
ソフトウェア開発サイクルの変革と
AI Agent の動向

はじめに

AI技術の発展が、社会のあらゆる分野に深い影響を及ぼすだろうということは、さまざまな人がさまざまな形で語っています。

今回のセミナーは、「あらゆる分野」にではなく、僕を含むソフトウェア開発につらなる世界に、AIがどのような変化をもたらすのかを考えようとしたものです。

IT技術は、インターネットにしろスマートフォンにしろSNSにしろ、世界のあり方を大きく変えてきました。

IT技術者にとって、IT技術は、こうした変化の能動的な主体であり、それ以外の世界は変化を受ける受動的な客体であると考えても、大きな不都合はありませんでした。

ただ、AIという武器を得た新しいIT技術が、変えようとしている世界の一つに、他ならぬITの世界そのものが含まれていることには注意が必要です。

新しいIT技術が、変革の対象として選ぼうとしているのが、ソフトウェアの生産現場です。対象リストの中で、その優先順位はとても高いと僕は考えています。

第一部： ソフトウェア開発でのAI利用の動向

セミナーの第一部では、まず、ソフトウェア開発でのAI利用の動向を確認したいと思います。

この分野へのIT技術者の関心は高く、さまざまなプロダクトが登場しています。そこでは、AI ベンダートップの楽観的な見通しと強気な発言が特徴的です。膨大な投資がなされています。

同時に、セキュリティに関心を持つ人の間では、かつてないほどの規模で AIに対する警戒心が広がっているように見えます。また「攻撃者としてのAI」からシステムを守る「防御者としてのAI」という議論も活発に展開されています。

第二部： ソフトウェア開発とAI Agent

これまで、AIの利用はコード補完のような特定のタスクに限定されていましたが、現在ではより広範なソフトウェア開発ライフサイクル（SDLC）全体に関与し、自律的にタスクを実行する「AIエージェント」へと進化しつつあります。

これらのAIエージェントは、単なるコード生成ツールを超え、設計、テスト、デバッグ、コードレビューといった複雑なエンジニアリングタスクを支援、あるいは自動化する可能性を秘めています。

この第二部では、ソフトウェア開発におけるAIエージェントの導入の現状を見ていきたいと思います。

また、AI agent プログラミングの特徴を見ていきます。

第三部： ソフトウェア開発の課題と未来

セミナーの第三部では、こうしたAI Agent を利用したAI開発ツールが抱える問題を取り上げます。

一つには、現在の主要なAI開発ツールは、主にコード補完やチャット支援に重点を置いており、機能は限定的です。

大規模プロジェクトの複雑な構造やモジュール間の連携をAIに直接指示することは、まだできません。何よりも、人間が、機械にどういう形で、ソフトウェアの「仕様」を提示すべきか明確な指針が存在していません。

もう一つには、LLMは驚異的なコード生成能力を示す一方で、生成されたコードは、信頼性の問題をいくつか抱えています。

こうした中で、LLMのコード生成能力と形式手法の厳密性を統合するアプローチに注目が集まっています。こうした動きは、ソフトウェア開発の自動化において革新的な可能性を秘めていると、僕は考えています。

もう一つ、大事な問題があります。それは、こうした変化が進む中で、人間と機械との関係がどのように変化し、特に人間には、どのようなスキルが必要とされるかということです。

第一部

ソフトウェア開発でのAI利用の動向

第一部：ソフトウェア開発でのAI利用の動向

AI利用の急速な拡大

- ザッカーバーグの予測
- AIベンダーの動向

広がる危機感　RSAC 2025

- 攻撃者としてのAI / 防御者としてのAI
- AIとNHIによる脅威の拡大


AI利用の急速な拡大
ザッカーバーグの予測

"AI Will Do Most Of Coding Soon,
Better Than Top Coders”

ザッカーバーグの予測

2025年5月、MetaのCEOであるマーク・ザッカーバーグ氏は、今後12～18ヶ月以内に同社のLlamaプロジェクトのコードの大部分がAIによって書かれるようになるだろうという、注目すべき予測を発表しました。

この発言は、単なるコード補完を超え、AIがテスト実行、バグ発見、そして熟練した人間のエンジニアをも凌駕する可能性のある高品質なコード生成を行う未来を示唆しています。

自動化を推進する経済的インセンティブ

ソフトウェアエンジニア、特に大手テック企業のエンジニアの高い人件費と、AIによる生産性向上の実証は、企業がAIコード生成の限界を押し広げる強い経済的インセンティブを生み出しています。

Metaの「効率化の年」に見られるように、企業は積極的に効率改善を追求しています。特にザッカーバーグ氏がターゲットとする中級レベルのエンジニアが行うコーディングタスクを自動化することは、コスト削減と効率向上の直接的な道筋を提供します。

これは、AIへの大規模な投資を促進し、潜在的に労働力の調整につながる可能性があり、「支援」と「代替」の力学において、代替の側面を加速させる可能性があります。

Metaのビジョンと戦略

ザッカーバーグ氏の予測の背景には、Meta独自のAI戦略があります。この戦略は、オープンソースによるリーダーシップと、内部開発の加速という二つの柱によって特徴づけられます。

同社は、LlamaシリーズやCode Llamaといった強力な基盤モデルをオープンソースとして公開し、業界標準の確立とイノベーションの促進を目指す一方で、CodeComposeのような内部ツールやLlama研究を加速するための特化型AIエージェントの開発にも注力しています。

オープンソースでのリーダーシップ

Metaは、Llama 2、Code Llama、そしてLlama 3といった強力な基盤モデルを、研究や商用利用を可能にする寛容なライセンスの下で公開してきました。

その目的は、イノベーションの推進、コミュニティによる評価を通じた安全性向上、エコシステムの標準化、そして開発者コミュニティとの良好な関係構築にあると述べられています。

同社が開催したLlamaConイベントも、このオープンソースへのコミットメントを示す象徴的な動きです。

内部開発の加速

オープンソースへの貢献と並行して、Metaは内部でのAI開発を加速させるための、特化したプロプライエタリなAIエージェント（コーディングエージェント、AIリサーチエージェント）の開発に注力しています。

これらのエージェントは、Llama開発を加速させることを明確な目的としています。既に社内で展開されているAI支援ツールであるCodeComposeや、SQL生成に特化したSqlComposeは、この内部戦略の具体的な現れです。

基盤としてのインフラ投資

この二元的戦略を支えるのが、AIインフラへの巨額投資です。

Metaは2025年に600億ドルから650億ドルの設備投資を計画しており、その大部分はAIトレーニングクラスター、データセンター、次世代チップに向けられます。

具体的には、130万基のNvidia製GPUの導入、2ギガワット超の電力を消費する巨大データセンターの建設計画、そして効率化のためのカスタムシリコン（MTIAチップ）の開発などが含まれます。

この圧倒的な計算能力は、最先端モデルのトレーニングだけでなく、ザッカーバーグ氏が構想するような強力なAIコーディングエージェントを社内で大規模に展開するために不可欠です。

戦略と予測の連関

Metaの戦略は、ザッカーバーグ氏の発言と密接に結びついています。

内部向けAIエージェントは、Llamaプロジェクトに関する18ヶ月目標を達成するための直接的な手段です。オープンソースの取り組みは、これらのエージェントの基盤となるモデルやエコシステムを構築・強化します。

そして、インフラ投資は、この野心的な計画を実行するために必要な計算能力を提供します。


AI利用の急速な拡大
AIベンダーの動向

AIベンダーの動向

ソフトウェア開発における人工知能（AI）の利用は、単純なコード補完を超え、ソフトウェア開発ライフサイクル（SDLC）全体にわたる複雑でエージェント的なタスクを包含する方向へと急速に進化しています。

この分野は、先に見たMetaだけでなく、Google、Microsoft、OpenAI、Anthropic、Amazonといった主要なテクノロジー企業にとって戦略的に重要であり、各社は製品開発と基礎研究の両方に多額の投資を行っています。

ソフトウェア開発におけるAIの加速的進化

先のセッションで取り上げたMeta社のAIへの取り組みとザッカーバーグ氏の予測を背景に、業界全体ではAIが単なる開発者のアシスタントから、コーディングタスクにおける潜在的な協力者、さらには自動化主体へと移行しつつある広範なトレンドが見られます。

基本的なコード提案から、関数全体の生成、単体テスト、ドキュメンテーション作成、デバッグやコードレビューへの参加へと機能が拡大しています。これにより、生産性の大幅な向上が期待される一方で、過度の依存やソフトウェアエンジニアリングの役割の将来に対する懸念も提起されています。

Agent 的シフト

この進化の核心には、「エージェント的シフト」とも呼べるパラダイムの変化があります。OpenAI、Anthropic、Amazon、Microsoft、Google など、複数の企業がコーディングのための「エージェント的」AIシステムを明示的に開発または議論しています。

これらのエージェントは、受動的な提案を超えて、複数ステップのタスクを能動的に実行し、ツールと対話し、自己修正さえ行います。研究論文もこのシフトを反映しており、エージェント的なワークフローや、長期間タスク完了能力の評価 に焦点を当てています。

この動きは、初期のAIコーディングツールがコード補完に焦点を当てていた段階から、より大きなコードブロック、関数、テストを生成する能力へと進化し、さらにリーダーたちが「エージェント的コーディング」の概念を提唱するに至った自然な流れを示しています。

Amazonのような企業がエージェントAI専門のグループを設立したことも、このトレンドを裏付けています。

これは、AIがソフトウェア開発プロセスに単に支援するのではなく、積極的に参加する未来への収束を示唆しています。その結果、競争の主戦場はコード提案の質から、複雑なソフトウェアエンジニアリングワークフローを自動化するAIエージェントの有効性と信頼性へと移行しています。

このセッションでは、主要なAIベンダーの取り組みとビジョンを紹介しようと思います。

主要なAIベンダーの取り組みとビジョン

このセッションでは、次のAIベンダーの取り組みとビジョンを紹介しようと思います。

- Google
- Microsoft
- OpenAI
- Anthropic
- Amazon

このセッションのコンテンツは、いくつかのDeep Seek エンジンの出力を、丸山が、自分の好みで編集したものです。

GoogleのGemini Code Assist

Googleの戦略は、そのエコシステムへの深い統合に特徴づけられます。Gemini Code AssistがFirebase、BigQuery、Apigee、Cloud Runなどと緊密に連携していること は、単なるスタンドアロンのコーディングツールを超えて、Google Cloud Platform（GCP）自体の価値提案を高めることを目的とした戦略を示唆しています。

特定のGCPサービスと連携した機能（例：ApigeeでのAPI開発、BigQueryでのデータ分析、Firebaseでのアプリ開発） は、OpenAIのモデル中心アプローチやMicrosoftのGitHub Copilotを通じたIDE/開発者中心アプローチとは対照的です。

これは、GCPに深く投資している顧客にとっては、シームレスな統合によりGemini Code Assistが非常に魅力的になる一方で、エコシステム外のユーザーにとっては、よりプラットフォームに依存しないツールと比較して魅力が薄れる可能性があることを意味します。

また、Googleは軍事/防衛関連組織にAI技術を提供している可能性を示唆しており、同社は2018年の特定の「レッドライン」を削除する形でAI原則を公式に改訂しました。この改訂は、地政学的または商業的な競争圧力、あるいは新たな技術的能力によって推進された可能性のある内部的な戦略転換を示唆しています。

Microsoft と GitHub Copilot

GitHubの巨大な開発者コミュニティとプラットフォームを活用します。自然言語を通じて非技術系ユーザーにもアクセス可能にすることで、ソフトウェア開発の民主化に焦点を当てています。アクセシビリティを強く重視しています（身体的障害を持つ開発者のためのCopilot Voice、すべての人に利益をもたらす）。ビジョンには、パーソナライズされたデジタルコンパニオンとしてのCopilotが含まれます。

開発者の生産性を大幅に向上させることを目指しています。（Nadella氏：MSコードの30%がAIによって書かれている；Scott氏：2030年までに95%になると予測）。

Nadella氏は、AIの影響が生産性向上に完全に現れるまでには「何年もかかる」と認め、電力の普及と比較しています。

Microsoftのリーダーシップは、「誰もが」開発者になれるようにすることと、既存の開発者の生産性を劇的に向上させること（Nadella氏の30%、Scott氏の95%予測）の両方を強調しています。

これらの目標は補完的に見えるかもしれませんが、緊張を生む可能性もあります。開発の民主化はコーディングへの参入障壁を下げることを目指しますが、Copilotのような生産性ツールは経験豊富な開発者をより速く、より効率的にすることを目的としています。

もしAIがコードの95%を書くようになれば、これは本当に開発を民主化するのでしょうか、それとも主に専門家を補強するのでしょうか？

Nadella氏の電力との比較 は、生産性向上のためには長期的なシステム全体の変化が必要であることを示唆しており、単なるコード生成以上のもの（ワークフローの変更、マネジメントの適応）が必要であることを意味します。

これは、Microsoftが二方面戦略を追求していることを示唆しています。つまり、初心者向けのツールを十分にシンプルにしながら、専門家向けの機能を深化させ、同時に、予測される生産性向上を実現するためにはAIツール自体以上のものが必要であることを認識し、より広範なソフトウェア開発文化とプロセスに影響を与える可能性があるということです。

アクセシビリティへの焦点は、物理的な障壁を取り除き、すべてのユーザーにとってインタラクションを簡素化する可能性があるため、両方の目標に貢献します。

OpenAI

特に推論とコーディングにおいて、AIモデルの能力の限界を押し広げ、他の人が構築するための基盤となる強力なベースモデルをAPI経由で提供することに焦点を当てています。

ビジョンには、AIが超人的なコーディング能力を達成し、エージェント的コーディングが複雑なタスクを引き継ぐことが含まれます。

Altman氏は、長期的には人間のソフトウェアエンジニアの需要が減少する可能性があると予測しています。広告ではなく、プレミアムツール/エージェントを通じて収益化することに焦点を当てています。

OpenAIの主要な焦点は、発表やリーダーシップの発言に反映されているように、基盤となるモデル（GPT-4 → GPT-4.1 → 将来のモデル）の生の能力を進歩させることにあります。

製品機能は、しばしばこれらのモデルの改善を示すものです。改善は、モデルのメトリクス（SWE-bench、Aider diff、MMLU、コンテキストウィンドウサイズ）の観点から枠組み化されています

Altman氏のビジョンは、AIが「最高のコーダー」になり、「エージェント的コーディング」を可能にすることに焦点を当てており、これは将来のより能力の高いモデルへの依存を意味します。収益化戦略は、強力なモデル/エージェントへのアクセスに対して課金することに焦点を当てています。

これは、Google/Amazonのプラットフォーム統合やMicrosoftの開発者エコシステムへの焦点とは対照的です。

OpenAIは、優れたコアモデルのインテリジェンスが主要な差別化要因になると賭けています。彼らの競争優位性は、基礎的なモデルパフォーマンスにおけるリーダーシップを維持することにかかっています。

彼らの戦略は、特定の開発者ツールに関するものではなく、コーディングタスクのための最も強力なAI「エンジン」を提供することに重点を置いており、他の人がそれを統合したり、その上に構築したりすることができます。

彼らの進歩は、LLM研究とスケーリングにおけるブレークスルーに大きく依存しています。

Anthropic

安全性、セキュリティ、信頼性を強く重視した（「Constitutional AI」）強力なAIモデルを開発します。API（AWS、GCP）および直接対話（claude.ai、アプリ）を通じてモデルを提供します。

ビジョンには、コード生成と並行して、複雑な推論、視覚分析、多言語タスクを処理するAIが含まれます。

Amodei氏は、AIがコーディングを非常に急速に引き継ぐと予測していますが（数ヶ月で90%、1年で100%）、雇用の影響や責任ある開発の必要性についても懸念を表明しています。

AnthropicはAIの安全性と責任ある開発を使命として設立されました。しかし、CEOのDario Amodei氏は、AIがコーディングタスクを急速に自動化するという、最も積極的な予測のいくつかを行っています。これは潜在的に破壊的な能力です。

Anthropicの創設原則はAIの安全性であり、OpenAIとの差別化を図っています。彼らのメッセージは、Constitutional AIのような信頼性、信頼性、安全機能を強調しています。

しかし、Amodei氏は、AIが12ヶ月以内にコードの100%を書くと予測しています。このような急速な自動化は、非常に高い能力を意味しますが、同時に重大な社会的混乱（雇用の喪失）も意味し、これは純粋な安全第一のアプローチとは潜在的に矛盾するように思われます。

これは、Anthropicが高い能力を安全に達成できると信じているか、あるいは能力の進歩は避けられず、彼らの焦点は移行を責任を持って管理することにあることを示唆しています。

Amodei氏が表明した雇用の影響に関する懸念は、この緊張を認めています。したがって、Anthropicは、安全性重視のブランドと、構築しているAIの潜在的に破壊的な力とを調和させるという課題に直面しています。

彼らの成功は、コーディングのようなタスクのための非常に能力の高いAIが、技術的進歩と並行して社会的影響を乗り越えながら、信頼性高く倫理的に展開できることを実証することにかかっているかもしれません。

Googleのような投資家との関係は、このダイナミクスにさらなる層を加えています。

Amazon (AWS)

DevOpsライフサイクル全体にAIを統合し、開発者の専門知識を補強し、コード品質を向上させ、パフォーマンスを最適化し、セキュリティを確保し、クラウドの採用/移行を加速します。

AWSエコシステム統合とAWSサービスに最適化されたツールの提供に焦点を当てています。エージェント的AIへの強力な推進（Nova Act、AWS CEO直属のSwami Sivasubramanian氏率いる専門グループ、Andy Jassy氏直属の内部AGIチーム）。

AIエージェントをAWSにとって潜在的な「数十億ドル規模のビジネス」と見なすビジョン。Amazon.comからの運用経験を活用します。

AmazonのAI開発者ツール（CodeWhisperer、CodeGuru）とエージェントイニシアチブ（Nova Act、Sivasubramanian氏のグループ）は、AWSとDevOpsの文脈で強く位置づけられています。

そのビジョンは、クラウドへの移行を加速し、AWS上で実行されるアプリケーションを最適化することを強調しています。Amazonの主要な収益源はAWSです。CodeWhispererのようなAIツールは、AWSサービス向けに明示的に最適化されています。

CodeGuruは、アプリケーションのコード品質とパフォーマンス向上に焦点を当てており、これは暗黙的にAWSインフラストラクチャ上で実行されるものに利益をもたらします。DevOps Guruは、アプリケーションの運用パフォーマンスと可用性の向上を直接対象としており、これはクラウドユーザーにとって中心的な関心事です。

Sivasubramanian氏の下の新しいエージェント的AIグループはAWS内に位置づけられ、クラウドインフラストラクチャ管理やコーディングタスクの自動化といった潜在的な応用分野があります。

したがって、Googleと同様に、AmazonはAI開発者ツールとエージェントを、主にその中核的なクラウドプラットフォームであるAWSの価値と定着性を高めるために戦略的に展開しているように見えます。AmazonのAI開発者戦略は、そのクラウド支配と深く結びついています。

成功は、ツールの採用だけでなく、これらのツールがAWSの利用をどれだけ効果的に促進し、AWS上の顧客の運用効率を向上させ、自社のクラウドと統合された同様のAI機能を提供する競合他社に対してどれだけ防御できるかによって測られるでしょう。


広がる危機感　RSAC 2025
攻撃者としてのAI / 防御者としてのAI

RSAC 2025には、過去最大の 44,000人が参加したという

講演者700名以上、セッション数450以上という

攻撃者としてのAI / 防御者としてのAI

2025年のRSAカンファレンス（RSAC）は、サイバーセキュリティに対する人工知能（AI）の多面的な影響というテーマが明確に支配的であった。

議論は、AIが防御を革新する可能性から、洗練された攻撃ベクトルとしてのAIの憂慮すべき出現、そして実際に主要なセキュリティ懸念事項としてのAIに至るまで、多岐にわたった。

僕が感じた、「攻撃者としてのAI」や「AIがセキュリティの最大の敵になるだろう」という内容に対する「ショック」は、カンファレンスで中心的かつ広範に議論されたテーマを反映している。

RSAC 2025からの重要な示唆は、AIを理解し、統治し、保護することが、もはやニッチな懸念ではなく、ソフトウェア開発に携わる人々を含むすべての利害関係者にとって基本的な必須事項であるということである。

カンファレンスにおけるAI関連コンテンツの膨大な量と、その根底にある懸念のトーンは、テクノロジーの能力が、そのリスクを確保または完全に理解する集合的な能力よりも速く進歩しているという事実に、業界が取り組んでいることを示唆している。

RSAC 2025は、サイバーセキュリティの未来がAIと密接に結びついていることを明確に示したカンファレンスであった。

攻撃者としてのAI
脅威の根本的な変化

RSAC 2025では、AIが攻撃者としてますます巧妙化し、サイバーセキュリティの様相を根本的に変えつつあるという懸念が前面に押し出された。

「攻撃者としてのAI」というテーマは、脅威の状況における根本的な変化を意味しており、生産性向上のために設計されたツール自体が攻撃の経路となり得ることで、正当なソフトウェアの挙動と悪意のあるソフトウェアの挙動の境界線を曖昧にしている。

これは、ソフトウェアコンポーネントや開発ツールに対する信頼の再評価を必要とする。

従来のセキュリティモデルは、ソフトウェアツールが明示的に侵害されない限り無害であると仮定することが多い。

Zenityの調査は、AIツールが従来の侵害なしに改ざんされる可能性があることを示している。Diana Kelley氏の「シャドーAI」は、本質的に安全でない可能性のある内部開発AIを指摘している。これは、「攻撃者」が常に外部エンティティであるとは限らず、AIシステム自体であるか、AIシステムが知らず知らずのうちに攻撃を可能にしている可能性があることを意味する。

攻撃者としてのAI
AIを活用したサイバー犯罪の急速な拡大

カンファレンスでは、AIが高度な国家レベルの攻撃者だけのツールではなく、高度な攻撃能力をますます民主化していることが強調された。

RSACで注目されたFortinetの2025年グローバル脅威ランドスケープレポートでは、「AIを活用したサイバー犯罪が急速に拡大している」と明確に述べられている。

これは、より多くの攻撃者が、ポリモーフィックマルウェアの作成、非常に説得力のあるフィッシングキャンペーン、ディープフェイクの生成にAIを活用できることを意味する。

アカウントのAIジャック

Michael Bargury氏によるZenityのセッション「Your Copilot Is My Insider」は、AIアシスタント（Copilot）がユーザーアカウントを侵害することなく「AIジャック」され、悪意のあるインサイダーに変貌し、新たな初期アクセスベクトルとして機能する方法が実証された。

このセッションは、AIが攻撃者を支援するのではなく、AIが侵害されたエンティティになるというパラダイムシフトを示しているため、極めて重要である。調査によると、Copilotは、その行動や出力をリモートで制御するように改ざんされ、スピアフィッシングを行わせたり、ユーザーに誤った情報に基づいた意思決定をさせたりすることが可能であり、これらすべてが直接的なアカウント侵害なしに行われることが示された。

シャドーAI

Diana Kelley氏のパネル「Shadow AI: Shining the Governance Light on AI」では、組織内で適切な監視なしに開発または使用されるAIシステムの急増が取り上げられ、重大な統制されていないリスクが生じていると指摘された。

シャドーAIは、抑制されないAI導入から生じる広範な攻撃対象領域を表しており、AIが内部から「攻撃者」になることを容易にする。

このパネルでは、透明性、AI部品表（AI-BOM）、説明可能性、そしてリスクを低減し倫理的なAI利用を確保するためのガバナンス戦略が検討された。このような議論の必要性自体が、現在のガバナンスが不足していることを示唆している

AI生成コードの脆弱性

ソフトウェア開発のバックグラウンドに特に関連する繰り返される懸念は、AI生成コードが脆弱性を導入または増幅するリスクである。

これは、AIモデルが安全でないコードでトレーニングされた場合や、生成されたコードが厳密に検証されない場合に発生する可能性がある。

AIによる防御の強化
AIセンチネル

攻撃者としてのAIの脅威が増大する一方で、RSAC 2025では、AIがサイバーセキュリティ防御を強化するための強力なツールとしての可能性も強調された。

AIによるAIの監視: RSACで提案された戦略の1つは、他のAIツールの不正行為や侵害を監視するためにAIツールを設計することであった。これは、AIシステム自体が専門的な監視を必要とするという理解が深まっていることを反映している。

高度な脅威検出とインテリジェンス：SecAIのような企業は、自然言語対話、詳細な情報統合、文脈に応じたセキュリティ推論にAIを活用して、厳選された脅威インテリジェンスを提供するプラットフォームを発表した。ElasticのSIEM移行のための新しいAI機能も、AIが分析タスクを強化していることを示している。

AI駆動型セキュリティオペレーション
Microsoft: Security Copilot

MicrosoftのSecurity Copilotとその新しいAIエージェント（現在プレビュー版）に関する発表は大きな注目を集め、セキュリティスタック全体での検出、調査、対応の強化におけるAIの役割を示した。

Conditional Access Optimization Agent、Phishing Triage Agent、Alert Triage Agentsなどのエージェントは、大量のタスクを自動化し、フィードバックから学習し、ゼロトラストフレームワーク内で動作することを目指しており、人間の防御者がより複雑な脅威に集中できるようにする。

これは、「攻撃者としてのAI」の物語に対する対抗策を提供し、サイバーセキュリティにおけるAIの建設的な利用を示している、

進化するゼロトラスト
新たな現実への適応

自律的に行動したり改ざんされたりする可能性のある洗練されたAIの台頭は、ゼロトラストアーキテクチャに極度の圧力をかけている。ゼロトラストは、AIエージェントとその相互作用を厳格に認証および認可するように進化し、意図された機能にもかかわらず、潜在的に信頼できないエンティティとして扱う必要がある。

Ellis氏の批判 は、現在のゼロトラストでは不十分である可能性を示唆している。AIエージェントが「AIジャック」されたり、な望ましくない行動を示したりする場合、ゼロトラストの「検証」側面は継続的であり、最初のアクセスだけでなく機械/AIの行動にも適用されなければならない。「最小権限」の原則は、そうでなければ膨大な量のデータにアクセスする可能性のあるAIエージェントにとって不可欠である。

「Having Zero Trust to Give: What Should Have Been Next?」

ndy Ellis氏のセッション：「Having Zero Trust to Give: What Should Have Been Next?」 は、標準的なゼロトラスト原則（「侵害の想定」が誤った信頼決定につながるなど）を批判し、実践的なアプローチ（強力な認証（エンドポイントとユーザー）、管理者権限の削減、摩擦のないアクセスプロビジョニング）を提唱した。

これは、基本的なセキュリティ戦略に関する、ユーザーが注目したセッションである。Ellis氏は、ゼロトラストはテクノロジーを実装するだけでなく、信頼モデルを再考することであると主張した。彼の実践的なアプローチは、より堅牢なゼロトラスト体制に向けた具体的なステップである。

AIエージェントのためのゼロトラスト

ゼロトラストの原則（決して信頼せず、常に検証する。最小権限）は、データやシステムへの広範なアクセスを持つ可能性のあるAIエージェントを扱う場合にさらに重要になる。

例えば、MicrosoftのSecurity Copilotエージェントは、ゼロトラストフレームワークに沿って動作すると述べられている。

AIレッドチームの現状

AIによる攻撃の脅威が現実のものとなる中、AIシステムの安全性と堅牢性を評価するためのレッドチーム活動の重要性が増している。RSAC 2025では、AIレッドチームの有効性、方法論、そして人的要因に関する活発な議論が交わされた。

「ほとんどのAIレッドチームは役に立たない ― AI Villageからの教訓」 Cattell氏は、LLMのようなAIモデルがもたらす多様で急速に進化する脅威に対して、専門チームによる従来のプライベートなレッドチーム活動は不十分であると主張した。

Cattell氏は、モデルに関する多様な問題は、より多くの人々がレッドチーム活動を行い評価する方法を知るまで解決されないと強調した。彼は、バグバウンティやライブハッキングイベントのような公開参加型の取り組みを、MLシステム用に修正して実施し、害に対処し、研究者コミュニティを成長させることを提唱した

安全なAIの未来の構築

Diana Kelley氏のセッション:「Principles of GenAI Security: Foundations for Building Security In」。

このセッションは、GenAIのユニークな攻撃対象領域とリスク、この新しいパラダイムにおけるセキュリティ原則の進化、エージェント型AIシステムの保護、リスク管理、安全なアーキテクチャの構築に関する実践的ガイダンスに焦点を当てた。

