AQUA RESEARCH · TECHNICAL REPORT 002

AIが作ったアプリを、AIの審査で正しく測れるか

コードを書くエージェントの改善と、評価の揺れ・落とし穴

Can an LLM Judge Tell Whether Generated Apps Got Better? Improving a Production Code-Generation Agent, and the Noise and Pitfalls in Measuring It

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

40 → 100%

グラフを頼まれたとき、本物のグラフ部品を使った割合（実行環境の説明を正しただけで）

45 → 94%

画面の文字が日本語だった割合（日本語の頼みに対して）

14.6〜31.3%

まったく同じ条件で3回測った、AIの審査役の合格率。この揺れが、改善の差を覆い隠した

ABSTRACT

We improved the “Dev Mode” of YUA, a Japanese chat assistant in production that plans, writes, and reviews multi-file React apps with a team of LLM agents, and tried to measure the effect with the 24-task benchmark of our previous report. Changes included sandbox hardening, a live in-chat preview, an in-frame error watcher with bounded self-repair, loading real npm libraries through an import map, correcting the agents’ description of the runtime (it had claimed libraries were mere stubs), and a short set of quality rules. Code-level measures moved sharply: the share of chart tasks implemented with a real charting library rose from 40% to 100% from the corrected runtime description alone ($p < 10^{-5}$), and the share of apps whose UI text was Japanese rose from 45% to 94% with the quality rules ($p < 10^{-9}$). The share of apps that rendered and survived a first interaction rose from 91.7% to 97.9% ($p = 0.05$). Yet an LLM judge’s pass rate did not separate the conditions: three identical runs of the unchanged system scored 14.6%, 16.7%, and 31.3%, a spread larger than any difference between conditions. We also report six pitfalls on the measuring

side, including a blank-screen detector that mistook box-only drawings for empty pages—an error that invalidated the only self-repair event claimed in our previous report.

要旨

筆者らが運営する日本語のチャットAI「YUA」には、複数のAIが計画・分担・見直しをして、ファイルが複数あるReactのアプリを作る「開発モード」がある。本稿では、その開発モードを改善し、前回の報告と同じ24題で効果を測った。主な変更は次のとおりである。プレビューの枠の守りを固める。会話の中でアプリをそのまま動かす。エラーを見張り、回数を限って自動で直す。npm の部品を本物で読み込む。AIに伝えていた「実行環境の説明」を正す（部品は見せかけだと書いていた）。出来の良さの決まりを短く足す。その結果、コードには大きな変化が出た。グラフを本物の部品で作った割合は、説明を正しただけで 40% から 100% に上がった（$p < 10^{-5}$）。画面の文字が日本語だった割合は、決まりを足すと 45% から 94% に上がった（$p < 10^{-9}$）。動作成功も 91.7% から 97.9% に上がった（$p = 0.05$）。ところが、AIの審査役の合格率では条件の違いを見分けられなかった。何も変えていないシステムを同じ条件で3回測っただけで、合格率は 14.6%・16.7%・31.3% と揺れた。これは条件のあいだのどの差よりも大きい。あわせて、測る側で見つかった6つの落とし穴を報告する。その一つは、色の箱だけで描かれた画面を「真っ白」と取り違える見張りの誤りである。これにより、前回の報告で唯一とした自動修復の例が誤りだったとわかった。

1 はじめに

前回の報告（Technical Report 001）では、チャットの中で動く小さな画面（アーティファクト）を壊れにくくした。今回はその続きとして、ファイルが複数あるアプリを作る「開発モード」に同じ考えを広げた。あわせて、出来の良さそのものを上げる工夫も試した。

出来の良さを測る手軽な方法は、別のAIに画面とコードを見せて採点させること（LLM-as-a-Judge）である。しかし今回、コードを直接数えれば明らかな変化が、AIの審査では見えなかった。本稿の中心は、この食い違いである。貢献は次の3つである。

1. 本番のコード生成エージェントに加えた変更と、それぞれの効き目を切り分けた測定（変更前・画面と説明だけ・決まりも足す、の3条件を同時に測定）
2. 同じ条件をくり返し測ったときの、AIの審査役の合格率の揺れの実測
3. 測る側で見つかった6つの落とし穴。前回の報告の誤りの訂正を含む

