メタバース新書

書籍タイトル

cover sample for A5,

with bleed margin 3mm

デジタルハリウッド・パブリッシャーズ

AI Native ハッカソン徹底攻略

3 人のエンジニアが語る、AI と共創するものづくりの実践知

Kuu、Sae、Lemio　著

2026-02-07 版　　発行

はじめに

3 人のエンジニアがハッカソンと AI の交差点で見つけたこと

この本は、ハッカソンを愛する 3 人のエンジニアが、AI 時代のものづくりについて語り合った記録です。

私たちは何度もハッカソンに参加し、運営し、審査してきました。その中で、AI の登場によってハッカソンの風景が大きく変わりつつあることを肌で感じています。Vibe Coding で誰でもプロトタイプを作れる時代に、ハッカソンで何を競い、何を楽しむのか。エンジニアのマインドセットはどう変わるべきなのか。そして、非エンジニアにとってのハッカソンの可能性はどこまで広がるのか。

2026 年 2 月、ブッカソン（本のハッカソン）という場で、私たちはこれらの問いに向き合いました。本書はその対話から生まれました。

この本は「対話」で読む技術書である

対話形式を採用した理由——１人の語りに２人が問いを投げる構成

本書は、一般的な技術書とは異なる構成を採用しています。各章では 1 人の著者が主に語り、残りの 2 人がツッコミや質問を入れる対話形式で進みます。

1 人で書くと出てこない気づきが、対話からは自然に生まれるからです。「それってどういうこと？」という素朴な質問が、読者にとっても理解の助けになります。また、ディープリサーチでは得られないリアルな経験——実際にハッカソンで AI を使って感じたこと、うまくいかなかったこと——は、対話の中でこそ引き出されます。

本書の読み方と各章の役割分担

本書は 4 つの章と付録で構成されています。

- **第 1 章「AI ネイティブという不可逆な変化」**（Kuu）—— AI がエンジニアの仕事やマインドセットをどう変えるのか、そして AI を「ガードレール」で囲んで活用する開発手法について語ります。
- **第 2 章「アイデアは『オタク知識×技術知識』の掛け算で生まれる」**（Lemio）—— ハッカソンでのアイデア発想法、自己表現としてのものづくり、そしてアイデアを仕様に落とし込む技術について語ります。
- **第 3 章「ハッカソンという『枠組み』が人を動かす」**（Kuu・Lemio）—— ハッカソンの構造的な力、非エンジニアへの門戸開放、運営の実務ノウハウについて語り

ます。

- **第 4 章「AI 活用の実践テクニック」**（全員）—— 第 1〜3 章の知見を横断し、ハッカソンでも仕事でも使える AI 活用テクニックをまとめます。

どの章から読んでも構いません。興味のあるテーマから読み始めてください。

想定する読者——エンジニアもそうでない人も

本書は以下のような方を想定しています。

- ハッカソンに参加したことがあり、AI 時代の変化を感じているエンジニア
- ハッカソンに興味はあるが、まだ参加したことがない方
- AI ツールを使ってものづくりを始めたい非エンジニア
- ハッカソンの運営・企画に携わる方
- 「手を動かすことの価値」を再確認したいすべてのクリエイター

技術的な予備知識は必要ありません。対話形式で進むので、わからない用語があっても文脈から理解できるよう心がけています。

目次

第 1 章

AI ネイティブという不可逆な変化——エンジニアの仕事はもう元に戻らない

AI が当たり前になった世界で、エンジニアに求められるマインドセットはどう変わるのか。具体的なツールの使い方ではなく、永続的に必要となる考え方の転換について、Kuu が語ります。

1.1 「AI を使う」から「AI と共に考える」への転換点

2026 年に起きているのは一過性のブームではなく思考様式の変化である

Kuu：自分が特に書きたいのは、AI ネイティブになった時にどういうマインドチェンジが求められるのか、というところです。2026 年 2 月の話だけじゃなくて、おそらく永続的に変わるもの。

Lemio：それってエンジニアだけの話？

Kuu：エンジニアのマインドがまず大きいですが、他の職種も含めた話です。ハッカソンでの人間の役割分担がだんだん消えていって、将来的には全員が同じ一つの職種くらいまで染み出して溶けると思うので、全員向けの話でもあります。

かつて AI は「便利な道具」だった。検索エンジンの延長線上で、わからないことを聞けば答えが返ってくる辞書のような存在だ。しかし 2025 年以降、Coding Agent の台頭によって状況は決定的に変わった。AI は「聞けば答える道具」から「指示すれば実装する協働者」へ進化した。

この変化の本質は、人間の役割の移行にある。従来の開発では、人間がコードを書き、AI に補完や提案を求めていた。今は違う。人間は「何を作るか」「なぜ作るか」を考え、AI が「どう作るか」を実行する。人間がコーディングすることを諦める——これは後退ではなく、発想の転換である。

Sae：「諦める」って言い方、ちょっと抵抗ありますね。

Kuu：僕も最初は抵抗がありました。でも実際に AI のコーディング速度が人間を超えた時点で、人間がコードを書く行為は「頑張ること」ではなく「非効率を選ぶこと」に

なった。だから発想を転換して、人間は企画・UX 設計・ユーザーヒアリングなど、AI にできない領域に集中する。これが AI ネイティブ時代のマインドセットの起点です。

1.2　職種の境界線が AI によって溶解する

エンジニア・デザイナー・PM の肩書きが AI 時代に意味を失う理由

Kuu：AI のおかげで、職種という肩書きがどんどん取り払われる感じはありますし、エンジニアでなくても貢献できる形が増えてきています。

具体例を挙げよう。ハッカソンの現場では、デザイナーが Vibe Coding で動くプロトタイプを自分の手で作り始めている。Figma でモックを描いた後、AI に「このデザインを React で実装して」と指示すれば、数分で触れるプロトタイプが手に入る。PM が API ドキュメントを AI に渡し、「この API を使ったデモアプリを作って」と頼めば、エンジニアの手を借りずに動作検証ができる。

Sae：実際にハッカソンでデザイナーの方がアプリを実装していた場面を見ました。以前なら「実装はエンジニアに任せる」が当然だったのに。

Kuu：そうなんです。AI がコーディングを担うことで、「誰が実装するか」ではなく「誰がアイデアを持っているか」「誰がユーザーを理解しているか」が重要になる。肩書きではなく、その人が持つ知識と視点がチームへの貢献を決める時代になった。

「コードが書けない人」がプロダクトを作れる時代の到来

Kuu：「コードが書けない」は、もはやプロダクトを作れない理由にならない。Vibe Coding の本質は、自然言語でアイデアを伝えれば動くものが出てくる点にある。

Lemio：具体的にはどんなケースがありますか？

Kuu：たとえば、マーケティング担当者が「顧客データを可視化するダッシュボードがほしい」と AI に伝え、数回のやり取りで Web アプリを完成させる。営業担当者が「商談管理のツールがほしい」と言えば、簡易的な CRM が数時間で動く。ハッカソンでも、プログラミング経験のない参加者が AI と対話しながらプロトタイプを形にする場面は珍しくなくなった。

重要なのは、非エンジニアが「エンジニアの代わり」をするのではなく、自分自身の専門領域の知見をプロダクトに直接注ぎ込めるようになった点だ。ドメイン知識を持つ人が、翻訳者（エンジニア）を介さずに自分のアイデアを形にできる。これは品質の向上にもつながる。

1.3　AI をガードレールで囲む開発手法

最新の API リファレンスを起点にしないと AI は古い情報で嘘をつく

Kuu：AI をそのまま使うと、古い情報でハルシネーションを生み出します。だからこそ「ガードレール」という考え方が重要なんです。AI に自由に動いてもらう前に、正しい方向に走れるよう枠を作っておく。

Lemio：ガードレールを引いた上で AI に暴れさせることで、より良い結果が出ると。

Kuu：そうです。この「まず枠を作ってから走らせる」という発想が、AI 時代のエンジニアに求められるマインドセットの一つだと思います。具体的な方法——たとえば公式ドキュメントの URL をプロンプトに含める、バージョンを明示するといったテクニックは第 4 章「AI 活用の実践テクニック——ハッカソンでも仕事でも使える知恵」で詳しく紹介します。

ガードレール思考とは、「AI に仕事を任せる前に、正しい方向に走れる枠組みを整えておく」という考え方である。高速道路のガードレールが車の自由な走行を妨げず、かつ崖から落ちないよう守るのと同じだ。