この講演は、参加者がユニークなGenAIの攻撃対象領域を理解し、セキュリティ原則を適応させ、エージェント型AIの保護、リスク管理、安全なアーキテクチャの構築に関する実践的なガイダンスを得ることを目的としていた 。これは、GenAIを扱う開発者にとって基本的なことである。

「基盤」と「原則」への重点は、業界が対症療法的なパッチ適用を超えて、AIシステムの積極的かつアーキテクチャ的なセキュリティをゼロから確立する必要性を認識していることを示唆している。

Kelley氏のセッション や「Generative AI Security」 のような書籍は、「基盤」「原則」「セキュリティの組み込み」に焦点を当てている。これは、アドホックなセキュリティ対策がGenAIやエージェント型AIの複雑さに対して不十分であるという認識を示している。これは、より体系的で設計レベルのアプローチを求めるものである。

コミュニティの力

主要なオープンソース財団や標準化団体の強力な存在感と協力は、AIセキュリティはサイロでは解決できないというコンセンサスを示している。

共有ツール、知識、ベストプラクティスを開発するためには、集団的で透明性のあるコミュニティ主導の取り組みが必要である。OWASP、OASIS、Linux Foundation、OpenSSF、CNCFの関与は、AIセキュリティ課題の規模と複雑さが、独自のクローズドなソリューションよりもオープンな協力を必要とすることを広範に認識していることを示している。

これは、テクノロジーの他の分野における成功したオープンソース運動を反映している。

社会的および組織的適応
「The Machines Are Learning, But Are We?」

「機械は学習しているが、我々はどうなのか？」という問いは、AI時代の核心的な不安を要約している。それは、指数関数的な速度で学習し進化するテクノロジーを理解し、管理し、倫理的に導くための人間（個人、組織、社会）の能力に関するものである。

これは、技術的なセキュリティを超えて、倫理的枠組み、法的構造、一般の理解を含む。AIの急速な進歩（機械学習）は当然のことである。セッションのタイトルは、人間の対応に直接疑問を投げかけている。

これは、対応する人間と社会の学習と適応なしには、技術的な解決策だけでは不十分であることを意味する。


広がる危機感　RSAC 2025
RSAC2025 レポート
AIとNHIによる脅威の拡大

RSAC 2025

2025年のRSA Conference (RSAC 2025) は、4月28日から5月1日まで、米国カリフォルニア州サンフランシスコのモスコーンセンターで開催されました 。今回で34回目を迎えるこの年次カンファレンスは、サイバーセキュリティ業界における世界最大級のイベントとしての地位を確立しています。

特筆すべきは、参加者数が過去最高の約44,000人に達したことです。この数字には、730名を超える講演者、650社以上の出展企業、そして400名以上のメディア関係者が含まれており、一部情報では講演者700名以上、セッション数450以上とも伝えられています。このような大規模な参加者数は、世界的な不確実性が続く中でも、サイバーセキュリティの重要性がますます高まっていること、そしてRSACが業界随一の国際的会合としての独自の地位を築いていることを明確に示しています。

RSAC 2025のテーマは「Many Voices. One Community. (多くの声、一つのコミュニティ。)」でした 。このテーマは、13世紀の詩人ルーミーの言葉「ランプはそれぞれ違っていても、光は同じ」に触発されたものであり、共有された脅威に対する統一された力としての協力の強さを称賛するものです。

このテーマは単なるキャッチフレーズではなく、ますます複雑化し相互接続する脅威に対処するために不可欠とされる、より包括的で協力的なグローバルサイバーセキュリティエコシステムを育成するというRSACの戦略的方向性を反映しています。

実際に、RSACをグローバルプラットフォームとして位置づける動きはテーマ設定にも現れており、その役割拡大は時宜を得たものと評価されています。基調講演者の多様性や、オープンソースソリューション、官民連携 に関する議論も、「Many Voices」の側面と協調の重要性を裏付けています。

主要キーノート・スピーカー

登壇者には、Craigslist創設者のクレイグ・ニューマーク氏、英国AI安全研究所CTOのジェイド・ルン氏、大統領特別補佐官兼国家安全保障会議サイバー担当シニアディレクターのアレクセイ・ブラゼル氏といった著名人に加え、多数のサイバーセキュリティ専門家、イノベーター、ゲストスピーカーが含まれていました。

技術リーダーからは、シスコのエグゼクティブバイスプレジデント兼最高製品責任者であるジーツ・パテル氏、マイクロソフトセキュリティ担当コーポレートバイスプレジデントのヴァス・ジャッカル氏、CrowdStrikeのCEO兼創設者であるジョージ・カーツ氏、Google Cloudのサンドラ・ジョイス氏、NVIDIAのダニエル・ローラー氏などが登壇しました。

政府・政策分野からは、元CISA長官のジェン・イースタリー氏、元米国国家サイバー長官のクリス・イングリス氏、元NSAサイバーセキュリティ長官のロブ・ジョイス氏などが名を連ねました。

研究者・学術分野からは、ウィットフィールド・ディフィー氏らによる暗号研究者パネルや、ブルース・シュナイアー氏、SANS Instituteのパネルが注目を集めました。

これらのキーノートスピーカーの顔ぶれは、技術幹部、政府高官、著名な研究者、さらにはエンターテイメント界の人物まで多岐にわたり、サイバーセキュリティの影響が純粋な技術領域を超えて、政策、ビジネス戦略、社会全体の関心事へと拡大していることを反映しています。

トラック構成と主要技術領域

RSAC 2025では、29のトラックが設けられ、多岐にわたるサイバーセキュリティの領域が網羅されました。

公式トラックリストには、「AIと機械学習」「分析と運用」「ビジネスとリーダーシップ」「クラウド、DevOps、アプリケーションセキュリティ」「暗号、プライバシー、データ保護」「グローバルな視点」「人的要素」「ID、アクセス管理、プライバシー」「インフラストラクチャと運用」「最新のセキュリティトレンド」「リスク管理とガバナンス」「戦略とアーキテクチャ」「脅威インテリジェンスとインシデント対応」などが含まれていました。これらに加え、「サイバー人材とリーダーシップ開発」 や「ハッカーと脅威」 といったトラックも言及されています。

このトラック構成は、AI、暗号、クラウドセキュリティといった高度な技術分野だけでなく、ビジネスリーダーシップ、人的要因、グローバルポリシー、人材育成といった重要な非技術的側面も網羅する、サイバーセキュリティへの包括的なアプローチを反映しています。

「AIと機械学習」「ID、アクセス管理、プライバシー」「クラウド、DevOps、アプリケーションセキュリティ」といったトラックは、カンファレンス全体を通じて観察された主要テーマと一致しており、これらの分野が業界内で特に注目され、開発が進められている領域であることを示唆しています。

例えば、IDセキュリティ関連のセッションが332、AI関連のセッションが159も予定されていたことからも、これらの分野への注力の度合いがうかがえます。

RSAC 2025 主要テーマと技術トレンド

RSAC 2025では、サイバーセキュリティの未来を形作るいくつかの重要なテーマと技術トレンドが浮き彫りになりました。

特に、人工知能（AI）の進化、非人間アイデンティティ（NHI）の急増、そしてそれらを取り巻くセキュリティ課題が中心的な議題となりました。

エージェント型AIの台頭とその影響

RSAC 2025における最も支配的な話題は、サイバーセキュリティソリューションへのAIの統合であり、2,800件以上のセッション応募のうち40%がAI関連のトピックに焦点を当てていました。

特筆すべきは、議論が生成AIから完全に自律的な「エージェント型AI」システムへと移行したことです。エージェント型AIは、人間のアナリストと協力して、目標を達成するためにタスクを独立して特定し、推論し、動的に実行できると定義されています。

エージェント型AIの出現は、サイバーセキュリティにおけるパラダイムシフトを意味し、AIを分析ツールから自律的な運用エンティティへと移行させます。

これは、前例のない防御能力を提供する一方で、同時に斬新で複雑な攻撃ベクトルとガバナンスの課題を生み出すという、深刻なデュアルユースの示唆を持っています。

主要ベンダーによるエージェント型AIの急速な開発・推進と、その関連リスクや運用統合の課題に対する広範なセキュリティコミュニティの理解・準備の遅れとの間には、明白な緊張関係が存在します。これは、技術の可能性と、その展開・管理の現実との間に「イノベーションと準備のギャップ」があることを示唆しています。

AIとセキュリティについての主要なレポート

Knosticの共同創設者であるSunil Yu氏は、この進化をOODAループ（Observe, Orient, Decide, Act）モデルを用いて説明し、エージェント型AIが意思決定を含む4つのフェーズすべてを実行できるようになったと指摘しました。

マイクロソフト（Security Copilotエージェント）、Google（Geminiセキュリティオファリング、SecOpsにおけるAIエージェントのビジョン）、シスコの幹部らは、自社のAI搭載セキュリティツールが脅威の検出と対応をどのように変革しているかを披露しました。

しかし、シスコのジーツ・パテル氏は、自律型AIエージェントが「これまでに見たことのないまったく新しいクラスのリスク」をもたらすと警告しました。意図の誤解、独立して動作するシステムの制御、予測不可能性、悪用可能性、透明性、監視、明確なガードレールの必要性などが懸念事項として挙げられました。

Forresterのアナリストは、セッションではエージェント型AIの出力管理に必要な人的課題やスキルが見過ごされがちであると指摘しています。

非人間アイデンティティ (NHI) セキュリティ

RSAC 2025におけるもう一つの革新的な議論は、エンタープライズ環境における非人間アイデンティティ（NHI）の「爆発的増加」に集中しました。

これには、APIキー、サービスアカウント、OAuthトークン、マシンID、AIエージェントなどが含まれます。マシン間の通信が人間のデジタルインタラクションを上回るようになり、従来のアイデンティティ境界は時代遅れになっています。

クラウド導入、マイクロサービス、IoT、そして現在のAIエージェントによって加速されるNHIの急増は、広大で、ほとんど管理されておらず、非常に脆弱な攻撃対象領域を生み出しています。

これはアイデンティティランドスケープにおける根本的な変化であり、アイデンティティガバナンスとアクセス管理のための新しいパラダイムを必要としています。

市場はこのNHIの課題に急速に対応しており、複数のベンダーが専門的なソリューションを発表しています。

これは、NHIセキュリティがニッチな懸念事項から、リスクとコンプライアンスの両方の圧力によって推進される主流のサイバーセキュリティ優先事項へと移行していることを示しています。

NHIセキュリティについての主要なレポート

Transmit Securityの最高情報責任者であるDave Mahdi氏は、アイデンティティ、特にNHIがサイバーセキュリティにおける「重大な死角」であり、ほとんどの侵害がIAMの障害や悪用に起因すると強調しました。驚くべきことに、ほとんどの組織はNHIの20%未満しか追跡していません。

State of Non-Human Identity Securityレポートによると、5組織に1組織がNHI関連のセキュリティインシデントを経験しています。

Oasis Securityはこの課題に対応するため、NHIの作成、ガバナンス、セキュリティを自動化するNHI Provisioningを発表しました。

Anetacは、人間と非人間のアイデンティティ脆弱性を統合管理するプラットフォームを拡張しました。

RSAは、人間と非人間のアイデンティティに対応する新しいアイデンティティセキュリティポスチャ管理（ISPM）機能を発表しました。

Teleportは、人間と非人間のアイデンティティがクラウドリソースと安全に対話できるようにするソリューションを提示しました。

RSAC Innovation Sandbox Contest

RSACの恒例行事であるInnovation Sandboxコンテストは、今年で20周年を迎えました。

このコンテストは、サイバーセキュリティ業界を変革する可能性を秘めた画期的な技術を持つスタートアップ企業にとって、主要なプラットフォームとしての地位を確立しています。

過去19年間で、上位10社のファイナリストは合計で90件以上の買収と164億ドル以上の投資を受けてきました。

特筆すべきは、RSAC 2025で新たに導入された投資プログラムにより、上位10社のファイナリストそれぞれに500万ドルの投資が行われたことです。

また、コンテストへの応募件数は2024年から40%以上増加し、200社以上のスタートアップが応募しました。P

ProjectDiscoveryのCOOであるAndy Cao氏は、「受賞は、セキュリティ分野でオープンソースが可能であることの認識を示すものです。私たちはセキュリティを民主化することに興奮しています」と述べています。

ProjectDiscoveryの優勝

2025年のコンテストでは、ProjectDiscoveryが「最も革新的なスタートアップ2025」に選ばれました。

同社は、セキュリティチームが脆弱性を迅速に発見・修正するためのオープンソースツールを提供しています。

Nucleiを搭載したプラットフォームは、攻撃対象領域の監視と脆弱性管理を自動化し、組織が現実世界の脅威に先んじるのを支援します 8。Nucleiテンプレートには、GCP設定チェックなどが含まれています。ProjectDiscoveryは、従来の脆弱性スキャナーよりも迅速に重大な脆弱性に対応できるとされています。

オープンソースとコミュニティの重要性

ProjectDiscoveryの勝利とファイナリストへの新たな直接投資は、脆弱性管理のような重要なサイバーセキュリティ課題に取り組むためのオープンソースでコミュニティ主導のアプローチを業界がますます高く評価していることを強く示唆しています。

これはセキュリティツールの民主化とイノベーションの加速につながります。Innovation Sandboxコンテストへの応募件数の大幅な増加とコンテストの実績は、活気あるスタートアップエコシステムを示しており、同コンテストが新興企業にとって強力な触媒となり、将来の市場の方向性を形作る可能性があることを示唆しています。

第二部

ソフトウェア開発とAI Agent

第二部： ソフトウェア開発とAI Agent

ソフトウェア開発へのAI Agent導入の動向

Multi AI Agent プログラミングの基本
A2A: Travel Agent サンプル
ADK: コード開発パイプライン・サンプル

System Message instruction とセキュリティ


ソフトウェア開発とAI Agent
ソフトウェア開発へのAI Agent導入の動向

ソフトウェア開発へのAI Agent導入の動向

先のセッションでは、AIの能力の飛躍的な発展がさまざまな分野で進む中で、AIが持つcode生成能力をソフトウェア開発に利用しようという関心が高まっていることを見てきました。

その関心のフォーカスは、個別のプログラムの個別的な開発でのAIの利用にとどまらず、もっと広いソフトウェア開発サイクルそのものでのAIの利用の可能性に向かっています。

AIをソフトウェア開発に利用しようという流れ、あるいは、AIそのものを開発しようという流れの中で、一つの変化が起きました。それは、AIの働きを「AI エージェント」として捉える見方が急速に広がったことです。

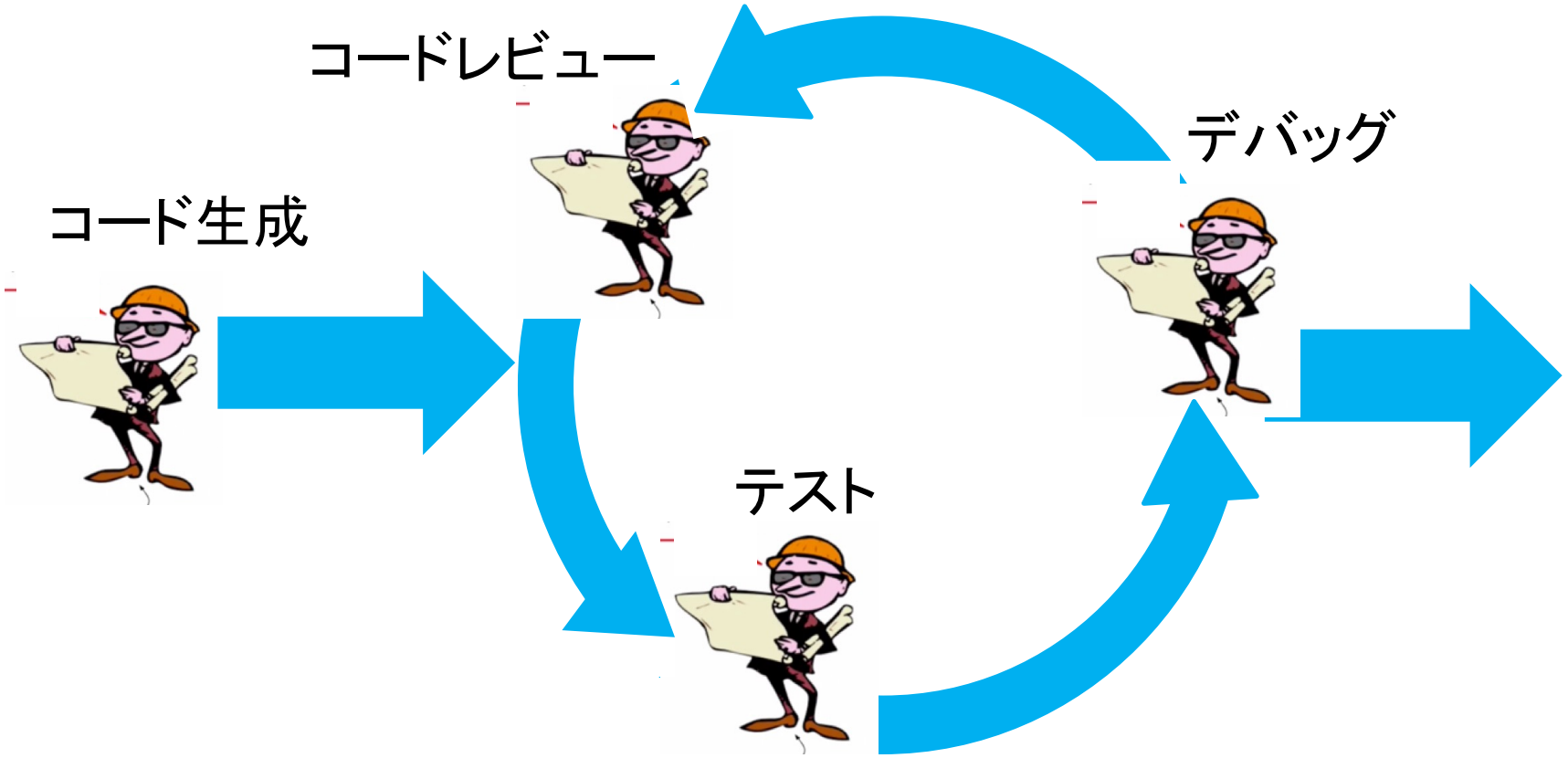
AIエージェントの登場と進化

これまで、AIの利用はコード補完のような特定のタスクに限定されていましたが、現在ではより広範なソフトウェア開発ライフサイクル（SDLC）全体に関与し、自律的にタスクを実行する「AIエージェント」へと進化しつつあります。

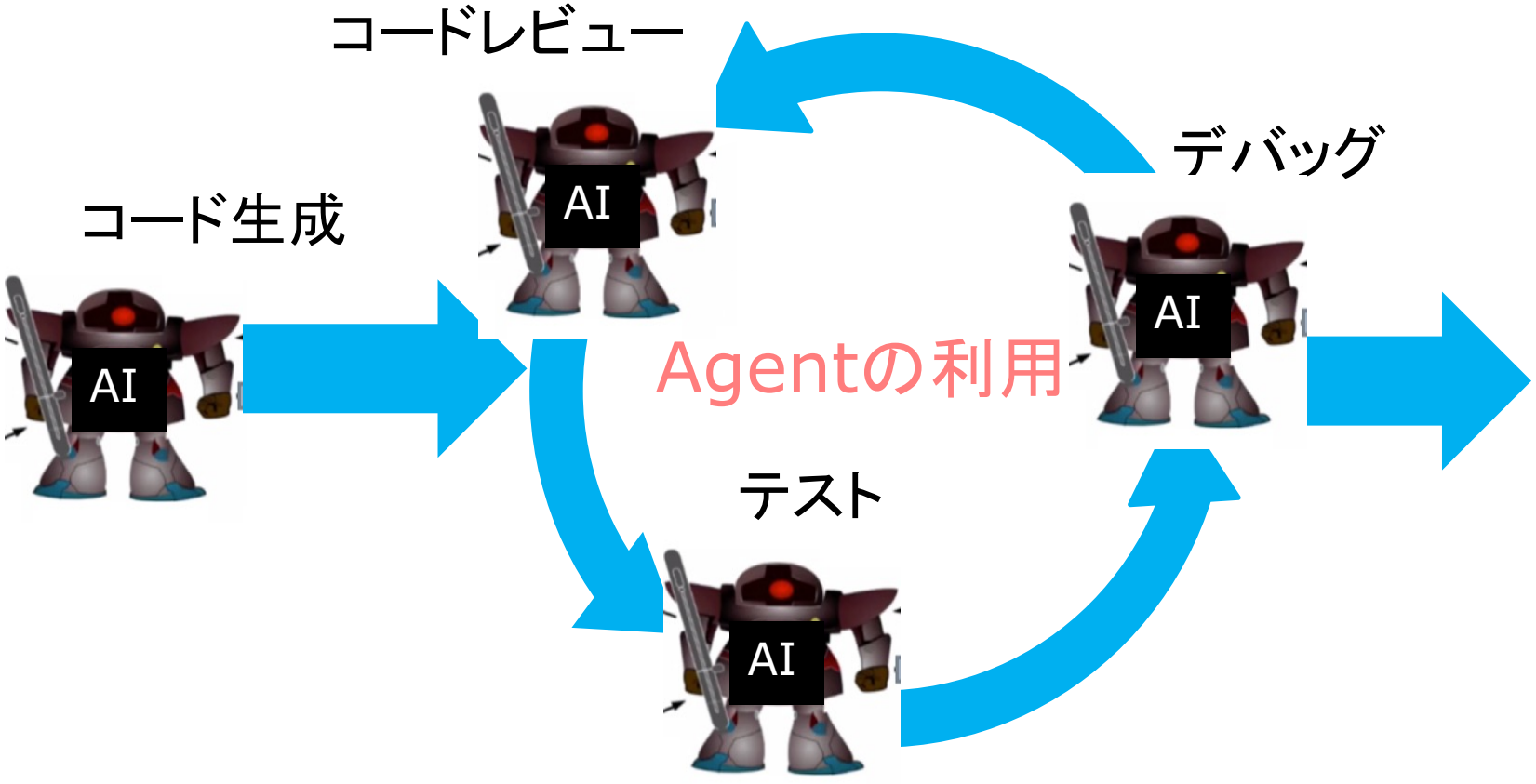
これらのAIエージェントは、単なるコード生成ツールを超え、設計、テスト、デバッグ、コードレビューといった複雑なエンジニアリングタスクを支援、あるいは自動化する可能性を秘めています。

このセッションでは、ソフトウェア開発におけるAIエージェントの導入の現状を、特にコード補完、テスト、デバッグ、コードレビューの各領域に焦点を当てて見ていきたいと思います。

ソフトウェア開発サイクルの単純化されたモデル



人間が果たしている役割をAgentに置き換える



AIによるコード補完の現状と
エージェント化の動向

AIによるコード補完は、開発者の生産性向上に大きく貢献する技術として広く普及しています。

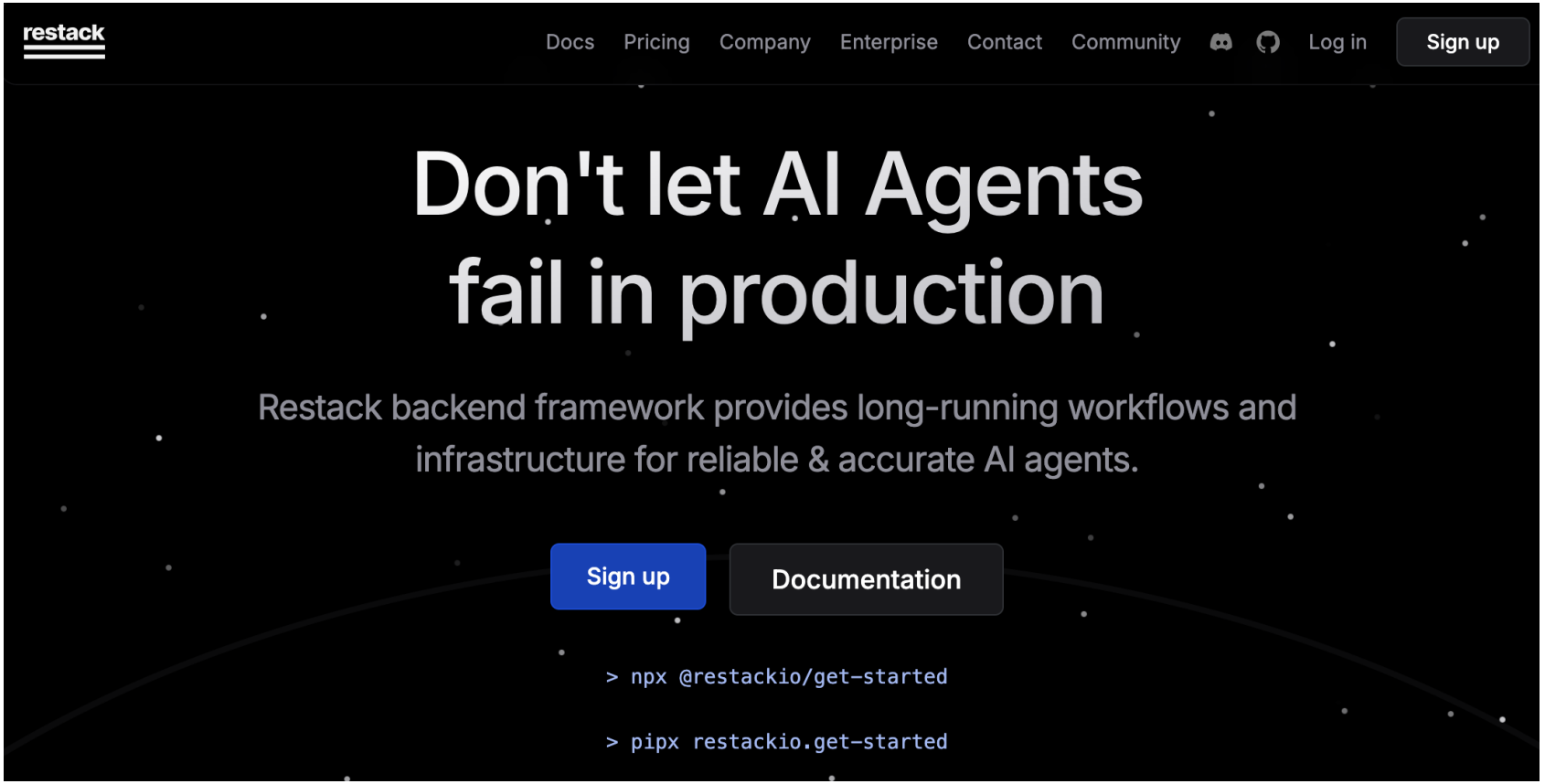
主要なAIコード補完サービスと エージェント化の動向

代表的なAIコード補完サービスには、GitHub CopilotやAmazon CodeWhispererなどがあります。これらのサービスは、単なる補完機能に留まらず、より高度なエージェント的振る舞いを見せ始めています。

- **GitHub Copilot**

OpenAI Codexを基盤とし、膨大な公開GitHubリポジトリで訓練されています。入力されたコメントや関数記述を解釈し、単純なコード行から複雑な関数まで提案可能です 。ユーザーのフィードバックを通じて学習し、提案の質を向上させます。

