

AIの開発チームは何人がいいか

役割の分け方と、実際に動かして確かめる役の実測

How Many Agents Does a Code-Generation Team Need? Measuring Role Decomposition and an Execution-Grounded Checker in Production

湯村 信司 (Shinji Yumura) AQUA合同会社 2026年10月11日

88.5 → 68.8
%

5人を3人に減らしたときの、
審査役の合格 ($p = 0.001$)

-22%

3人にしたときの料金。最初の
版も 25秒 → 17秒 と早くなっ
た

+39% / ±0

動かして確かめる役を足した
ときの料金と、出来の差
(77.1% と 78.1%)

ABSTRACT

YUA's Dev Mode builds small React apps with a team of five LLM agents: a planner, three specialists split by file type (styles, types, components) written in sequence, and a reviewer that scores the code and requests revisions. Suspecting this was more ceremony than a 2–4-file app needs, we compared it against two alternatives on a 24-task benchmark (96 generations per condition, conditions run concurrently, Gemini 3.5 Flash-Lite). A three-agent team—planner with explicit acceptance criteria, a single builder, and a checker that runs the app in the user's browser and verifies the criteria against the rendered page—was 22% cheaper and produced its first version 8 seconds sooner, but its pass rate from an LLM judge fell from 88.5% to 68.8% ($p = 0.001$). Adding the same execution-grounded checker to the five-agent team left quality unchanged (77.1% vs. 78.1%) while raising cost by 39% and time-to-final by 13 seconds. Two-thirds of the criteria the checker flagged were visual properties it could not observe from the page text and element counts it received. The same five-agent system scored 88.5% and 77.1% in two sessions, so only concurrent comparisons were interpretable. Our earlier conclusion that the reviewer's scores were uninformative, drawn on an older model, did not hold here.

要旨

YUA の開発モードは、5人のAIのチームで小さな React のアプリを作っている。計画する役、ファイルの種類（見た目・型・画面の部品）で分けて順に書く3人、そしてコードを読んで点数をつけ、直しを頼む見直し役である。2〜4ファイルのアプリには大げさだと考え、ほかの2つの形と比べた。比べ方は、24題で各条件96回、条件を同じ時刻に動かし、作るAIは Gemini 3.5 Flash-Lite とした。

1つ目の形は3人のチームである。計画する役が「できあがりの条件」を決め、書く役が1人で全部を書き、確かめる役が利用者のブラウザでアプリを実際に動かして、画面と条件を見比べる。この形は料金が22%安く、最初の版も8秒早く出た。しかし、審査役の合格は 88.5% から 68.8% に下がった ($p = 0.001$)。

2つ目の形は、同じ確かめる役を5人のチームに足したものである。出来は変わらず (77.1% と 78.1%)、料金は39%、仕上がるまでの時間は13秒増えた。確かめる役が「満たしていない」とした条件の3分の2は、渡された画面の文字と部品の数からは確かめようのない見た目の条件だった。また、まったく同じ5人のチームが、測った日によって 88.5% と 77.1% になった。同じ時刻に比べた結果だけが読めるということである。前に、古いモデルのデータから「見直し役の点数は出来と関係ない」と述べたが、今回はそうならなかった。

1 はじめに

複数のAIに人の会社のような役割を持たせて、ソフトウェアを作らせる試みが広がっている。分担すれば一人ひとりの仕事は小さくなる。一方で、呼ぶ回数・待ち時間・つなぎ目の食い違いは増える。人数を増やすことがどれだけ得なのかは、作るものの大きさと、使うモデルに左右されるはずである。

前回の報告（002）では、開発モードの画面の側を改善した。そこで作られるアプリのファイルの数は、ほとんどが2〜3個だった。それなのに、5人で分担している。筆者は「5人は多すぎる」と考え、役割を「ファイルの種類」ではなく「仕事の段階」で分けた3人の形を作った。さらに、読んで点数をつける見直し役の代わりに、実際に動かして確かめる役を置いた。本稿は、この予想が外れた実験の報告である。

2 比べた3つのチーム

形	役割の流れ	確かめ方
5人 (本番)	計画 (Yua) → 見た目 (Noa) → 型と道具 (Toa) → 画面の部品 (Kaito) を順に → 見直し (Ren)。Ren は点数が基準に届かなければ直しを頼む	読んで点数 (Ren)
3人	計画とできあがりの条件 (Yua) → 全部のファイルを1人で (Noa) → 確かめる (Toa)	実際に動かして、条件を確かめる
5人+ 確認 役	5人の流れに、計画のできあがりの条件と、確かめる役を足す	読んで点数 (Ren) + 実際に動かして確かめる