この考え方がなぜ重要か。AI は指示されたことを高速で実行するが、指示の前提が間違っていれば、間違った方向に全速力で走る。人間がコードを書いていた時代は、書きながら「あれ、おかしいな」と気づけた。しかし AI に任せると、成果物が出てくるまで問題に気づかないことがある。だからこそ、走らせる前の枠組み作りが従来以上に重要になる。

具体的なガードレールには、Agent Skills（AI の能力を補強するスキルファイル）の整備、使用する API の情報整理、プロジェクトのルール文書化などがある。たとえば`.trae/skills/`のようなディレクトリにスキルファイルを配置し、AI が参照すべき iOS 開発や Web 開発の知見をあらかじめ用意しておく。AI はこのスキルファイルを参照することで、プロジェクト固有のルールに沿ったコードを生成できる。

Sae：AI に任せる前の準備がむしろ増えている、とも言えますね。

Kuu：その通りです。ただし、一度整えたガードレールはチーム全体で再利用できる。一人が整備すれば、チーム全員の AI 活用の質が上がる。これは従来の「各自がコードを書く」モデルにはなかった効率化だ。

JS/Python のライブラリ依存地獄を AI が悪化させる構造的問題

Kuu：JavaScript や Python のエコシステムでは、ライブラリの更新速度が速い。npm や pip で管理されるパッケージは日々バージョンが上がり、API の破壊的変更も珍しくない。ここに AI が加わると、問題が構造的に悪化する。

Lemio：どういう意味ですか？

Kuu：AI の学習データには「ある時点で最新だったコード」が含まれている。しかし実際のプロジェクトで使うライブラリのバージョンは学習データと異なる場合が多い。結果として、AI が自信満々に生成したコードが「動くけど非推奨の API を使っている」「メソッド名が変わっていて動かない」といった問題を引き起こす。

対策の核心は、AI に最新の情報を渡すことである。使用するライブラリのバージョンを明示し、公式ドキュメントの URL をプロンプトに含める。API リファレンスを起点に AI に書かせることで、古い情報に基づく生成を防げる。この具体的なテクニックは第 4 章「AI 活用の実践テクニック——ハッカソンでも仕事でも使える知恵」で詳しく扱う。

ここで伝えたいのはテクニックそのものではなく、マインドセットの部分だ。「AI が書いたコードは正しいはず」と信じるのではなく、「AI には最新の情報がない可能性があ

る」と常に疑う姿勢。そして疑うだけでなく、正しい情報を AI に提供する責任が人間にある、という認識を持つことが重要である。

CI パイプラインと Lint を AI のガードレールとして活用する

Kuu：ガードレールを引くために、既存の CI や Lint のような仕組みは今でも使えますし、むしろ重要性が上がっている気がします。

CI パイプラインと Lint は、AI 時代において新たな役割を獲得した。従来は「人間が書いたコードの品質を機械的にチェックする仕組み」だったが、今は「AI が書いたコードを自動で検証し、問題があれば AI 自身に修正させる仕組み」として機能する。

Kuu：ここで面白いのは、CI が落ちた時の対応です。人間がエラーログを読んで修正するのではなく、AI に「Please fix CI failing」と伝えるだけで修正が完了する。CI と AI の組み合わせは、自動修正ループを作り出す。

Lemio：それって無限ループにならないんですか？

Kuu：実際には 2〜3 回のやり取りで収束することがほとんどです。Lint の指摘も同様で、「Please fix Lint errors」で AI が修正する。ポイントは、CI や Lint がガードレールとして「何が正しいか」を定義し、AI がその基準に合わせて修正するという構造だ。人間は基準を設定し、AI が基準を満たすコードを書く。この分業が成り立つのは、CI や Lint が「合格/不合格」を明確に判定できるからである。

マインドセットとして重要なのは、CI や Lint を「人間のためのツール」ではなく「AI のためのガードレール」として捉え直すことだ。テストカバレッジ、型チェック、コードフォーマットといった機械的に検証可能な基準を厚くすればするほど、AI の出力品質が安定する。

1.4 ツールの選び方——特定ツールではなく「強いツールの条件」を知る

特定のツール名ではなく「強いツールの特性」を見極める

Kuu：具体的なツールの使い方というよりは、こういう特性のツールが強い、というのを書いていきたい。ティップスみたいな「こういう時はこう」というのはおそらくティップス化しづらいかなと思っています。

AI 開発ツールは半年で勢力図が塗り替わる。特定のツール名に依存した知識はすぐに陳腐化する。だからこそ、「強いツールに共通する特性」を知っておくことが、ツールを選び続けるための武器になる。

強い AI 開発ツールに共通する特性は以下の通りだ。

- **スキルファイルによる拡張性** —— ツールの振る舞いをファイルで定義・拡張できること。たとえば Agent Skills のように、プロジェクト固有の知見をファイルとして配置し、AI の能力を強化できる仕組みがあるかどうか。

- **非同期実行への対応** —— 複数のタスクを並列で実行できること。後述する非同期 Coding Agent の恩恵を受けるには、ツール側の対応が不可欠である。
- **コンテキスト理解の深さ** —— プロジェクト全体のコードベースを理解した上で提案・生成できること。単一ファイルの補完ではなく、リポジトリ全体の構造を踏まえた出力ができるかが差を生む。
- **エコシステムの広さ** —— プラグインや拡張機能のコミュニティが活発であること。ツール本体の機能だけでなく、周辺の生態系が充実しているかを見る。
- **API と CLI の一貫性** —— プログラマブルに制御できること。GUI だけでなく、CLI や API で操作できれば、CI パイプラインやスクリプトに組み込める。

Sae：この基準って、AI 以外のツール選びにも当てはまりそうですね。

Kuu：本質的にはそうです。ただ AI ツールに特有なのは、「スキルファイル対応」と「非同期実行」の 2 点。この 2 つがあるかないかで、チームの生産性が桁違いに変わる。ツール名を覚えるのではなく、特性で評価する癖をつけておけば、次のツールが出てきた時にもすぐに判断できます。

1.5 Vibe Coding が変えるハッカソンの難易度と面白さ

Vibe Coding とは何か——感覚駆動でプロトタイプを生み出す開発スタイル

Lemio：最近 Vibe Coding が流行ってきて、ハッカソンが退屈に感じることがあって。

Kuu：ただのアプリケーションだと本当に簡単に 3 秒でポッとできてしまうので、それ以外の技術を絡めたものが求められます。

Vibe Coding とは、自然言語で「こんな感じのものがほしい」と伝え、AI がコードを生成し、人間はその出力を見て「もうちょっとこうして」と調整する開発スタイルである。コードの詳細を人間が制御するのではなく、感覚（Vibe）で方向性を示し、AI が具体化する。

従来の開発スタイルとの最大の違いは、人間がコードを書かない点にある。従来は「設計→実装→テスト」の各段階で人間がコードに直接触れていた。Vibe Coding では「設計→ AI に指示→出力を確認→フィードバック」というサイクルになる。人間の仕事は、コードを書くことからフィードバックを与えることに変わった。

Kuu：重要なのは、Vibe Coding は「雑にやる」という意味ではないことです。「コードの詳細ではなく、ユーザー体験やプロダクトの方向性に集中する」という、より上位の抽象度で開発するスタイルだと捉えるべきです。

Lemio：でも、AI が生成したコードの品質が心配にならないですか？

Kuu：だからこそガードレールが必要なんです。先ほど話した CI や Lint、スキルファイルの整備があれば、Vibe Coding でも品質は担保できる。むしろ、ガードレールがない Vibe Coding は危険だと思います。

AI で「作る」が簡単になった今、ハッカソンに求められる新しい挑戦

Kuu：基本的な実装がすぐ終わるからこそ、新しいフォーメーションや挑戦に時間を割けます。もし詰まるハッカソンで冒険したら何も作れないリスクがありますが、余裕がある前提なら挑戦できる。

Lemio：心理的安全性が担保された上で、「最悪ここまでは作れる、だからこそ新しいことに挑戦しよう」という発想ですね。

Vibe Coding の普及により、ハッカソンの競争軸が根本から変わった。以前は「限られた時間でどれだけ実装できるか」が勝負だった。しかし基本的な実装を AI が高速でこなせる今、差がつくのは「何を作るか」「なぜ作るか」という企画力と、複数の技術やサービスを組み合わせる統合力である。

Kuu：非同期 Coding Agent を活用すれば、生産性は一桁上がる。git worktree を使ってローカルで 2 並列、複数 PC で最大 6 並列の開発が可能になる。基本実装に時間を取られないからこそ、ハードウェア連携やリアルタイム通信など、挑戦的な技術に時間を割ける。