2 対象のシステム

開発モードでは、まず「計画する役」が、作るファイルと役割を決める。次に「書く役」3人（見た目・型と道具・画面の部品）が分担してファイルを書き、最後に「見直す役」が点数をつけて直しを求める。文章を作るのは `gemini-3.1-flash-lite`、1回の答えはおよそ15〜30秒である。

できたアプリは、利用者のブラウザの中で動く。ブラウザの中で Babel が TypeScript と JSX を変換し、npm の部品は esm.sh から読み込む。このプレビューの仕組みは利用者が手元で試すためのもので、GitHub への反映（PR）は別の機能である。

3 加えた変更

変更は2つの組に分けた。**B** は画面の側と、AIへの「実行環境の説明」を正したもの。**C** は B に「出来の良さの決まり」と「まず作る」を足したものである。

3.1 守り（B）

プレビューの枠には `allow-scripts allow-same-origin` が付いていた。この組み合わせでは、作られたコードが親の画面（YUA の保存データやログインの状態）に触れられる。`allow-same-origin` を外し、枠からの知らせも送り元の枠を確かめてから受け取るようにした。

3.2 会話の中で動かす（B）

以前は「▶ プレビュー」を押さないと、できたアプリは見えなかった。最新の結果を会話の中でそのまま動かし、下の帯に次の操作を置いた。

- 版（版 2/3 など）
- 全ファイルのコピー
- ZIP で保存
- そのまま開ける1つの HTML で保存
- 全画面

以前からあった会話の中の小さなプレビューの部品は、使われていなかった。しかも枠の外から中身を書き込む作りで、sandbox の中では動かなかった。これは取り除いた。

3.3 見張りと自動修復（B）

枠の中に見張りの小さなスクリプトを入れた。拾うのは次の5つである。

- 例外
- 処理されなかった Promise の失敗
- 部品の読み込みの失敗
- 描画の3秒後になにも見えない「真っ白」
- 止められるダイアログ（画面の中のお知らせに置き換える）

React は開発用の版に替えた。製品用の版のエラーは「Minified React error #130」のような番号だけで、AI が直す手がかりにならないためである。

生成が終わり、エラーがあれば、System から「次のエラーが出ました。直してください」という依頼を自動で送る。決まりは次の3つである。

- 回数は、利用者が自分で送るまでに2回まで
- 昔の会話を開いただけのときは直さない
- この依頼は「会話か相談か」の判定を通さず、必ずコードを直す

3.4 本物の部品を読み込む（B）

esm.sh の部品（recharts など）は、中で `import 'react'` をしている。ところがプレビューには読み込み先の表（importmap）がなく、「react が見つからない」で読み込みに失敗していた。そのたびに、名前だけの見せかけの部品（スタブ）に入れ替わっていた。先に読み込んだ React をそのまま渡す小さな橋渡しを表に入れ、recharts・framer-motion・lucide-react が本物で動くことを確かめた。

3.5 実行環境の説明を正す（B）

AIへの指示には「提供ライブラリはスタブ版のみ。高度な機能は動作しない前提で実装」と書かれていた。これは、3.4 の不具合で実際にそうなっていた時期の名残と考えられる。この説明を、次の内容に改めた。

- 本物が動く部品と、その使いどころを書く
- グラフや3Dは div で自作しない
- alert は使わない

3.6 出来の良さの決まりと「まず作る」（C）

変更前の答えに対する審査役の指摘から、多かったものを7行の決まりにして、計画する役と書く役の指示に足した。

- 画面の文字は、頼んだ言葉で書く
- 頼まれた数や項目を省かない

- 頼まれた操作は、目に見える部品で用意する
- 開いた直後から中身が見える
- 指でも操作できる
- ゲームには、やり直すボタンを置く
- 細かい指定は、頼みの言葉どおりに数える

