個のスケールのためのAI時代のスキルの磨き方

あるいは「全部賭ける」ための発想

@mizchi
Colab Conf U35

自己紹介

- @mizchi | 竹馬光太郎 37歳(1988)
- ソフトウェアエンジニア/Node.js/React/フロントエンド
  今回のバストアップ画像はAIで捏造しました。なかったので...
- **経歴**: ゲーム開発、教育系ベンチャー、Qiita、サードパーティスクリプト/ノーコードエディタの開発
- **現在**: パフォーマンスチューニングの傭兵。AI周辺ツールの開発
- 色々と技術記事を書いてきた
  - 2014: なぜ仮想DOMという概念が俺達の魂を震えさせるのか | Qiita
    たぶんこれで日本で React が流行った
  - 2025: CLINEに全部賭けろ | Zenn
    たぶんこれで日本で Coding Agent が流行った

今日話すこと

1. 第一部: 苦しみと怒りによってイノベーションを発見する
2. 第二部: 結局プログラマは何をしているのか
3. 第三部: じゃあAIって一体何なんだよ

余談: このスライドについて

- この資料は https://gist.github.com/mizchi/aa16146240e566271e7c0c9fe9c6b3e8 を元に、面白半分で `GenSpark` で生成したものです。
- 読みづらくても `AIのベンチマーク` だと思って付き合ってください

第一部

苦しみと怒りによって
イノベーションを発見する

～高校生

- 2ちゃんねらーで、典型的なスクリプトキディ
  - 親のPCを勝手にBIOSリセットしてバックドアをつくる
  - 親がいない間にゲームをする
- 高校生まではプログラマ板(マ板)を見て、IT業界を避け気味
- 浪人中に梅田望夫: Web進化論を読んでから「Web業界面白そうじゃん」

原点: Twitterのアーリーアダプターが楽しそう

- 大学1年時点: 思い立った時点で回りにプログラマはいない
- 同世代のプログラマを見ていた @ymrl @r7kamura @udzura @yuiseki @chokudai ...
- はてな界隈のハッカーっぽい人達: @mala @dankogai @hamachiya @amachang @naoya ....
- **2008~: はてな界隈**
  - みんな本気で殴り合ってるのが楽しそう(**モヒカンの時代**)
  - はてな匿名ダイアリーで記事を書いて、どうやったらバズるかを学習

「はてな界隈のモヒカン族」
技術論争でまさかり（トマホーク）を振り回し、本気で殴り合う様子を表現

プログラマを目指す？

- CS専攻ではなくプログラミング学習が出遅れてる自分は、他の人と同じことをやっても差別化できない
- 逆張り気味で Python => Node.js と選択
  - 当時は Ruby 全盛だが、文法が複雑で独学に向かないという判断
- 文系大学生の有り余る可処分時間で、学習時間を捻出

大学時代 2008~

- **早稲田大学 人間科学部 人間情報科学科**
  - 高校は理系だが日本史が得意すぎて(偏差値80超)、文理両方受験可能な学部を選択
  - 当時の興味: **「無意識」や「バイアス」がどのように形成されるか**
- **松居研究室: 専攻 感性情報処理・暗黙知記述**
  - やってたこと: 学習者を二群に分け、教育メソッドの有効性を計測するための実験を繰り返す
  - => 教育工学会のトレンドが肌に合わず、大学院の内定を捨てて学部就職
- **アカデミックは「全然真面目にやってない」が今非常に役に立ってる**
  - なぜなら暗黙知記述とは**AIプロンプティングそのもの**だったので...
- **大学時代にやったバイト(B3)**
  - Python で論文を解析する自然言語処理
  - Android jar をデコンパイルしてリバースエンジニアリング

就職前の自己分析

- **とにかく自分は仕事を ゲーミフィケーション する必要がある**
  - 例:「受験勉強」が持続しないので、日本史小説を読み続けた
  - プログラマ職= パズルゲームを解いて金が貰えるなんて最高
- **強ADHD**
  - 一度熱中したら問題が解けるまで寝食忘れて没頭できる
  - 一度問題を理解したり、一度忘却したら、興味が持続しない
- **興味と仕事を無理矢理一致させるために、自分自身をハックする**
  - 仕事は「面白いパズルのお題をくれる環境」かつ、雇用主と Win=Win でないといけない
  - 挑戦的なテーマに挑戦するなら 普通の奴らの上をいく 必要がある