GitHub Copilot



https://www.restack.io/

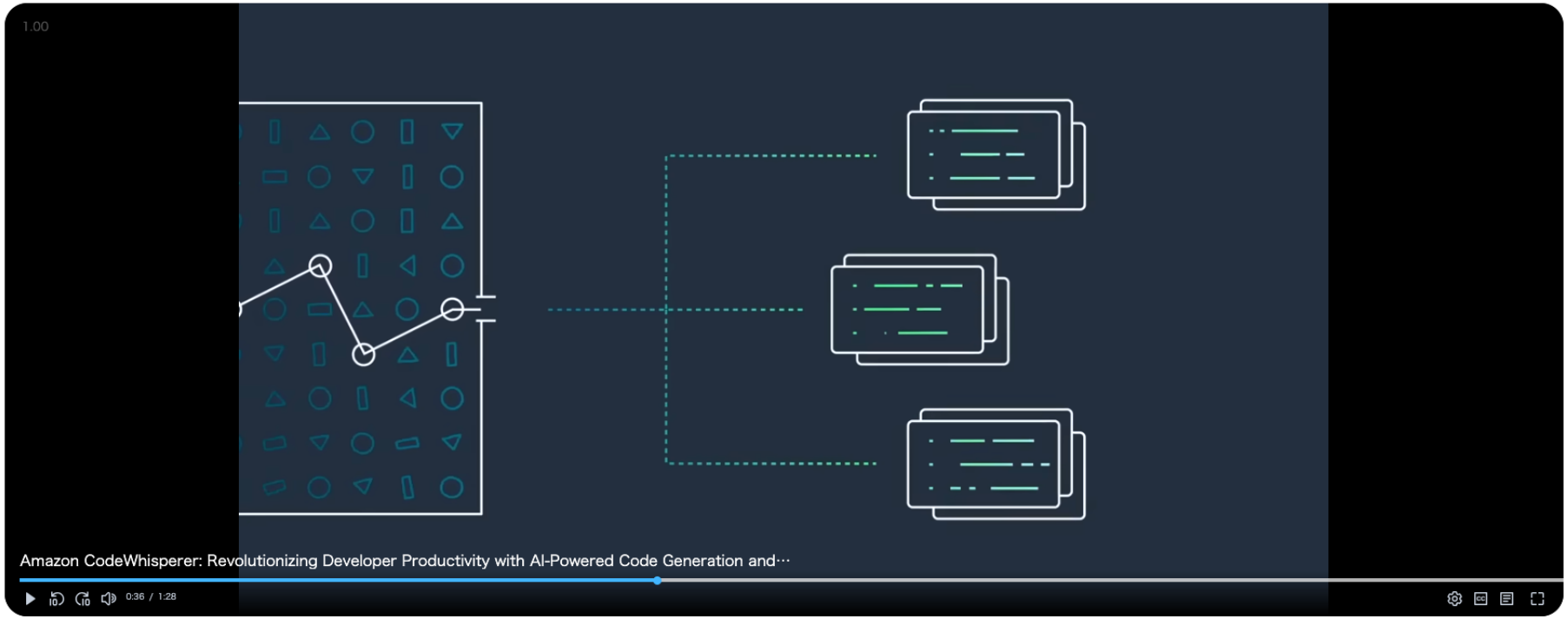
- **Amazon CodeWhisperer**

機械学習を活用し、文脈に基づいたコード推薦を行うことで開発者の生産性を向上させます。Amazonのコードやオープンソースコードを含む多様なデータソースで訓練されており、コメントからコードスニペットや関数全体を生成できます。

- **Cursor**

AIファーストのコードエディタであり、「エージェントモード」と呼ばれる機能を搭載しています。このモードでは、タスクをエンドツーエンドで完了させることができ、カスタム検索モデルを用いてコードベース全体を理解し、ターミナルコマンドの自動作成・実行（デフォルトでは確認を求める）、lintエラーの自動検出・修正といった自律的な動作が可能です。

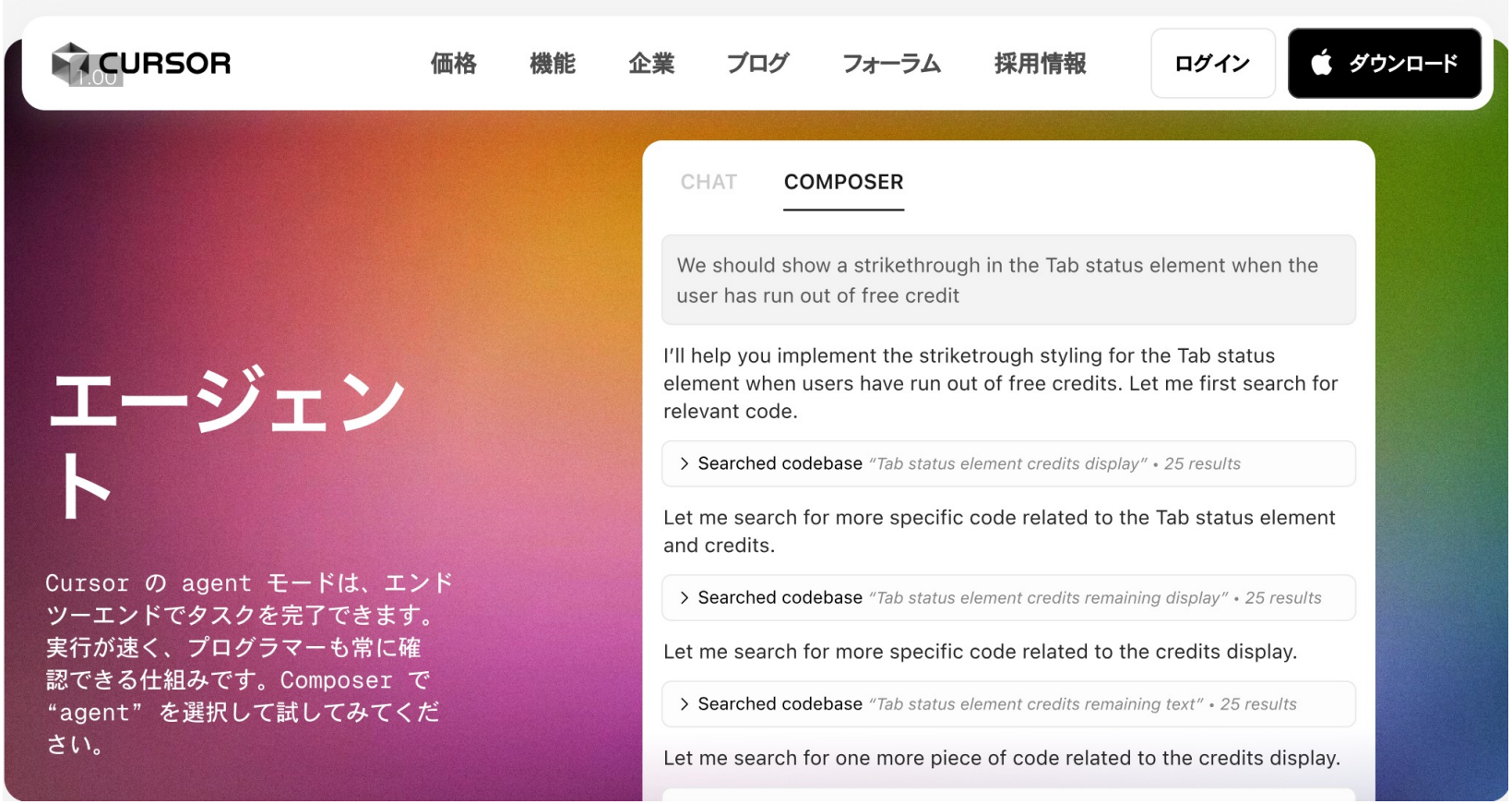
Amazon CodeWhisperer



Amazon CodeWhisperer: Revolutionizing Developer Productivity with AI-Powered Code Generation and Recommendations

https://aws.amazon.com/jp/awstv/watch/50a3d784916/

Cursor



https://www.cursor.com/ja/features

オープンソースのAIコーディング・エージェント

商用サービスに加え、オープンソースのAIコーディング・エージェントも活発に開発されています。

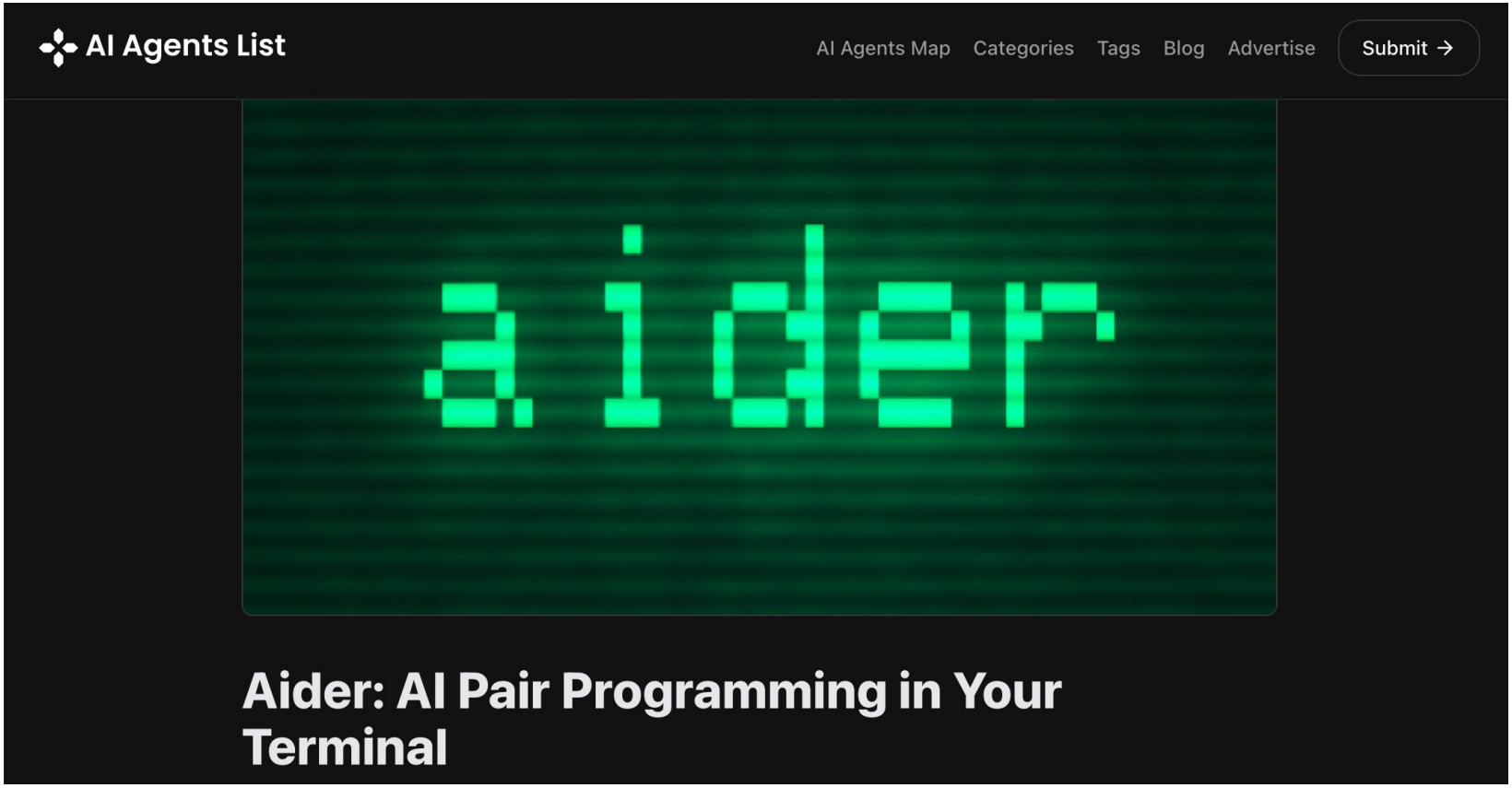
- **Aider**

ターミナルで動作するAIペアプログラミングツールで、GPTモデルなどと連携し、ローカルのGitリポジトリ内のファイルを編集します。

- **Roo-Code (旧 Cline)**

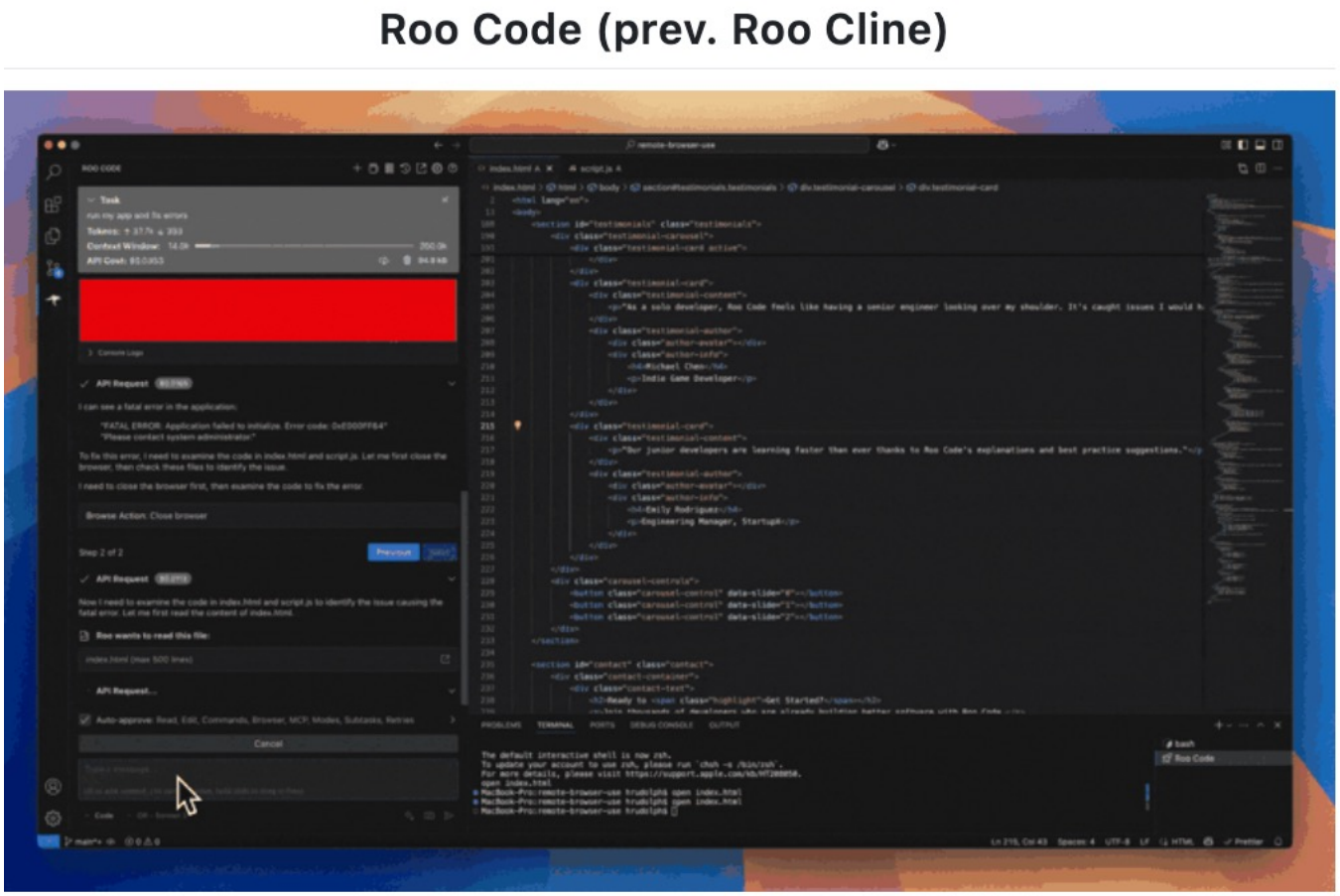
VS Code向けのオープンソース自律型コーディングアシスタントです 12。計画モードと実行モードを持ち、プロジェクト全体を読み込み、ファイル内を検索し、ターミナルコマンドを実行できます。OpenAI GPT-4やローカルモデルなど、指定されたAPI経由でLLMに接続します。

Aider



https://aiagentslist.com/agent/aider

Roo Code (prev. Roo Cline)



https://github.com/RooVetGit/Roo-Code/blob/main/README.md

- **OpenHands (旧 OpenDevin)**

人間の開発者と同様に、コードの編集、コマンドライン操作、ウェブブラウジング、API呼び出しなど、あらゆるタスクを実行できるAIソフトウェア開発エージェントプラットフォームです。チャットパネル、VS Code統合ワークスペース、Jupyterノートブックサポート、アプリビューア、ブラウザ、ターミナルといった包括的なインターフェースを備えています。

これらのオープンソースプロジェクトは、特定のLLMに縛られず、ローカル環境での実行やプライバシー重視の運用を可能にする点で重要です。開発者はこれらのフレームワークを基盤として、独自のAIエージェントを構築したり、既存の機能を拡張したりすることができます。

OpenHands (旧 OpenDevin)

OpenHands: An Open Platform for AI Software Developers as Generalist Agents

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin et al. *(4 additional authors not shown)*

Published: 23 Jan 2025, Last Modified: 19 Feb 2025 ICLR 2025 Poster Everyone Revisions BibTeX CC BY 4.0

Keywords: AI agents, evaluation, infrastructure, benchmark

Abstract:

Software is one of the most powerful tools that we humans have at our disposal; it allows a skilled programmer to interact with the world in complex and profound ways. At the same time, thanks to improvements in large language models (LLMs), there has also been a rapid development in AI agents that interact with and effect change in their surrounding environments. In this paper, we introduce OpenHands, a platform for the development of powerful and flexible AI agents that interact with the world in similar ways to a human developer: by writing code, interacting with a command line, and browsing the web. We describe how the platform allows for the implementation of new agents, utilization of various LLMs, safe interaction with sandboxed environments for code execution, and incorporation of evaluation benchmarks. Based on our currently incorporated benchmarks, we perform an evaluation of agents over 13 challenging tasks, including software engineering (e.g., SWE-Bench) and web browsing (e.g., WebArena), amongst others. Released under the permissive MIT license, OpenHands is a community project spanning academia and industry with more than 2K contributions from over 186 contributors in less than six months of development, and will improve going forward.

Primary Area: infrastructure, software libraries, hardware, systems, etc.

Code Of Ethics: I acknowledge that I and all co-authors of this work have read and commit to adhering to the ICLR Code of Ethics.

Submission Guidelines: I certify that this submission complies with the submission instructions as described on https://iclr.cc/Conferences/2025/AuthorGuide.

Anonymous Url: I certify that there is no URL (e.g., github page) that could be used to find authors' identity.

https://openreview.net/forum?id=OJd3ayDDoF

「AIソフトウェアエンジニア」 Devinの登場

「世界初のAIソフトウェアエンジニア」として注目を集めているのが、Cognition AIによって開発されたDevinです。Devinは、自然言語の指示に基づいて、計画、コーディング、デバッグ、デプロイといったソフトウェア開発タスクを自律的に実行できるとされています。シェル、コードエディタ、ブラウザを含むサンドボックス環境で動作し、Slack、Linear、GitHubといった開発ツールと連携します。

2025年4月にはDevin 2.0が発表され、インタラクティブな計画機能、エージェントネイティブなIDEエクスペリエンス、コードベースを探索するDevin Search、リポジトリの自動インデックスとドキュメント生成を行うDevin Wikiといった新機能が導入されました。

Devinの技術アーキテクチャは、Cognitionのクラウド上で動作する「頭脳」と、ユーザー環境（VPCまたはSaaS）で実行されるワークスペース（シェル、ブラウザ、コードエディタを含む）で構成されます。

初期の評価では、Devinが一部のタスクで期待された性能を発揮できなかったり、複雑な問題で苦戦したりするケースが報告されています。Answer.AIによる20件のタスク評価では、成功したのは3件のみであったと報告されています。一方で、特定のタスク（NotionデータベースからGoogle Sheetsへのデータ取り込みなど）では有能さを示した例もあります。2025年初頭のSWE-benchにおける未検証の実行では13.86%の解決率を示したとされますが、その後の検証済みベンチマークスコアは公開されていません。Devin 2.0の登場により、これらの課題への対応と性能向上が期待されています。

Devin AI: Redefining Software Development with the World's First AI Engineer

By Amit Kumar March 18, 2024

https://www.hashstudioz.com/blog/devin-ai-redefining-software-development/

AIエージェントによるテストの自動化

ソフトウェア開発ライフサイクル（SDLC）において、テストは品質を担保するために不可欠なプロセスです。AIエージェントは、このテストプロセス全体（計画、生成、実行、分析、改善）を自動化し、効率化する可能性を秘めています。

オープンソースAIテスト・エージェント

オープンソースコミュニティからも、テスト自動化に特化したAIエージェントが登場しています。

- **AI Testing Agent (furudo-erika)**

このプロジェクトは、APIテストの自動化を目的としたAIエージェントです。LangChainフレームワークをベースとしたチャットボット（agent.py）を通じてユーザーと対話し、LLM（OpenRouter経由）と連携して以下のタスクを実行します。

- テスト計画の生成
- テストコードの生成
- テストの実行
- フィードバックによる改善

AI Testing Agent: Open Source AI Agent for Software Testing

This repository contains an AI Testing Agent that interacts with an LLM (via OpenRouter) to automatically:

1. Generate a Test Plan for your API
2. Generate Python test code (using pytest) based on that plan
3. Run the generated tests
4. Accept user feedback to refine or extend the test suite

By default, the AI agent assumes your API has some specific REST routes (e.g., /api/endpoint), but you can customize everything using prompts.

Contents

https://github.com/furudo-erika/ai-testing-agent

- **Keploy**

Keployは、eBPF（Extended Berkeley Packet Filter）技術を活用して、アプリケーションのコード変更なしにAPIコールと依存関係（データベース呼び出しなど）をキャプチャするオープンソースのテスト自動化プラットフォームです。

- アーキテクチャ: Keployは、アプリケーションと外部依存関係の間のミドルウェアとして機能し、APIリクエストとレスポンスを傍受します。
- テスト生成: 記録モードとテストモード
- AIの役割 (ut-gen): Keployは、Meta社のLLM研究論文の実装に基づいたユニットテストジェネレータ（ut-gen）を導入しています63。この機能は、コードのセマンティクスを理解し、意味のあるユニットテストを自動生成することを目指しています。

Keploy

https://keploy.io/blog/community/executing-ebpf-in-github-actions

商用AIテストツールにおけるエージェント的機能

TestRigorやRainforest QAのような商用ツールも、AIを活用してテスト作成とメンテナンスを支援しています。これらのツールは、自然言語でのテスト記述や、UIの視覚的解釈に基づくテスト実行など、エージェント的な側面を持っています。特にTestRigorは、平易な英語のプロンプトで自動テストを作成できる機能を提供しています。

AIエージェントによるテスト自動化は、テスト作成の労力削減、テストカバレッジの向上、メンテナンスコストの削減といった多くの利点をもたらします。オープンソースと商用の両方で、より自律的でインテリジェントなテストエージェントの開発が進んでおり、ソフトウェアの品質保証プロセスに大きな変革をもたらすことが期待されます。

AIエージェントによるデバッグの自動化

デバッグはソフトウェア開発において時間と労力を要する作業ですが、AIエージェントの活用により、このプロセスも自動化・効率化が進んでいます。AIデバッグエージェントは、コードパターンを分析し、過去のデータから学習し、予測分析を用いて潜在的な問題を未然に警告することができます 。リアルタイムでのバグ検出や、バグの自動修正提案・実装も可能です。

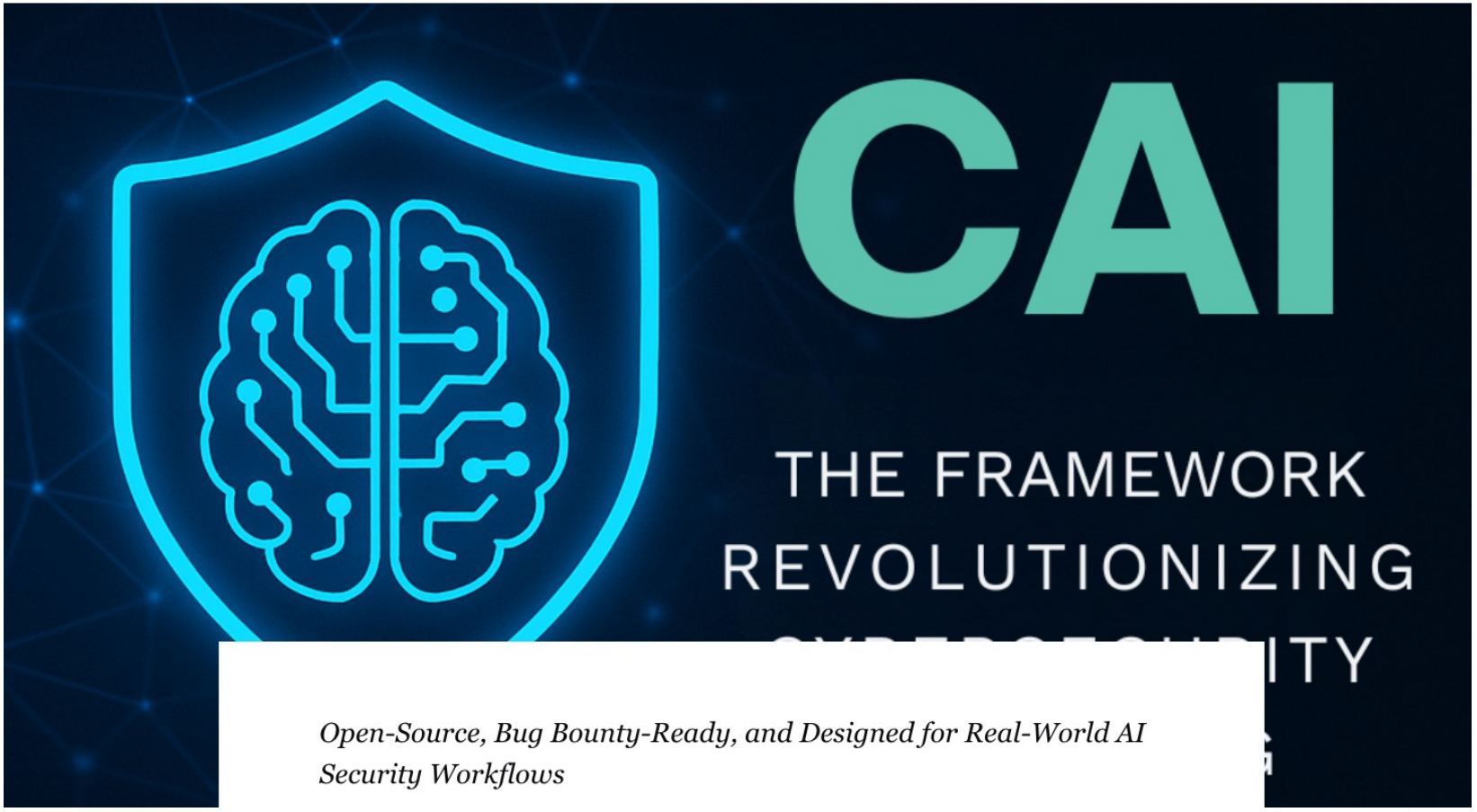
オープンソースのAIデバッグ・エージェント

- **Cybersecurity AI (CAI)**

Alias Roboticsによって開発されたCAIは、特にサイバーセキュリティのユースケースに焦点を当てたオープンソースフレームワークです。

  - エージェント: LLMによって駆動され、ReAct (Reasoning and Action) パターンに従って推論と実行を行います。
  - ハンドオフ: 特定のタスクを専門のエージェントに委任するメカニズムです。
  - パターン: 複数のエージェントが協調してタスクを達成するための構造化された設計（例：Swarm、Hierarchical）を定義します 。
  - Human-In-The-Loop (HITL): 重要な意思決定ポイントで人間の介入を許容する設計が組み込まれています。

CAI: The framework revolutionizing cybersecurity AI testing



https://news.aliasrobotics.com/cai-the-framework-revolutionizing-cybersecurity-ai-testing/

- **SWE-agent**

プリンストン大学の研究者らによって開発されたSWE-agentは、GitHubのIssueを読み込み、関連するコードリポジトリ内で自律的にバグ修正を試みるシステムです。このエージェントの核となるのが、Agent-Computer Interface (ACI) です。

  - ACIアーキテクチャ: ACIは、LLMエージェントがソフトウェアエンジニアリングタスクを遂行するためにコンピュータと対話するためのカスタムインターフェースです。
  - タスク実行フロー: SWE-agentは、GitHubのIssueから問題文を抽出し、ReActフレームワークに基づいて思考とコマンド生成を繰り返します。
  - 呼び出し方法: run.pyスクリプトを使用して起動し、モデル名や問題文（GitHub IssueのURLなど）を引数として渡します。

SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering

John Yang* **Carlos E. Jimenez*** **Alexander Wettig** **Kilian Lieret**
Shunyu Yao **Karthik Narasimhan** **Ofir Press**