あわせて、「会話か・相談か・作るか」を決める判定の指示も変えた。作るものがはっきりした「〜を作って」は、細部が足りなくても、無難な形でまず作る。

---

4 評価の方法

お題は前回と同じ24題で、それぞれ2回ずつ頼んだ（1回の測定で48回）。A・B・C の3つの版を別々のフォルダで立ち上げ、**同じ時刻に**測った。これを2回くり返した。A はこのほかに、変更を始める前にも1回測っている（合わせて3回）。

2回目は、1回目で見つかった B・C の不具合を直してから測った。直したのは「真っ白」の誤判定（6章 P2）と、自動で直す依頼が判定を通っていたことである。「まず作る」の判定の変更は、2回目の C から入っている。

判定は次の3種類で行った。

- **自動の判定**：前回と同じ4つ（画面が出た・エラーなし・真っ白でない・最初のボタンを押しても壊れない）
- **審査役**：`gemini-3.8-flash` に、頼みごと・画面の写真・全ファイルのコードを見せて、0〜2点で採点させた。3回採点させて、真ん中の点を使う（前回と同じ）
- **コードの上の指標**（今回足した）。どちらも、できたファイルを機械的に数える
  - グラフのお題5つで、recharts・chart.js を import しているか
  - 画面の文字（JSX の中の文字）の過半が日本語か

5 結果

指標	A：変更前	B：画面と説明	C：B＋決まり
グラフで本物の部品を使った	12/30（40%）	**20/20（100%）**	**20/20（100%）**
画面の文字が日本語	49/109（45%）	36/72（50%）	**76/81（94%）**
動作成功（自動の判定4つすべて）	132/144（91.7%）	86/96（89.6%）	**94/96（97.9%）**
カフェの紹介ページを作った	0/6	0/4	2/4（「まず作る」のあとは 2/2）
審査役の合格（2点）	**30/144（20.8%）**	**21/96（21.9%）**	**28/96（29.2%）**
答えまでの時間（中央値）	14〜22秒	17〜24秒	17〜26秒

表1　すべての回を合わせた結果。p値はフィッシャーの正確確率検定（両側）。主な比較は次のとおり。グラフの部品は A と B で $p = 8.4\times10^{-6}$、日本語の画面は B と C で $p = 7.8\times10^{-10}$、動作成功は A と C で $p = 0.05$、審査の合格は A と C で $p = 0.17$。日本語の画面の分母は、画面の文字が見つかった答えだけ。

コードの上の指標は、変更の狙いどおりに大きく動いた。実行環境の説明を正しただけ（B）で、グラフを本物の部品で作る割合は 40% から 100% になった。A では、残りの18件のうち14件が div を並べ、4件が SVG を手で描いていた。日本語の画面は、決まりを足した C で 94% になった。B では 50% のままだった。動作成功は C で 97.9% になり、2回目の C は48回すべてが成功した。