2011 付近の Node.js(v0.2~0.4)

Python

- 最初に覚えた `Python` は「仕事がない」「仕事が選べない」

Node.js

- `JS` という言語は(モダンレガシー問わなければ)潰しが効く

- **`Node.js (2009~)` という若くて勢いのあるコミュニティがある**
  - V8 でサーバーサイドNode.jsが現実的に
  - WebSokect サーバーを作れる socket.io が唯一無二という競争力
  - 就活ポートフォリオとして、一人で MMO を作った

自分だけのビルドを作る

- 理想的には「コンピューターサイエンスの全てに精通」したいが、間に合わない

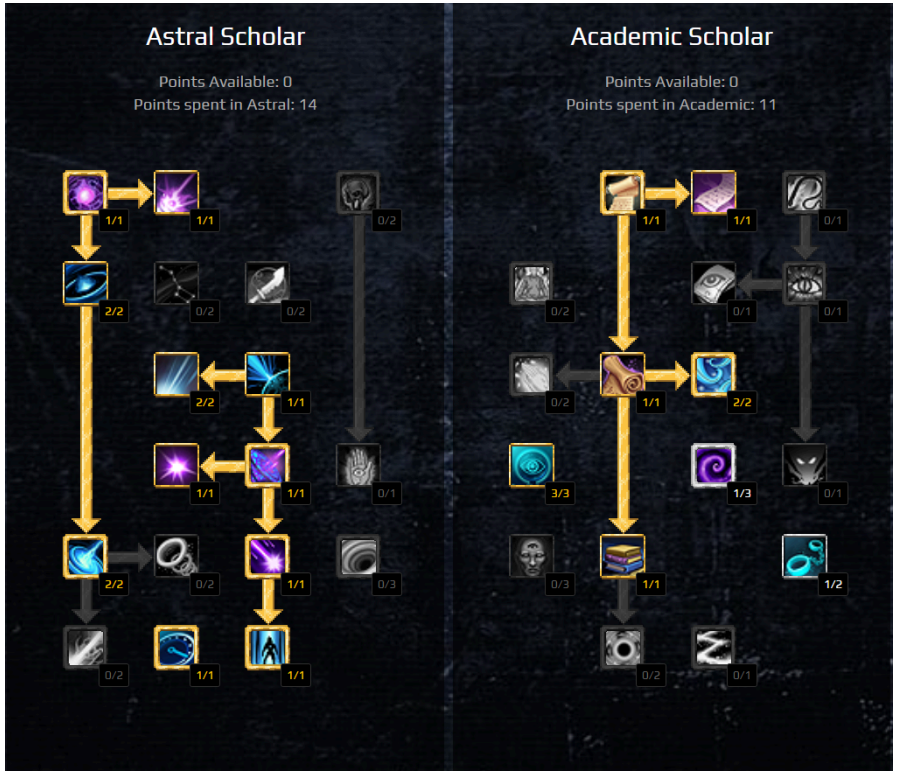
- キャリア形成＝有限時間の有限リソース最適化で最強のビルドを目指す

  → 学習リソース(やる気、時間、体力)は有限であり、適正もある

  → 初期状態では先行者には勝てないが、同じスタートラインの人には勝ちたい

  → 旬なビルドを見つければ、先行者にも部分的に勝てる

- JSは貧弱な言語だったが、他の言語のプラクティスを移植すると「無双」出来そう



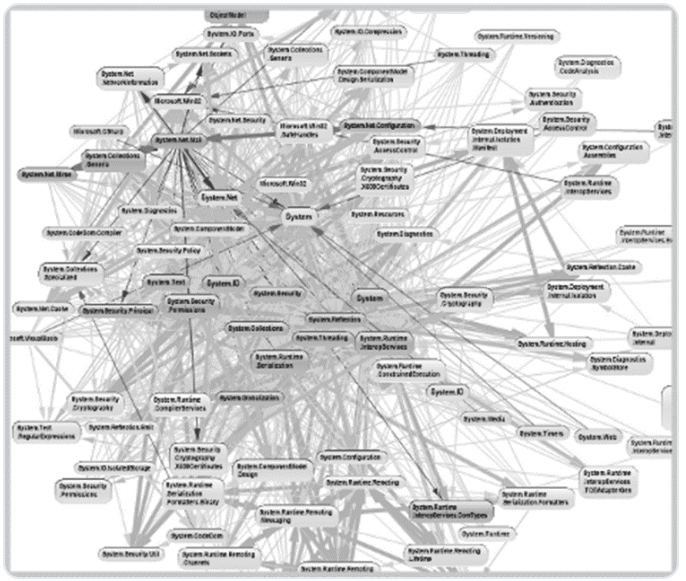
キャリア形成をゲームのスキルツリーのように最適化して「自分だけの最強ビルド」を作る