Princeton Language and Intelligence, Princeton University

Abstract

Language model (LM) agents are increasingly being used to automate complicated tasks in digital environments. Just as humans benefit from powerful software applications, such as integrated development environments, for complex tasks like software engineering, we posit that LM agents represent a new category of end users with their own needs and abilities, and would benefit from specially-built interfaces to the software they use. We investigate how interface design affects the performance of language model agents. As a result of this exploration, we introduce SWE-agent: a system that facilitates LM agents to autonomously use computers to

https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf

その他のAIデバッグツールと研究

GitHub CopilotやSnykCodeといったリアルタイムツールは、開発者がコードを書いている最中にバグを検出し、IDE内で修正を支援します。

Qodo AI（旧CodiumAI）は、テスト生成とコードの振る舞い検証を自動化することで、デバッグプロセスを支援します。また、

MarsCode Agentのような研究プロジェクトでは、マルチエージェントコラボレーションや構造化されたアプローチを用いて、バグの再現、根本原因分析、パッチ生成、検証といったデバッグの各段階を自動化する試みがなされています。これらのエージェントは、動的デバッグと静的解析を組み合わせ、コード知識グラフを活用して関連コードを効率的に検索し、修正を行います。

MarsCode Agent: AI-native Automated Bug Fixing

Yizhou Liu[1*] **Pengfei Gao**[1*] **Xinchen Wang**[2†] **Jie Liu**[1]
Yexuan Shi[1] **Zhao Zhang**[1] **Chao Peng**[1‡]
[1]ByteDance [2]Harbin Institute of Technology, Shenzhen
pengchao.x@bytedance.com
https://se-research.bytedance.com/

Abstract

Recent advances in large language models (LLMs) have shown significant potential to automate various software development tasks, including code completion, test generation, and bug fixing. However, the application of LLMs for automated bug fixing remains challenging due to the complexity and diversity of real-world software systems. In this paper, we introduce `MarsCode Agent` , a novel framework that leverages LLMs to automatically identify and repair bugs in software code. `MarsCode Agent` combines the power of LLMs with advanced code analysis techniques to accurately localize faults and generate patches. Our approach follows a systematic process of planning, bug reproduction, fault localization, candidate patch generation, and validation to ensure high-quality bug fixes. We evaluated `MarsCode Agent` on SWE-bench, a comprehensive benchmark of real-world software projects, and our results show that `MarsCode Agent` achieves a high success rate in bug fixing compared to most of the existing automated approaches.

https://arxiv.org/pdf/2409.00899

AIエージェントによるコードレビューの自動化

コードレビューは、ソフトウェアの品質維持、知識共有、チームのコーディング標準の遵守を目的とした重要なプラクティスです。しかし、手動でのレビューは時間と労力を要し、特に大規模なプロジェクトや迅速な開発サイクルにおいてはボトルネックとなることがあります。

オープンソースAIコードレビュー・エージェント

- **CodiumAI PR-Agent (Qodo Merge open-source)**

このツールは、プルリクエスト（PR）の分析とフィードバックを自動化することを目的としたAI搭載エージェントです。GitHub、GitLab、Bitbucketなどの複数のGitプロバイダーをサポートし、CLI、GitHub Action、GitHub Appなど多様な方法で利用可能です。

- アーキテクチャと機能: PR-Agentは、PRの差分をLLMが処理しやすい形式に変換する「PR圧縮戦略」と、モジュール式でカスタマイズ可能なツールを実現する「JSONプロンプティング戦略」を特徴としています。

- LLMとの対話: PRのコンテキスト（コード差分、説明など）と実行されたコマンドに基づいてプロンプトを生成し、GPT、Claude、Deepseekなどのサポートされ ているLLMに送信します。LLMからの応答は、PRへのコメントとしてフィードバックされます。
- 呼び出し方法: GitHubのPRコメント内で@CodiumAI-Agent /reviewのようにメンションすることで、特定のツールを呼び出すことができます。

Qodo Mergeというホスト版も提供されており、より高度な機能やプライバシー、サポートが利用可能です。CodiumAI PR-Agentは、実用的な利用に重点を置き、迅速かつ手頃な価格で価値を提供することを目指しています。

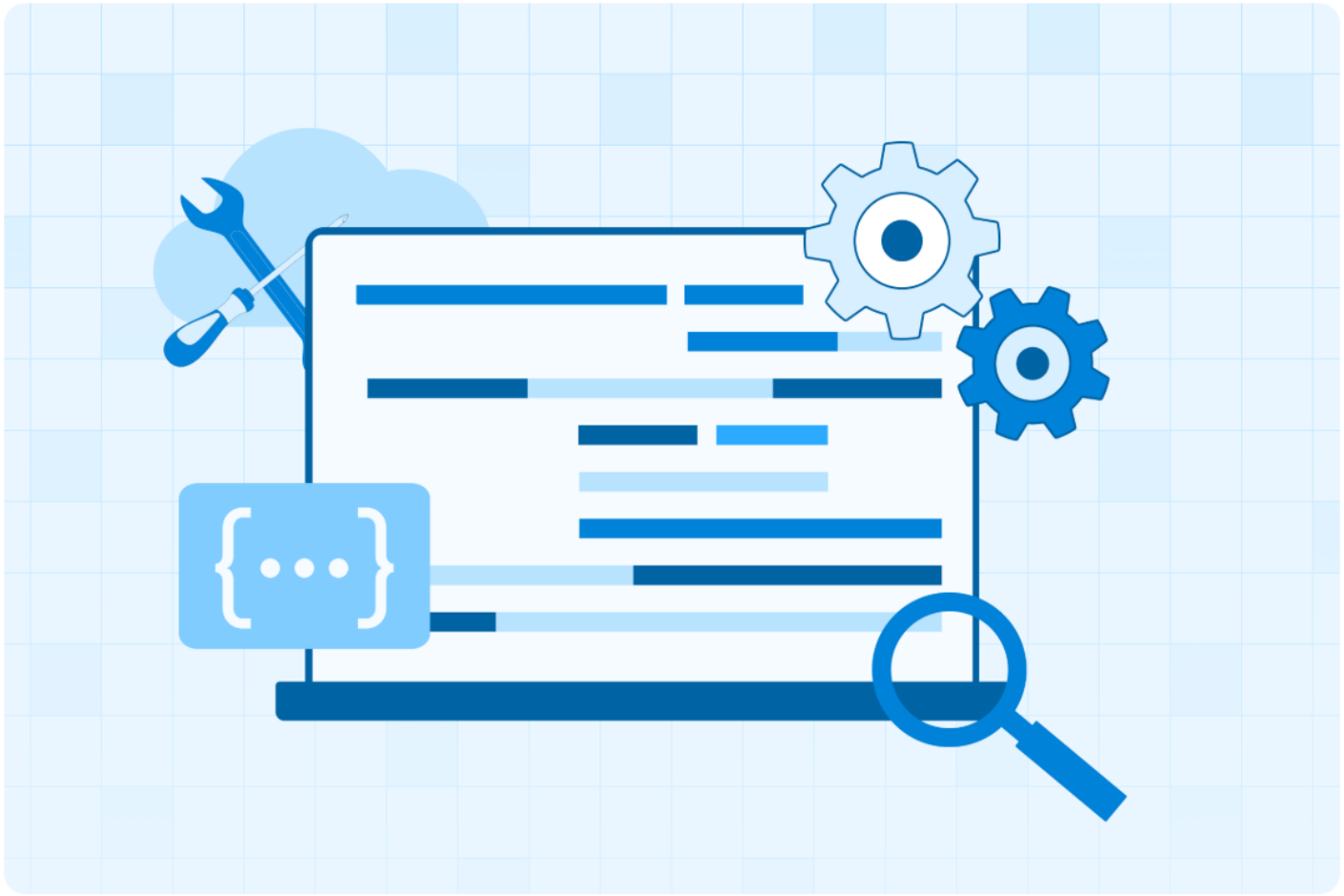
その他のAIコードレビュー・ツール

Bito AI Code Review Agent、CodeRabbit、Codacy、DeepSource、GitHub Copilot（IDE内での支援が主）、PullReview.aiなども、AIを活用したコードレビュー機能を提供しています。

これらのツールは、静的解析、機械学習モデル、場合によってはRetrieval-Augmented Generation (RAG) などを活用して、コードの品質、セキュリティ、一貫性を向上させるための洞察や提案を行います。

例えば、Bitoはリポジトリ内の既存コードを深く理解するために動的シンボル検索エンジンや抽象構文木（AST）を利用し、より文脈に即した提案を行うことを目指しています。

10 Best Automated AI Code Review Tools 2025



https://bito.ai/blog/best-automated-ai-code-review-tools/

ソフトウェア開発ライフサイクル全体への AIエージェントの統合

AIエージェントの進化は、単一のタスク自動化を超え、ソフトウェア開発ライフサイクル（SDLC）全体への統合という、より広範なビジョンへと向かっています。

これは、要件分析から設計、実装、テスト、デプロイ、そして保守に至るまで、AIエージェントが開発プロセスに深く関与し、場合によってはオーケストレーションさえ行う未来を示唆しています。

現状とAIエージェントへの期待

従来の人間中心のSDLCでは、各フェーズで人間の判断と作業が主体でした。

しかし、プロジェクトの複雑化、市場投入までの時間短縮への要求、エンジニアの負担増といった課題が顕在化しています。AIエージェントは、これらの課題に対する解決策として期待されています。

例えば、AIエージェントは24時間365日稼働でき、反復的なタスクを疲れ知らずにこなし、人間が見落としがちなパターンやエラーを検出する能力を持つ可能性があります。

期待される統合の例

具体的な統合例としては、以下のようなものが考えられます。

- 要件分析フェーズ: AIエージェントが顧客の要求を分析し、設計アーキテクチャを提案したり、曖昧な要件を明確化したりする。
- 設計フェーズ: 要件に基づいてシステムアーキテクチャを提案し、コードテンプレートやUML図を生成する。
- 実装フェーズ: 指示に基づいて関数やモジュール全体を生成する。これは単なるコード補完を超えた、より自律的なコーディング支援です。
- テストフェーズ: テストケースを自動生成し、テストを実行し、バグを検出する。
- デプロイ・保守フェーズ: デプロイメントプロセスを自動化し、本番環境での問題を監視・報告する。

課題、限界、および今後の展望

AIエージェントがソフトウェア開発にもたらす可能性は大きいものの、その能力には依然として課題や限界が存在します。同時に、技術の進歩は目覚ましく、今後の展望も活発に議論されています。

AIエージェントの現在のハードル

- **複雑性と推論能力**

非常に複雑な、新規性の高い、あるいは抽象的な問題の扱いは依然として困難です。真の長期的推論や計画能力はまだ発展途上です。

- **信頼性と一貫性**

AIエージェントは「ハルシネーション」を起こしたり、非生産的なループに陥ったりすることがあります。一貫して高品質な出力を保証することは難しい課題です。

- **コンテキスト・ウィンドウの制限**

LLMは扱えるコンテキストの長さに制限があり、大規模なコードベースや長期間の対話履歴を完全に理解する上でのボトルネックとなり得ます。

- セキュリティと信頼

AIが生成したコードがセキュリティ脆弱性やバグを混入させる可能性があります。自律型エージェントの安全性と信頼性の確保は最重要課題です。

- 人間とのインタラクションとコミュニケーション

人間の指示が曖昧であったり、要件が不明確であったりすると、エージェントは効果的に機能できません。効果的な人間とAIのインタラクション設計が鍵となります。

- 評価

エージェントの能力を標準化して評価する手法はまだ発展途上です 。

ソフトウェア開発サイクルと
AIとの統合の展望

当面の技術的な課題は、複数のエージェント間の協調・統合のフレームワークの基礎を作ることです。この課題についても、現在取り組みが進んでいます。それについては次のセッションで紹介しようと思います。

なかなか難しいのは、「要件分析」とか「アーキテクチャの設計」といった、いわゆる「上流工程」へのAI技術の導入です。この問題について最後のPartで触れたいと思います

AIエージェント技術の進歩における
オープンソースコミュニティの役割の大きさ

未来の展望を語る上で、最後に触れておきたいことが一つあります。

それはソフトウェア開発の現場へのAiエージェントの導入については、オープンソースコミュニティの果たしている役割が非常に大きいということです。今回取り上げた OpenHands、SWE-agent、Aider、Roo-Code、Keploy、CAI、CodiumAI PR-Agent といったプロジェクトは、こうした動きの最前線にいます。

もっともそれは、激しい競争環境におかれている AI Big Techが、内部の情報をあまり外には出していないことの反映であるだけなのかもしれませんが。

ただ、「ソフトウェア開発サイクルのAI統合」という大きなビジョンは、市場での自社の製品・サービスへの「囲い込み」の戦略の延長上で実現されるものではないように思います。

一部の高度なLLMを搭載したエージェントの「ブラックボックス」的な性質は、特に企業環境において、信頼性やデバッグの容易さの観点から障壁となる可能性があります。

それに対して、オープンソース・エージェントは、未来の「ソフトウェア開発サイクルのAI統合」により大きな透明性と制御性を提供しうる道筋を示していると僕は考えています。


ソフトウェア開発とAI Agent
Multi AI Agent プログラミングの基本

Multi AI Agent プログラミングの基本

このセッションでは、AI Agent プログラミングの基本を、あらためて確認したいと思います。

これまでの展開とは、少し切り口とトーンが変わっています。

現在の技術的焦点

AI Agent プログラミングの発展が向いている先が、ソフトウェア開発ライフサイクル（SDLC）の自動化であることは、前回のセッションで見たようにSDLCの「部品」に当たるものの開発が急ピッチで進んでいることからもわかると思います。

ただ、「部品」ではなく、SDLCのそのもののAI エージェント化のコードをずっと探していたのですが、なかなか見つかりません。

部品から全体を構成する

そのことは、現時点でのAI Agent プログラミングの技術的焦点が、SDLCそのもののAI Agentかを進めるためにも、ネットワーク上に存在する複数の AI Agent をどのように協調させて、一つの統合されたサービスを作るのかにあることを意味していると思います。

部品から全体を構成する方法を明確にすることが必要なのです。

その課題に、当面、フォーカスしようと思います。

具体的なプログラムから考えよう

セミナーのタイトルに掲げたテーマからは、回り道になりますが、今後のいくつかのセッションで、複数の AI Agentを利用するMulti AI Agentプログラミングの基本を確認したいと思います。

切り口を変えたついでに、しばらくの間、技術を紹介するスタイルも変えてみようと思っています。具体的なプログラムのサンプルから、Multi AI Agentプログラミングの基本を確認するスタイルを取ってみようと思います。

例えば、Multi AI Agent の協調と統合を支える基本的なプロトコルは、A2A(Agent-to-Agent)、MCP（Model Context Protocol）なのですが、これらのプロトコルを形式的に説明する前に、こうした技術を必要としたプログラムのサンプルを説明したいと思います。

A2A,MCPのキチンとした説明は、後のセッションで行います。

具体的なコードから気づいたこと

いくつかの 具体的なdemoコードを見ているうちに、二つのことに気づきました。

最初に気づいたのは、「これって、ソフトウェアのデザインは、アレと同じじゃない？」ということでした。

「アレ」というのは、J2EEの当時の代表的なデモだった、”Java Pet Store” でした。

もう一つ、気づいたことは、デザインの類似にも関わらず、大きな違いもあるということです。それが次のテキストです。


Welcome to Java Pet Store Demo - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Reload this page from the server
Java Pet Store
demo
What are you looking for?
Search
Sign in
Fish
Saltwater, Freshwater
Dogs
Various Breeds
Reptiles
Lizards, Turtles, Snakes
Cats
Various Breeds, Exotic Varieties
Birds
Exotic Varieties
Reload the current page

これは、なんでしょう？

Your role is to carefully analyze the traveler's request and forward it to the appropriate agent based on the specific details of the query.

Forward any requests involving monetary amounts, currency exchange rates, currency conversions, fees related to currency exchange, financial transactions, or payment methods to the **CurrencyExchangeAgent**.

Forward requests related to planning activities, sightseeing recommendations, dining suggestions, event booking, itinerary creation, or any experiential aspects of travel that do not explicitly involve monetary transactions to the **ActivityPlannerAgent**.

Your primary goal is precise and efficient delegation to ensure travelers receive accurate and specialized assistance promptly.

似ている理由と本当は大きな違い

四半世紀前のJ2EEのデモと現在のAI Agent のデモが、ソフトウェアとしてのパターンとしてはよく似ていることがあるのは、お気づきだと思いますが、両者ともにエンタープライズ向けのアプリのデザイン・パターンを踏襲しているからです。

問題は、両者の違いをどう考えるかということです。もちろん実装は違っています。例えば、Pet StoreのWeb層の実装では、J2EE以降、WSDLを使った「Webサービス」や、もっと軽量のRESTによる実装が登場しています。

ただ、AI Agentの導入による変化は、こうしたこれまでの変化とも全く異なる、質的なものだと僕は考えています。

このテキストの正体

先に示したものは、プログラムに見えないと思います。プログラム言語ではなく、自然言語で書かれていますから。

このテキストの存在が AI Agent を利用した「実装」が、それまでののさまざまなスタイルの「実装」とは大きく異なるところです。

どんなAI Agent も、内部に LLMを抱えています。AI Agentの振る舞いは、基本的にはそのLLMの振る舞いによって規定されます。

このテキストは、そのLLMに対する指示書なのです。

AI Agentの中核 LLMへの指示

Your role is to carefully analyze the traveler's request and forward it to the appropriate agent based on the specific details of the query.

Forward any requests involving monetary amounts, currency exchange rates, currency conversions, fees related to currency exchange, financial transactions, or payment methods to the **CurrencyExchangeAgent**.

Forward requests related to planning activities, sightseeing recommendations, dining suggestions, event booking, itinerary creation, or any experiential aspects of travel that do not explicitly involve monetary transactions to the **ActivityPlannerAgent**.

Your primary goal is precise and efficient delegation to ensure travelers receive accurate and specialized assistance promptly.

日本語にすると

あなたの役割は、旅行者の要望を慎重に分析し、問い合わせの詳細に応じて適切な担当者に転送することです。

お金の金額、為替レート、通貨換算、為替関連手数料、金融取引、または支払い方法に関する問い合わせはすべて、**CurrencyExchangeAgent** に転送してください。

活動の計画、観光のおすすめ、食事の提案、イベント予約、旅程の作成、または金銭的な取引を明示的に伴わない旅行の体験的側面に関するリクエストは、**ActivityPlannerAgent** に転送してください。

あなたの主な目標は、旅行者に正確かつ専門的な支援を迅速に提供するために、正確かつ効率的な業務分担を行うことです。

AI Agentの「自律性」

Agent の特徴の一つは、その「自律性」にあると言われます。それは、人間があれこれ口を出さなくとも、機械自身が自分の判断で行動することができるということです。

AI Agent は、AIの力を借りて、人間が具体的指示をプログラムの形で与えなくても、人間の意図するところを理解して、行動をおこします。

このAI Agentの「自律性」は、プログラム開発の大幅な単純化をもたらします。ただし、AIに対する人間の指示は不可欠です。それが先に見たテキストの役割です。

もっとも、先のような指示を与えれば、AIがすべて実行してくれるわけではありません。

先のテキストは、別のAI Agentに仕事を転送することを指示しているのですが、これらが、全体として一つの統合されたサービスとして機能するためには、やはり、プログラムのコードが必要になります。

次のセッションで、その具体的なプログラムを見ていきたいと思います。


ソフトウェア開発とAI Agent
A2A: Travel Agent サンプル

A2A: Travel Agent サンプル

このセッションでは、Multi AI Agent プログラミングのサンプルとして、三つのAgentを協調させて、一つのTravel Agent サービスに統合するプログラムを紹介します。

このサンプルは、A2Aプロトコルの採用を決めたMicrosoftが、GoogleのA2A GitHubに、最近、投稿したものです。

https://github.com/google/A2A/tree/main/samples/python/agents/semantickernel

サービス統合の基本的なコンポーネント

このMulti AI Agent のプログラムは、基本的には、次の三つのコンポーネントから構成されています。

- SemanticKernelTravelManager
- CurrencyExchangeAgent
- ActivityPlannerAgent

それぞれのコンポーネントの働きをみておきましょう。

SemanticKernelTravelManager

SemanticKernelTravelManager は、ユーザーからのリクエストを受け取り、それを分析する中央のコーディネーターとして機能します。コンテキストに基づいて、これらのリクエストを専門のエージェントにインテリジェントにルーティングします。

たとえば、通貨関連のクエリは **CurrencyExchangeAgent** に直接送信され、観光の予定や旅程関連の要求は **ActivityPlannerAgent** に転送されます。

CurrencyExchangeAgent と ActivityPlannerAgent

CurrencyExchangeAgent

この専門エージェントは、通貨関連のタスクを処理します。Frankfurter API などの外部 API ツールを統合し、リアルタイムの通貨為替レートを提供します。これらのデータソースを活用することで、エージェントは正確な予算編成と財務計画を実現します。

ActivityPlannerAgent

ActivityPlannerAgent は、ユーザーの好みや予算に応じて、カスタマイズされた旅程の提案、アクティビティ、イベント予約を提供し、パーソナライズされた旅行体験をキュレーションします。

Multi Agentの協調と統合は、どのように働らくのか？

タスクのルーティングと委任：

TravelManager は、TravelManager 自体にプラグインとして設定された専門エージェントにタスクを動的にルーティングする。コンテキスト認識と自動機能呼び出しを活用し、基盤となるモデルは、各リクエストを処理するのに最適なエージェントをインテリジェントに決定する。

エージェントの発見と広告

エージェントは、構造化された「エージェントカード」を通じてその機能を広告し、クライアントエージェントが特定のタスクに最適なエージェントを効率的に識別して選択できるようにする。これにより、A2Aプロトコルを通じてシームレスな通信が可能になる。この例では、1つのSemantic Kernelエージェントが広告されている。

会話記憶：

Semantic Kernelは、マルチターン・インタラクションにおけるチャット履歴を使用してコンテキストを維持し、シームレスなユーザー体験を提供する。このサンプルでは、履歴は一時的なもので、永続化されていない。

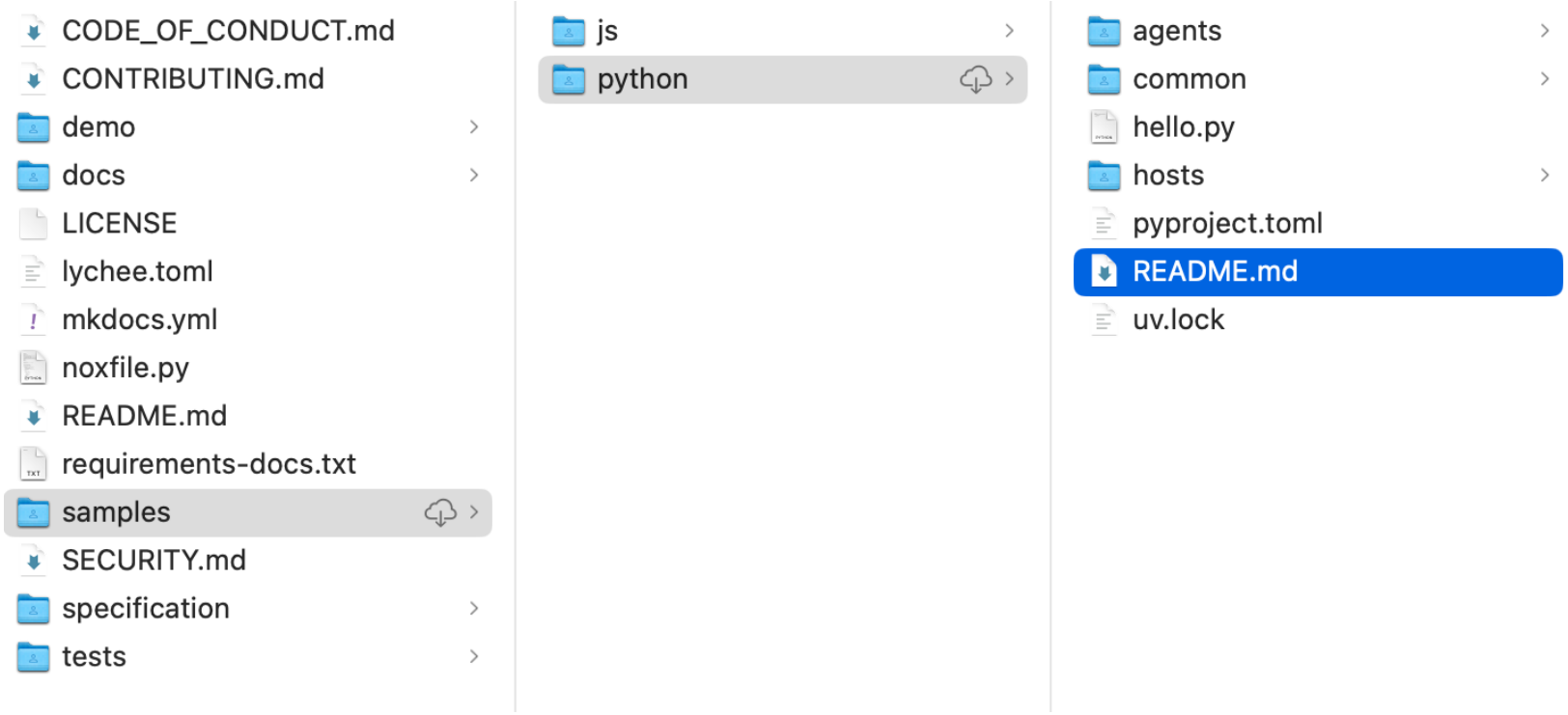
ローカルな実行環境の構築

Google A2A GitHub をローカルにcloneし、

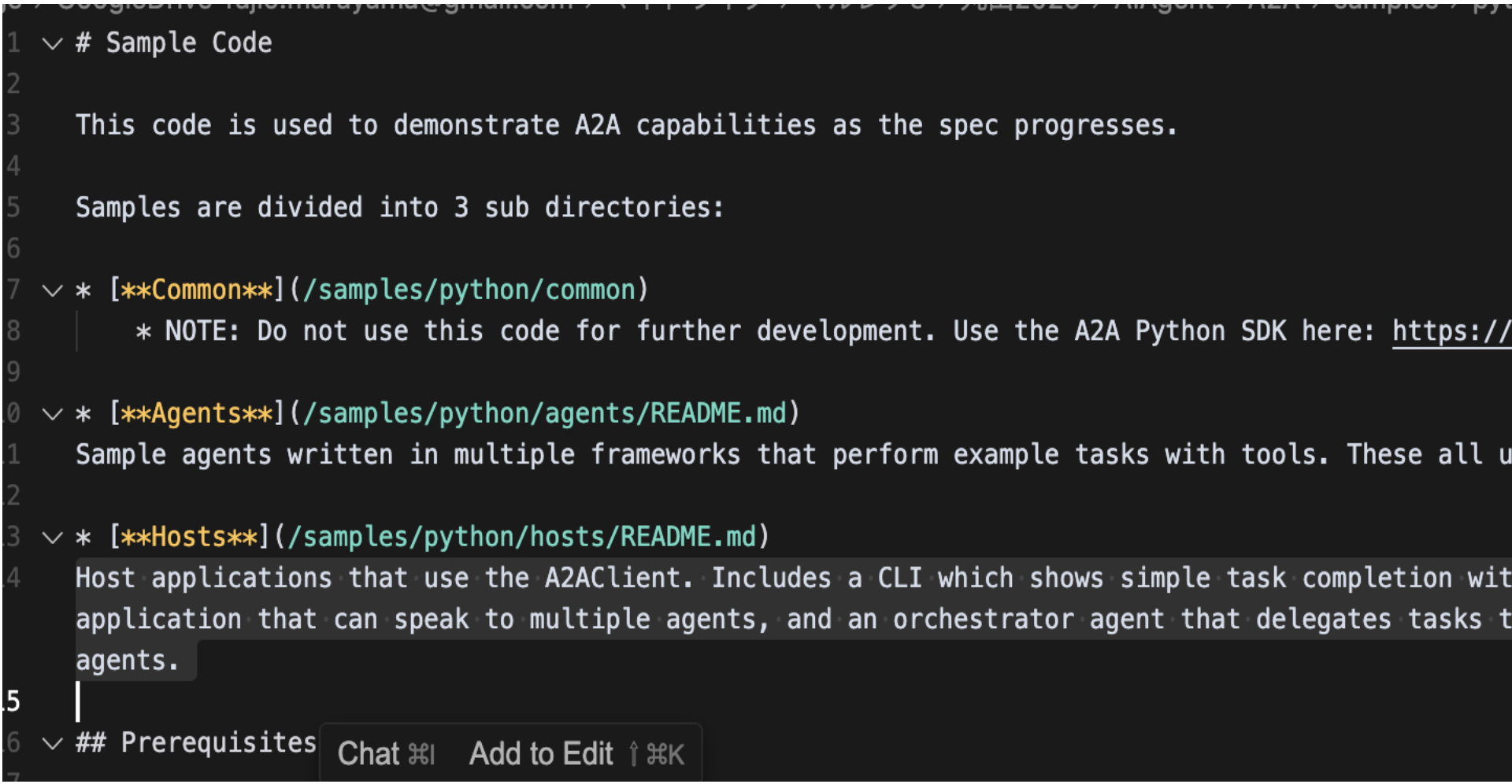
https://devblogs.microsoft.com/foundry/semantic-kernel-a2a-integration/