Sae：でも挑戦するとリスクも上がりますよね。

Kuu：そこがポイントです。AI があれば「最悪ここまでは作れる」というベースラインが保証される。心理的安全性が担保された状態で冒険できる。ハッカソンのブランチ戦略としても、PR 不要で master に直接 push し、mono repo で single branch にする。フィードバックループを最速にすることで、挑戦と安定の両立が可能になる。

ハッカソンの面白さは「制約の中でどう工夫するか」にあった。AI によって「実装」という制約が緩和された今、新しい制約——たとえば「社会課題の解決」「未知の技術スタックの組み合わせ」「非エンジニアとの協働」——にチャレンジできる余地が生まれている。ハッカソンは退屈になるのではなく、競争の次元が上がったのだ。

第 2 章

アイデアは「オタク知識×技術知識」の掛け算で生まれる

ハッカソンでのアイデア発想法から、チームでの合意形成、アイデアを仕様に落とし込む技術まで。Lemio が自身の経験をもとに、ハッカソンにおける「自己表現としてのものづくり」を語ります。

2.1　ハッカソンは競技ではなく「自分を披露する場」である

勝ちに行くのではなく「作りたいもの」を作る姿勢が結果を出す

Lemio：今思うと、アイデアを出してハッカソンで勝つためにやっているのかなと。意外と自己表現の場でもあるのかなと感じます。

Kuu：ちょっとアーティストですね。

Lemio：「今これがないから作りたい」というアイデアを貯めておいて、ハッカソンの場で出す。そのたびに作品が生まれるのは、自己表現なのかもしれません。

ハッカソンに参加する動機を「勝つため」と定義すると、審査員の好みに合わせた無難なプロダクトを作りがちになる。一方で「自分の作りたいもの」を軸にすると、プロダクトに作者の情熱が宿る。この差はデモの説得力に直結する。

審査員が見ているのは、技術の巧みさだけではない。「なぜこれを作ったのか」という動機の強度である。自分が本気で欲しいものを作っている人の発表には、聞く者を引き込む力がある。「誰かに刺さるもの」を狙って作ったものより、「自分に刺さるもの」を全力で作り込んだほうが、結果として他人にも刺さる。

Lemio が「アイデアは腐る前にハッカソンで放出する」と表現するように、良いアイデアを温めすぎると先を越される。ハッカソンは、頭の中にある「いつか作りたいもの」を強制的にアウトプットする装置だ。締切があるからこそ、「もう少し考えてから」の言い訳が通用しない。その制約が、自己表現の密度を上げる。

2.2 アイデア発想の源泉——趣味と技術の引き出しを掛け合わせる

日常の「これ不便」「これ面白い」を技術で解決可能な形に変換する思考法

Lemio：オタクの知識と技術の知識を両側から攻め合わせて、できそうなところでプロダクトが生まれる。

Kuu：それをどうやって生み出しているのか、普段からやっていることや、情報キャッチアップの方法を言語化してほしい。

Lemio：技術的実現性がわかっているからこそ、「なぜ実現できないのか」を「どういう切り口なら実現できるか」に分解していけます。その過程を言語化したいですね。

アイデアは「こういうものが欲しい」という妄想と「今の技術で何ができるか」という知識の交差点に生まれる。この 2 つを意識的に掛け合わせることが、ハッカソンにおけるアイデア発想の核である。

Lemio はこのプロセスを「オタク妄想力×技術トレンド＝今作れるもの」と表現する。フィクションの中で描かれた未来の技術が、現実のどの技術と対応するかを日常的に観察する習慣が、アイデアの引き出しを増やす。たとえばアニメ「ARMS」に登場する AI アシスタント「アリス with チャット」は、ChatGPT の登場で現実になった。名探偵コナンの蝶ネクタイ型変声機は、リアルタイムボイスチェンジャー技術で再現可能になった。ガンダムの小型 AI「ハロ」は、スタックチャンや Romi といった対話型ロボットとして実現しつつある。

重要なのは、フィクションを「夢物語」として消費するのではなく、「将来実装されるかもしれない仕様書」として読む姿勢だ。日常の中で触れるアニメ、映画、漫画、小説の中に、まだ世の中にないプロダクトのヒントが眠っている。

具体的なステップとしては、3 つの段階がある。

1. 日常の観察から「こんなものがあったら面白い」という着想を集める。通勤中の不便、趣味で感じた課題、フィクションで見た未来技術など、何でもよい
2. 技術トレンドのキャッチアップで「今何が実現可能か」を把握する。新しい API の発表、OSS の動向、研究論文の成果を常にウォッチする
3. 着想と技術を突き合わせ、「今の技術で実現可能な範囲」を見極める。完全な再現が無理でも、本質的な体験の一部を切り出せれば十分なプロダクトになる

過去のハッカソン作品から見るアイデア発想の実例

Lemio の過去のハッカソン作品は、まさに「フィクション×現実技術」の掛け算で生まれている。

たとえば名探偵コナンの蝶ネクタイ型変声機。「コナンの道具が実際に使えたら面白い」という妄想から始まり、それを INPUT → PROCESS → OUTPUT のモデルに落とし込む。音声を入力し、API やスクリプトで処理し、変換された音声を出力する。「欲しい」

という感情を、具体的な入出力仕様に翻訳する作業である。

Lemio：蝶ネクタイ型変声機を作ったとき、最初は漠然と「コナンのあれがほしい」だったんです。でも「あれ」を分解すると、マイクから音声を取って、ピッチを変えて、スピーカーから出す。入力と出力が明確になれば、あとは間をつなぐ技術を探すだけです。

Kuu：入力と出力を言葉にできれば、AI が間を埋めてくれる時代ですからね。

この「入力と出力を定義する」思考法は、AI 時代においてさらに威力を発揮する。かつては INPUT と OUTPUT の間を自分でコーディングする必要があった。今はその間を AI が埋めてくれる。つまり「何を入れて、何が出てほしいか」を言語化できれば、実装の大部分を AI に委ねられる。「コードを書く」から「設計を言語化する」へのパラダイムシフトだ。

もう一つの例として、東のエデンの「エデンシステム」がある。カメラで映した対象を画像認識で特定し、関連情報をオーバーレイ表示する——この構想は SAM3 や YOLO といった画像認識技術の進化により、現実的な精度で実装可能になった。Google Genie 3 のような生成 AI を使えば、「絵本の中に入り込む靴」のような体験すら、インタラクティブな仮想空間として再現できる。

うまくいかなかった経験もある。アイデアが壮大すぎて、ハッカソンの限られた時間では動くデモまで持っていけなかったケースだ。ドローンスウォームでガンダムの「ファンネル」を再現しようとしたような場合、ハードウェア制御の複雑さが想定を超え、デモとして見せられる段階に到達できない。こうした失敗から学べるのは、「フィクションの 100% 再現」を目指すのではなく、「本質的な体験の一部を切り出す」ことの重要性だ。

1 人で考えるより「雑談」から生まれるアイデアの方が強い理由

Lemio：1 人で考えていると、自分の知識の範囲でしかアイデアが出ないんですよね。でも誰かと話していると、相手の一言で全く違う方向に発想が飛ぶことがある。

Sae：雑談って、テーマが定まっていないからこそ自由に飛べるんですよね。ブレインストーミングのように「アイデアを出しましょう」と構えると、かえって出てこないこともあります。

Kuu：この本の企画自体、3 人の雑談から生まれましたからね。

1 人のブレインストーミングには構造的な限界がある。自分の知識と経験の範囲内でしか発想が広がらないため、似たようなアイデアが堂々巡りしやすい。

雑談が強い理由は 3 つある。第一に、異なる知識領域の衝突が起きること。自分が知らない技術やフィクションを相手が持っていれば、掛け合わせの組み合わせが爆発的に増える。第二に、言語化の過程でアイデアが精錬されること。頭の中のぼんやりした構想を相手に説明しようとする行為自体が、アイデアの輪郭を明確にする。第三に、「それ面白い」というリアクションがモチベーションを生むこと。1 人で考えたアイデアは自信が持てないが、他人が面白がってくれた瞬間、実装への推進力が生まれる。

ハッカソンの開始直後にチームメンバーと雑談する時間を意識的に設けることは、アイデアの質に直結する。いきなり「何を作るか」を議論するのではなく、「最近面白かったもの」「日常で不便に感じていること」を自由に話す時間を 10 分でも取るだけで、発想の