最初の仕事: @ffu_さんとの思い出

- 新卒で4社ぐらい応募し、udzuraさんに誘われてゲーム会社に就職
- 最初のメンターである藤村大介さん(現 STORE CTO) に鍛えられる
  - Ruby on Rails をコアとして、関数型言語のエッセンスを積極的に取り込む
  - プログラミング言語とOSSの発展を人文学的に捉える視点
- 最初の大規模プロジェクト
  - すでにあるUnity のゲームクライアントを(短期間で) HTML5 に移植
  - プログラマだけで20人ほどの大規模(寄りの)開発

Node.js得意ならJS設計できるよね

- 別のチームの **設計破綻したプロト** を引き取るところからスタート
  - ベースは Backbone + jQuery
  - 1画面作った段階でコードが破綻して機能追加が不可能に
- 手を動かしながら段階的に設計をする
  - Ruby経験者が多いので `CoffeeScript` による OOP+MVC(MVVM) を提案
  - mocha + jsdom による自動テストの導入
  - 当時まだ採用用例が少ない GitHub PR によるワークフローを採用



設計破綻したプロトタイプのスパゲッティコード状態

当時の悩みと、わかったこと

当時の悩み

- 多人数同時開発だと自然とコードは壊れていく
- 集団開発で、コード(+人間)の `「コンフリクトの解決」` が一番大変
- 「俺の経験が足りないから...?」 => いろんな人に相談する
- フロントにサーバーのMVCが適用できない(と主張すると異常者扱い)