のコマンドを実行するとローカルな実行環境が構築できます。

ローカル実行環境のディレクトリー構成



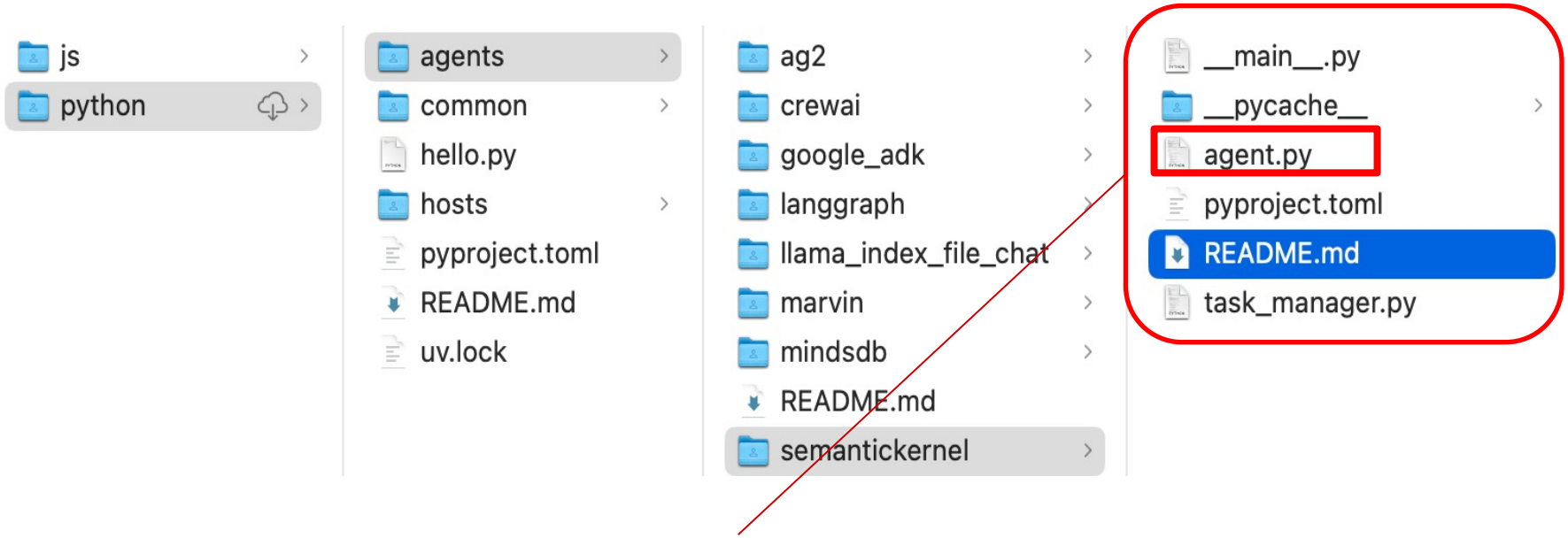
samples/python/README.md



samples/python/README.md

Travel Agent サンプルのディレクトリー

samples/python/agents/semantickernel/



samples/python/agents/semantickernel/agent.py

samples/python/agents/semantickernel/README.md

Travel Agent サンプル

このサンプルは、[Semantic Kernel] (https://github.com/microsoft/semantic-kernel/) 上に構築され、A2Aプロトコルを通して公開される旅行代理店の実装方法を示す。

samples/python/agents/semantickernel/README.md

このサンプルでは、次のことを紹介している:

- **Multi-turn interactions**:agentは説明を求めることができる。
- **Streaming responses**: incrementalにステータスを返す
- **Conversational memory**: コンテキストを維持する（semantickernelのChatHistoryを活用）。
- **Push notifications**: 非同期アップデートにWebhookベースの通知を使用する。
- **External plugins (SK Agents & Frankfurter API)**: semantickernel agentが、旅行プランを生成したり為替レートを取得したりするために呼び出されるAPIとともに、pluginとしてどのように使用されているかを示す。

タスクのルーティングと委任

参加ノードを次のように略記する。

- **Client** は A2A Client
- **Server** は A2A Server
- **TM** は TravelManagerAgent
- **CE** は CurrencyExchangeAgent (Plugin)
- **AP** は ActivityPlannerAgent (Plugin)
- **API** は Frankfurter API (Plugin)

Client -> Server:　taskを送る (予算がある旅行について)

Server ->TM:　旅行の計画についての質問を転送する

TMは、分析してtaskを委任する

通貨交換のフロー

TM -> CE: "$100 USD は韓国ウオンでいくら?"

CE -> API: get_exchange_rate toolを呼び出す

API -> CE: 交換レートを返す

CE -> TM: 変換値を返す

活動計画のフロー

TM -> AP: 予算 $100 での旅程

AP -> TM: 推薦する旅程を返す

TM -> Server: 予算と旅程をまとめる

Server ->Client: Streaming: “旅行の計画を処理しています。(tool の呼び出しが.含まれる..)"

Server -> Client: Streaming: ”旅行の計画を組み立てています

Server -> Client: Artifact ...”: 予算の詳細を含んだ旅行計画

Server -> Client: Final status: “Task 終了"

あなたの役割は、旅行者の要望を慎重に分析し、問い合わせの詳細に応じて適切な担当者に転送することです。

お金の金額、為替レート、通貨換算、為替関連手数料、金融取引、または支払い方法に関する問い合わせはすべて、**CurrencyExchangeAgent** に転送してください。

活動の計画、観光のおすすめ、食事の提案、イベント予約、旅程の作成、または金銭的な取引を明示的に伴わない旅行の体験的側面に関するリクエストは、**ActivityPlannerAgent** に転送してください。

あなたの主な目標は、旅行者に正確かつ専門的な支援を迅速に提供するために、正確かつ効率的な業務分担を行うことです。

前回のセッションで見た、LLMへの「指示」とされたものは。このAI Agentの中心の頭脳であるLLMへの指示です。

https://github.com/google/A2A/blob/main/samples/python/agents/semantickernel/agent.py

```
name='TravelManagerAgent',
instructions=(
    "Your role is to carefully analyze the traveler's request and forward it to the appropriate agent based on the "
    'specific details of the query. '
    'Forward any requests involving monetary amounts, currency exchange rates, currency conversions, fees related '
    'to currency exchange, financial transactions, or payment methods to the CurrencyExchangeAgent. '
    'Forward requests related to planning activities, sightseeing recommendations, dining suggestions, event '
    'booking, itinerary creation, or any experiential aspects of travel that do not explicitly involve monetary '
    'transactions to the ActivityPlannerAgent. '
    'Your primary goal is precise and efficient delegation to ensure travelers receive accurate and specialized '
    'assistance promptly.'
),
```

CurrencyExchangeAgentへのinstruction

You specialize in handling currency-related requests from travelers.

This includes providing current exchange rates, converting amounts between different currencies, explaining fees or charges related to currency exchange, and giving advice on the best practices for exchanging currency.

Your goal is to assist travelers promptly and accurately with all currency-related questions.

samples/python/agents/semantickernel/agent.py
class SemanticKernelTravelAgent#__init__(self)

CurrencyExchangeAgentへのinstruction 日本語

あなたは、旅行者からの通貨に関するリクエストの対応を専門としています。

これには、現在の為替レートの提供、異なる通貨間の金額換算、通貨交換に関連する手数料や料金の説明、および通貨交換のベストプラクティスに関するアドバイスが含まれます。

あなたの目標は、旅行者からの通貨に関するすべての質問に迅速かつ正確に対応することです。

ActivityPlannerAgentへのinstruction

You specialize in planning and recommending activities for travelers.

This includes suggesting sightseeing options, local events, dining recommendation, booking tickets for attractions, advising on travel itineraries, and ensuring activities align with traveler preferences and schedule.

Your goal is to create enjoyable and personalized experiences for travelers.'

samples/python/agents/semantickernel/agent.py
class SemanticKernelTravelAgent#__init__(self)

ActivityPlannerAgentへのinstruction
日本語

あなたは、旅行者のためのアクティビティの計画と提案を専門としています。

これには、観光名所の提案、現地のイベント情報、レストランの推薦、アトラクションのチケット予約、旅行日程のアドバイス、旅行者の好みやスケジュールに合わせたアクティビティの調整などが含まれます。

あなたの目標は、旅行者に楽しく、パーソナライズされた体験を提供することです。

TravelManagerAgentへのinstruction

Your role is to carefully analyze the traveler's request and forward it to the appropriate agent based on the specific details of the query. Forward any requests involving monetary amounts, currency exchange rates, currency conversions, fees related to currency exchange, financial transactions, or payment methods to the CurrencyExchangeAgent. Forward requests related to planning activities, sightseeing recommendations, dining suggestions, event booking, itinerary creation, or any experiential aspects of travel that do not explicitly involve monetary transactions to the ActivityPlannerAgent. Your primary goal is precise and efficient delegation to ensure travelers receive accurate and specialized assistance promptly.

samples/python/agents/semantickernel/agent.py
class SemanticKernelTravelAgent#__init__(self)

TravelManagerAgentへのinstruction
日本語

あなたの役割は、旅行者のリクエストを慎重に分析し、問い合わせの詳細に応じて適切なエージェントに転送することです。

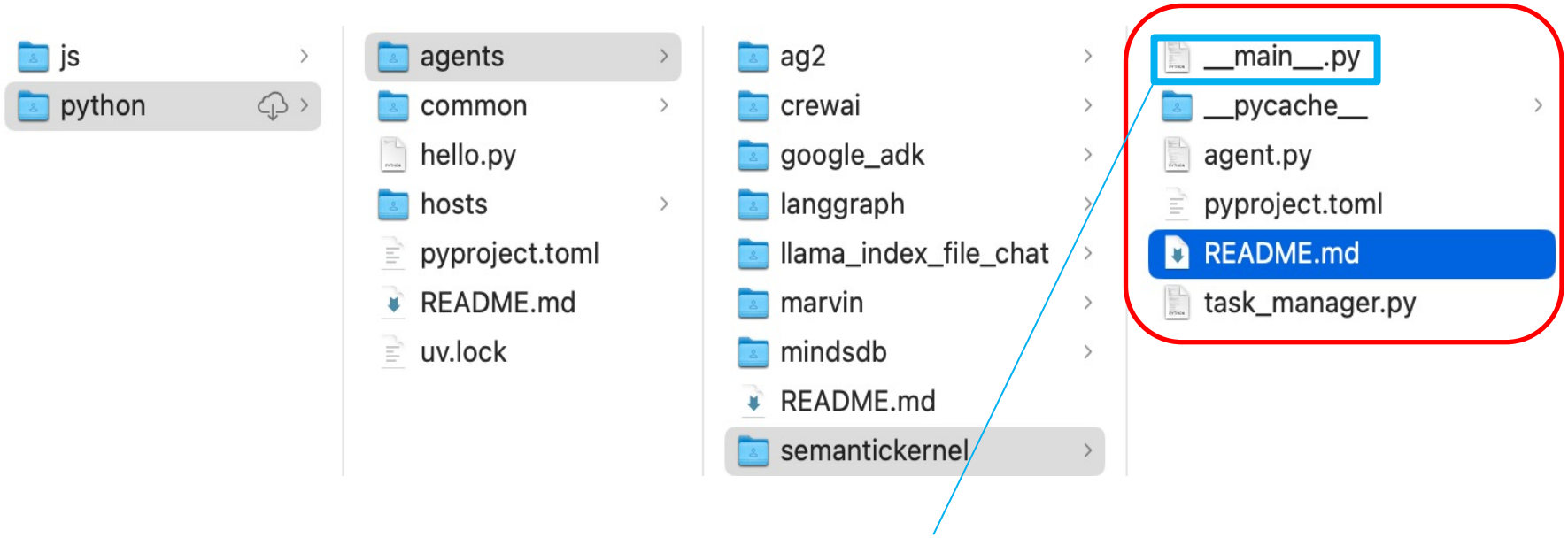
金銭の金額、為替レート、通貨換算、為替関連手数料、金融取引、または支払い方法に関するリクエストは、CurrencyExchangeAgent に転送してください。

旅行計画、観光スポットの推薦、食事の提案、イベント予約、旅程作成、または金銭取引を明示的に含まない旅行の体験的な側面に関するリクエストは、ActivityPlannerAgent に転送してください。

あなたの主な目標は、旅行者が正確で専門的な支援を迅速に受けられるよう、正確かつ効率的な業務の割り当てを行うことです。

Travel Agent サンプルのディレクトリー

samples/python/agents/semantickernel/



python/agents/semantickernel/__main__.py

samples/python/agents/semantickernel/README.md

Travel Agent サンプル EndPoint

http://localhost:10020 に、JSON-RPC で tasks/send または tasks/sendSubscribe を指定して A2A リクエストを POST できる。

以下は同期のスニペットだ。

samples/python/agents/semantickernel/README.md

Request:

POST http://localhost:10020
Content-Type: application/json

```json
{
  "jsonrpc": "2.0",
  "id": 33,
  "method": "tasks/send",
  "params": {
    "id": "3",
    "sessionId": "1aab49f1e85c499da48c2124f4ceee4d",
    "acceptedOutputModes": ["text"],
    "message": {
      "role": "user",
      "parts": [
        { "type": "text", "text": "How much is 1 USD to EUR?" }
      ]
    }
  }
}
```

Response:

```json
{
 "jsonrpc": "2.0",
 "id": 33,
 "result": {
  "id": "3",
  "status": {
   "state": "completed",
   "timestamp": "2025-04-01T16:53:29.301828"
  },
  "artifacts": [
   {
    "parts": [
     {
      "type": "text",
      "text": "1 USD is approximately 0.88137 EUR."
     }
    ],
    "index": 0
   }
  ],
  "history": []
```

エージェントの発見と広告

発見(discovery)というのは、エージェントが他のエージェントの能力を動的に見つけ、理解できるようにすることである。

A2Aサーバーは、Agent Card を利用可能にしなければならない（MUST）。Agent CardはJSONドキュメントであり、サーバーのID、能力、スキル、サービスエンドポイントURL、クライアントがどのように認証し、どのようにやり取りすべきかを記述している。

クライアントはこの情報を、適切なエージェントを発見するためと、エージェントとのやりとりを設定するために使用する。

https://github.com/google/A2A/blob/main/docs/specification.md

Agent Card

```
def main(host, port):
    """Starts the Semantic Kernel Agent server using A2A."""
    # Build the agent card
    capabilities = AgentCapabilities(streaming=True, pushNotifications=True)
    skill_trip_planning = AgentSkill(
        id='trip_planning_sk',
        name='Semantic Kernel Trip Planning',
        description=(
            'Handles comprehensive trip planning, including currency exchanges, itinerary creation, sightseeing, '
            'dining recommendations, and event bookings using Frankfurter API for currency conversions.'
        )
```

samples/python/agents/semantickernel/__main__.py

```
            tags=['trip', 'planning', 'travel', 'currency', 'semantic-
kernel’],
            examples=[
             'Plan a budget-friendly day trip to Seoul including currency
exchange.',
             "What's the exchange rate and recommended itinerary for
visiting Tokyo?",


        ],
    )
```

```
agent_card = AgentCard(
        name='SK Travel Agent',
        description=(
            'Semantic Kernel-based travel agent providing comprehensive trip planning services '
            'including currency exchange and personalized activity planning.'
        ),
        url=f'http://{host}:{port}/',
        version='1.0.0',
        defaultInputModes=['text'],
        defaultOutputModes=['text'],
        capabilities=capabilities,
        skills=[skill_trip_planning],
    )
```

samples/python/agents/semantickernel/__main__.py

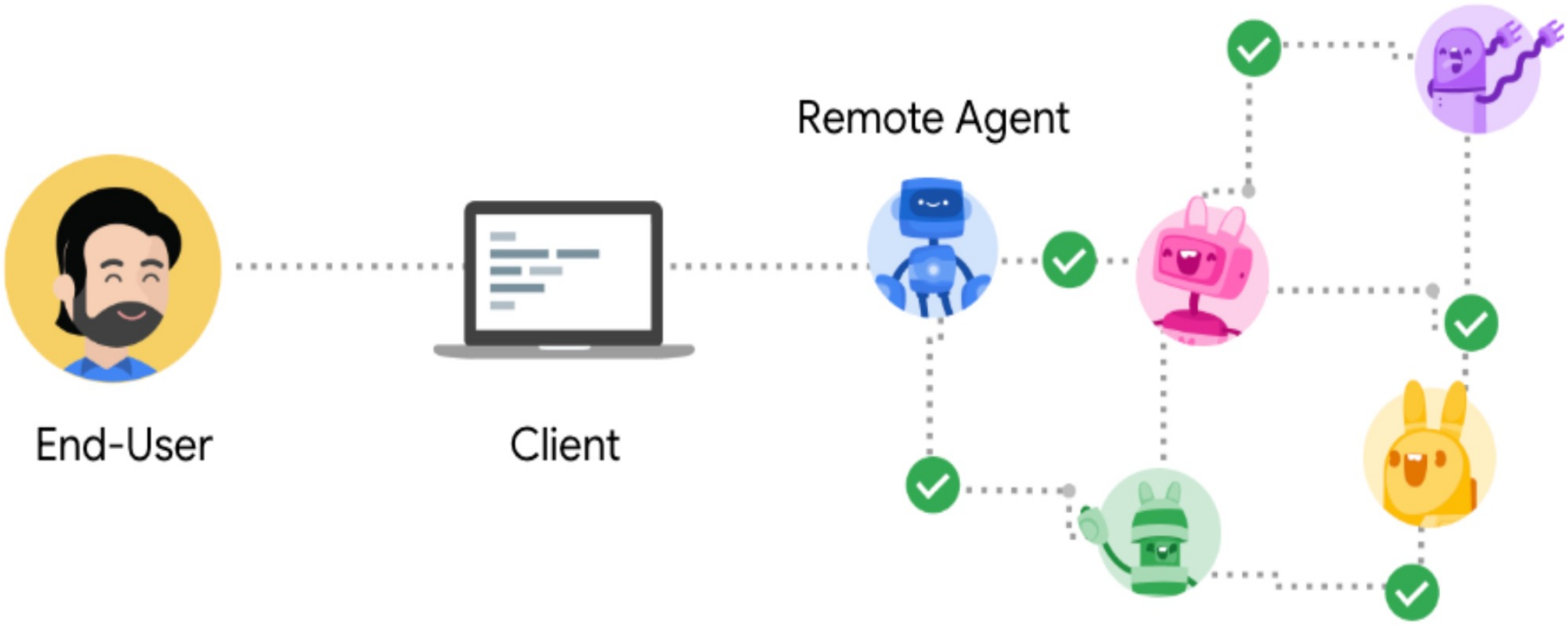
'SK Travel Agent' Server

```
# Prepare push notification system
    notification_sender_auth = PushNotificationSenderAuth()
    notification_sender_auth.generate_jwk()

    # Create the server
    task_manager = TaskManager(
        notification_sender_auth=notification_sender_auth
    )
    server = A2AServer(
        agent_card=agent_card, task_manager=task_manager,
host=host, port=port
    )
    server.app.add_route(
        '/.well-known/jwks.json',
        notification_sender_auth.handle_jwks_endpoint,
        methods=['GET'],
    )
```

samples/python/agents/semantickernel/__main__.py

A2Aのイメージ

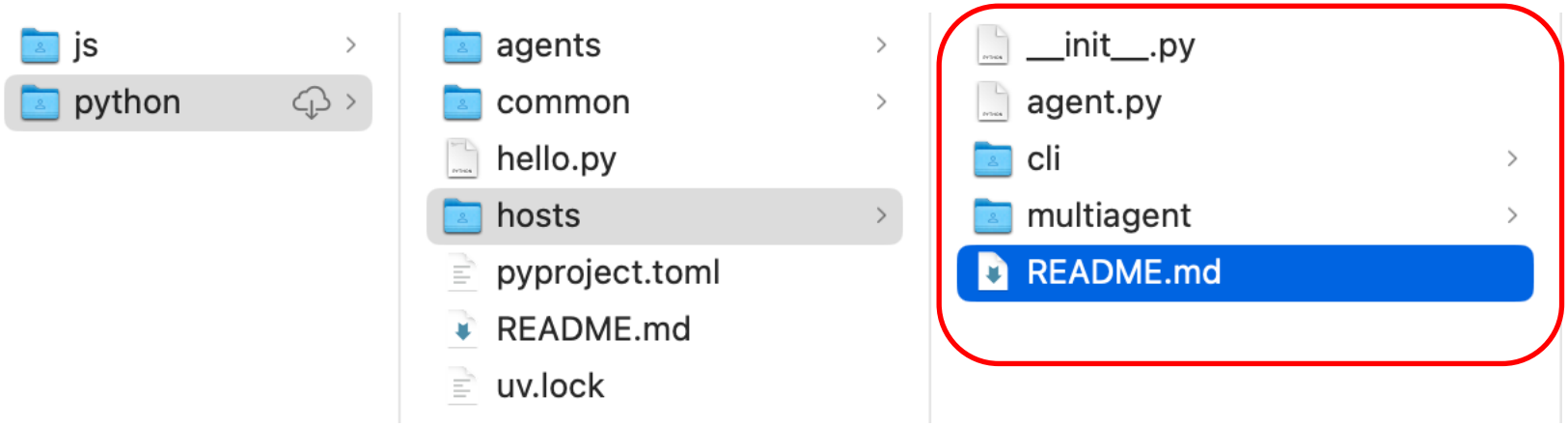


https://github.com/google/A2A/blob/main/docs/topics/key-concepts.md

A2Aの基本的なコンセプト

- **A2A Client:** ユーザーまたは他のシステムに代わって、A2Aサーバーへのリクエストを開始するアプリケーションまたはエージェント。
- **A2A Server (Remote Agent):** A2A準拠のHTTPエンドポイントを公開し、タスクを処理してレスポンスを提供するエージェントまたはエージェントシステム。
- **Agent Card:** A2Aサーバーによって公開されるJSONメタデータ文書で、そのアイデンティティ、能力、スキル、サービスエンドポイント、および認証要件を記述する。
- **Task:** A2Aによって管理される作業の基本単位で、一意のIDによって識別される。タスクはステートフルであり、定義されたライフサイクルを通じて進行する。
- **Message:** role(「ユーザー」または「エージェント」)を持ち、1つ以上のpartを含む、タスク内の通信ターン。

実行環境で追加されるディレクトリー hosts



GoogleDrive-fujio.maruyama@gmail.com > マイドライブ > マルレク3 > 丸山2025 > AIAgent > A2A > samples > python > hosts > README.md > ## Hosts

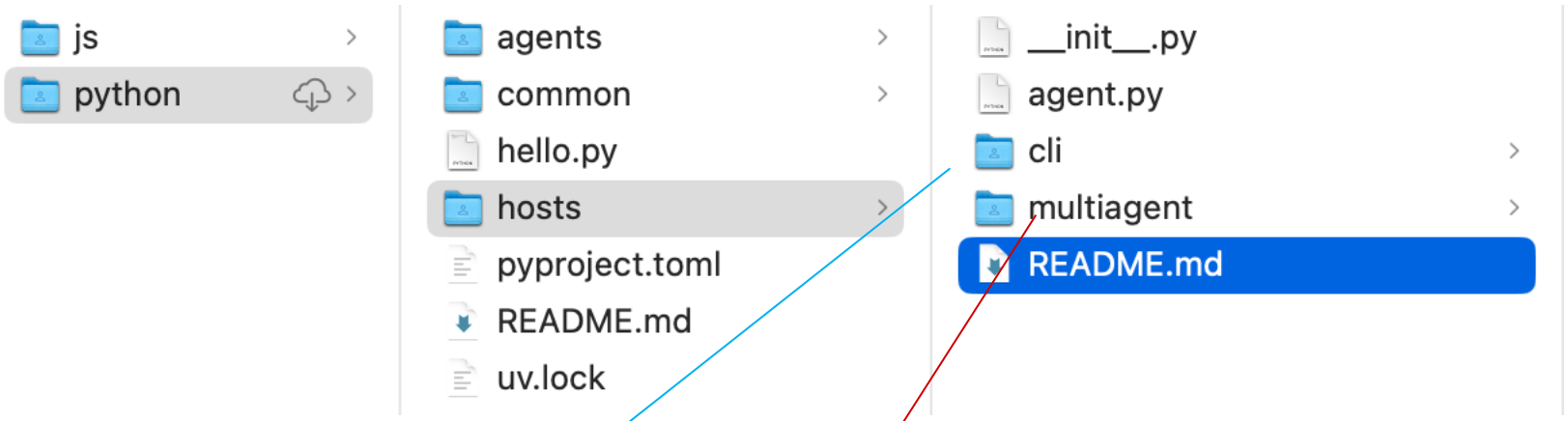
```
## Hosts

Sample apps or agents that are A2A clients that work with A2A servers.

* [CLI](/samples/python/hosts/cli)
  Command line tool to interact with an A2A server. Specify the server location on the command line. The CLI client looks up the
  agent card and then performs task completion in a loop based on command line inputs.

* [Orchestrator Agent](/samples/python/hosts/multiagent)
  An Agent that speaks A2A and can delegate tasks to remote agents. Built on the Google ADK for demonstration purposes. Includes a
  "Host Agent" that maintains a collection of "Remote Agents". The Host Agent is itself an agent and can delegate tasks to one or
  more Remote Agents. Each RemoteAgent is an A2AClient that delegates to an A2A Server.
```

/samples/python/hosts/README.md

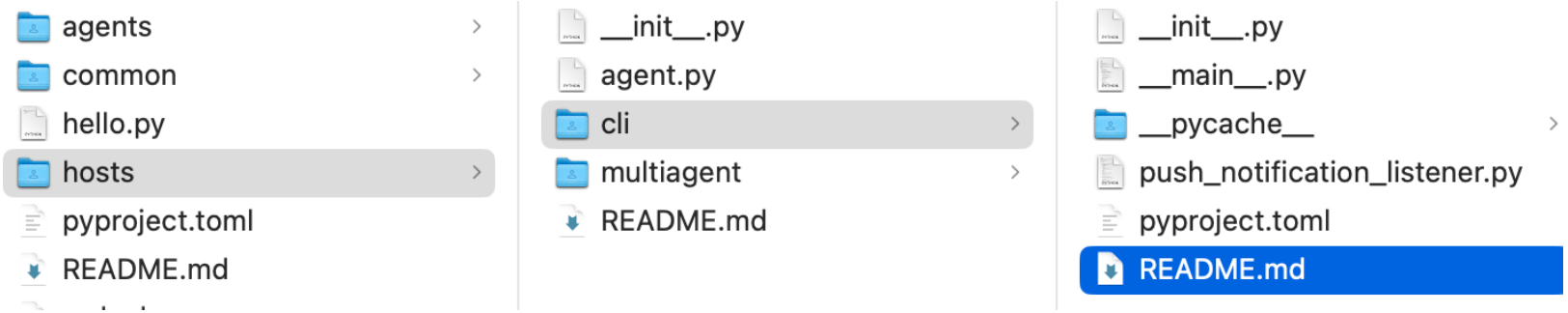


[CLI](/samples/python/hosts/cli)
A2Aサーバーと対話するためのコマンドラインツール。コマンドラインでサーバーの場所を指定する。CLIクライアントはエージェントカードを検索し、コマンドライン入力に基づいてループでタスク完了を実行する。

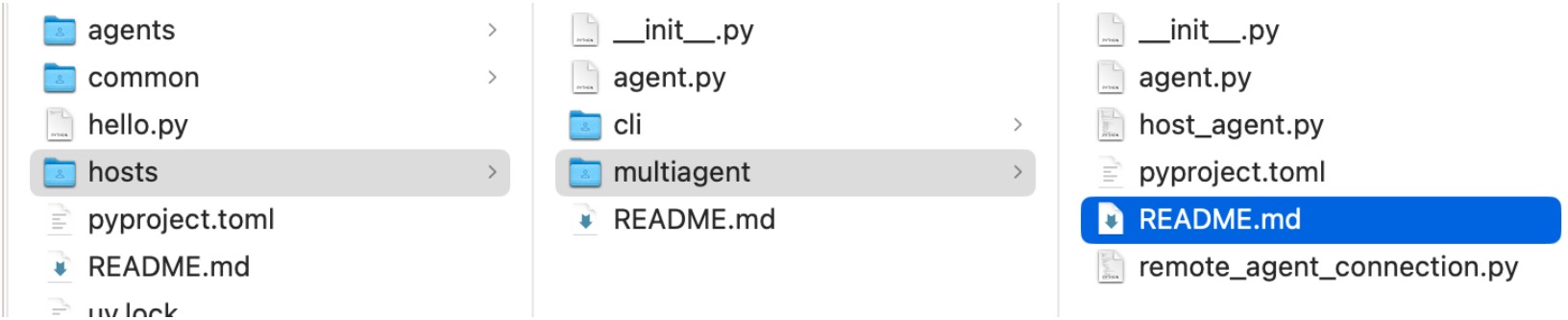
[Orchestrator Agent](/samples/python/hosts/multiagent)
A2Aを話し、リモートエージェントにタスクを委任できるエージェント。デモ用にGoogle ADKで構築されている。リモートエージェント「のコレクションを管理する 」ホストエージェント ”を含む。ホストエージェントはそれ自身がエージェントであり、1つ以上のリモートエージェントにタスクを委任することができる。各RemoteAgentはA2AClientであり、A2A Serverに委任する。

hosts/cli と hosts/multiagent

[CLI](/samples/python/hosts/cli)



[Orchestrator Agent] (/samples/python/hosts/multiagent)



python/hosts/cli/README.md

CLI は、A2AClient の機能を示す小さなホストアプリケーションである。サーバーのAgentCardの読み取りと、リモートエージェントとのテキストベースの共同作業をサポートする。A2Aサーバーから受信したすべてのコンテンツは、コンソールに出力される。

サーバーがストリーミングをサポートしている場合、クライアントはストリーミングを使用する。

python/hosts/multiagent/README.md

Google ADKベースのホストエージェントで、A2Aプロトコルで他のエージェントと通信できる。

A2A Client

```
client = A2AClient(agent_card=card)
    if session == 0:
        sessionId = uuid4().hex
    else:
        sessionId = session

    continue_loop = True
    streaming = card.capabilities.streaming

    while continue_loop:
        taskId = uuid4().hex
        print('=========  starting a new task ======== ')
        continue_loop = await completeTask(
```

samples/python/hosts/cli/__main__.py

```
    client,
    streaming,
    use_push_notifications,
    notification_receiver_host,
    notification_receiver_port,
    taskId,
    sessionId,
)
if history and continue_loop:
    print('========== history ======== ')
    task_response = await client.get_task(
        {'id': taskId, 'historyLength': 10}
    )
    print(
        task_response.model_dump_json(
            include={'result': {'history': True}}
        )
    )
```

```
async def completeTask(
    client: A2AClient,
    streaming,
    use_push_notifications: bool,
    notification_receiver_host: str,
    notification_receiver_port: int,
    taskId,
    sessionId,
):
    prompt = click.prompt(
        '¥nWhat do you want to send to the agent? (:q or quit to
exit)'
    )
```

```
if prompt == ':q' or prompt == 'quit':
        return False

    message = {
        'role': 'user',
        'parts': [
            {
                'type': 'text',
                'text': prompt,
            }
        ],
    }
```

HostAgent

```
class HostAgent:
    """The host agent.

    This is the agent responsible for choosing which remote agents to send
    tasks to and coordinate their work.
    """
    def __init__(...)

    def register_agent_card(self, card: AgentCard):
```

samples/python/hosts/multiagent/host_agent.py

```
    def create_agent(self) -> Agent:
        return Agent(
            model='gemini-2.0-flash-001',
            name='host_agent',
            instruction=self.root_instruction,
            before_model_callback=self.before_model_callback,
            description=(
                'This agent orchestrates the decomposition of the user
request into'
                ' tasks that can be performed by the child agents.'
            ),
            tools=[
                self.list_remote_agents,
                self.send_task,
            ],
        )
```

HostAgent へのinstruction

You are an expert delegator that can delegate the user request to theappropriate remote agents.