表1 比べた形。どの形も、作るAIは同じ (gemini-3.5-flash-lite。書く・計画・見直しは考える深さ medium)。

確かめる役は、次の流れで動く。

1. アプリができあがると、利用者のブラウザの見えない枠の中で動かす。
2. エラーが出たら、そのエラーを直すよう頼む。
3. エラーがなければ、画面の様子をAIに渡し、計画が決めた条件 (3〜8個) を1つずつ「満たしている／いない」で答えさせる。画面の様子とは、表示された文字、ボタンの文字、入力の子品の種類と数、図 (canvas・svg) の数、表の行やリストの項目の数である。
4. 満たしていない条件があれば、直しを頼む。直しの依頼は、利用者が自分で送るまでに合わせて2回までとした。

3 測り方

- **お題**：前回までと同じ24題 (電卓・グラフ・3D・ゲームなど)。それぞれ2回ずつ頼み、これを2回くり返した (各条件96回)。
- **同時に測る**：比べる2つの形を別々の場所で立ち上げ、同じ時刻に測った。実験1 (5人と3人) と実験2 (5人と5人+確認役) は、別の時間帯に行った。
- **記録**：次の5つを記録した。
 - 動いたかどうか (前回までと同じ4つの判定)
 - AIを呼んだ回数と、役ごとのトークン (入力・出力・考えた分)。Gemini が返す使用量をそのまま数えた

- かかった時間：最初の版まで、自動の直しまで終わって仕上がるまで
 - 確かめる役の結果
 - 本物のグラフ部品と日本語の画面（コードを機械的に数えた）
- **審査役**：gemini-3.8-flash に、最後の版の画面の写真とコードを見せ、0～2点で3回採点させた。その真ん中の点を使った（前回と同じ）。
 - **料金**：3.5 Flash-Lite の価格（入力 \$0.30、出力と考えた分 \$2.50、いずれも100万トークンあたり）で計算した。

4 実験1：5人と3人

指標	5人	3人
審査役の合格（2点）	85/96（88.5%）	66/96（68.8%）
1回目・2回目の合格	44・41	35・31
動いた	96/96	94/96
AIを呼んだ回数（1題あたり）	6.38	5.24
入力トークン（1題あたり）	15,081	6,314
出力・考えた分のトークン	4,340・3,117	4,407・2,071
料金（100題あたり）	\$2.32	\$1.81
最初の版まで（中央値）	25秒	17秒
仕上がるまで（中央値）	40秒	41秒
本物のグラフ部品・日本語の画面	20/20・86/89	19/20・88/90

表2 実験1。合格の差は $p = 0.0013$ （フィッシャーの正確確率検定）。3人では、確かめる役が96回中20回で満たしていない条件を見つけた（571条件中79）。

3人の形は、料金・入力トークン・最初の版までの時間で優れていた。入力トークンが半分以下になったのは、主に、5人の形では各役に計画と先に書かれたファイルを毎回渡しているためと考えられる。仕上がるまでの時間は同じだった。3人の形は、確かめて直す回が加わった分だけ追いついたことになる。

しかし、出来ははっきり5人が上だった。差は、確かめる役の直しから来たものではない。3人の形の合格は、直しが入らなかった回で 47/67（70%）、入った回で 19/28（68%）とほぼ同じだった。差は、1人で全部を書いたときの最初の出来から来ている。5人の形では、計画にもとづいて見た目と型を先に書き、それを見ながら画面の部品を書く。さらに見直し役が、およそ3回に1回、直しを頼んでいた（見直し役の呼び出しは1題あたり1.29回）。

5 実験2：確かめる役を足す

実験1では、分担と見直しの効き目を、確かめる役の効き目と切り分けられなかった。そこで、5人の形はそのままにして、確かめる役だけを足した形と比べた。

指標	5人	5人+確認役
審査役の合格（2点）	74/96（77.1%）	75/96（78.1%）
1回目・2回目の合格	34・40	37・38
動いた	96/96	96/96
AIを呼んだ回数（1題あたり）	6.38	9.55
料金（100題あたり）	\$2.33	\$3.22（+39%）
仕上がるまで（中央値）	44秒	57秒
確かめる役が直しを頼んだ回	—	30/96
確かめて「満たしていない」とした条件	—	651条件中102

表3 実験2。合格の差は $p = 1.0$ 。最後に「すべて満たしている」と確かめられたのは96回中76回、満たしていない条件が残ったのは12回。

確かめる役は96回中30回で直しを頼み、そのぶん呼び出し・料金・時間が増えた。それでも、審査役の合格は変わらなかった。

確かめる役が「満たしていない」とした102条件を、言葉で簡単に分けると、69は見た目の条件だった。「時計の文字盤が表示されている」「1オクターブの鍵盤が並んでいる」「12か月分の棒がある」などである。残りは、操作の条件が23、その他が10だった。確かめる役に渡しているのは画面の文字と部品の数だけで、絵の形は見えない。指示では「確かめられない条件は満たしているとする」と決めていたが、守られないことが多かった。多くの直しは、確かめようのないものを根拠に頼まれた可能性が高い。