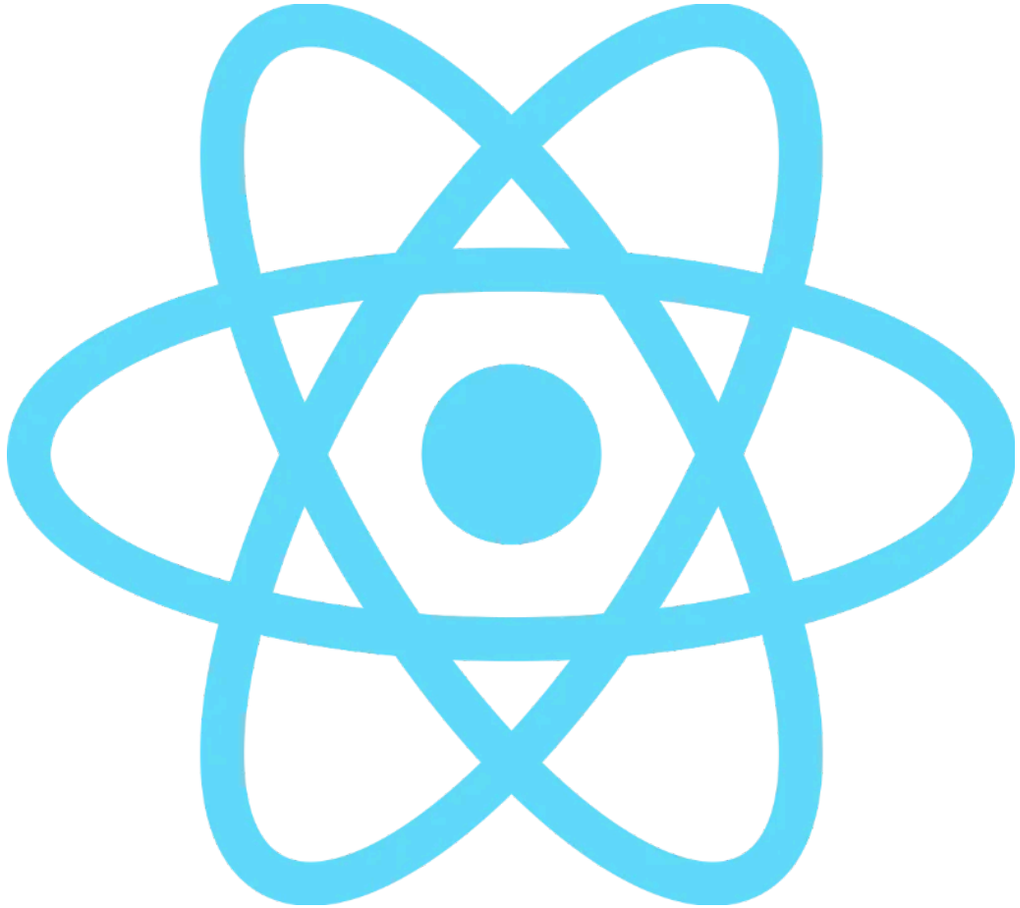
わかったこと

- `速度と複雑性を両立する設計手法` は、既存のコミュニティの中にない
- 外の言語コミュニティとそれを援用するため AltJS に解を求める
- いろんな言語に手を出す： `Clojure(Script)`, `Scala(JS)`, `Haxe`, `PureScript`, `js_of_ocaml`, `TypeScript`, `Flow`
- 勉強になったが、どれも `フロントエンドの実現手段としては、しっくりこない`

2014: Reactとの出会い

- 藤村さんから「関数型界隈で話題の面白いJSライブラリがある」と React を紹介される
- 最初は筋悪に感じた。「またライブラリ独自テンプレートか？」
- 複雑化した MVVM で同等の表現はできてはいた
- しばらく寝かせてから **自分の抱える問題の大部分を解決する** のに気づく
  - 自動差分適用はサーバーサイドMPAのメタファとワークフローが適用できる
  - 差分適用が自動化されれば、関数型言語の `UI=View(Model)` 発想が援用できる
- 半年ほど手元で検証
  - APIが導く設計の良さで、学習と実行のオーバーヘッドを踏み倒せると判断

2014 **Reactとの運命的な出会い**

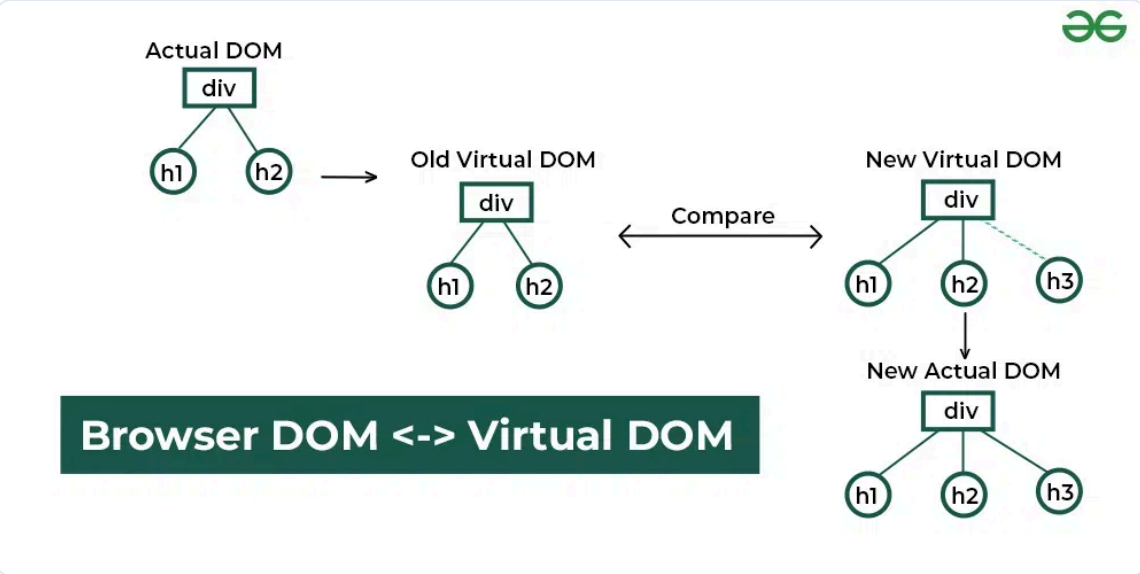


React - A JavaScript library for building user interfaces

「UIをコンポーネント化する発想が革新的だった」

なぜ仮想DOMが魂を震わせるのか

- なぜ仮想DOMという概念が俺達の魂を震えさせるのか | Qiita (2014)
- これ以上自分が苦しまないために React の優位性を訴えた
- デファクトの jQuery を否定するなら、相応の理屈が必要
- 自分が使うためにも **「コミュニティを説得するしかない」** と判断