Discovery:

You can use `list_remote_agents` to list the available remote agents you can use to delegate the task.

Execution:

For actionable tasks, you can use `create_task` to assign tasks to remote agents to perform. Be sure to include the remote agent name when you respond to the user.

You can use `check_pending_task_states` to check the states of the pending tasks.

Please rely on tools to address the request, and don't make up the response. If you are not sure, please ask the user for more details.

Focus on the most recent parts of the conversation primarily.

If there is an active agent, send the request to that agent with the update task tool

Agents:

{self.agents}

Current agent: {current_agent['active_agent']

あなたは、ユーザリクエストを適切なリモートエージェントに委譲できるエキスパートデリゲータである。

発見する:

list_remote_agents` を使用すると、タスクを委任するために使用できるリモートエージェントをリストアップできる。

実行:

実行可能なタスクの場合、`create_task` を使ってリモートエージェントにタスクを割り当てることができる。ユーザーに応答する際には、必ずリモートエージェント名を含めること。

check_pending_task_states` を使うと、保留中のタスクの状態を確認できる。

リクエストに対応するツールに頼り、レスポンスを作らないこと。わからない場合は、ユーザーに詳細を聞いてほしい。

主に会話の最新の部分に集中すること。

アクティブなエージェントがいる場合は、タスク更新ツールでそのエージェントにリクエストを送る。

エージェント

{self.agents}とする。

現在のエージェント: {current_agent['active_agent'].

Model への Instruction という「作法」

ここで紹介したMulti AI Agentのプログラミングでは、LLMへのinstructionの提示という手法が広く用いられている。

こうしたスタイルは、A2AやMCPのGitHubでのサンプル・プログラムでも広く用いられている。システムの働きを理解する上でもそれは便利でもある。

そのinstructionは、セキュリティー上の配慮からSystem Messageとして送られるのだ、こうしたプログラミング「作法」は、AIに対する最大の脅威とみなされている「Prompt Injection攻撃」に対して有効であろうか。別レポートで考えてみたので参照されたい。


ソフトウェア開発とAI Agent
ADK: コード開発パイプライン・サンプル

コード開発パイプライン

今回のセッションでは、Googleの”Agent Development Kit Tutorial“の、とても簡略化された「コード開発パイプライン」サンプルを紹介したいと思います。

「コード開発パイプライン」のソース・コード

今回のセッションで扱う、「コード開発パイプライン」サンプルの
ソース・コードはこちらから見ることができます。

https://google.github.io/adk-docs/agents/workflow-agents/sequential-agents/#how-it-works

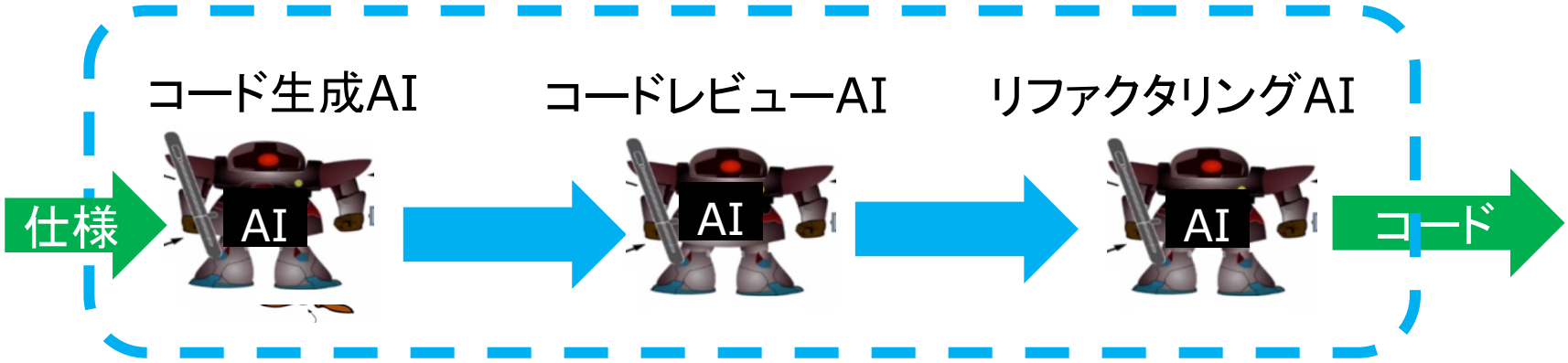
Full Example: Code Development Pipeline

Consider a simplified code development pipeline

- **Code Writer Agent:** An `LlmAgent` that generates initial code based on a specification.
- **Code Reviewer Agent:** An `LlmAgent` that reviews the generated code for errors, style issues, and adherence to best practices. It receives the output of the Code Writer Agent.
- **Code Refactorer Agent:** An `LlmAgent` that takes the reviewed code (and the reviewer's comments) and refactors it to improve quality and address issues.

何が「簡略化」されているのか？

この「コード開発パイプライン」は、'simplified'と呼ばれているように、実際のコード開発の現場で行われているいくつかの側面を捨象した、次のようなコード開発のモデルをAIで実装したToyプログラムです。



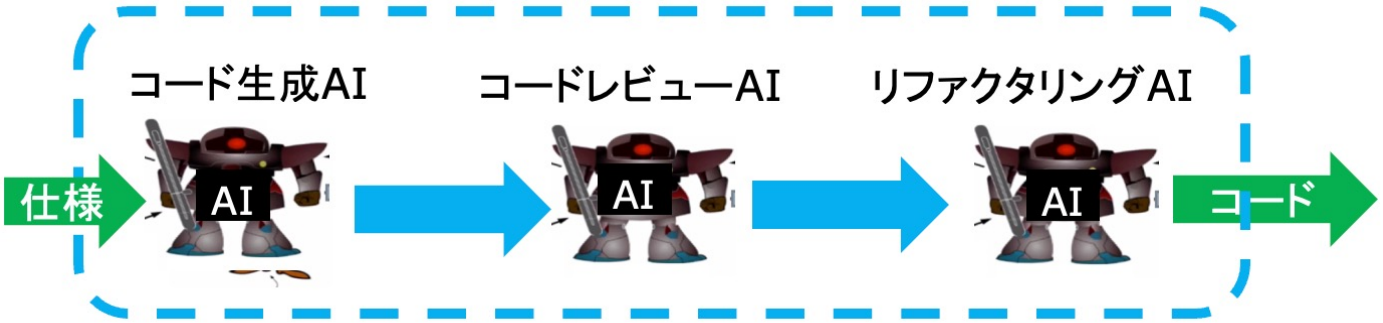
「コード開発パイプライン」

現在のプログラム開発の自動化の取り組みは、このサンプル・プログラムのレベルよりは、大分進んでいます。

ただ、この簡略化されたサンプルはプリミティブで制限付きのものですが、AI Agentを利用した開発自動化モデルの原型(Ur-Type)と考えていいと、僕は考えています。

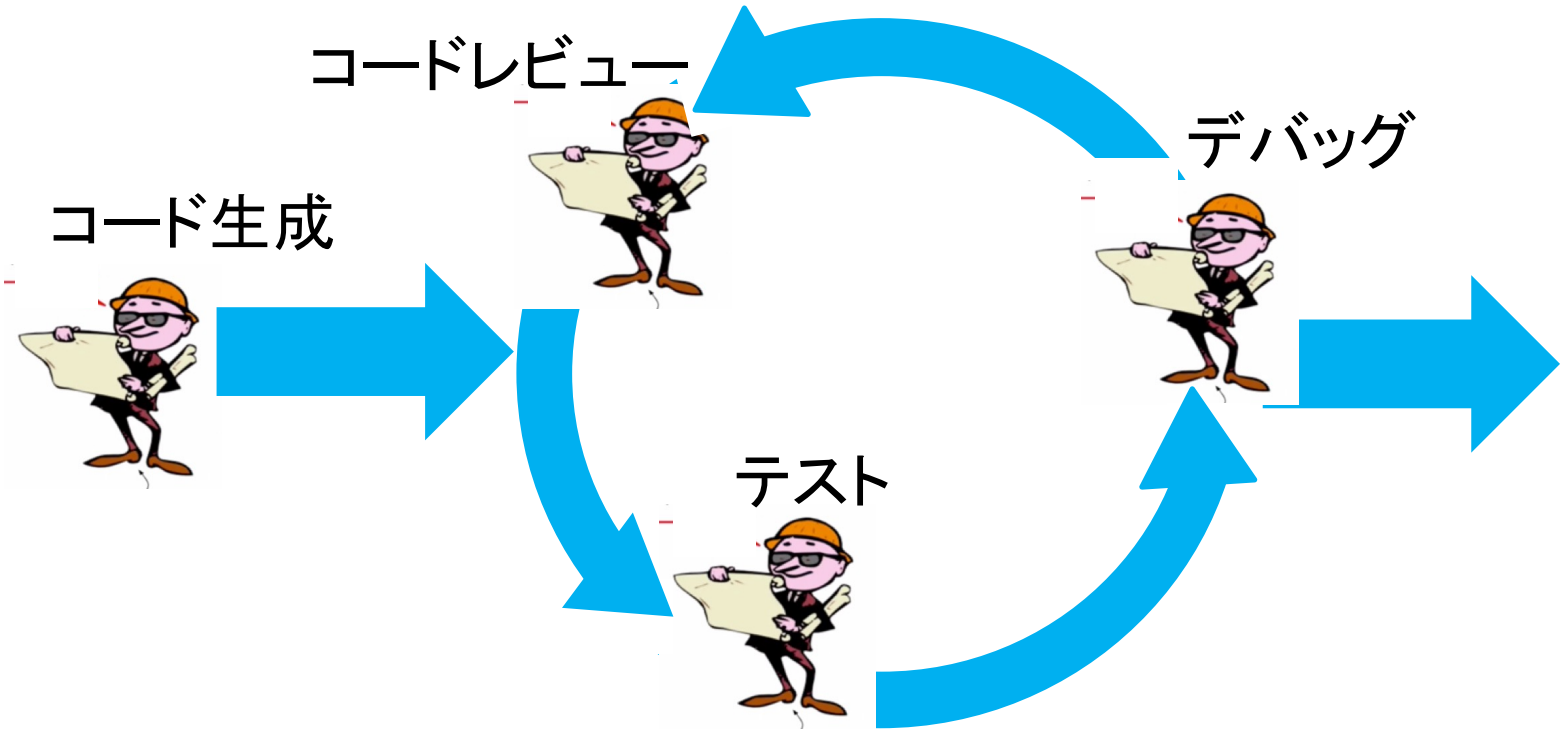
そうした「進化」を確認するためにも、まず、最初に、このモデルでは何が「簡略化」されているのかを考えてみたいと思います。

この「コード開発パイプライン」のモデルには、「テスト」工程も「デバッグ」工程もありません。



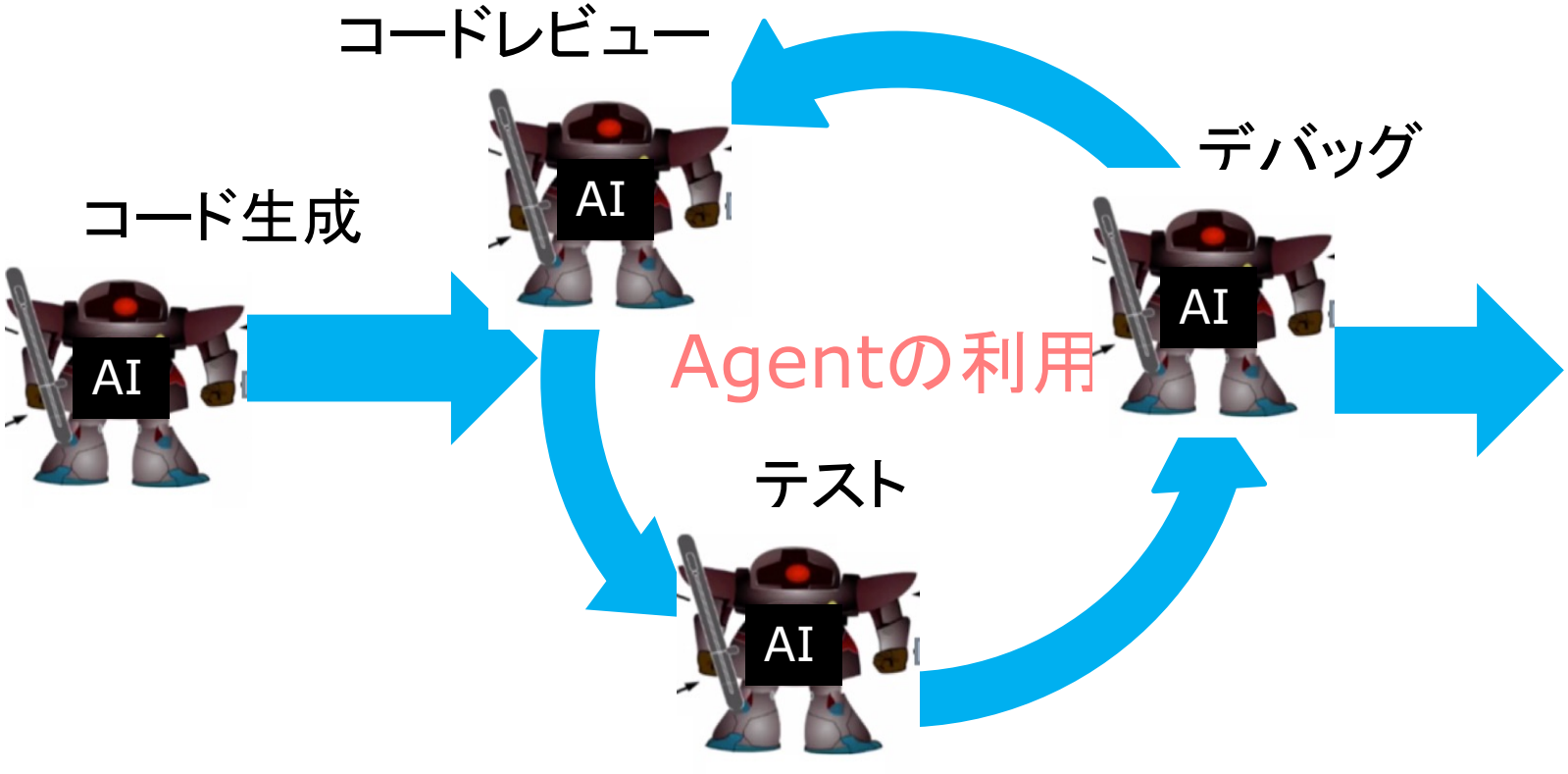
人間によるソフトウェア開発サイクルのモデル

人間が行っているソフトウェア開発サイクルのモデルは、次のようなものです。ここでもいろんな単純化が行われているのですが、それについては、セミナーの別のパートで紹介します。



現在の開発自動化の取り組み

現在、AIベンダーが行っていることは、先の単純化された開発モデルの中で、人間が果たしている役割を AI Agentに置き換えることだと思います。



現在(2025年5月時点の公開情報に基づく))の開発自動化の取り組みについては、今回のセミナーに向けたセッション「ソフトウェア開発へのAI Agent導入の動向」の資料で、

https://drive.google.com/file/d/1NZKV4hTy8SrUuJvvT0Sxllk9boAjg6eY/view

ソフトウェア開発におけるAIエージェントの導入の現状を、特に

- コード補完
- テスト
- デバッグ
- コードレビュー

各領域に焦点を当てて紹介しています。

開発自動化の取り組みの現在の時点でのまとめ

主要AIエージェントの機能と性能(SWE-bench)は、次の表にまとめました。

これらについては、「AI Agent導入の動向（音声による概要）」が簡便にまとめています。参照ください。

https://www.marulabo.net/wp-content/uploads/2025/05/AI%E3%82%A8%E3%83%BC%E3%82%B8%E3%82%A7%E3%83%B3%E3%83%88%E3%81%AE%E5%B0%8E%E5%85%A5.mp3

エージェント (モデル)	**SWE-bench Resolved % (セット)**	結果日付	主なエージェント機能
SWE-agent 1.0 (Claude 3.7 Sonnet)	33.83 (Verified)	2025-02-27	ACI、ファイル操作、テスト実行
OpenHands + CodeAct v2.1 (claude-3-5-sonnet-20241022)	29.38 (Verified)	2024-11-03	コード編集、コマンド実行、ブラウザ操作
Amazon Q Developer Agent (v20241202-dev)	29.99 (Verified)	2025-01-31	不明（商用エージェント）
Devin (GPT-4o ベースと推定、未検証)	13.86 (Full, unverified)	2024年初頭	計画、コーディング、デバッグ、デプロイ（シェル、エディタ、ブラウザ統合）
SWE-agent + GPT 4 (1106)	12.47 (Test)	2024-04-02	ACI、ファイル操作、テスト実行

サンプル・プログラムを読む
主な流れ

サンプルプログラムは、「簡略化されたコード開発パイプライン」を ADKのSequentialAgent を用いて実装しています。

```
# This agent orchestrates the pipeline by running the sub_agents in order.
code_pipeline_agent = SequentialAgent(
    name="CodePipelineAgent",
    sub_agents=[code_writer_agent, code_reviewer_agent, code_refactorer_agent],
    description="Executes a sequence of code writing, reviewing, and refactoring.",
    # The agents will run in the order provided: Writer -> Reviewer -> Refactorer
)
```

サンプル･プログラムを読む
三つのsub-agent

このパイプラインは、以下の3つのサブエージェントで構成されています。

1. **Code Writer Agent**
ユーザーの要求に基づいて初期コードを生成します。与えられた指示に従い、Pythonコードブロックのみを出力し、その結果をgenerated_code というキーで状態（state）に保存します。

2. **Code Reviewer Agent**
前の Code Writer Agent が生成したコードを状態から読み取り、そのコードに対して建設的なフィードバックを提供します。レビュー結果は箇条書きで簡潔に出力され、review_commentsというキーで状態に保存されます。

3. **Code Refactorer Agent**

元のコードと Code Reviewer Agent からのレビューコメントを状態から読み取り、それに基づいてコードをリファクタリングします。最終的なリファクタリングされたコードは、refactored_code というキーで状態に保存されます。

CodeWriterAgentのプログラム

```
# Code Writer Agent
# Takes the initial specification (from user query) and writes code.
code_writer_agent = LlmAgent(
    name="CodeWriterAgent",
    model=GEMINI_MODEL,
    # Change 3: Improved instruction
    instruction= ... ... ,（後述）
    description="Writes initial Python code based on a specification.",
    output_key="generated_code" # Stores output in state['generated_code']
)
```

CodeReviewerAgentのプログラム

```
# Code Reviewer Agent
# Takes the code generated by the previous agent (read from state) and provides feedback.
code_reviewer_agent = LlmAgent(
    name="CodeReviewerAgent",
    model=GEMINI_MODEL,
    # Change 3: Improved instruction, correctly using state key injection
    instruction= ... ... ,（後述）
    description="Reviews code and provides feedback.",
    output_key="review_comments", # Stores output in state['review_comments']
)
```

CodeRefactorerAgentのプログラム

```
# Code Refactorer Agent
# Takes the original code and the review comments (read from state) and refactors the code.
code_refactorer_agent = LlmAgent(
    name="CodeRefactorerAgent",
    model=GEMINI_MODEL,
    # Change 3: Improved instruction, correctly using state key injection
    instruction= ... ... ,  （後述）
    description="Refactors code based on review comments.",
    output_key="refactored_code", # Stores output in state['refactored_code’]
)
```

CodeWriterAgent への instruction

You are a Python Code Generator.

Based *only* on the user's request, write Python code that fulfills the requirement.

Output *only* the complete Python code block, enclosed in triple backticks (```python ... ```).

Do not add any other text before or after the code block.

CodeWriterAgent への instruction
日本語

あなたは Python コードジェネレータです。
ユーザーの要求のみに基づいて、その要件を満たす Python コードを記述してください。
完全な Python コードブロックのみを、3 つのバックティック（```python ... ```）で囲んで出力してください。
コードブロックの前後に他のテキストを追加しないでください。

CodeReviewerAgentへの instruction

You are an expert Python Code Reviewer.

Your task is to provide constructive feedback on the provided code.

Code to Review:

```python
{generated_code}
```

Review Criteria:

1. **Correctness:** Does the code work as intended? Are there logic errors?
2. **Readability:** Is the code clear and easy to understand? Follows PEP 8 style guidelines?
3. **Efficiency:** Is the code reasonably efficient? Any obvious performance bottlenecks?
4. **Edge Cases:** Does the code handle potential edge cases or invalid inputs gracefully?
5. **Best Practices:** Does the code follow common Python best practices?

Output:

Provide your feedback as a concise, bulleted list. Focus on the most important points for improvement.

If the code is excellent and requires no changes, simply state: "No major issues found."

Output *only* the review comments or the "No major issues" statement.

CodeReviewerAgentへの instruction
日本語

あなたは Python コードレビューのエキスパートです。
あなたの仕事は、提供されたコードについて建設的なフィードバックを提供することです。

レビューするコード:
```python
{generated_code}
```

レビュー基準:

1. **正確性:** コードは意図したとおりに動作するか？論理的なエラーはないか？
2. **可読性:** コードは明確で理解しやすい？PEP 8 スタイルガイドラインに従っているか？
3. **効率性:** コードは適度に効率的か？明らかなパフォーマンスのボトルネックはないか？
4. **エッジケース:** コードは潜在的なエッジケースや無効な入力を適切に処理するか？
5. **ベストプラクティス:** コードは一般的な Python のベストプラクティスに従っているか？

出力:

フィードバックを簡潔な箇条書きで記載してください。改善すべき最も重要な点に焦点を当ててください。

コードに問題がなく、変更の必要がない場合は、「大きな問題は見つかりませんでした」と記載してください。

レビューコメントまたは「大きな問題は見つかりませんでした」という記述のみを出力してください。

CodeRefactorerAgentへの instruction

You are a Python Code Refactoring AI.

Your goal is to improve the given Python code based on the provided review comments.

```
**Original Code:**
```python
{generated_code}
```

Review Comments:
{review_comments}
```

Task:

Carefully apply the suggestions from the review comments to refactor the original code.

If the review comments state "No major issues found," return the original code unchanged.

Ensure the final code is complete, functional, and includes necessary imports and docstrings.

Output:

Output *only* the final, refactored Python code block, enclosed in triple backticks (```python ... ```).

Do not add any other text before or after the code block.

