

Developers Summit 2026

# 「おとうさん、AIが発達したらクビじゃない？」 ——47歳、生涯エンジニア宣言。

2026/02/19

NECソリューションイノベータ株式会社

森松 祐樹

# 自己紹介



## 森松 祐樹（もりまつ ゆうき）

NECソリューションイノベータ株式会社  
イノベーションラボラトリ 探索グループ



アプリケーションエンジニア



プログラミング歴 35年（47歳）



C# / Java / Python / JavaScript / TypeScript / クラウド基盤



生成AIを用いた次世代の開発プロセスを探索中

### 信条

「触れなければ、始まらない。動かなければ、わからない。」

私たちは、  
ひとりひとりが持つ「らしさ」を探究し、  
その可能性が無限に広がる社会の実現を  
目指します。

イノベーションラボラトリーの

■技術者、■研究者、■事業開発者は、

仲間やパートナーと共に、思考と実験を重ねながら、

ヒトがより輝く明日を創ります。

ヒトを知り、ひとの可能性を広げる、人間中心ラボ

**INNOVATION LABORATORIES**

■エンジニアリング ■研究開発 ■事業開発の専門家集団

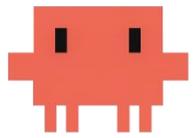




# 「おとうさん、AIが発達したら クビじゃない？」

—— ある日の夕食での、娘の一言

# 本当に、エンジニアは不要になるのか？



Claude



Codex



Gemini



GitHub Copilot

AIがコードを書くことは当たり前になり、開発工程全てへの適応が進んでいる。

## では、エンジニアの価値はどこに残り続けるのか？

ClaudeはAnthropic, PBCの商標です。  
CodexはOpenAI, Inc.の商標です。  
GeminiはGoogle LLCの商標です。  
GitHub CopilotはGitHub, Inc.の商標です。



それは、

今までの経験から生まれた、「なぜ」を問う力

# 本日本話すること

01

## AIの能力と限界

AIができること・できないこと、  
そして人間に残る価値

02

## 40代の経験値 × AI

AIを脅威から仲間に変える「経験」  
という武器

03

## AI時代を生き抜く

経験を活かしてAIに「任せる」方  
法

SECTION 01

# AIの能力と限界

# AIが「できる」こと

## 01

### 知識を持っている

膨大な情報を学習済み。経験の浅い人より知識は豊富。

- 新人が1週間かけて調べることを、AIは知識から即時に回答
- 「あのライブラリの使い方」  
→ サンプルコード付きで即答
- 英語ドキュメントも日本語で要約

## 02

### パターンを見つける

類似事例から回答を生成。「こういう時はこうする」が得意。

- NullPointerException  
→原因と修正パターンを列挙
- 「ビルドが通らない」  
→依存関係の競合を検出
- 「認証機能を作りたい」  
→ OAuth/JWT/セッションを提示

## 03

### 高速に処理する

人間の何倍も速く作業。大量のタスクも短時間でこなす。

- 1000行のコードレビュー  
30分 → 30秒
- 100ファイルのリファクタリング  
1日 → 10分
- API仕様書からコード生成  
半日 → 即座

# AIが「できない」こと

## 01

### 指示の背景を理解しない

「なぜこれを作るのか」「本当に必要なのか」を問えない。指示通りに作るだけ。

- 「ログイン機能を作って」  
→ 作るが、本当に必要なのはSSO統合だった
- 要件の矛盾に気づかず実装

## 02

### 責任を取れない

AIが出力した成果の責任は、使った人間が負う。

- 「このOSSを採用すべきか」  
→ 最終判断は人間
- セキュリティホールを生んでも訴えられるのは人間
- 「このPRをマージしてよいか」  
→ 承認は人間

## 03

### 暗黙知を持ってない

チームの経緯、顧客の本音、組織のルールは学習データにない。

- 「あの顧客は実はこう考える」を知らない
- 「前回の失敗」から学んでいない
- 「この関数は歴史的経緯でこう書いた」を知らない

# 例)指示の背景を理解しない

## プロンプト

設計書からコードを生成してください。  
設計書100行あたりのステップ数は1KStepが妥当。  
基準値に近づくまで、実装してください。

↓ 設計書 1,000行を入力

AIが作成したコード: 10KStep 完璧 😊

↓ 静的コード解析を試みる

作成されたコード : 10KStep  
コードの重複率 : **50%** ええっ 😱

## なぜこうなったのか

“量”のKPIを満たすために、最も低コストで確実な手段の**コードのコピー**に寄った

## 改善対策

重複抑制・品質ゲートを追加

## 改善したプロンプト

設計書からコードを生成してください。  
設計書100行あたりのステップ数は1KStepが妥当。  
基準値に近づくまで、実装してください。  
**ただし、行数水増し(コピペ・同型パターン量産)は禁止。  
重複が発生しやすい処理は必ず共通化してください。**