Virtual DOM vs Real DOM: Reactの仮想DOMは差分のみを効率的に適用し、パフォーマンスを劇的に向上させる革新的な仕組み

キャリア前半による学び

- **良かったこと**: Node.js の成熟速度と、自分の成長曲線が一致していた
  - フロントエンドのパッケージマネージャがない => npm + browserify/webpack
  - jQuery では巨大SPAの状態を管理できない => React
  - 野良ライブラリに型がないと AltJS が機能しない => 後の DefinitelyTyped(`@types/*`)
- 手を動かしていて、不満を貯めていたからこそ、それを解決する`新技術の価値`に気づけた
- 「`UIにおける速度と複雑性の両立`」という非常に良いテーマを持てた

引用: Simple Made Easy

> 人間には限界がある
>
> - 理解することなく信頼できるものは作れない
> - 一度にごくわずかな数のことしかできない
> - 絡まったものは一緒に考えるしかない
> - 複雑さ(complexity)は理解を損なわせる

Rich Hickey, 2011

なぜ技術記事を書くか: コミュニティに介入する覚悟

- 同僚やコミュニティを説得できなければ存在しないのと同じ
- Webエンジニアは OSS という巨大な潮流 の中で生きている
  - コミュニティと正しい問題意識を共有できていれば、それは時間で解決される
- **強い言葉を使うなら、その説明責任を果たす**
  - 「実体のないハイプ技術」は駆逐されるべき
  - 「自分の直感が正しいという確信」「誰かにその審美眼を認めてもらう信頼」は、手を動かして苦しむことでしか手に入らない
- (承認欲求も間違いなくある)