幅が変わる。

■コラム: Lemio の過去プロジェクト紹介——アイデアの種がどう育ったか

Lemio が過去のハッカソンやプロジェクトで取り組んだ作品は、すべて「フィクション×技術トレンド」の掛け合わせから生まれている。いくつかの例を紹介する。

蝶ネクタイ型変声機: 名探偵コナンの変声機をリアルタイムボイスチェンジャーで再現。「声を変えたい」という入力と出力を定義し、音声処理 API で間をつないだ。ハッカソンでのデモでは、実際にその場で声を変えて見せることで、審査員の体験を直接揺さぶった。

画像認識オーバーレイ: 東のエデンの「エデンシステム」に着想を得て、カメラ映像にリアルタイムで情報をオーバーレイ表示するプロトタイプ。画像認識モデルの進化により、以前は精度が足りなかった機能が実用レベルで動くようになった。

対話型ロボット: ガンダムの「ハロ」をモチーフに、LLM を搭載した小型対話ロボットの制作。通信建設の現場で培ったハードウェアの知識と、LLM アプリエンジニアとしてのソフトウェアの知識が掛け合わさった作品。

共通しているのは、「完全な再現」ではなく「本質的な体験の抽出」を目指している点だ。フィクションのすべてを再現する必要はない。ユーザーが「おお、あれだ」と感じる体験の核を特定し、その核だけを技術で実装する。この割り切りが、限られた時間のハッカソンでは特に重要になる。

2.3　チームでアイデアの合意を取るのが最も難しい

個人のビジョンとチームの方向性がぶつかる瞬間

Kuu：初めましてで組んだ時のアイデアの合意形成も結構難しいですよね。

Lemio：「それ面白い」「それでいけそう」とみんなが納得するアイデアなのか、みんなで練り上げて生まれてくるアイデアなのか。その違いは大きいですね。

チームでのアイデア合意には 2 つのパターンがある。「誰かのアイデアに全員が乗る」パターンと、「全員で議論して練り上げる」パターンだ。

前者は意思決定が速い。強いビジョンを持つメンバーがアイデアを提示し、他のメンバーが「面白い、やろう」と合意する。ただしこのパターンには落とし穴がある。「面白い」と言ったメンバーが本心から賛同しているのか、単に空気を読んで同意しただけなのかが判別しにくい。表面的な合意のまま開発に入ると、途中で「実はあまりピンときていない」というメンバーのモチベーションが下がり、チームが機能不全に陥る。

後者は時間がかかるが、全員のオーナーシップが高い。対立が起きても、議論を通じて全員が納得した結論には推進力がある。ただし、議論が長引きすぎると開発時間を圧迫するリスクがある。

Lemio：初対面のチームでは、まず「誰が何に詳しいか」を共有するのが大事です。メ

ンバーの得意分野がわかると、アイデアの方向性が自然に絞られる。

Kuu：全員が「自分ごと」として捉えられるアイデアかどうかが、合意形成のカギですね。

合意形成を円滑にするための実践的なコツがある。まず、アイデア出しの段階では否定をしない。「それは技術的に無理」「審査員にウケない」といった批判は、発想の幅を狭める。次に、アイデアを選ぶ段階では「全員が一つ以上の貢献ポイントを持てるか」を基準にする。特定のメンバーしか手を動かせないアイデアは、チーム全体の合意を得にくい。

もっとも効果的なのは、「プロトタイプで語る」手法だ。言葉だけの議論では堂々巡りになりやすいが、5 分で作った粗いプロトタイプを見せれば、「これのこの部分を変えたい」「ここを膨らませたい」と具体的な議論に移行できる。AI 時代のハッカソンでは、Vibe Coding で数分のうちにプロトタイプを生成できるため、「まず作って見せる」ことの敷居が大幅に下がっている。

2.4 アイデアから仕様書へ——構想を実装可能な形に翻訳する技術

「面白い」だけでは開発が始まらない——アイデアと仕様の間にある崖

Kuu：アイデアと開発の間の仕様決めは大事だと思っています。「このアイデアをやりたい」と言っても、仕様を考えて実装まで持っていけないことがあるんですよね。

Lemio：仕様に落とし込めないチームは多いと思います。アイデアを考えても地に足がついていなくて、「どうやって実現するか」を考えた結果、ありきたりなものになるパターンですね。

「アイデアと仕様の間にある崖」は、ハッカソンで最も多くのチームがつまずくポイントだ。アイデア自体は面白いのに、仕様に落とし込む段階で 2 つの失敗パターンに陥る。

第一の失敗パターンは、仕様化できずに時間を浪費するケースだ。「面白いけど、どう作ればいいかわからない」まま議論が空転し、開発に着手できない。全員がアイデアの抽象的な面白さには合意しているが、具体的な実装イメージを持てていない。

第二の失敗パターンは、仕様化の過程でアイデアが矮小化するケースだ。「AI で○○を自動化する」という壮大なアイデアが、技術的制約を考慮するうちに「既存 API を叩いてデータを表示するだけの Web アプリ」に縮小される。結果として、アイデアの核心的な面白さが仕様から抜け落ちる。

どちらの失敗も、「アイデアと技術を行き来する力」の不足が原因だ。アイデアだけを考える人と技術だけを考える人が分離していると、この崖を越えられない。理想は、1 人の頭の中で「こんな体験を作りたい」と「この技術を使えば実現できる」を同時に考えられることだ。Lemio が「オタク知識×技術知識」と言うのは、まさにこの二面性を指している。

仕様に落とし込むステップ——ユーザー体験→画面遷移→ API 設計

Lemio：僕は最初に「入力がこの状態で、この処理をして、この結果を出す」という流れを全部決めてから、仕様書を作って開発します。わからない部分だけ、「ここからこうしたいけど、どの技術スタックを使えばいいか」を相談して壁打ちしながら進めますね。

抽象的なアイデアを実装可能な仕様に変換するには、段階的な具体化が必要だ。Lemio が実践する INPUT → PROCESS → OUTPUT モデルは、そのもっともシンプルで強力なフレームワークである。

ステップ 1: ユーザー体験を 1 文で定義する。「ユーザーが○○すると、○○が起きる」という形式で、プロダクトの核心的な体験を言語化する。蝶ネクタイ型変声機の例なら、「ユーザーがマイクに向かって話すと、別人の声でスピーカーから出力される」となる。

ステップ 2: INPUT・PROCESS・OUTPUT を分離する。ステップ 1 の体験を、入力（ユーザーが提供するもの）、処理（システムが行うこと）、出力（ユーザーが受け取るもの）に分解する。この分離により、「処理」の部分を技術的にどう実装するかという問題に集中できる。

ステップ 3: 処理を既知の技術要素に分解する。「処理」をさらに小さなステップに分け、各ステップに対応する技術（API、ライブラリ、サービス）を特定する。わからない部分は「ここは未定」と明示し、チームメンバーや AI に相談する。全体像を先に固め、不明点をピンポイントで潰していく。

ステップ 4: 画面遷移や操作フローを可視化する。紙のスケッチでもホワイトボードでも、ユーザーが触る画面の流れを視覚化する。これによりチーム全体が同じ完成イメージを共有でき、各メンバーがどの部分を担当するかが自然に決まる。

このプロセスのポイントは、「わからない部分を明示する」ことだ。仕様が曖昧なまま開発に入ると手戻りが発生する。わからないところを「わからない」と認め、そこだけを集中的に調査・相談することで、仕様の精度を上げながら時間を節約できる。

AI を使ってアイデアを仕様書に変換するワークフロー

AI 時代の開発では、アイデアから仕様書への変換プロセスにも AI を活用できる。「コードを書く」から「設計を言語化する」へとパラダイムが移行した今、入力と出力を言葉にできれば AI が間をつないでくれる。

具体的なワークフローは以下の通りだ。

フェーズ 1: アイデアの壁打ち。まず、アイデアを AI に投げて壁打ちする。「○○というアイデアがあるのだが、技術的に実現可能か」「類似のプロダクトやサービスは存在するか」といった質問で、アイデアの実現性と新規性を素早く検証する。

フェーズ 2: INPUT → PROCESS → OUTPUT の定義。アイデアの方向性が固まったら、AI に対して「このアイデアを INPUT・PROCESS・OUTPUT に分解してほしい」と依頼する。AI が提示した分解案をたたき台に、チームで議論して修正する。ゼロから考えるより、AI の提案を修正するほうが圧倒的に速い。