CodeRefactorerAgentへの instruction
日本語

あなたは Python コードリファクタリング AI です。

あなたの目標は、提供されたレビューコメントに基づいて、与えられた Python コードを改善することです。

元のコード:
```python
{generated_code}
```

レビュー コメント:
{review_comments}

タスク:

レビューコメントの提案を慎重に適用して、元のコードをリファクタリングして。

レビューコメントに「大きな問題は見つかりませんでした」と記載されている場合は、元のコードをそのまま返して。

最終的なコードが完全で機能し、必要なインポートとドキュメント文字列が含まれていることを確認して。

出力:

最終的なリファクタリング済みの Python コードブロックのみを、3 つのバックティック（```python ... ```）で囲んで出力して。

コードブロックの前後には、他のテキストを追加しないでください。

SDLC 自動化への課題

今回見たサンプルは、ソフトウェア開発の自動化へのAI Agent利用のプリミティブなUr-Typeとして見ると興味深い。

ただ、「ソフトウェア開発ライフ・サイクル -- Software Development Life Cycle (SDLC) 全体の自動化」を目指すには、細かな技術的な課題だけでなく、もっと大きな課題があることを示しているように思う。

それについては、次回のセッションで取り上げようと思う。

第三部

ソフトウェア開発の課題と未来

第三部： ソフトウェア開発の課題と未来

セミナーの第三部では、こうしたAI Agent を利用したAI開発ツールが抱える問題を取り上げます。

一つには、現在の主要なAI開発ツールは、主にコード補完やチャット支援に重点を置いており、機能は限定的です。

大規模プロジェクトの複雑な構造やモジュール間の連携をAIに直接指示することは、まだできません。何よりも、人間が、機械にどういう形で、ソフトウェアの「仕様」を提示すべきか明確な指針が存在していません。

もう一つには、LLMは驚異的なコード生成能力を示す一方で、生成されたコードは、信頼性の問題をいくつか抱えています。

こうした中で、LLMのコード生成能力と形式手法の厳密性を統合するアプローチに注目が集まっています。こうした動きは、ソフトウェア開発の自動化において革新的な可能性を秘めていると、僕は考えています。

もう一つ、大事な問題があります。それは、こうした変化が進む中で、人間と機械との関係がどのように変化し、特に人間には、どのようなスキルが必要とされるかということです。

いくつかの方向は提案されているのですが、問題を難しくしているのは、機械が進化するスピードと人間が成長するスピードには、大きなギャップがあることだと思います。

第三部： ソフトウェア開発の課題と未来

人間はAIに何を伝えるのか？

- 大規模プロジェクトへの仕様の提示

LLMと形式手法の統合の動向

- LLMと形式手法の統合の動向
- 形式手法で利用される証明支援システム：CoqとLean

人間と機械の役割の変化を考える

補論 -- Agent論の二つの系譜

- AI Agent 概念の成立と進化
- 数学的 Agent 論


ソフトウェア開発の課題と未来
人間はAIに何を伝えるのか？

人間はAIに何を伝えるのか？

AIエージェントは、大規模ソフトウェア開発に革命をもたらす変革的な可能性を秘めていますが、同時に、複雑な人間の意図をこれらのエージェントに効果的に伝達するという根本的な課題も抱えています。AIの能力が向上するにつれて、人間からAIへの仕様提示の精度がますます重要になります。

このセッションでは、AIエージェントがより自律的になるにつれて、その重要性が増している「人間は機械に何を伝えるべきか？」という問題を考えたいと思います。

“Vibe coding” by Andrej Karpathy 2025/02/02 on X

There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

僕が「バイブ・コーディング」と呼んでいる新しい種類のコーディングがあるんだ。バイブに完全に身を任せ、指数関数を受け入れ、コードの存在すら忘れてしまう。LLM（例えば、Cursor ComposerやSonnet）が良くなりすぎているから可能なんだ。また、私はComposerとSuperWhisperで会話するだけなので、キーボードに触れることすらほとんどない。サイドバーのパディングを半分に減らしてくれ」なんてアホなことを頼むのは、それを探すのが面倒だからだ。いつも "Accept All "で、もう差分は読まない。エラーメッセージが出ても、コメントなしでコピーペーストするだけで、たいていはそれで解決する。コードは普段の私の理解を超えて大きくなっていくから、しばらくは本当に目を通さなければならない。時にはLLMがバグを修正できないこともあるので、私はそのバグがなくなるまで、ただそのバグを回避したり、ランダムな変更を求めたりするだけだ。週末にやるプロジェクトとしては悪くないが、それでもかなり面白い。プロジェクトやウェブアプリを作っているんだけど、別にコーディングしているわけじゃないんだ。

AI agentの登場と「仕様」の性質の変化

従来、仕様は人間の開発者のための文書でした。しかし、AIエージェントの登場により、仕様は自動化されたプロセスのための直接的な入力へと変化しています。この変化は、「効果的な」仕様とは何かを再評価する必要性を生じさせます。

つまり、仕様は人間による検証のために理解可能であると同時に、機械による生成のために解釈可能でなければなりません。目標に基づいてタスクを実行するAIエージェントに関する研究 は、仕様が純粋に手続き的なものから、目標指向の定義へと進化する可能性を示唆しています。

AIエージェントが仕様を自律的な行動の入力として利用するという観察と、従来の仕様が主に人間同士のコミュニケーションと解釈を目的としていたという対比から、人間による可読性と機械による処理可能性という二重の要件が、仕様の構造化と形式化の方法に変化を促していると考えられます。

目標ベースのエージェントは、仕様が「どのように行うか」という命令的なものよりも、「何を達成するか」という宣言的なものになる可能性を示唆しており、これは、これらの二重のニーズのバランスを明示的にとる新しい仕様言語やフレームワークにつながる可能性があります。

ソフトウェア開発ライフサイクルにおける AIエージェントの能力の現状評価

AIエージェントの導入事例として、JPMorgan Chaseでは、AI搭載のコーディングアシスタントを統合し、コード提案、レビュー、デバッグを効率化することで、開発者の生産性を20%向上させたと報告されています。これは、AIが特定のタスクにおいて明確な成果を上げていることを示しています。

しかしながら、AIエージェントの能力には大きなばらつきがあることも明らかになっています。カーネギーメロン大学の研究では、Claude 3.5 SonnetやGPT-4o、Gemini 2.0 Flashといった最先端のAIエージェントが、シミュレートされた作業環境において、タスクのごく一部しか完了できず（Claude 3.5 Sonnetでさえ24%）、その性能は期待を大きく下回りました。

これは、SWE-benchのような特定の狭いベンチマークでの高い性能とは対照的です。

具体的な失敗例としては、ポップアップウィンドウを閉じられない、指定された時間待機することを誤解釈する、常識や社会的スキルの欠如、ウェブブラウジング能力の低さ、タスク完了に関する自己欺瞞などが挙げられています。

この研究は、現在のAIエージェントが、コーディングのような特定の技術的タスクには長けているものの、大規模ソフトウェア開発の多くの側面で必要とされる総合的な理解力や実世界のインタラクション能力に欠けていることを示唆しています。

"TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks"
https://arxiv.org/abs/2412.14161
2025/05/19 公開

AnthropicによるClaude Codeに関する研究 も、AIの利用実態に関する興味深い洞察を提供しています。

この研究では、Claude Codeの会話の79%が「自動化」（AIがタスクを直接実行）であり、「拡張」（AIが人間の能力を補強・共同作業）は21%でした。これは、エージェント型AIの普及に伴い、タスクの自動化が一層進む可能性を示唆しています。

また、ウェブ開発（JavaScript、HTML、UI/UX）での利用が多く、比較的単純なアプリケーション開発の分野でAIによるディスラプションが早期に起こる可能性が示されています。採用状況については、スタートアップ企業（33%）が既存の大企業（13%）よりも積極的にこれらのツールを活用しており、俊敏な組織ほど最先端AIツールの恩恵を受けやすいことがうかがえます。

Anthropic Economic Index: AI’s Impact on Software Development
https://www.anthropic.com/research/impact-software-development

「ラストマイル」問題：複雑な人間の意図を解釈する上でのAIの現在の限界

AIエージェントは、自然言語の固有の曖昧さや、人間が直感的に理解するニュアンスに富んだ、しばしば明言されない文脈の扱いに苦慮しています。

カーネギーメロン大学の研究で示されたように、AIは多くの実世界のソフトウェア開発タスクに不可欠な常識的推論を欠いています。

また、AIエージェントは、共同開発やユーザーニーズの理解において極めて重要となり得る、ニュアンスに富んだ人間とのインタラクションに苦労します。

これらの観察から、AIエージェントは、特定の文脈におけるコード生成のような、明確に定義された狭いタスクにおいては高い能力を発揮します（例：GitHub Copilotによる関数本体の提案）。

しかし、タスクがより広範になり、複雑な環境（完全なOSやウェブブラウザなど）とのインタラクションを必要としたり、常識的な推論に依存したりする場合、その性能は著しく低下します。

これは、現在のAIが汎用的なソフトウェアエンジニアというよりは、むしろ専門化されたツールの集合体に近いことを示唆しています。

これらの失敗は、多くの場合、文脈理解の欠如、常識の欠如、あるいは訓練データを超えた複雑で予測不可能な環境をナビゲートする能力の欠如に起因しています。

大規模開発は、多くが広範な理解と適応性を必要とする、多数の相互に関連したタスクを含みます。

現在のAIの能力プロファイルは、AIがコンポーネントの支援はできるものの、複雑で進化する要件の下でこれらをまとまりのある大規模システムに編成し統合することは、依然として人間の重要な仕事であることを示唆しています。

これらの「統合的」および「適応的」な側面に対する意図の仕様化が、主要な課題となります。

AIエージェントにおける「知能のコスト」

カーネギーメロン大学の研究では、最も性能の高かったClaude 3.5 Sonnetがタスクあたり平均30ステップ近く、計算コストとして6.34ドルを要したのに対し、性能で劣るGeminiはタスクあたりわずか79セントだったと報告されています。

これは、ソフトウェア開発におけるAIエージェントの評価に新たな次元、すなわち、大規模プロジェクトで多数のタスクに対して高性能だがリソース集約的なモデルを使用することの経済的実現可能性という次元を導入します。

高性能なAIエージェントがタスクあたりの計算コストにおいて著しく高価になる可能性があるという観察 を踏まえると、大規模ソフトウェア開発には何千、何百万もの「タスク」（コードスニペットから設計判断まで）が含まれることを考慮しなければなりません。

もし最も先進的なAIエージェントへの依存が広範になれば、累積的な計算コストは相当なものになり、一部の組織やプロジェクトタイプにとっては生産性の利点を上回る可能性があります。

これは、人間がAIに何をすべきかだけでなく、タスクにどれだけリソース集約的に取り組むべきかを指定する必要性、あるいは重要度の低いタスクをより安価で能力の低いモデルが処理する階層型AIシステムの必要性につながるかもしれません。

これは、「機械に何を伝えるべきか」という問題にさらなる層を加えることになります。

人間の意図をAIエージェントに伝える現在の方法

- **コード生成のためのAIプロンプトの最適化**: 具体的に記述し、文脈（言語、依存関係、入出力）を提供し、反復的に改良することの重要性が指摘されています。例えば、「ソート関数を書いて」という一般的な要求よりも、「クイックソートを用いてO(n log n)の計算量で配列をソートするPython関数を書いて」というプロンプトの方が効果的です。
- **「Vibe Coding」:** 特にUI/UXにおいて、開発者が望ましい結果を自然言語で記述し、AIが実装の詳細を担当するというアプローチです。これは、より高レベルで抽象的な意図伝達への移行を示唆しています。
- **フィードバックループ:** ユーザーが検証を行い、エラーをAI（例：Claude Code）に送り返すという一般的なインタラクションパターンであり、意図伝達を反復的な改良プロセスとしています。これはClaude CodeではClaude.aiの約2倍（35.8%対21.3%）一般的でした。
- **入力モダリティ:** AIエージェントは、テキスト入力、音声コマンド、視覚データからデータを収集します。

現在の方法は大規模システムでは不十分

- **自然言語固有の曖昧性:** 自然言語は使いやすい一方で、特に大規模システムの複雑な要件については、複数の解釈を許容しがちです。
- **人間の指定者における「知識の呪い」:** 人間は「自明」と考える詳細を省略しがちですが、それはAIにとっては自明ではありません。
- **プロンプトエンジニアリングのスケーラビリティ:** 小規模なタスクには効果的ですが、大規模システム内の何千ものコンポーネントに対して正確なプロンプトを作成し管理することは、大きな課題となります。
- **一貫性の維持:** 多数のプロンプトや「Vibe Coding」セッションを通じて伝えられる意図が、大規模で進化するコードベース全体で一貫性を保つことは困難です。

これらの課題を考慮すると、現在の意図伝達方法は、特に大規模で複雑なシステムにおいては、人間の意図をAIに正確かつ完全に伝える上で限界があると言えます。

「対話」を通じて意図を伝えるというアプローチ

「フィードバックループ」の普及や、プロンプトの反復的な改良の必要性は、現在のAIエージェントへの意図伝達が、完成した設計図を手渡すようなものではなく、むしろ継続的な会話に近いことを示唆しています。

AIが試行し、人間が修正または改良し、このサイクルが繰り返されるのです。従来の仕様化が要件から設計へのより一方向的な「モノローグ」と見なせるのとは対照的です。

特に大規模開発における複雑なタスクでは、初期の仕様は不完全であったり、部分的に曖昧であったりする可能性があります。それを解釈しようとするAIの試みと、人間の修正が、仕様化プロセス自体の一部となるのです。

これは、開発者のワークフロー、ツール（これらの「対話」の反復的な改良と履歴追跡のためのより良いサポートが必要）、そして開発者に必要なスキル（効果的なAI「コーチング」へとシフト）に影響を与えます。

大規模システムでは、これらの対話を複数の抽象化レベルで管理するための構造化された方法が必要になるかもしれません。

意図の階層性

さらに、意図の具体性に関して、新たな階層構造が出現しつつあると考えられます。

「Vibe Coding」は非常に高レベルで抽象的な意図の形式を表し、詳細なプロンプトエンジニアリングはより具体的です。

大規模システムでは、単一の具体性レベルで十分であるとは考えにくく、高レベルのアーキテクチャ上の意図は抽象的に指定され、コンポーネントレベルの詳細はより正確なプロンプトや、場合によっては形式的な仕様を必要とする階層が出現するかもしれません。

様々な意図伝達方法が存在し、抽象的な「Vibe Coding」6 から具体的なプロンプト まであるという観察と、大規模システムが本質的にその構造（コンポーネント、モジュール、サブシステム、システム全体の目標）において階層的であることを考慮すると、そのようなシステムに対する効果的な意図の仕様化は、この階層を反映し、異なる時点で異なる方法/詳細レベルを使用する必要があるという仮説が成り立ちます。

将来のツールは、アーキテクトが高レベルの目標と制約を定義し、開発者がこれらを特定のモジュールに取り組むAIエージェントのためのより詳細な指示に改良することを可能にする、多レベルの仕様化をサポートする必要があるかもしれません。

モジュラー設計と仕様の階層性
ソフトウェア工学の原則とAI agent

ソフトウェア工学の原則は、AIに「何を」構築するかを直接指示するものではありませんが、人間の意図をどのように構成し分解すべきかという「構造」を提供します。モジュール性や階層的仕様といった原則を意図の仕様化に適用することで、AIにとってより管理しやすくなるという考え方があります。

例えば、大規模システム全体を一度に指定するのではなく、モジュールとその相互作用を指定したり、分解される高レベルの目標を指定したりします。これは、AIがこれらの構造化された仕様の範囲内で動作することを学習し、人間がこれらの構造を活用する方法で意図を指定することを学習するという、共進化を示唆しています。「仕様」自体がモジュール化され、階層化されるのです。

インターフェースの問題から見えてくるもの

しかし、AIが生成するモジュラーシステムにおいては、「インターフェース定義」という新たな課題が生じます。モジュール性は重要ですが、その有効性はモジュール間の明確に定義された安定したインターフェースにかかっています。

AIエージェントがこれらのモジュールを生成する場合、これらのインターフェースの一貫性、正確性、安定性を確保することが新たな課題となります。AI生成モジュールの他のモジュール（AI生成または人間生成）に対する「契約」をどのように指定すればよいのでしょうか。

意図の仕様化には、AIが利用可能な厳密なインターフェース定義が含まれる必要があります。複雑なアーキテクチャや再利用可能なコンポーネントが必要な大規模システムの場合、自然言語で機能を記述するだけでは不十分です。

意図の仕様化は、コードが機能的に何をすべきかだけでなく、コードをどのように構造化すべきかについてもAIを明示的に導く必要があリマス。これは、単なる平易な自然言語以上のものが必要であることを示しています。

これは、形式手法やDSLの潜在的な役割につながります。

仕様の形式化

AIの創発的または予期せぬ振る舞いの可能性を考えると、形式仕様は、AIが「何をすべきか」そして「何をしてはならないか」を定義する、正確で曖昧さのない「契約」として機能することができます。これは、安全性と倫理的配慮にとって特に重要です。

自然言語は初期のアイデア出しには適していますが、形式手法は意図の譲れない境界を固めることができます。

大規模なシステムの場合、ハイブリッドなアプローチが必要になるかもしれません。一般的な要件には自然言語を使用し、主要な特性とインターフェースには形式仕様を使用します。これにより、意図は「伝える」ことよりもAIを「制約する」ことに関するものになります。

XAI --「説明可能なAI」

AIが失敗したのか、それとも我々の仕様が悪かったのかをどのように知るかという問題を考えると、合成プロセスが説明可能であれば、仕様からコードへのAIの「推論」を追跡できるという解決策が浮かび上がります。

XAI は、AIが仕様をどのようにコードに変換したかを明らかにすることができ、人間が自分の意図が誤解されたり、十分に指定されていなかったりした箇所を特定することを可能にすることを目指します。

Building Reliable AI Agents with Modular and Scalable Frameworks
https://magnimindacademy.com/blog/building-reliable-ai-agents-with-modular-and-scalable-frameworks/

AI agent の為のDSL

DSLは、人間が複雑なドメイン固有の意図を、汎用言語や自然言語よりも簡潔かつ曖昧さなく表現することを可能にします。AIエージェントにとって、DSLはより制約され、意味的に豊富な入力を提供し、潜在的により正確で信頼性の高いコード生成につながります。

AIエージェントは、特に複雑でドメインが豊富な大規模システムの場合、オープンエンドな自然言語よりも特定のDSLをより効果的に「理解」するように訓練または設計できます。

将来の研究は、DSLとそれを解釈できるAIエージェントを共同設計することに焦点を当てる可能性があり、「機械に何を伝えるべきか」をより強力かつ対象を絞ったものにするでしょう。

「人間は機械に何を伝えるべきか」

現状では、AIは特定のタスクにおいて生産性向上に貢献しているものの、複雑でニュアンスに富んだ、あるいは大規模な人間の意図を完全に把握する能力には限界があります。

この「意図のギャップ」を埋めるためには、ソフトウェア工学の基礎原則を適応させるとともに、プログラム合成、形式手法、説明可能なAIといった分野における新たな研究を積極的に取り入れていく必要があります。

特に、自然言語の曖昧さを克服し、AIにとって解釈可能かつ人間にとって検証可能な仕様をどのように構築するかは、重要な課題です。


ソフトウェア開発の課題と未来
LLMと形式手法の統合の動向

ソフトウェア開発のパラダイムシフト

近年の大規模言語モデル（LLM）の進化は、ソフトウェア工学の領域に大きな変革をもたらしつつある。

LLMは、人間が生成するような自然なテキストだけでなく、ソースコードの生成においても顕著な能力を示しており 、既存コードの修正や関数全体の新規生成といったタスクに広く活用され始めている。

これは、ソフトウェア開発のあり方におけるパラダイムシフトと言える。

LLM固有の信頼性の課題

その高い能力にもかかわらず、LLMには固有の信頼性の課題が存在する。

LLMは、構文的には正しく、意味論的にもっともらしく見えるコードを生成するが、それが期待通りに実行される、あるいは指定された要件を満たすとは限らない。また、生成されたコードが特定の実行パスでのみエラーを露呈する場合など、その特定と修正が困難な場合がある。

これは生成されたコードの信頼性を損ない、重大なリスクを導入する可能性がある。

形式手法への期待

このようなLLMの信頼性の課題に対し、ソフトウェアの完全性を保証する手段として形式手法（Formal Methods）が有望視されている。

形式手法は、数学的な論理を用いてシステム特性に関する決定的な保証を提供するものであり、システムがその特性を満たすことを形式的に証明するか、具体的な反例を提示することで、誤検出を排除することができる。

この厳密性は、特に安全性やセキュリティが重視されるクリティカルなソフトウェアにおいて不可欠である 9。

LLMと形式手法の統合の狙い

LLMの創造性と形式手法による検証の厳密性を統合するこのハイブリッドアプローチの核心は、LLMの生成能力とパターン認識能力を活用しつつ、その信頼性の欠如を形式手法の数学的な確実性によって補完することにある。

これにより、より高い信頼性と正確性をもってソフトウェア開発タスクを自動化することを目指す。

LLMと形式手法の統合を推進する動きは、単にコード品質を向上させるだけでなく、従来は形式手法単独の高いコストと専門知識、あるいはAI単独の信頼性の低さゆえに実現不可能だった、ソフトウェア工学における新しい形態の自動化を可能にすることに根本的な意義がある。

LLMと形式手法の統合がもたらすもの

LLMは、形式手法において人間にとって煩雑な仕様記述や証明生成の一部を自動化することで、形式手法利用の敷居を下げることができる。一方、形式手法はLLMの出力を検証する。

この相乗効果により、特定モジュールの検証済みコード生成、仕様からのテストオラクル自動生成、あるいはレガシーコードの形式検証支援といったタスクの自動化が期待される。

これにより、形式検証はクリティカルシステム向けのニッチな活動ではなく、AIによって強化され、通常のソフトウェア開発ライフサイクルに広く組み込まれる未来が展望される。

LLMの説明可能性の欠如を補う

さらに、LLMの「ブラックボックス」的な性質は、説明可能な形式手法との統合を動機付ける重要な要因である。

AIが生成した成果物、特に影響の大きいソフトウェアにおいては、透明性と検証可能性が最優先事項となる。LLMがコードや証明を生成した際、形式検証器はそれをチェックできる。正しければ形式証明が説明となり、誤っていれば検証器がエラーを特定する。

この組み合わせは、LLM固有の説明可能性の欠如という倫理的・実践的な主要懸念に対処する。これは、規制産業や安全性が重要なシステムにおいて、「AIがそう言ったから」という理由が許容されない場合に極めて重要であり、検証可能なAIは監査可能なAIとなる。

相互に相補的な役割を担う

LLMの限界（信頼性の低さ、形式的保証の欠如）と従来の形式手法の限界（スケーラビリティ、高い手作業コスト）は、著しく相補的である。この相補性が、両者の統合研究の根本的な推進力となっている。

LLMは、形式手法において人間にとって煩雑な成果物（コード、仕様、証明ステップ）の生成を自動化できる。形式手法は、LLMの出力を検証し、欠けている厳密性を提供する。

各アプローチが他方の弱点を直接的に補うため、強力な相乗効果の可能性が生まれる。この統合が成功すれば、どちらか一方のアプローチだけでは達成できなかった、より強力で信頼性の高いソフトウェア自動化ツールが実現する「両方の世界のベスト」シナリオにつながる可能性がある。

形式的証明の二つの主要技術

形式的証明には、次の二つの技術がある。

定理証明 (Theorem Proving): システムの特性やプログラムの振る舞いを数学的な定理として表現し、これらの定理の形式的な証明を構築する。Lean、Coq、Isabelle といった証明支援系が用いられる。これらのシステムは、証明の正しさをほぼ瞬時に検証できる。

モデル検査 (Model Checking): システムモデルの全ての可能な状態を系統的に探索し、与えられた特性が成立するかどうかを検査する。有界モデル検査（Bounded Model Checking, BMC）は、探索を事前に定義された境界内に限定し、SAT/SMT ソルバーを利用する。

統合システムにおける形式手法の選択

形式手法の選択,定理証明 対 モデル検査は、対処しようとしているLLMの特定の弱点やタスクの性質によってしばしば決定される。

例えば、定理証明器は数学的推論や証明生成タスク（APOLLO、DeepSeek Prover-V2）に一般的であり、論理ベースの検証（モデル検査やSMT解決に近い）はLLM応答の事実精度（AWS Automated Reasoning）に使用される。

以下、この分野での先駆的試みを紹介する。

AWS Automated Reasoning

AWSは、生成AIサービスに自動推論を統合する初の主要クラウドプロバイダーとなることを目指している。

- Amazon Bedrock Guardrails内のAutomated Reasoning checksを利用する。
- 確率論的アプローチではなく、形式数学的論理および定理証明方法論に依存し、与えられた仮定の下で何が証明可能で何が不可能かについて決定的な保証を提供する。
- ドメイン専門家が、ドメイン知識（例：人事ポリシー、運用ワークフロー）をエンコードしたAutomated Reasoning Policiesを平易な言語で作成する。これらのポリシーは論理ステートメントに変換される。

- システムは、これらのポリシーに対してLLM出力を検証し、不正確さや述べられていない仮定を特定し、検証可能な精度の証明を提供する 7。
- ソルバーは、充足可能（特性が成立、例付き）、充足不可能（特性が満たせない、矛盾する制約付き）、または不明のいずれかAmazon Bedrock Guardrails内のAutomated Reasoning checksを利用する。
- 確率論的アプローチではなく、形式数学的論理および定理証明方法論に依存し、与えられた仮定の下で何が証明可能で何が不可能かについて決定的な保証を提供する。

現状と能力

現在ゲート付きプレビュー段階であり、米国西部（オレゴン）AWSリージョンで利用可能。アクセスにはAWSアカウントマネージャー経由での許可が必要。

説明可能な検証結果を生成する：有効、無効、またはデータなし、詳細な調査結果、考慮されたルール/変数、無効なコンテンツに対する修正提案付き。

平易な言語でのポリシー入力を可能にしつつ、数学的に厳密な検証を維持するハイブリッドアーキテクチャの一部である。

AWSは、暗号実装やIAM Access Analyzerの形式検証に内部的にまた自動推論を使用している（LLMが生成したAWS S3バケットアクセスポリシーコードの検証に言及）。

AWSのアプローチは、LLMの内部生成プロセスに直接影響を与えて正しさを保証するのではなく、事前に人間が作成したドメイン知識に対してLLMの出力を事後的に検証することに焦点を当てている。これにより、形式手法はLLMの「保護者」または「ファクトチェッカー」として位置付けられる。

ポリシーはドメイン専門家によって作成され、LLMの出力はその後これらのポリシーに対して検証される。

これは、DeepSeek Prover-V2やAPOLLOのような、LLMに形式的に正しい証明を直接生成させるか、それを修復させることを目指すアプローチとは異なる。

Google APOLLOプロジェクト

LLMを使用して（Leanのような言語で）完全に正しい形式証明を生成するという困難なタスクに取り組む。

これは通常、広範なサンプリング（多数回のプロンプト試行）を必要とする。形式検証システムは証明をほぼ瞬時にチェックできるが、生成がボトルネックとなっている。

Apolloのアプローチ

1. LLMが初期証明を生成。
2. エージェントが証明を分析し、構文エラーを修正（Syntax Refiner、sorryプレースホルダー挿入用のSorrifier）。
3. Leanが証明中の誤りを特定。
4. 失敗した補助定理を分離。
5. 自動ソルバーを活用。
6. 残りのゴールに対して低いトップKバジェットでLLMを呼び出し。
7. 修復された部分証明を再結合し、反復的に再検証。

現状と成果

- 7Bパラメータモデルにおいて、サンプリングバジェットを1000未満に抑えつつ、**miniF2Fベンチマークで75.0%という新たな最先端の精度**を確立。
- **Goedel-Prover-SFTの最先端精度を65.6%に向上**させ、サンプル複雑性を25,600から数百に大幅削減。
- 汎用モデル（o3-mini、o4-mini）の精度を3-7%から40%以上に向上。
- ターゲットを絞ったコンパイラ誘導型の修復が、効率と正しさの両方で劇的な向上をもたらすことを実証。

成功の意義

APOLLOの成功は、LLMと形式検証ツール（Leanコンパイラ）間の緊密なフィードバックループの力を浮き彫りにしている。この「ニューロシンボリック」な協力関係、すなわちシンボリックツールがニューラルモデルを誘導し修正するアプローチは、LLMの生の出力に頼るよりも効果的である。

APOLLOのパイプラインは、誤りを特定し修復を誘導するためにLeanコンパイラを明示的に使用する。これは、検証器を最終チェックとしてのみ使用する可能性のあるアプローチとは対照的であり、ここでは検証器が生成/修復プロセスに積極的に参加する。

コンパイラは、LLMがその試みを洗練させるために使用できる、正確で実行可能なフィードバックを提供する。

この反復的な、コンパイラをループに組み込んだアプローチは、形式数学や厳密なAPIや仕様への準拠を必要とする複雑なコード生成のような、高度に構造化されたドメインでLLMを効果的にするための鍵となるパラダイムである可能性が高い。

APOLLOは、Goedel-Prover-SFTのサンプル複雑性を25,600から数百に削減した。高いサンプル複雑性は、LLMを何度も実行することを意味し、多大な計算コストと時間を要する。

「サンプル複雑性」の大幅な削減は、実用上極めて重要な成果である。これにより、LLMベースの定理証明が計算的により実現可能かつ経済的に実行可能になり、実用化に近づいた。

生成プロセスをより「インテリジェント」にすること（力ずくのサンプリング対誘導型修復）により、APOLLOはLLMをより効率的に使用する。

この効率向上は、探索空間が広大な、より大規模で複雑な定理やソフトウェア検証タスクに取り組む上で不可欠である。また、これらの技術を使用したい研究者や開発者にとっての参入障壁を下げることにもつながる。

DeepSeek Prover-V2
再帰的定理証明パイプライン

1. **補助定理分解 (Lemma Decomposition):** 大規模汎用モデル**DeepSeek-V3**に、複雑な定理をより単純なサブゴール/補助定理に分解させ、自然言語による証明スケッチを生成させると同時に、これらをsorryプレースホルダー付きのLean 4ステートメントに形式化させる。これは人間の証明構築方法を模倣している。
2. **サブゴール証明探索 (Sub-goal Proof Search):** より小規模で特化された**7Bプローバーモデル**（DeepSeek-Prover-V2-7B）を使用して、分解された各サブゴールの証明探索を行い、計算負荷を軽減する。
3. **証明合成とコールドスタートデータ (Proof Composition & Cold-Start Data):** 全てのサブゴールが解決されると、それらの証明を合成して元の問題の完全な形式証明を構築する。この完全な証明を、補助定理分解のためのDeepSeek-V3の思考連鎖（Chain-of-Thought, CoT）と組み合わせることで、**コールドスタート推論データ**を作成する。