第二部

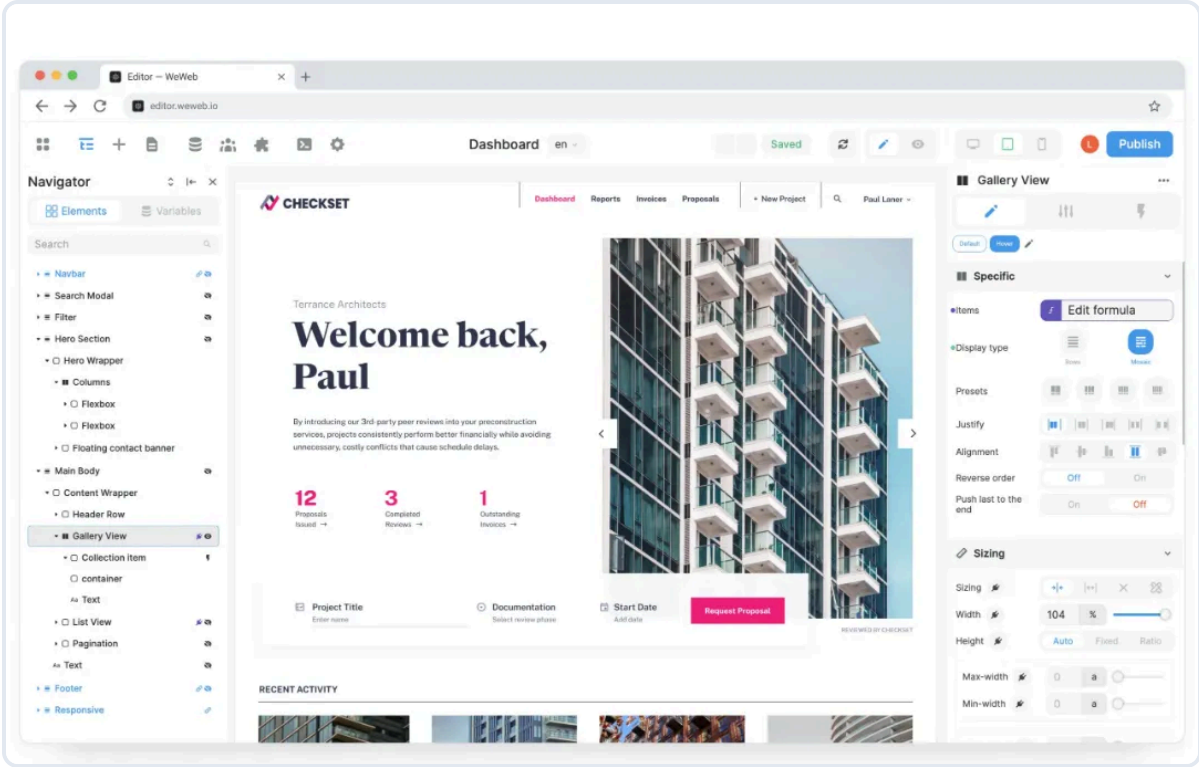
結局プログラマは何をしているのか

2017~: フロントエンドの富豪的開発の歪みの時代

- React以降、SPAで富豪的なリッチクライアントを作れる時代になった
- エコシステムが複雑に進化して、`フロントエンドの専門職`が発生した
  - これには対モバイルアプリという文脈もある
- **Next/Nuxt** がその複雑な層を一旦隠蔽したが...
  - また漏れ出しており、`RSC`のような更に複雑な層が追加されている(未解決問題)
  - チューニングのレイヤーが`CDN`や`サーバーサイド`に及んでおり、言語的な議論が活発化
- **2019~:**
  - 自分の価値は`フロントのパフォチュ`によって発揮すべきと考えた
  - 歴史的に負債が多い`サードパーティスクリプト`に挑戦(前職プレイド)

ノーコードエディタ開発の経験

- 非エンジニアでも、DnDでUIを簡単に(本当に？)作れる環境
  - これは「ホームページエディター」や「RPGツクール」に連なる失敗の歴史
  - 熟達するほどに存在が邪魔になる
- 経験的に GUI は人間の理不尽が直撃したが、GUIを生成するノーコードはもっと顕著
  - **プログラマとして発想**では、データ構造とUIの対応をDSL化したい
  - **プログラマとして訓練を受けない人の要望**は、「物事には論理と構造がある」が前提にない
  - 特定のユースケースに特化することは簡単だが、一つ機能を提供するごとに、それ以外の伸びしろを失う
  - プログラマの発想で一般化しすぎると、それはプログラミング言語になり、高い学習コストに

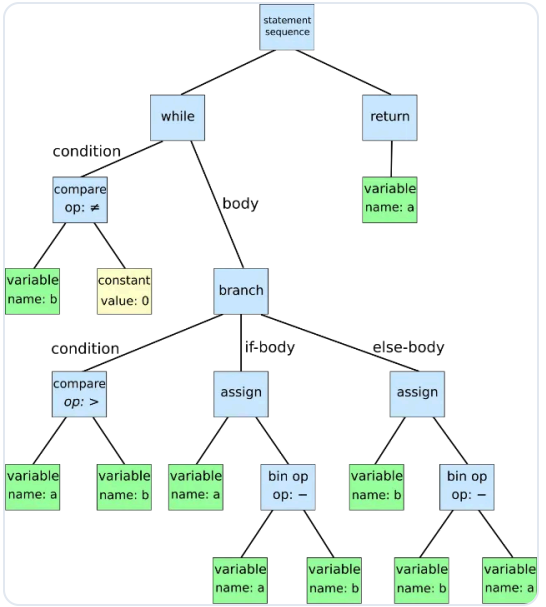


典型的なノーコードエディタのインターフェース例

プログラマとは何をしているのか

- プログラマは**プログラミングが万能ツールという幻想**を捨てている
  - 学習の初期段階で、制約を受け入れるイニシエーションがある
  - これを認められないと、プログラミング入門に失敗する

- **木構造と遷移グラフ**で対象の概念をモデル化する
  - 遷移グラフは、可読性のあるチューリングマシンの表現の一つ
  - 手続き型の操作的意味論か、静的関数型の表示的意味論かの派閥はあるが、概ねそう



抽象構文木（AST）の例: プログラマは木構造やグラフでモデル化し、コードに落とし込みます。

プログラマの世界への期待値

- 全員が手段としてのプログラミングは覚える必要はないが、「どの制約下にいるか」の世界観は共有したい
  - 光速度等の物理限界は普通は踏み倒せない
  - それらに対して、「キャッシュ」や「結果整合」という概念がある
  - 条件分岐や、データ構造が「ある」
  - Google/Apple のやってることは安易に真似できない
- 打率が高い仕事をしたい
  - 「素人アイデア」は制約から解き放たれているのではなく、思い込みに縛られている
  - 良い企画者やドメインエキスパートは、プログラミングを経由せずに同等の概念を獲得している
- GUIという幻想
  - GUI という「現象」だけではなく、「現象」の裏にある「物自体」の含む構造に思考が及ぶ必要がある (カント)