フェーズ 3: 技術スタックの選定。「この処理を実現するために使える API、ライブラ

リ、サービスを提案してほしい」と AI に聞く。ここで注意すべきは、AI が提案する技術が最新とは限らない点だ。提案された技術の公式ドキュメントを必ず確認し、現在も利用可能で、ハッカソンの期間内に使いこなせるかを検証する。

フェーズ 4: 仕様書の生成。ステップ 1〜3 の結果をまとめ、AI に仕様書のドラフトを生成させる。画面遷移、API のエンドポイント設計、データフローを含む仕様書を、自然言語で指示するだけで作成できる。この仕様書をチーム全員で確認し、認識のずれを早期に潰す。

Lemio：入力と出力を言葉にできれば、AI が間を埋めてくれる。この「言語化する力」が、AI 時代のハッカソンで最も重要なスキルだと思います。

Sae：逆に言えば、「何がほしいか」を言語化できない人は、AI をうまく使えないということですよね。

Lemio：そうです。だからこそ、普段からフィクションや日常の中で「これがほしい」を具体的に考える訓練が、直接的にハッカソンの成果につながるんです。

2.5　評論家にならず手を動かす人が最後に勝つ

「それ無理じゃない？」と言う人より「とりあえず作ってみた」人が評価される

Kuu：評論家にはなりたくないですね。何が良くて何がダメなのか、「すごい」と言うけれど本当にすごいのか、従来と何が違うのか。動かさないとわかりません。自分の手で動かして、生の声で語ってほしいですよね。

ハッカソンの現場では、「それは技術的に難しいのでは」「前に似たサービスがあって失敗した」と指摘する人より、「とりあえず 30 分で動くものを作った」と見せる人のほうが圧倒的に価値が高い。

評論と実装の差は、情報量の差だ。頭の中で「無理そう」と判断するとき、その判断の根拠は過去の経験や断片的な知識にすぎない。一方、実際に手を動かすと、「ここは想像通りに動いた」「ここは予想外の壁がある」「ここに思いがけない可能性がある」と、100 倍の解像度で状況を把握できる。

AI 時代では、この差はさらに大きくなった。「作ってみる」コストが劇的に下がったからだ。Vibe Coding で 30 分あれば動くプロトタイプが手に入る。その 30 分を惜しんで 1 時間議論する行為は、「実装が重かった時代」の習慣の名残である。

Lemio：「アイデアは腐る前にハッカソンで放出する」と僕はよく言うんですが、同じことがハッカソンの中でも言えます。議論は腐る前にプロトタイプにする。プロトタイプにした瞬間、議論の質が変わるんです。

Kuu：「すごい」という感想も、動くものを触ったうえでの「すごい」と、話を聞いただけの「すごい」では、まったく別物ですからね。

評論家にならないためのもっとも確実な方法は、「発言する前に手を動かす」という習慣を持つことだ。「これは面白いかもしれない」と思ったら、まず 5 分でプロトタイプを作る。「この技術は使えるかもしれない」と思ったら、まずサンプルコードを動かす。そ

の手触りの情報が、チームへの発言の説得力を根本的に変える。

ハッカソンで最後に勝つのは、最も優れた評論をした人ではない。最も多くのプロトタイプを試し、最も多くの失敗から学び、その知見を最終プロダクトに凝縮した人だ。

第 3 章

ハッカソンという「枠組み」が人を動かす——運営と参加の実践知

ハッカソンはなぜ人を惹きつけるのか。その構造的な力から、非エンジニアへの門戸開放、軽量ハッカソンの提案、運営の実務ノウハウまで。ハッカソンの「枠組み」が持つ力を掘り下げます。

3.1 ハッカソンが持つ構造的な力——締切・他者・発表の三要素

2〜3 日の制限時間が「完璧主義」を強制的に解除する

Kuu：ハッカソンはフレームワークとしてすごく良いと思うんですよ。2、3 日で締め切りを作って、初対面の人たちと何かを作る。必ず発表しなければならないというプレッシャーも良い。

TODO: 時間制限が完璧主義を打破し、「とりあえず動くもの」を作る文化を生み出す仕組みについて解説する

全員がデモを見せる義務がある制約がアウトプットの質を底上げする

TODO: 発表義務がもたらすポジティブなプレッシャーと、アウトプットの質向上への影響を解説する

3.2 ハッカソンは非エンジニアにこそ開かれた場である

Vibe Coding の登場で「コードが書けない人」の参加障壁が消えた

Lemio：フランスで学生ハッカソンを視察したとき、Vibe Coding が初めてという参加者もいました。今までハッカソンはエンジニアのものでしたが、一般の方に広げていくのは面白いかもしれません。

Kuu：プロダクトを自分たちで作れなかった人たちは、すごく楽しんでいるんですよ。

作りたいものがたくさん溜まっていて、それが形になったときの感動は大きいです。

TODO: フランスの学生ハッカソン視察での具体的な体験と、非エンジニア向けハッカソンの参加者の反応を詳述する

デザイナー・企画者・ドメイン専門家がハッカソンで発揮できる固有の価値

Lemio：デザイナーが実際にアプリまで作れるようになると面白いですよね。エンジニアとの会話も円滑になりますし。

Kuu：「ここまでは簡単にできて、ここからはエンジニアの技術が必要」という境界線を体感できるのも価値があります。

Kuu：Apple Vision Pro ハッカソンでは、Vision Pro を持っていない人が、数年先に普及するデバイスを実際に触って、しかもアプリまで作れました。クラウドのクレジットが提供されるハッカソンもあり、特に学生にとっては貴重な機会です。

Lemio：新しい技術を触ってみたいと思いつつ、時間やお金がなくて挑戦できていない人が、最新技術のすごさを体感できる。しかも多くの人と話しながら進められるのは醍醐味ですね。

TODO: 各職種がハッカソンで持ち込める価値（ドメイン知識、UX 視点、ビジネス視点）と、Vibe Coding により職種間の相互理解が深まる効果を解説する。メンターの配置、チーム構成の工夫、敷居を下げるネーミング（「ブッカソン」等）の事例を紹介する。Apple Vision Pro ハッカソン等の具体例を通じて、最新技術に触れる場としてのハッカソンの価値、特に非エンジニアや学生にとって高価なデバイスやクラウド環境に無料でアクセスできることが参加障壁を下げる効果にも焦点を当てる

3.3 軽量ハッカソンのすすめ——5 時間で開催する即興型イベント

「新しい API が出た、今夜ハッカソンしよう」という文化の作り方

Kuu：エンジニアはもっと気軽に、「今日この技術が発表されたから、このあと 5 時間でハッカソンやりませんか」と招集するような、カジュアルなスタイルもいいかもしれないですね。

TODO: 即興ハッカソンの具体的な開催手順と、最低限必要な準備を解説する

即興ハッカソンで得られる「鮮度の高い技術キャッチアップ」

Kuu：エンジニアこそ、新しい技術が出たら必ずその技術を使ったハッカソンを小まめに開催した方がいいかもしれないですね。

TODO: 最新技術の発表直後にハッカソンを行うことで得られる学びの質について解説する

3.4 ハッカソン運営の実務——タイムライン・グルーピング・審査

デモ前のテスト時間を確保しないと発表が崩壊する

Kuu：事前に発表のテストをしっかり行って、テクニカルイシューを起こさせないことが大事ですね。

Lemio：他のハッカソンでは事前の練習がないことが多いですね。接続したら動画が再生できないとか。

Kuu：僕も 1 回、テストでは音が出たのに本番で音が出なかったことがあります。

TODO: デモ前テストの重要性と、具体的なチェック項目を解説する

グルーピング手法——ランダム vs スキルミックス vs テーマ別

TODO: チーム分けの各手法のメリット・デメリットを比較する

審査基準と賞の設計——多くの参加者が受賞できる仕組みが次回参加を生む

Kuu：ハッカソン運営をする側としては、賞をなるべく増やしていきたいです。小さな賞でも、受賞することでハッカソンを好きになる理由が生まれます。ハッカソンの沼にはまるきっかけになりますね。

Lemio：作ったものに正解はないので、本来はみんな優勝みたいなものです。作りきった人たちは全員優勝、というところはありますね。

Lemio：全員のピッチを聞くと、練習できていないチームの発表は中身が薄いのに時間が長くなりがちです。

Kuu：予選・決勝方式もありだと思います。予選でプロダクトを触って評価し、ファイナリストだけがピッチをする。プロダクトの完成度とピッチの質の両方を評価できます。

Lemio：予選側は動画を作る必要が出てくるので、参加者の負担は増えますが、1〜2 ヶ月かけたハッカソンであればファイナリストのピッチは精練されていて、聴く側も飽きません。

