architecture-blueprint-for-intelligent-enterprise
31. 1月 1, 1970にアクセス、 https://github.com/VCDesign-org/vcdesign-core
32. 1月 1, 1970にアクセス、 https://github.com/VCDesign-org/boa-core
33. Responsibility-driven design - Wikipedia, 1月 17, 2026にアクセス、 https://en.wikipedia.org/wiki/Responsibility-driven_design
34. 2. Object-Oriented Design Principles, 1月 17, 2026にアクセス、 https://scg.unibe.ch/assets/download/lectures/p2-2012/P2-02-OODesign.pdf
35. Paper2Poster: Towards Multimodal Poster Automation from Scientific Papers - arXiv, 1月 17, 2026にアクセス、 https://arxiv.org/html/2505.21497v2
36. Automating hallucination detection with chain-of-thought reasoning - Amazon Science, 1月 17, 2026にアクセス、 https://www.amazon.science/blog/automating-hallucination-detection-with-chain-of-thought-reasoning

37. Hallucination Detection and Mitigation in Large Language Models - arXiv, 1月 17, 2026にアクセス、 https://arxiv.org/html/2601.09929v1
38. CICube - Optimize CI costs with targeted insights, 1月 17, 2026にアクセス、 https://cicube.io/
39. Cyclomatic complexity and cognitive complexity - DEV Community, 1月 17, 2026にアクセス、 https://dev.to/packmind/cyclomatic-complexity-and-cognitive-complexity-4c03