2つのアプローチ

専門家として非専門家に寄り添い続ける

- 双方向に対話をする。(権力構造に屈せず)根拠を届ける環境を作る
- 非効率性を指摘することも専門家の責務

全ての人類をニュータイプにする

- 進歩的/教育的アプローチ

※両方やって摩擦の少ない社会を目指しましょう、要はバランスというやつです

開発者が面倒くさい理由

- **知ることによって認識が変質し、それ以前には戻れない**
- プログラマとして計算機という「神視点の制約」を代弁するのは、傲慢さとして映る
  - 純粋なユーザー視点を保つのは困難で、ある種の二重思考や統合失調的ですらある
- ゲーム業界の人は開発者/プレーヤーの葛藤の言語化が上手い
  - 桜井政博のゲーム作るには
  - FF14: 吉田Pのプロデューサーレターライブ

第三部

じゃあAIって
一体何なんだよ

「CLINE に全部賭けろ」以前

2024/2

Meta のテストコード自動生成

カバレッジという明確なテスト指標があれば、AIは人間の手を離れて自走できる

https://arxiv.org/abs/2402.09171

2024/4

ノーコード用に自動検証ループによる自動マークアップを試作

自然言語からの変換で、論理構造の記述を任せる

生成結果が不安定なので、テストコードの品質と網羅性が大事と直感

https://github.com/mizchi/previs

2024/9

「プログラマ vs AI 生存競争」

最終的に手数で負けるのは既定路線

https://mizchi-20240920-nextbeat-slide.pages.dev/

2024/11

CLINE の存在を認識

2025/02 たまに使いつつ、「イケる」と確信

「CLINEに全部賭けろ」

- https://zenn.dev/mizchi/articles/all-in-on-cline
- ※当時まだ **CLINE以外のろくなコーディングエージェント** がないので、CLINEと言ってる
- Reactのときと同じく **「明らかに便利で、これが標準になってくれないと逆に困る」** という動機
- もはや言及しないことでセンスが疑われるという危機感すらあった

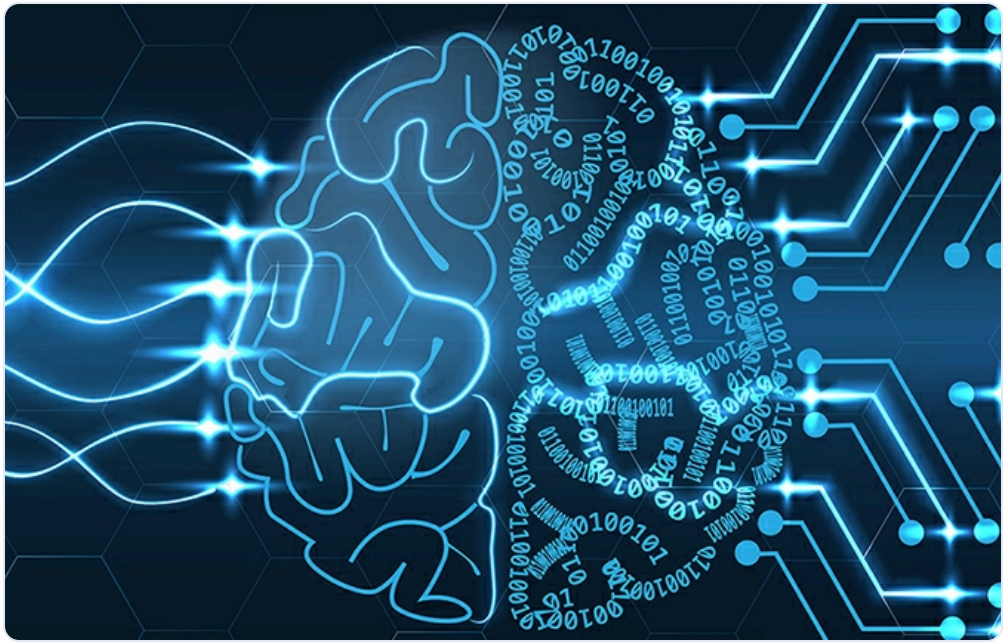
「全部賭ける」決断 - 技術選択における覚悟の象徴

CLINEに全部賭けろ以降

- 2025/7 既に存在が当然になりすぎて、**当時の感動を思い出せなくなりつつある**！
- 実利的なプログラマにも実用性が認められ、周辺ツールの開発が加速し続けている状態
- おそらく本物の**破壊的なイノベーション**であり、2024/2025でプログラミングは別物
    - 「蒸気機関」「鉄道」「電気」「原子力」「インターネット」とかそういう次元
- 好む好まざるに関係なく、不可逆な変化であり元には戻れない