TODO: 賞の設計哲学（技術賞・アイデア賞・プレゼン賞の分離）と、受賞体験がリピーターを生むメカニズムを解説する。予選・決勝方式の審査のメリット・デメリットと、規模に応じた審査設計の指針も述べる

時間管理のコツ——なぜハッカソンは 2 時間押すのか、押さない運営の秘訣

Lemio：他のハッカソンは絶対に 2 時間ぐらい押すんですよ。なんで押さないんですか？

Kuu：拍手で盛り上げるんです。みんなが「よかったよ」という雰囲気になると、「ここで終わった方がいい」という空気ができあがります。それでオンタイムに収まるんです。

Kuu：開発が簡単になった今、アイデアを練る時間を十分に取らないと、ありきたりな作品になりがちです。「開発しなきゃ」という焦りが、アイデアのブラッシュアップを後回しにさせてしまう。

Lemio：逆に、ある時間まではアイデアを必ず議論しなければならない「開発禁止期間」を設けて、そこから先は開発に集中するというルールはどうですか。

Kuu：それはメリハリがあっていいですね。「ここまではアイデアを練ってください、ここからはアイデアを変えないでください」と明確に区切ることで、両方の質が上がりそうです。

TODO: ハッカソンが時間を押す原因と、タイムキーピングの具体的なテクニックを解説する。アイデア出しと開発の時間配分について、開発禁止期間の導入メリットと具体的な運用方法も述べる

3.5 ハッカソンに出続ける理由——承認・キャリア・アドレナリン

成果が即座に評価される快感は通常業務では得られない

Kuu：ファイナリストに選出されたとき、仲間同士で「おめでとう」と熱狂できたのは大きいですね。やみつきになりました。

Lemio：勝つのはいいですよね。アドレナリンが高まりますし、認められるのはやはり大きいです。

TODO: ハッカソンでの即時フィードバックが参加者のモチベーションに与える影響を分析する

ハッカソン実績がポートフォリオとして転職・昇進に効く

TODO: ハッカソン実績がキャリアに与える具体的な影響（就活、スポンサー企業との接点、スキル証明）を解説する

「勝つ」以外のモチベーション——仲間・学び・自己更新

Kuu：プログラミングが好きで、ダラダラ作るよりも作りきった方が楽しい。たまに優勝できるのが良くて、ギャンブル性が高いんです。特に AI が出てきて、優位性がある中で戦えるのは最高じゃないですか。

TODO: 勝利以外のモチベーション（学び、出会い、新技術の習得、自己表現）を整理する

第 4 章

AI 活用の実践テクニック――ハッカソンでも仕事でも使える知恵

第 1〜3 章で語られた知見を横断し、ハッカソンでも日常業務でも活用できる AI 活用の実践テクニックを 3 人でまとめます。具体的なツール名ではなく、考え方とアプローチに焦点を当てます。

4.1　AI に正しく仕事をさせるための前提条件を整える

公式ドキュメントの URL をプロンプトに含めるだけで回答精度が劇的に上がる

Kuu：第 1 章「AI ネイティブという不可逆な変化――エンジニアの仕事はもう元に戻らない」でガードレールの考え方を話しましたが、具体的な方法として一番効くのは、最新の API リファレンスの URL をプロンプトに含めることです。AI に読ませるだけで回答精度が全然違います。

Lemio：ドキュメントの URL を渡すだけでそんなに変わるんですか？

Kuu：変わります。特に JS や Python はライブラリのバージョン更新が速いので、公式ドキュメントを起点にしないと AI が古い API で書いてきます。URL を渡す、バージョンを明示する、この 2 つだけでつまずきにくくなります。

AI が生成するコードの品質は、与えるコンテキストの質に比例する。公式ドキュメントの URL をプロンプトに含める手法は、最も手軽でありながら効果が大きい。

具体的な手順は単純だ。使いたいライブラリや API の公式ドキュメントページの URL を、プロンプトの冒頭に「参考ドキュメント」として記載する。たとえば「以下の公式ドキュメントを参照してコードを書いてください」と一文添えるだけで、AI は学習データに頼らず、最新の仕様をもとにコードを生成する。

この習慣は、コンテキストの「鮮度管理」と呼べるものだ。情報には賞味期限がある。AI の学習データは数ヶ月前のスナップショットであり、日々更新されるライブラリの API とは乖離がある。公式ドキュメントの URL を渡す行為は、AI の持つ古い知識を最新の情報で上書きする作業にほかならない。

ここで大事なのは、AI の出力が期待と異なったときの受け止め方だ。「AI が間違えた」

と嘆くのではなく、「自分のプロンプト設計に不足があった」と捉える。AI は与えられたコンテキストの範囲で最善を尽くしている。出力の品質が低いなら、入力の品質を疑う。この発想の転換が、AI 活用の上達を加速させる。

古いライブラリバージョンのコードを AI が自信満々に生成する問題

Kuu：AI は学習データの時点で最新だったバージョンのコードを自信満々に書いてきます。動くけれど非推奨の API を使っていたり、そもそもメソッド名が変わっていたりする。

Lemio：それ、動くだけに気づきにくいですよね。

Kuu：だからこそ、使うライブラリのバージョンとドキュメントをプロンプトで明示する習慣が大事です。

この問題の厄介さは「動いてしまう」ことにある。エラーが出ればすぐに気づくが、非推奨 API で書かれたコードは正常に動作する。しかし将来のバージョンアップで突然壊れたり、セキュリティパッチが当たらなかったりする。技術的負債が静かに蓄積する。

典型的な例を挙げる。Python の `requests` ライブラリではセッション管理の推奨パターンが変わっているし、React ではクラスコンポーネントから Hooks への移行が完了している。しかし AI は学習データの割合に引きずられ、古い書き方で生成することがある。

対策は 3 つだ。第一に、プロンプトでバージョンを明示する。「Python 3.12、React 19 を使用」と書くだけで、AI は対応する API を選択しやすくなる。第二に、`package.json` や `requirements.txt` をプロンプトに含める。依存関係の全体像を AI に渡すことで、バージョン間の整合性が取れたコードが生成される。第三に、生成されたコードの `import` 文を確認する。非推奨のモジュールからインポートしていないか、廃止されたパッケージを参照していないかをチェックする習慣をつける。

Agent Skills を整備して AI の行動を仕組みで制御する

Kuu：プロンプトに毎回同じことを書くのは非効率です。Coding Agent には「Agent Skills」と呼ばれる仕組みがあって、プロジェクト固有のルールや慣習を事前に設定ファイルとして渡せる。これを整備しておくと、AI が毎回正しい前提で動いてくれます。

Sae：設定ファイルというのは、どのくらいの粒度で書くんですか？

Kuu：「このプロジェクトでは TypeScript を使う」「テストは Vitest で書く」「コミットメッセージは Conventional Commits に従う」くらいの粒度です。プロジェクトの README に書くような情報を、AI が読める形式で置いておくイメージですね。

Agent Skills の整備は、AI に対するオンボーディング資料の作成と同じだ。新しいメンバーがチームに入ったとき、プロジェクトの規約やコーディングスタイルを伝えるように、AI にもプロジェクト固有の文脈を渡す。

この仕組みの利点は、再現性にある。プロンプトに毎回「Prettier でフォーマットしてください」「エラーハンドリングは Result 型を使ってください」と書く代わりに、設定

ファイルに一度書いておけば、以降のすべてのやり取りで AI がその前提を踏まえて動く。チームメンバー全員が同じ設定ファイルを共有すれば、AI の出力品質がチーム全体で均一化される。

API 情報の提供も Agent Skills の重要な要素だ。プロジェクトで使っている外部 API のエンドポイント一覧、認証方式、レスポンスの型定義を設定ファイルに含めておくと、AI は API コールのコードを正確に生成できる。特にハッカソンでは、スポンサー企業が提供する API を初めて使うことが多い。API ドキュメントの URL と基本的な使い方を Agent Skills に書いておくことで、チーム全員が AI を通じて素早く API を活用できる。

4.2 設計を言語化する—— AI に伝わる設計書の作り方

入力と出力を言葉にできれば AI が間をつなぐ

Lemio：AI の時代に変わったのは、設計を言語化するスキルの重要性です。入力と出力を明確に言葉にできれば、その間の処理は AI が埋めてくれる。逆に言えば、言語化できない設計は AI にも伝わらない。