💡 AIは「なぜその基準になっているか」を理解しない → 経験者の「検証力」が必要

# AIとの関係は、こう変わる

## 現状（ツール）

- 関係性：人間が指示 → AIが実行
- AIの役割：言われた通りに作る
- 人間の役割：全てを指示する

「ログイン機能を作って」  
→ 言われた通りに作る

## 近い将来(パートナー)

- 関係性：人間（暗黙知・経験知）  
→ 「問う視点」を与える
- AIの役割：  
その視点で問い返す
- 人間の役割：判断・責任をする

「UX重視」を伝え「なぜログイン必要？」  
→ AI「SSOの方が良いのでは？」  
→ 人間が判断

💡 AIから良い答えを引き出すには、人間が良い「問い」を持っている必要がある

コードを書く能力より、

**「なぜ必要か」を問い、判断し、責任を取る力**  
が重要になる。

SECTION 02

**40代の経験値 × AI  
= 最強の組み合わせ**

# 40代エンジニアの「あるある」言い訳



「AIにかなわない」

→ 「AIは何でもできる」と  
思っている



「時間がない」

→ 優先順位の問題



「今更遅い」

→ 始めない理由探し



「マネジメントに専念」

→ 技術から逃げる言い訳

全部、私も思っていました。

でも、AIを使い始めて気づいたんです。

「経験の"活かし方"が変わっただけだ。『問い』を立てる力は、むしろ武器になる」と。

# 経験値が「問い」の質を変える

## X 経験の浅い人の問い → ◎ 経験者の問い

### エラーが発生した

- X 「このエラー、どう直せばいい？」
- ◎ 「なぜこのエラーが起きる設計になっている？」

### ログイン機能を実装する

- X 「ログイン機能、どう作ればいい？」
- ◎ 「ユーザーが離脱しない認証体験とは？」

### 処理が遅い

- X 「どうすれば処理が速くなる？」
- ◎ 「本当にここがボトルネック？」

### アーキテクチャ選定

- X 「マイクロサービス、どう構成すればいい？」
- ◎ 「この規模でマイクロサービスは必要？」

## なぜ経験値が問いを変えるのか

1. 結果の予測ができる  
「こう聞いたらこう返ってくる」がわかる
2. 見立てを持てる  
「この辺が怪しい」という勘が働く
3. 先回りできる  
問題が起きる前に防ぐ指示ができる
4. 全体を俯瞰できる  
部分最適ではなく全体最適の問いを立てる

💡 AIは「知識豊富だが判断は任せられない新人」。経験者の問いが、その能力を引き出す。

# 経験値 × AI で生まれる価値

01

## 失敗経験 × AI

→ 的確なリスク予測

「この設計は過去にこう失敗した」という知見をAIに伝え、同じ轍を踏まない提案が得られる

02

## 顧客理解 × AI

→ 本質的な要件定義

「この業界の顧客は実はこう考える」という暗黙知をAIに伝え、表面的でない要求を引き出せる

03

## 技術の勘所 × AI

→ 効率的なレビュー

「ここがハマリポイント」という勘所をAIに伝え、重点的に検証すべき箇所を教えてくれる

 経験は「AIへの指示の質」と「出力を見抜く目」を決める。



SECTION 03

---

# AI時代を生き抜く

# AI時代の開発スタイルの変化

## 従来の開発スタイル

- サイクル：直列  
(要件定義→設計→実装→テスト)
- 品質観：「完成品」を一発で出す
- 失敗コスト：高い（手戻りが痛い）
- 人間の役割：全工程を担う

設計書を作成する  
コーディングをする

## AI時代の開発スタイル

- サイクル：反復  
(仮説→プロト→検証→修正)
- 品質観：「作り直し前提」で素早く出す
- 失敗コスト：低い（AIが再生成）
- 人間の役割：「なぜ」を問う・判断・責任に集中

プロンプトで指示する

💡 プロンプトを通じて「問い」を立てながら指示をすることが中心になってくる



35年間、コードを書き続けてきた私も、  
この1年間...