AI/Coding Agent とは何か

- The End of Programming as We Know It
- 「アイデアを何でも実装できる」ではなく「**アイデアの妥当性を一瞬で検証してくる**」やつ
  - スタート地点や認識が間違ってる場合は、やっぱり辿り着けない
  - 今まで検証に二週間かかっていたものを、**1時間で辿り着く**
- 高度なAIと付き合うには、**論理的帰結として自分が否定されることになれないといけない**
  - 今のAIは、人間を肯定しすぎる。そこに性能の限界を感じる
- 「**破壊的イノベーション**」なのは間違いない



AIと人間の脳をつなぐ神経ネットワーク -
「アイデアの妥当性を一瞬で検証」する仕組み

AIとは「破壊的イノベーション」の典型である

> **破壊的技術とは：**
>
> 従来の価値基準の下では従来製品よりも性能を 低下させる が、新しい異なる価値基準のもとで いくつかの優れた特長 を持つ新技術のことである。優れた特長は 低価格・シンプル・使い勝手のよさ などであることが多い。

> **イノベーションのジレンマ：**
>
> 大企業にとって、新興の事業や技術は、 小さく魅力なく映る だけでなく、カニバリズムによって 既存の事業を破壊 する可能性がある。
>
> 既存の商品が優れた特色を持つがゆえに、その特色を改良することのみに目を奪われ、 顧客の別の需要に目が届かない 。そのため、大企業は、 新興市場への参入が遅れる傾向 にある。

出典：Wikipedia「破壊的技術」「イノベーションのジレンマ」

プログラマへのスキル要求の変化

シンタックス/セマンティクス

シンタックスの重要性 ↓
セマンティクスの重要性 ↑

メタプログラミング

より高次なメタプログラミング能力 ↑↑
判断基準を生成するプロンプティング

CI/自動テスト

CI・自動テストの設計能力 ↑↑

AI生成結果の解析

膨大なAI生成結果を動的/静的に解析する技術
破綻したプロジェクトのリカバリー能力

ドメイン専門知識

特定分野のエキスパートとしての言語化能力 ↑↑

セキュリティサンドボックス

セキュリティサンドボックスを設計する技術 ↑

↑ 重要性増加　↓ 重要性減少

AI時代の理解水準

1. 概念を認識している
2. 補助されれば使える
3. 能動的に適用可能な領域を発見できる
4. 自分で作れる
5. 自分で改善できる

2までの学習コストは低い。専門家として求められるのは3以上。

AI時代の学び方: 動画による視覚的理解

- 例 ソートアルゴリズムの可視化
  - https://www.youtube.com/watch?v=kPRA0W1kECg&t=3s
- 例 GPU の動作原理
  - https://www.youtube.com/watch?v=fKK933KK6Gg&t=88s
- 例 Transformer の可視化
  - https://www.youtube.com/watch?v=KlZ-QmPteqM
- **「概念や存在を認識していること」**の価値が大きい
- arxiv の URL を食わせれば、AI は実装できる

諦念：どこかは破綻する

- **歴史的に**、低レイヤーの性能向上で得られた伸びしろは、より上位レイヤーで消費される
  - スクリプティング言語
  - OS仮想化
  - リッチフロントエンド
- **AIによって得られた生産性も**、それが耐えうる限りで必ず別のレイヤーで消費される
  - 不安定なコンポーネントの増加により、最終的なユーザー体験は犠牲になる
  - 選択肢自体は増えるはずだが....

今の懸念

- 特に `Claude Code` 以降、CLIツールとしてより専門家がより深く使う方向へ進化
  - あまりに景色が違うが、この違いを言語化するのが困難
- 大企業や投資家はGUIや既存のワークフローの自動化を望んでいるが、方向性が違う
  - `VLM` が安価になるまで、このギャップは深まり続けるという予想
- 例えばスライド生成にAIを使ったが、視覚的な能力はない

技術選択ポリシー

常に破壊的イノベーションの側を選択する

星取表なんて見るな

破壊的イノベーションは最初は一点突破であり、網羅性は後から付いてくる

ハイプに騙されない

手を動かして自分で評価する。インプリンティングを疑う

結論

> **今は<u>コーディングエージェント</u>に**
> **全部賭けろ**