Kuu：「何を入れて、何が出てくるか」を定義するのが人間の仕事で、「どうやって変換するか」は AI の仕事になった、ということですよね。

Lemio：そうです。「この画面にユーザーが名前を入力すると、パーソナライズされた挨拶が表示される」——この一文があれば、AI はフォーム、バリデーション、表示ロジックまで一気に書ける。設計を言語化する力が、そのまま AI の活用力になります。

「設計を言語化する」とは、頭の中にある曖昧なイメージを、AI が解釈できる精度まで具体化する行為だ。これはプログラミングスキルとは別の能力である。コードが書けなくても、「入力は CSV ファイル、出力は月別売上のグラフ、グラフは PNG で出力」と言語化できれば、AI はそのとおりに実装する。

このパラダイムシフトは、ハッカソンの戦い方を根本から変える。従来は「どう実装するか」に時間の大半を費やしていたが、今は「何を作るか」「どんな入出力を持つか」の設計に集中できる。設計さえ明確なら、AI が実装を担当する。

実践的なアプローチとして、機能ごとに「入力→処理→出力」の三行を書く方法がある。たとえば「入力：ユーザーの現在地（GPS 座標）、処理：半径 1km 以内のレストランを検索してレビュースコア順にソート、出力：レストラン名・距離・スコアのリスト」と書く。この三行だけで、AI は API コール、ソートロジック、データ構造の設計まで一貫したコードを生成できる。

非同期 Coding Agent で並列開発を実現する

Kuu：Coding Agent の最大の利点は、非同期で動かせることです。人間が一つのタスクに集中している間に、別のタスクを AI に並列で進めてもらう。これはハッカソンでの時間効率を劇的に上げます。

Sae：具体的にはどうやって並列に動かすんですか？

Kuu：git worktree を使います。一つのリポジトリから複数の作業ディレクトリを作って、それぞれに別の Coding Agent を割り当てる。Agent A にはフロントエンドを、

Agent B にはバックエンドを、自分はインフラの設定をやる、という具合です。

非同期 Coding Agent の活用は、一人の開発者が複数人分の作業を同時進行させる手法だ。従来の開発では、一人が同時に着手できるタスクは実質一つだった。コンテキストスイッチのコストが高く、複数の作業を切り替えながら進めると効率が落ちる。しかし Coding Agent を並列で動かせば、人間はディレクションに徹しながら、複数のタスクを同時に前進させられる。

git worktree は、一つのリポジトリから複数の作業ツリーを作成する Git の機能だ。通常のブランチ切り替えと異なり、物理的に別のディレクトリとして存在するため、それぞれのディレクトリで独立して Coding Agent を起動できる。メインの worktree では自分が UI 設計を進め、別の worktree では Agent A が API 実装を進め、さらに別の worktree では Agent B がテストコードを書く――こうした並列作業が可能になる。

ハッカソンでは時間が限られているからこそ、この手法の効果は絶大だ。3 人チームで 6 時間のハッカソンに参加する場合、各メンバーが 2〜3 の Coding Agent を並列で動かせば、実質的な作業量は数倍に膨らむ。ただし、並列化の前提として、タスクの分割と依存関係の整理が不可欠だ。依存関係が複雑なタスクを並列で進めるとコンフリクトが頻発し、かえって時間を浪費する。

master 直 push 戦略――ハッカソンではブランチ運用を捨てる

Kuu：ハッカソンでは、あえて master ブランチに直接 push する戦略を取ることがあります。通常の開発ではブランチを切ってプルリクエストを出すのが常識ですが、2〜3 日の短期決戦ではそのオーバーヘッドが致命的です。

Lemio：コンフリクトの解消に時間を取られるリスクはないですか？

Kuu：あります。だからこそ、タスクの分割が重要です。フロントエンドとバックエンドで明確にディレクトリを分けたり、ファイルの担当を決めたりして、物理的にコンフリクトが起きにくい構造を先に作る。それでもコンフリクトが起きたときは、AI に「Please fix conflict」と指示すれば、大半の場合は自動で解決してくれます。

通常の開発プロセスでは、コードレビューとブランチ管理が品質の担保に不可欠だ。しかしハッカソンという時間制約の下では、プロセスの厳密さよりもアウトプットの速度が優先される。

master 直 push 戦略の前提条件は三つある。第一に、チームメンバー全員が担当領域を明確に分割していること。第二に、CI パイプラインが設定されており、push のたびにビルドとテストが自動実行されること。第三に、コンフリクト発生時に AI で即座に解決する体制が整っていること。

CI が失敗した場合のプロンプトも定型化しておくとよい。「CI が失敗しています。以下のエラーログを確認して修正してください」――これだけで Coding Agent はエラーを解析し、修正コミットを生成する。「Please fix CI failing」というシンプルな指示が、ハッカソンにおける CI 修復の定型パターンになる。

この戦略はあくまでハッカソンや短期プロトタイピングに限定した手法であり、長期運用のプロダクトには従来のブランチ運用を推奨する。

4.3 AI を学習パートナーとして使い倒す

理解できるまで何度でも聞き直す—— AI は嫌な顔をしない最強の家庭教師

Lemio：わかるまで聞けばいいんです。「わからない部分をもう一度説明してください」と何度でも聞く。そのやり方をレクチャーするのが、ググり方を教えるのと同じ感覚ですね。

Kuu：やり方さえわかれば、自走できますしね。

AI を学習パートナーとして活用する最大の利点は、「何度でも同じことを聞ける」ことだ。人間の先生やメンターに同じ質問を繰り返すのは心理的なハードルがある。しかし AI は何度聞いても嫌な顔をしない。これは冗談ではなく、学習効率に直結する本質的な特性だ。

効果的な聞き方にはコツがある。「わかりません」とだけ伝えるのではなく、「ここまでは理解できたが、ここから先がわからない」と境界を示す。AI は理解の境界線を起点に、より噛み砕いた説明を返してくれる。

たとえば「Docker の仕組みがわかりません」よりも、「コンテナが仮想マシンと違うことは理解したが、イメージとコンテナの関係がわからない」と伝えた方が、的確な回答が得られる。この「わかる境界」を伝えるスキルは、AI 活用だけでなく、人間同士のコミュニケーションでも応用が利く。

この手法をチームに広めることは、かつて「ググり方を教える」のと同じ意味を持つ。検索エンジンの登場時、「何をどう検索するか」を知っている人と知らない人で情報アクセスの格差が生まれた。AI の時代も同様で、「AI にどう聞くか」を知っている人と知らない人で学習速度の格差が生まれる。聞き方のスキルを共有すること自体が、チームの底上げになる。

未知の概念を「絵を見せて聞く」ように学ぶ——わからないことがわかれば深掘りできる

Lemio：昔テレビで見たんですが、アイヌの方にアイヌ語を教えてもらうとき、ぐちゃぐちゃの絵を描いて見せて、相手の反応から語彙を学んでいく方法がありました。わからないことさえわかっていれば、知識を広げられます。AI も同じように使えばいいんです。

Kuu：自分がわからないことさえわかっていれば、わかっている部分とわからない部分の境界を伝えて、わからない部分だけを深掘りしていけば全体が見えてきます。

未知の概念に向き合うとき、最初の壁は「何がわからないかがわからない」状態だ。この壁を越える方法として、AI に断片的な情報を投げて反応を見るアプローチがある。

たとえば、新しいフレームワークの概念を学びたいとき、自分が知っている類似概念を AI に伝える。「React の state に似たものが Svelte にもあると思うが、何と呼ぶのか」と聞けば、AI は「Svelte ではリアクティブ変数と呼び、$: という構文で宣言する」と教え

てくれる。既知の概念を足がかりに、未知の領域を探索する手法だ。

この学び方の本質は、「自分が何を知らないかを知る」ことから始まる点にある。ソクラテスの「無知の知」と同じ原理だ。自分の知識の境界線を認識し、その境界を少しずつ外側に広げていく。AI は、その境界線の拡張を手助けする最適なパートナーとなる。

ハッカソンにおけるこの手法の価値は大きい。限られた時間の中で、初めて触る技術スタックを学ばなければならない場面は多い。「何がわからないかを整理して、AI に段階的に聞く」というアプローチを身につけておけば、新技術のキャッチアップ速度が格段に上がる。

音声 AI で曖昧な検索をする——オノマトペや身振りで伝える新しい検索体験

Kuu：テキスト検索だとテキストに言語化しなければなりませんが、音声なら擬音語やオノマトペを使って、ふわっとした形から絞り込んでいくことができます。

Lemio：昔見た YouTube の動画で、タイトルが全然わからなくても、「こういう感じのやつ」と探していったら出てきたりするんですよ。

テキスト検索には構造的な限界がある。検索したいものの正確な名前や用語を知らなければ、検索クエリを組み立てられない。しかし音声 AI を使えば、言語化が不完全な状態でも検索が成立する。