**1行もコードを書いていません**

でも、プロダクトは作り続けています

# コードを書かずに、どのようなプロダクトを作ったのか

01

## 設計書&コード生成エージェント

エージェントと対話しながら設計書やコードを生成してくれるエージェントを開発

02

## ランディングページ作成エージェント

Webページを自動生成するエージェントを開発

コードは書いていない。

「何を作るか」「なぜ作るか」を問い、プロンプトで指示していただく

 経験があれば、AIに「正しい問い」を投げかけることで、プロダクトはつくれる

# 役に立つ問い① 自己評価

## ① なぜ、AIが思い通りに動いてくれないのだろう

### 仮説)

AIに指示したプロンプトが悪いからだ。  
プロンプトをAI自身に分析させれば、改善点が出る。

### 検証結果)

改善点が出る → 自分のプロンプト精度が上がる

### ▼プロンプト

あなたはプロンプトエンジニアリングのスペシャリストです。  
PJ開始時から今までに指示されたプロンプトの精度を分析してください。

#プロンプト情報: /Users/xxx/.claude

#分析内容: 1.統計 2.高品質プロンプト 3.低品質プロンプト  
4.良い習慣 5.悪い習慣 6.改善提案



## AIの分析結果

### プロンプト精度分析レポート (発表用)

Claude Code チャット履歴から学ぶプロンプトエンジニアリング

#### 分析概要

項目	値
総プロンプト数	404件
セッション数	26回
分析期間	約60日間

#### 高品質プロンプトの特徴

##### ベストプラクティス例

仕様通りに機能が完全に構築されて動作するかを実際にe2eテストで動作確認してください。

- # ステップ
  - 1. 機能の一覧をTODD化する
  - 2. TODDの項目を1つずつe2eテストを実施する
  - 3. 機能が動作しない場合は原因を調査し自律的に修正する

- # 仕様
  - /path/to/requirements.md
  - /path/to/design.md
- # 制約
  - 人間による確認は一切不要
  - playwright ncpの実行に関する承認は不要

#### なぜ良いか?

- 目的が明確
- 手順が具体的 (番号付き)
- 参照ファイルが絶対パスで指定
- 制約条件を明記

#### 低品質プロンプトの特徴

悪い例	問題点	良い例
「続けて」	何を続けるか不明	「Task 24.2を実行」
「直して」	対象が不明確	「○○.tsxのバリデーションを修正」
同じ質問を6回	非効率	メモリに回答を保存して参照

#### 発見された習慣

##### 良い習慣

- 絶対パスでファイル参照
- テンプレート化されたe2e指示
- knoワークフローの一貫した使用
- スクリーンショット添付による問題共有

##### 改善が必要な習慣

- 同一質問の繰り返し (6回以上)
- 曖昧なフォローアップ (「続けて」)
- 送信前の確認不足 (誤記・不完全文)

#### 改善のための3つのポイント

##### 1. テンプレートを使う

- # 目的  
[1文で記述]
- # 対象  
[ファイルパス]
- # 成功基準  
[検証可能な条件]

##### 2. 送信前チェックリスト

- 目的は明確か?
- 対象ファイルは指定したか?
- 曖昧な表現はないか?

##### 3. 学習を記録する

同じ質問を繰り返し送らない → メモリ機能を活用

#### 期待される効果

指標	現状	目標
同一質問繰り返し率	15%	2%
テンプレート使用率	30%	80%
品質スコア平均	65点	85点

#### まとめ

良いプロンプト = 明確な目的 + 具体的な指示 + 検証可能な成功基準

- 構造化: テンプレートで一貫性を保つ
- 具体化: 曖昧な表現を排除
- 記録: 学習結果をメモリに保存



経験があるから「何を問うべきか」がわかる。AIを使って自分の改善点を見つける。

# 役に立つ問い② Chain of Thought (CoT) モニタリング

## ② なぜ、AIはこの回答をしたのだろう

### 仮説)

AIの判断理由を聞けば、こちらの意図との差異がわかるはず。「なぜ」と問うことで、AIの判断基準が見える。

### 検証結果)

指示の曖昧さや解釈のズレが明確になる → 対話で解消

### ▼ プロンプト例

今の回答について、**結論→根拠→判断手順**の順で、説明してください。  
推測した点があれば『推測』と明記してください。

※ 事前に「このファイルは何の形式？」と質問  
AIは「プロンプト（指示書）です」と回答  
→ なぜそう判断した？ と深掘り

## AIの回答例

### 【結論】

このファイルは、AIコード生成エージェント向けのプロンプト（指示書）です。

### 【根拠】

1. ファイル冒頭に「#コード生成タスク設計書」と明記
2. AIエージェントの行動を制御する詳細な指示が含まれている

### 【判断手順】

- ① ファイル冒頭のタイトルを確認
- ② 内容構造（セクション、指示文）を分析
- ③ 想定読者（AI）を特定

### → ここで気づく！

AIは「タイトル」を最重視していた。  
私が想定した「ファイル形式」（YAML/JSON等）とは違う観点で回答していた！



 AIに「なぜ？」と問うことで、認識のズレが見える。経験者は適切な問いを立てられる。

# 役に立つ問い③ 自律的にAIを動かす Harness手法

## ③ なぜ、AIは自律的に動き続けられないのだろう

### 仮説)

AIは指示の中に沢山の作業があると「どこまで進んだか」を忘れてしまう。人間と同じように細かくタスク化させ、タスク管理させながら動かすようにすれば、指示を完遂する。