6 わかったことと落とし穴

L1 役割を減らすと、安く速くなるが、出来は下がりうる

「分担しすぎ」という直感は、少なくとも今回のモデルとアプリの大きさでは外れた。順に組み立てて見直す流れには、呼ぶ回数とトークン以上の値打ちがあった。人数を決めるときは、料金・速さ・出来を同じ時刻に並べて測る必要がある。

L2 「動かして確かめる」は、見えるものしか確かめられない

実際に動かした結果は、読んで点数をつけるより確かな手がかりのはずだった。しかし、渡した情報（文字と部品の数）の外にある条件まで判定させると、確かめる役は誤った「満たしていない」を出し、不要な直しが増える。画面の写真を見せる、あるいは確かめる条件を文字と部品で確かめられるものに絞る、といった工夫が要る。

L3 古いモデルでの結論を持ち越さない

前のモデル（3.1 Flash-Lite）のデータでは、見直し役のつける点数は、審査役の評価とほとんど関係がなかった。筆者はそこから「見直し役は役に立っていない」と考えたが、今回のモデルでは見直しを含む5人の形のほうが良かった。モデルを替えたら、役割の値打ちも測り直す必要がある。

L4 同じシステムでも、測る日によって11ポイント動く

まったく同じ5人の形が、実験1では 88.5%、実験2では 77.1% だった（ $p = 0.055$ ）。これは前回の報告（002）で見た揺れと同じ種類のものである。別の日に測った数字どうしを比べると、ない差を作ってしまう。本稿の比較は、すべて同じ時刻に測ったものどうしで行った。

L5 作る途中で見つけた不具合

3人の形を作る途中で、次の不具合を見つけて直した。

- 直すとき（前のファイルがすでにある）に、書く役が書き終わる前に「完了」とみなしていた。中身の有無ではなく、書き終わったという知らせを待つようにした。
- 画面の様子を送るスクリプトの改行の書き方を誤り、枠の中の見張りが丸ごと止まった。枠に入るスクリプトが正しい JavaScript かを確かめるテストを足した。

- 色を選ぶ部品を、確かめる役が見落とした。入力の子品の種類も渡すようにした。

どれも、実際に動かして初めてわかったものである。

7 限界

- アプリは2〜4ファイルの小さなものだけ。大きなアプリや、既存のリポジトリを直す仕事では、結果が変わりうる。
- 作るAIは1種類（3.5 Flash-Lite）で、審査役も同じ系統（Gemini）。人による評価はない。
- 3人の形と5人の形は、役割の数と分け方を同時に変えている。人数の効き目と分け方の効き目は切り分けていない。
- 確かめる役は画面の写真を見ていない。写真を見せた場合の効き目は、今後の課題である。
- 審査役は最後の版だけを見る。確かめる役が直した前後の差は、審査では測っていない。

8 関連研究

役割を分けたAIのチームでソフトウェアを作る試みとして、MetaGPT [1] や ChatDev [2] がある。これらは人の会社の分業をなぞり、その有効性を示した。本稿は、同じ形を本番の小さなアプリ作りに当てはめ、人数を減らしたときの得失を料金・時間・出来で並べて測った。

実行の結果を使って直す試みには Self-Debugging [3] や Reflexion [4] がある。本稿の確かめる役もこの流れにあるが、手がかりが見えるものに限られると、誤った直しの依頼が増えることを示した。AIによる採点の揺れについては、前回の報告 [5] を参照されたい。

9 まとめ

2〜4ファイルのアプリを作るAIのチームでは、5人を3人に減らすと料金が22%下がり、最初の版が8秒早くなったが、審査の合格は約20ポイント下がった。実際に動かして確かめる役を5人に足しても、出来は変わらず、料金が39%増えた。出来に効いていたのは、確かめ方よりも、順に組み立てて見直す流れだった。

YUAの本番は5人の形のままとした。今後は、確かめる役に画面の写真を見せる形を試す。人数や役割の値打ちは、モデルを替えるたびに、同じ時刻に比べて測り直す。

参考文献

1. S. Hong, M. Zhuge, J. Chen, et al. “MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework.” *ICLR*, 2024. arXiv:2308.00352.
2. C. Qian, W. Liu, H. Liu, et al. “ChatDev: Communicative Agents for Software Development.” *ACL*, 2024. arXiv:2307.07924.
3. X. Chen, M. Lin, N. Schärli, D. Zhou. “Teaching Large Language Models to Self-Debug.” *ICLR*, 2024. arXiv:2304.05128.
4. N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, S. Yao. “Reflexion: Language Agents with Verbal Reinforcement Learning.” *NeurIPS*, 2023. arXiv:2303.11366.
5. 湯村 信司. 「AIが作ったアプリを、AIの審査で正しく測れるか」. AQUA Research Technical Report 002, 2026.