たとえば、UI のアニメーションパターンを探しているとき、「要素がふわっと現れて、スッと消える感じのやつ」と音声で伝えれば、AI は「フェードイン・フェードアウトのアニメーション」を提案してくれる。テキストで「CSS アニメーション フェード」と検索するには、まず「フェード」という用語を知っている必要がある。音声なら、オノマトペで伝えるだけでよい。

この手法は、デザインの領域で特に効果を発揮する。「なんかこう、カードがペラッとめくれる感じのトランジション」「ボタンを押したときにポヨンと跳ねるアニメーション」——テキスト検索では絶対にたどり着けない表現でも、音声 AI は意図を汲み取って適切な技術用語やライブラリに変換してくれる。

ハッカソンの文脈では、デザイナーとエンジニアの間のコミュニケーションにもこの手法が使える。デザイナーが「ここ、もうちょっとシュッとした感じにしたい」と言ったとき、その曖昧な要望を AI が「余白を増やしてフォントウェイトを細くする」という具体的な実装に変換する。曖昧さを許容する検索は、チーム内の意思疎通の潤滑油にもなる。

■コラム:「なんかこう、ぐにゃっとしたやつ」で検索できる時代

従来の検索は「正確なキーワードを知っている人」が有利だった。CSS の `border-radius` を知らなければ「角丸」にたどり着けないし、`box-shadow` を知らなければ「影をつける方法」を検索できなかった。しかし AI の登場で、「なんかこう、角がまるっとしてて、ちょっと浮いてる感じのカード」と言えば、`border-radius` と `box-shadow` の両方を含むコードが返ってくる。

面白い事例を紹介する。ある開発者が「データがドバーッと流れてくる感じの画

面」と音声で伝えたところ、AI はリアルタイムストリーミングのダッシュボード UI を提案した。別の開発者が「グラフがニョキニョキ生えてくるアニメーション」と伝えたところ、D3.js のアニメーション付き棒グラフのコードが返ってきた。正確な技術用語を知らなくても、感覚的な表現で AI が意図を読み取る時代がすでに来ている。

付録 A

ハッカソン参加・運営チェックリスト

本編で語られたノウハウを、すぐに使えるチェックリスト形式でまとめました。ハッカソンに参加する方も、運営する方も、この付録を手元に置いてご活用ください。

A.1　参加者向け——ハッカソン前日までに済ませておくこと

- 開発環境のセットアップと動作確認
- 使用予定の API キーの取得・設定
- Git 環境の準備（アカウント、SSH 鍵、基本操作の確認）
- AI ツール（チャット型 AI、コーディングアシスタント）のアカウント準備
- チームコミュニケーションツールの確認（Slack, Discord 等）
- デモ用のプレゼンツールの準備
- 当日の持ち物リスト確認（PC、充電器、変換アダプタ、延長コード等）

TODO: 各項目の詳細な手順を追記する

A.2　参加者向け——当日の持ち物とセットアップ手順

- ノート PC（フル充電済み）
- 充電器・電源タップ
- ディスプレイ変換アダプタ
- イヤホン・ヘッドセット
- 名刺（あれば）
- 飲み物・軽食

TODO: オンライン参加の場合のセットアップ手順も追記する

A.3　運営者向け——開催 2 週間前からの準備タイムライン

- **2 週間前**：会場確保、テーマ決定、参加者募集開始

- **1 週間前**：審査基準の策定、審査員への依頼、賞品手配
- **3 日前**：タイムテーブル最終確認、参加者への事前案内送付
- **前日**：会場設営、Wi-Fi・電源の動作確認、デモ環境テスト
- **当日朝**：受付準備、チーム分け、ルール説明資料の最終確認

TODO: 各項目の詳細とチェックポイントを追記する

A.4 運営者向け——当日のタイムテーブルテンプレート

1 日型ハッカソン（8 時間）の例

- 09:00 - 09:30　受付・チーム分け
- 09:30 - 10:00　オープニング・テーマ発表・ルール説明
- 10:00 - 11:00　アイデア出し・チーム内合意
- 11:00 - 17:00　開発タイム（昼食休憩含む）
- 17:00 - 17:30　発表準備・デモテスト
- 17:30 - 18:30　発表・デモ
- 18:30 - 19:00　審査・結果発表・クロージング

即興型ハッカソン（5 時間）の例

- 18:00 - 18:15　テーマ共有・ルール説明
- 18:15 - 18:45　アイデア出し
- 18:45 - 22:00　開発タイム
- 22:00 - 22:15　デモテスト
- 22:15 - 22:45　発表
- 22:45 - 23:00　投票・結果発表

TODO: 各時間帯の運営側の動きと注意点を追記する

A.5 AI 開発環境のセットアップ——最低限入れておくべきツールと設定

- チャット型 AI ツール（ブレスト・調査用）
- コーディングアシスタント（IDE 統合型）
- バージョン管理（Git + GitHub/GitLab）
- CI/CD 環境（GitHub Actions 等）
- API リファレンスのブックマーク

TODO: 各ツールの具体的な設定手順とおすすめの構成を追記する

あとがき

この本で伝えたかったこと—— AI 時代に変わるものと変わらないもの

この本は、2026 年 2 月のブッカソンで生まれました。

3 人で集まって、ハッカソンや AI について思いの丈を語り合い、その対話を本という形にまとめる——まさにハッカソンのフレームワークを使って本を書くという実験でもありました。

Kuu：AI で何でも作れるからこそ、手を動かすことが大事になってきています。

Lemio：作りたいものを作ろう。

ツールは変わり続けます。でも「作りたい」という衝動は変わりません。AI の時代だからこそ、限られた時間で人と協働して何かを作る「ハッカソン」の価値は高まっています。

テクノロジーの歴史を振り返れば、革新が起きるたびに「人間の仕事がなくなる」と言われてきました。印刷機が写本師を、自動車が馬車を、表計算ソフトが経理担当を置き換えたように、AI もまた多くの作業を自動化しています。しかし、どの時代にも変わらなかったものがあります。「何を作りたいか」「なぜ作りたいか」という問いに答える力——それは技術がどれだけ進歩しても、人間の内側からしか生まれません。

AI がコードを書き、デザインを生成し、文章を整える時代において、人間に残される最も価値ある能力は「衝動」だと私たちは考えます。「これが欲しい」「これを作りたい」「この問題を解決したい」——その衝動こそが、すべてのプロダクトの起点です。AI はその衝動を形にする速度を劇的に上げてくれますが、衝動そのものを生み出すことはできません。

ハッカソンは、その衝動を最も純粋な形で発揮できる場です。締め切りがあり、仲間がいて、発表の場がある。この三要素が揃うことで、普段は「いつかやろう」と先延ばしにしているアイデアが、具体的なプロダクトとして世に出る。AI の登場で、その「形にする」までの距離が格段に縮まりました。かつては技術的なハードルで諦めていたアイデアが、今なら週末の数時間で動くプロトタイプになる。

手を動かし、人と語り、形にする——その繰り返しが未来を作る。

AI が生成する文章と、人間の生の声から生まれる文章。その両方を活かしながら、この本は作られています。読者の皆さんに、私たちの熱量が少しでも伝わっていれば幸いです。

ハッカソンで会いましょう。

著者紹介

Kuu

Software Engineer @Mercari。AI 活用コンサルティング・受託開発 @Kuu Systems。趣味は海外旅行と Vibe Coding。ハッカソンでは「AI をガードレールで囲んで暴れさせる」スタイルで、非同期 Coding Agent の並列運用や master 直 push 戦略など、速度を最大化する開発手法を追求している。

Sae

TODO: 著者紹介を記入してください

Lemio

清水れみお。通信建設業界で 10 年の経験を積んだ後、LLM の可能性に魅せられてキャリアを転換し、LLM アプリエンジニアとして活動中。オタク知識と技術知識を掛け合わせたものづくりを実践しており、ハッカソンでは「入力と出力を言葉にできれば AI が間をつなぐ」という設計言語化のアプローチで、アイデアをプロダクトに変換する。

AI Native ハッカソン徹底攻略

3 人のエンジニアが語る、AI と共創するものづくりの実践知

2026 年 2 月 7 日　初版第 1 刷　発行

著　者　Kuu、Sae、Lemio