### 検証結果)

AIが自分で進捗を管理し、自律的にタスクを消化しながら長時間動くようになった

### ▼ プロンプト例

あなたはアプリケーションスペシャリストです。  
制約に則って、ペット管理サービスを作成してください。

#### # 制約

- ① 指示を **タスク分解** する
- ② タスクを **1つずつ確実に** 実行する
- ③ 完了後、**progress.md** を更新する



### 【タスク一覧】

- [X] 1. ペット登録機能
- [X] 2. ペット一覧画面
- [X] 3. 餌やりスケジュール機能
- [ ] 4. 健康記録機能
- [ ] 5. 通知機能

### 【特殊事項】

1. タスク1のペット登録機能は  
犬猫のみを想定して実装しました。

### → ポイント

これは私たちが仕事で当たり前に行っていること。  
「引き継ぎ資料を書く」「進捗を報告する」  
経験者なら、この仕組みの必要性がわかる。

 経験者が当たり前に行っている「タスク管理」を、AIにもさせる。それが自律の鍵。

# 経験を活かしてAIに「任せる」ステップ

## 01

### 過去の経験を言語化する

→ AIへの指示の質を決める

- 過去に解決した課題・知見
- 失敗から学んだ教訓
- 品質を上げるコツ
- 顧客・業界特有の暗黙知

## 02

### 小さく試して検証

→ 経験で出力を評価

- 過去に解決済みの課題で試す
- AIの得意・不得意を把握
- 指示の出し方を改善
- 失敗パターンを学ぶ

## 03

### 実務で本格活用

→ 判断と責任は自分が持つ

- AIの出力を経験で補正
- 品質基準は自分が持つ
- 若手にAI活用のコツを伝授
- 変化を恐れず学び続ける

 経験は「言語化」して初めてAIに伝わる。小さく始めて、大きく活かす。

# 経験者が磨くべき4つのスキル

## プロンプトエンジニアリング

経験から得た「正解のイメージ」をAIに伝える技術。  
曖昧さを排除した指示。

## AIアウトプットの検証力

AIの動作を理解し、出力を経験に照らして検証。  
「本当にそうか？」と問い続ける力。

## 抽象化・言語化能力

複雑な要件を構造化し、AIにも人間にも伝わる形で  
表現する力。

## ドメイン知識の深化

業界・業務の深い理解。  
AIには代替できない「現場感」こそ経験者の強み。

 AIが進化しても、この4つは人間の仕事として残り続ける。

娘に伝えたいこと

AIを使いこなせば、どこまでも行ける。  
だから——

「お父さんは大丈夫。」



## 生涯エンジニア宣言

年齢を言い訳にしない。  
経験を武器に、前提を疑い、新たな経験を積み上げる。  
AIと共に、より高みへ。

# Today's Takeaway

- 1 AIは「どのように」を担う。人間は「なぜ」を問い、判断し、責任を取る。
- 2 経験の量と質が、AIへの指示の質と出力を見抜く目を決める。
- 3 「問い」を立て、経験を言語化し、検証し、磨き続けることで、AI時代を生き抜ける。

# NECグループで、 エンジニアとして 次のキャリアを。

NECソリューションイノベータは、  
1万人超の国内トップクラスの  
エンジニアが在籍する、  
NECグループのITサービス事業を  
牽引する中核会社です。



創造しよう、  
想像を超えた未来を。

エンジニアキャリア採用Webサイト

<https://www.nec-solutioninnovators.co.jp/career/index.html>



# いつかを、 いまに、 変えていく。

すべては、いつか実現したい未来を描くことから始まる。

NECソリューションイノベータは、  
お客様価値を最大化するシステムインテグレータとして、  
まだ見ぬサービスを生み出すバリュー・プロバイダとして、  
めざす未来をともに描き、創っていきたい。

いつかを、いまに、変えていく。

私たちは常に挑戦していきます。



**NEC**

\Orchestrating a brighter world