現状と成果

- 最先端の性能: DeepSeek-Prover-V2-671BはMiniF2F-testで88.9%のパス率（Pass@8192）を達成。
- PutnamBenchの658問中49問を解決。
- ProofNet-test問題の37.1%を解決。
- AIMEコンペティションからの15問を含む325の形式化された問題からなる新しいベンチマークProverBenchを導入（DeepSeek-Prover-V2-671BはこれらのAIME問題のうち6問を解決）。

二つのLLMを使う

DeepSeekが大規模汎用LLM（DeepSeek-V3）を高レベルの推論と分解に使用し、より小規模な特化型LLMを詳細な証明探索に使用するという戦略は、能力と計算コストのバランスを取るためのインテリジェントなアーキテクチャ上の選択である。

この階層的アプローチは、人間の専門家による問題解決を模倣している。DeepSeek-V3（大規模モデル）が分解を行い、7Bモデルがサブゴールの証明探索を行う。

大規模モデルは広範な理解と計画（非形式的推論、スケッチ生成）に優れている。小規模モデルは、特定のより狭いタスク（形式証明ステップ生成）に対してより効率的にファインチューニングできる。

これは、モデルアーキテクチャ自体に適用された分割統治戦略である。

これにより、証明の全ての小さなステップに巨大なモデルを使用することによる非効率性を回避する。

このハイブリッドモデルアーキテクチャ（大規模ジェネラリスト＋小規模スペシャリスト）は、他の複雑なAI推論タスクのテンプレートとなり、性能とリソース利用の両方を最適化する可能性がある。

完全な形式証明とDeepSeek-V3からの対応する思考連鎖を統合することで「コールドスタート推論データ」を作成する手法は、複雑な推論タスク向けの高品質な訓練データを生成する巧妙な方法であり、これは特化型LLMの訓練におけるしばしばボトルネックとなる。

成功したサブゴール証明が結合され、この完全な形式証明がDeepSeek-V3のCoT分解と対にされる。これにより、高レベルの非形式的推論（CoT分解）と低レベルの形式証明ステップを明示的に結びつける例が作成される。

このデータは検証済みであるため「ゴールドスタンダード」であり、非形式的な計画を形式的な実行に変換する方法をモデルに直接教える。

Microsoft FLASHプロジェクト

LLMベースのエージェントが複数ステップのタスクを確実に実行するという課題、特にエンタープライズワークフローにおける課題に取り組む。

ステップバイステップモデルにおけるエラー伝播は、単一ステップの高い精度であっても、全体としては低い精度につながることを意味する（例：5ステップのタスク、各ステップ85%の精度 → 全体で44%の精度）

MicrosoftのFLASHプロジェクトは、Leanのような定理証明器という意味での「形式手法」を明示的に使用しているわけではないが、形式的推論の中核原理を具現化している。

すなわち、正確な状態追跡（状態監視）と検証済みの過去の結果からの学習（過去の失敗、すなわち明確に不正な状態からの事後分析統合）である。

これは、形式的推論原理のより運用指向、ワークフロー指向の応用である。「状態監視」と「事後分析統合」を使用する。状態監視は現在の状態に基づいて複雑な指示を分解する。事後分析は過去の失敗から学習する。

形式手法はしばしばシステム状態と有効な遷移の定義を伴う。

失敗からの学習は、エラーステートにつながるパスを特定し回避することに似ており、これはモデル検査（反例）や証明の改良で馴染みのある概念である。

FLASHは、コード検証や数学的証明に直接ではなく、プロセス実行と信頼性のドメインに形式的推論の概念を適用する。

これは、プロセスが時間とともに形式的に健全であることを保証することに関するものである。これは、形式的原則によってしばしば情報提供される堅牢なシステム設計の特徴である、

構造化され、状態を認識し、経験によって修正される実行モデルによってFLASHの「信頼性」がもたらされることを示唆している。これは、コードを超えた形式的思考のより広範な適用可能性を示唆している。

アプローチ

革新的な推論と反省プロセスを用いて、複雑な複数ステップタスクの指示追従精度を向上させる。

- 状態監視 (Status Supervision): ワークフロー実行の現在の状態にアクセスし、現在の状態に基づいて状態依存の指示をトリガーすることで、複雑な指示をより単純なものに分解する。これには、診断状態を評価するための追加の状態推論ステップが含まれる。
- 事後分析統合 (Hindsight Integration): LLMを活用して過去の失敗から自動的に「事後分析」を生成し、より多くの繰り返しケースが処理されるにつれてシステムのパフォーマンスを向上させる。この事後分析をエージェントの反省ステップに統合することで、以前に発生した問題を防止し、ワークフロー実行の信頼性をさらに向上させるのに役立つ。
- エージェントは、人間が作成した指示に確実に従い、ケース情報とツールをチェックして実行計画を生成/改良し、最終的に解決方法を自律的に提案する。

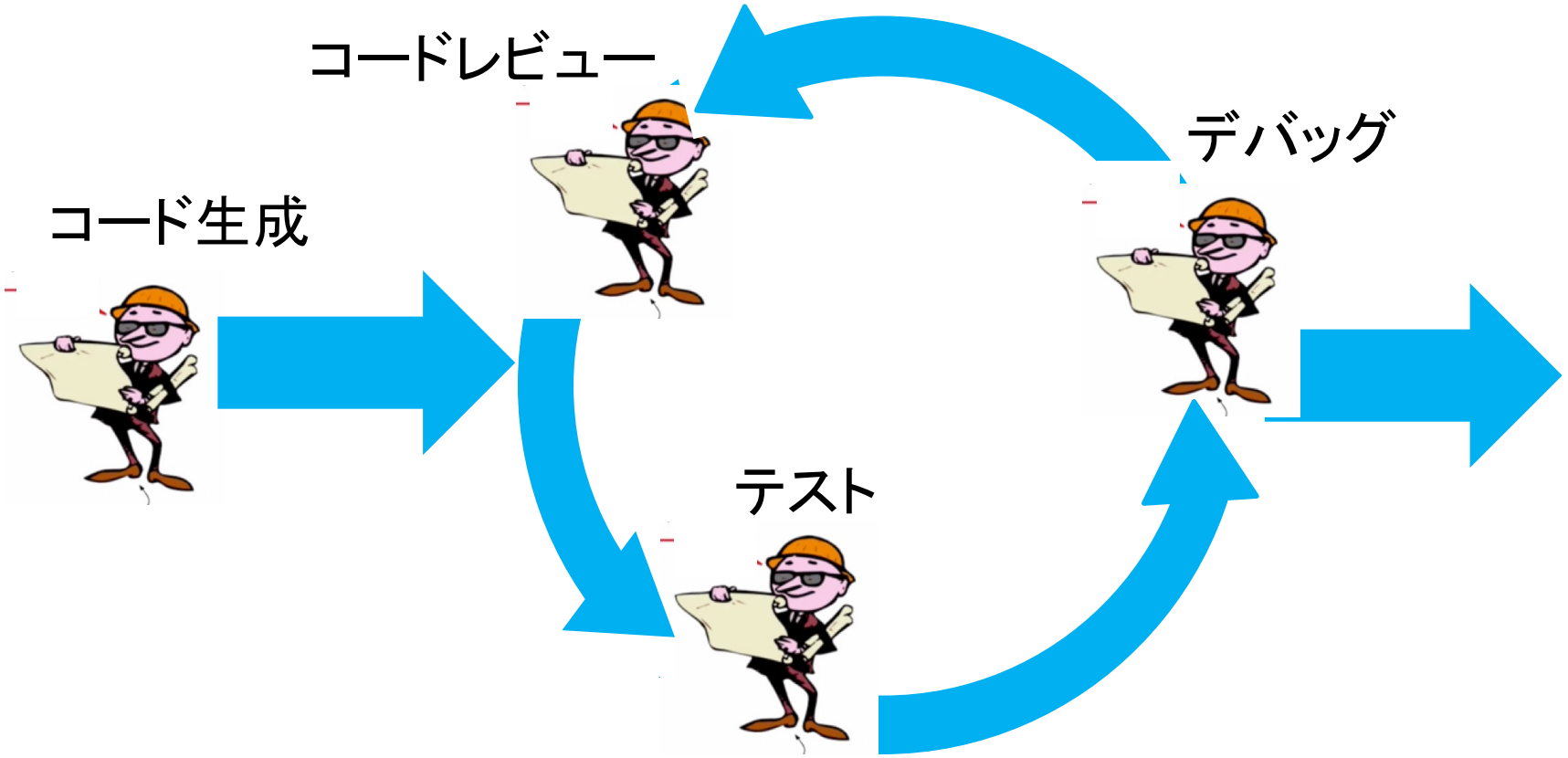
現状と応用

- 実世界のケーススタディで優れた精度を実証：顧客サポート、インシデント管理（クラウドサービスにおける繰り返し発生するインシデントの診断）、ビジネスワークフロー自動化。
- Ignite’24の基調講演でMAIAチップを搭載した製品サポートエージェントとして紹介された。
- Xuchao Zhangらによる論文「FLASH: A Workflow Automation Agent for Diagnosing Recurring Incidents」がFSE '24で発表された。Devjeet Royらによる初期バージョンまたは関連研究「FLASH: A Reliable Workflow Automation Agent」もFSE '24で予定されていた。

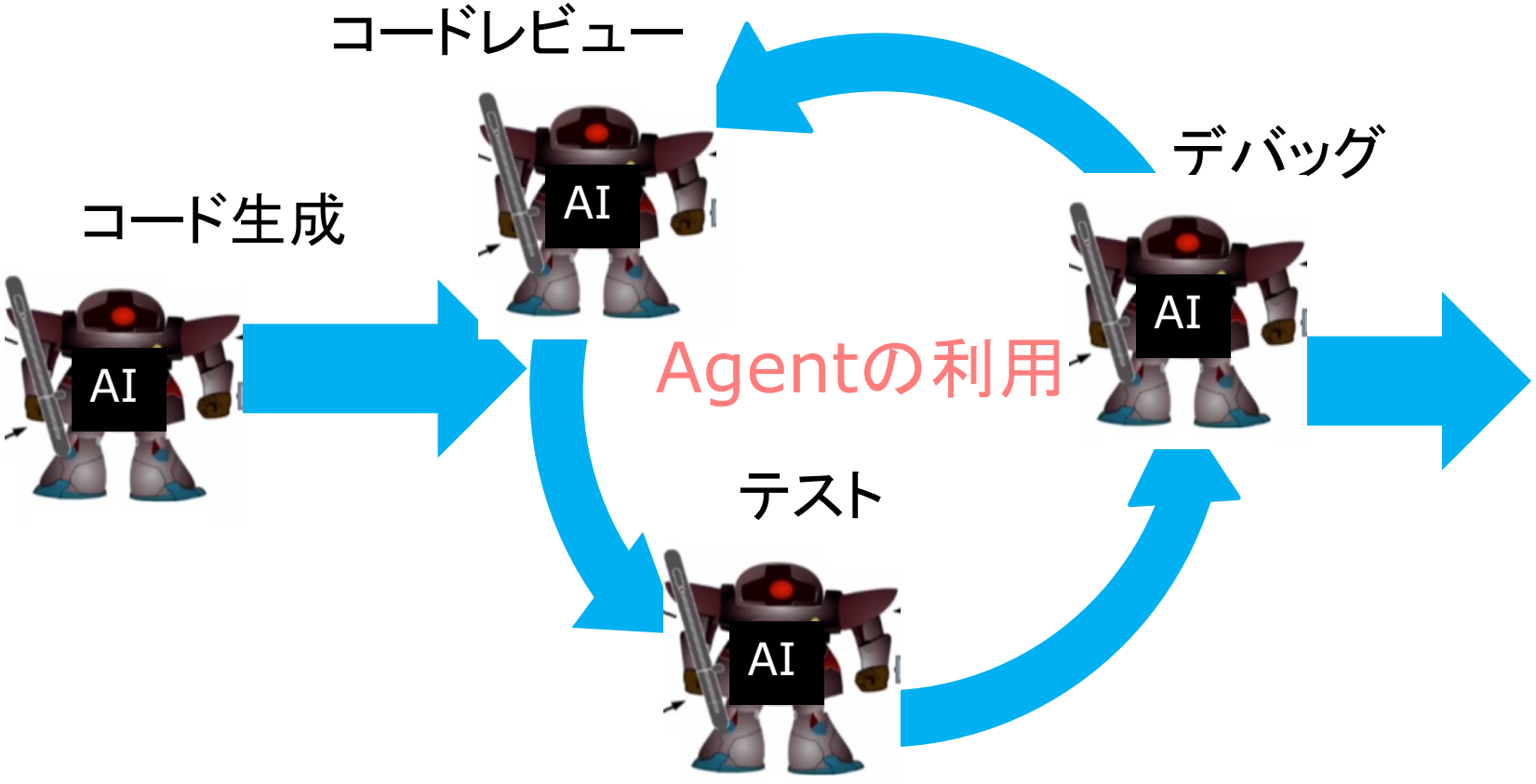
ソフトウェア開発の課題と未来

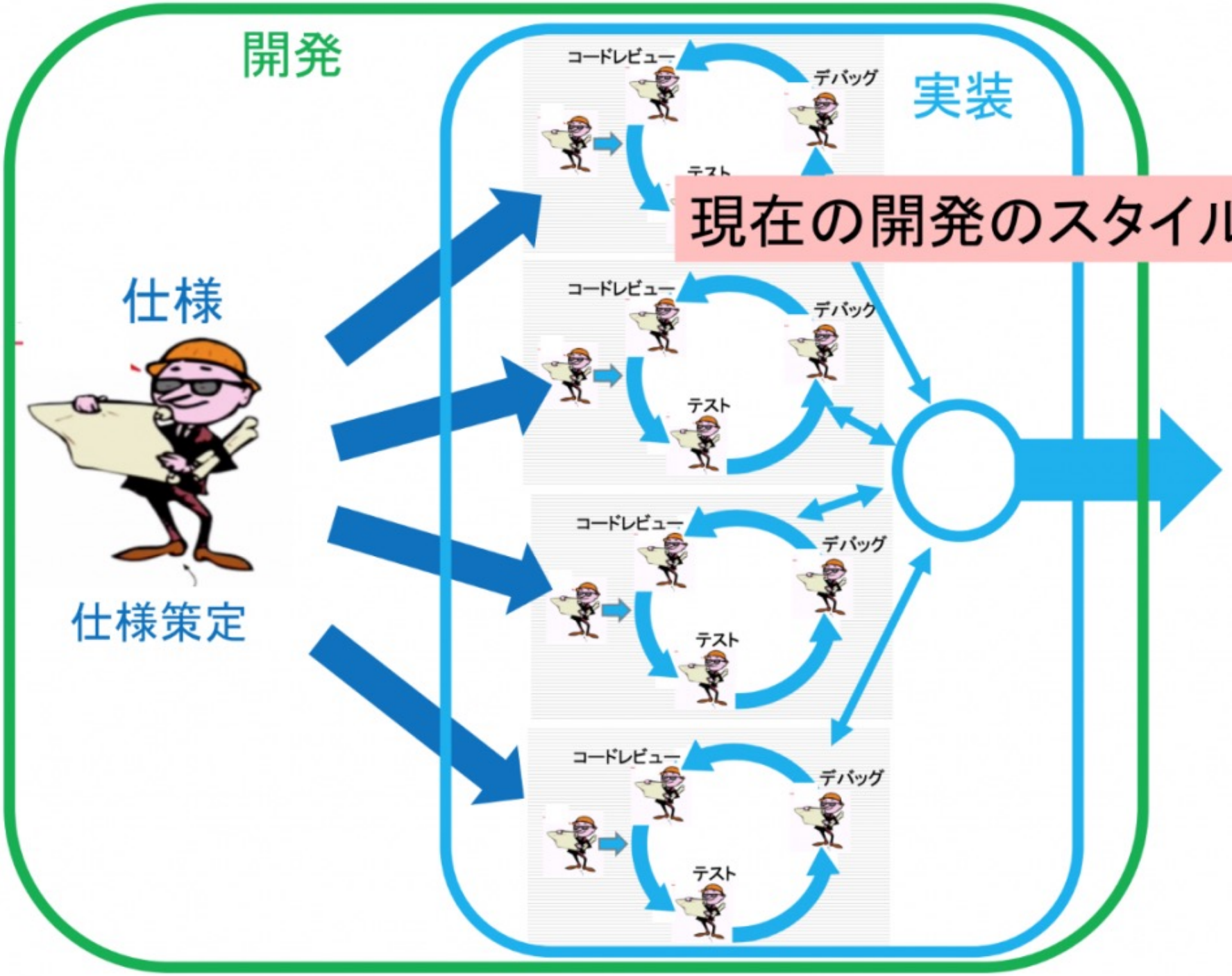
人間と機械の役割の変化を考える

ソフトウェア開発サイクルの単純化されたモデル

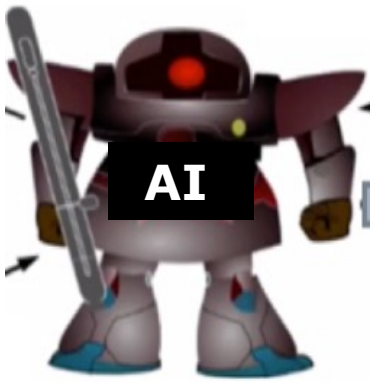


人間が果たしている役割をAgentに置き換える

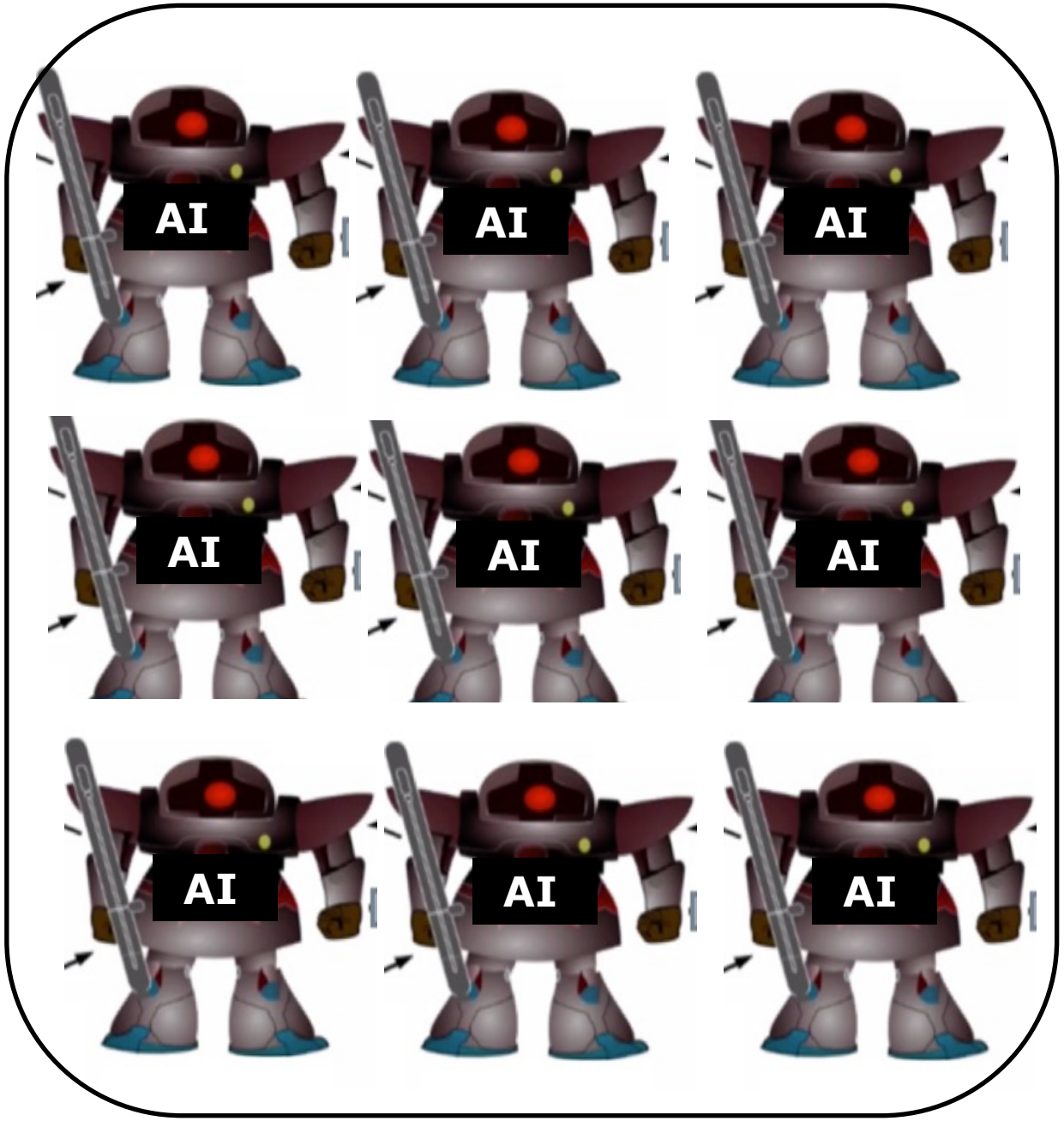



開発
実装
現在の開発のスタイル
仕様
仕様策定
コードレビュー
デバッグ
テスト
コードレビュー
デバック
テスト
コードレビュー
デバッグ
テスト
コードレビュー
デバッグ
テスト

代替

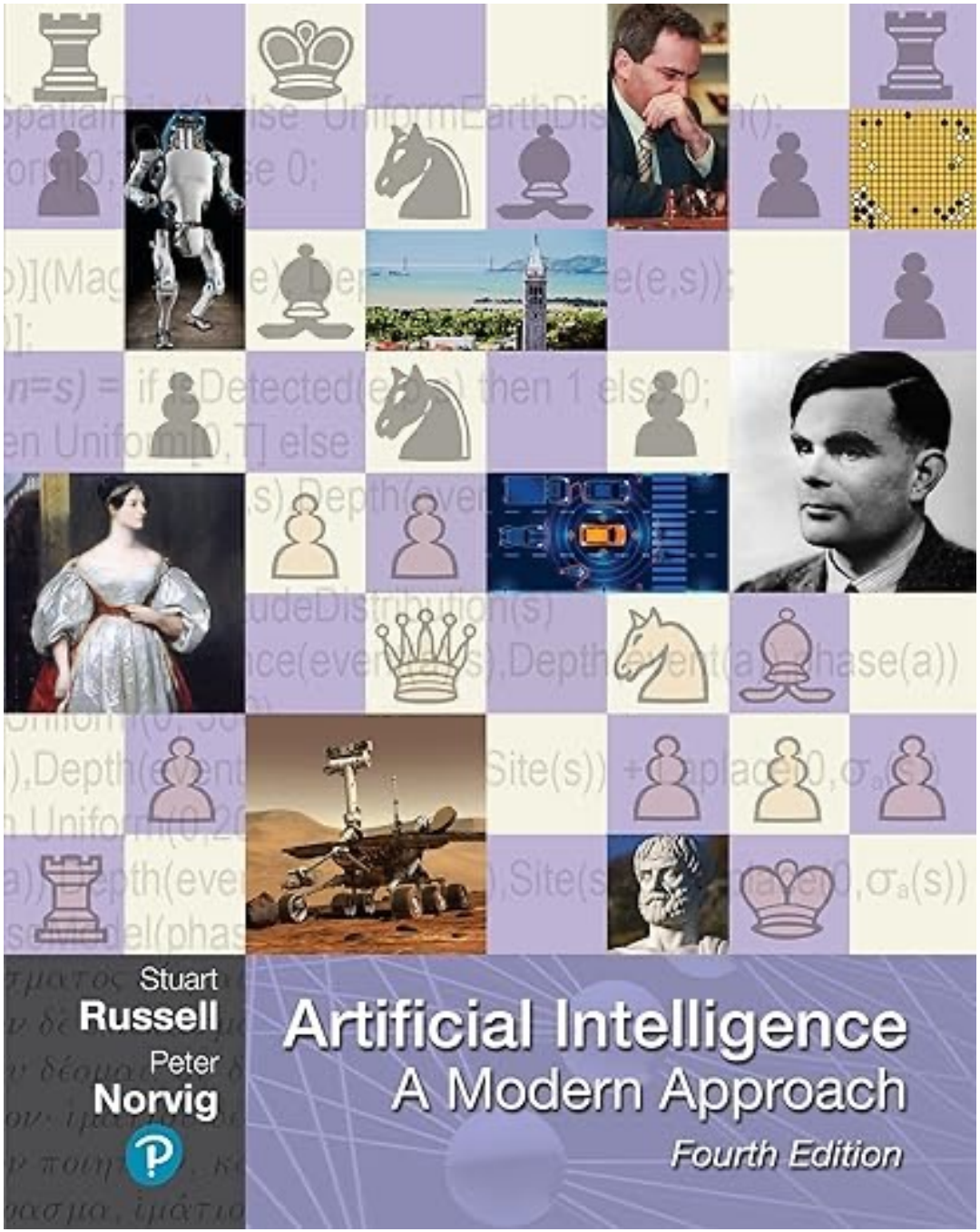


指示？

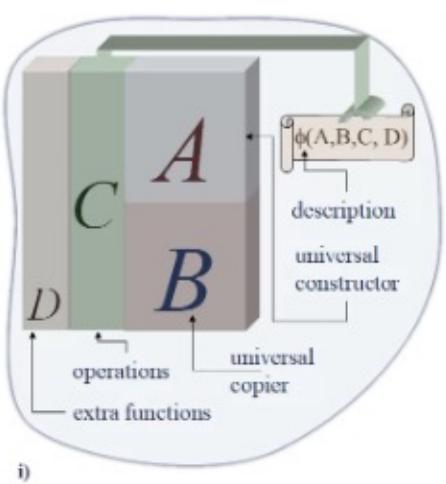


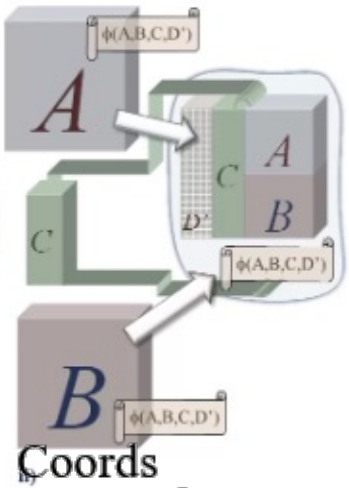

補論
Agent論の二つの系譜

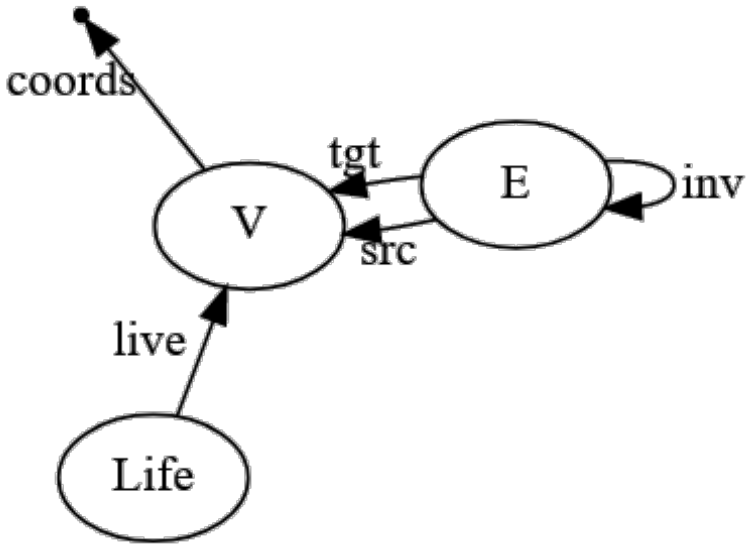
AI Agent 概念の成立と進化



数学的Agent論


A
C
B
D
φ(A,B,C, D)
description
universal
constructor
operations
universal
copier
extra functions
i)


φ(A,B,C,D')
A
C
A
B
D'
φ(A,B,C,D')
C
B
φ(A,B,C,D')


Coords
coords
V
tgt
src
E
inv
live
Life