一方、審査役の合格は 20.8%・21.9%・29.2% で、統計的にはっきりした差にならなかった。

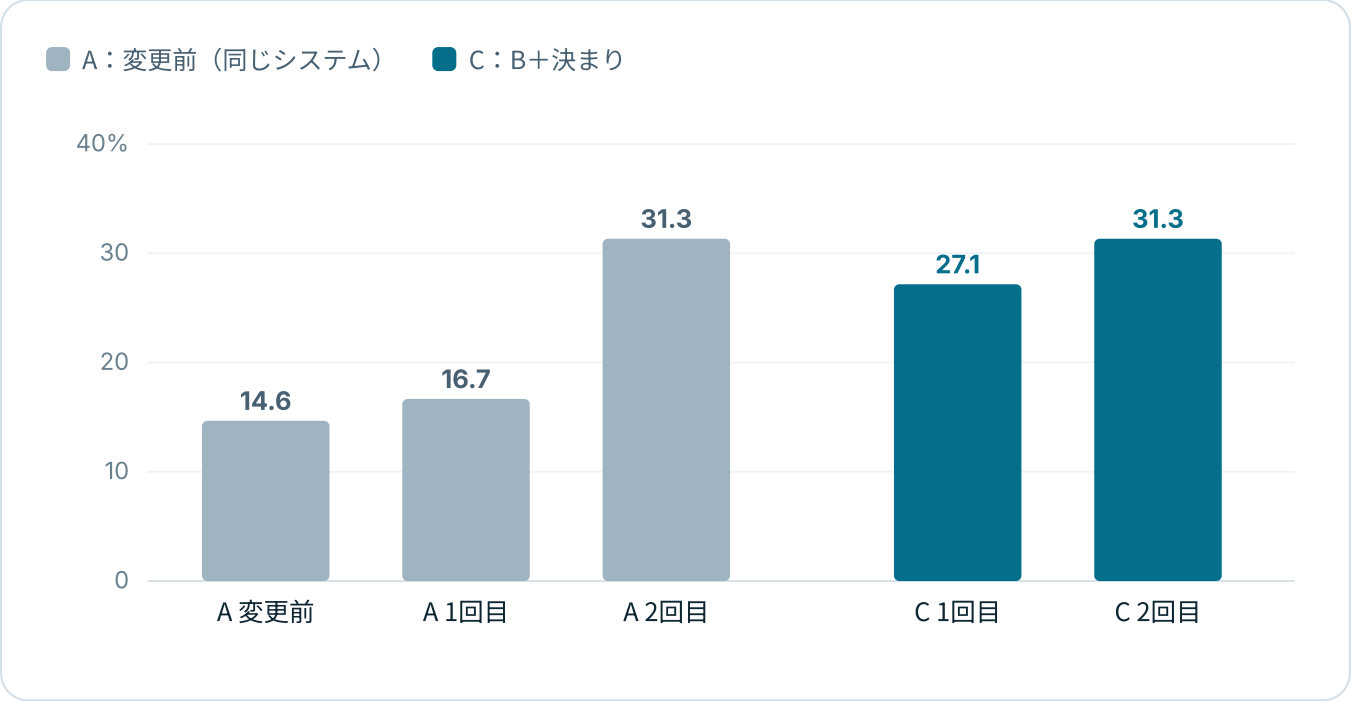


図1　審査役の合格率（各48回）。A は何も変えていないのに、回によって 14.6% から 31.3% まで動いた。同じ時刻に測った A と C の差は、1回目が 10.4 ポイント、2回目が 0 ポイントだった。

回	A	B	C	審査役の3回の採点が一致した割合
変更前（A のみ）	7/48	—	—	35/46
1回目	8/48	7/48	13/48	29〜39/46
2回目	15/48	14/48	15/48	27〜30（46〜48件中）

表2　回ごとの審査役の合格数。2回目は、3つの条件がそろって高い。条件の違いより、回（測った時刻）の違いのほうが大きく効いている。審査役が同じ画面に3回とも同じ点をつけたのは、56〜85%だった。

5.1 自動修復

2回目（見張りの誤りを直したあと）に自動修復が働いたのは、B で1件（3Dの立方体。2回直しても描かれず）、C で0件だった。前回のチャットと同じく、自動修復が必要になることはまれである。改善の多くは、壊れない作り（本物の部品、正しい説明、決まり）から来ている。

---

6 測る側の落とし穴

今回は、作る側よりも測る側の問題を多く見つけた。どれも、気づかなければ結論を変えていたものである。

P1 AIに伝えていた実行環境の説明が古かった

「部品はスタブ版のみ」という説明は、部品の読み込みが壊れていたこと（3.4）と一致していた。そのためAIは、正しく判断した結果として、グラフを div で自作していた。読み込みだけ直しても、説明が古いままでは本物の部品は使われない。説明を正すと、使用は 40% から 100% になった。**エージェントが環境について信じていることも、点検の対象である。**

P2 「真っ白」の判定が、色の箱だけの画面を取り違えた（前回の報告の訂正）

見張りは「文字・画像・canvas・ボタンなどがひとつもない画面」を真っ白と判定していた。しかし、ピアノの鍵盤・時計の針・CSS の立方体は、文字のない色の箱だけで描かれることが多い。1回目の B では、文字のない画面4件（時計2件・ピアノ・立方体）に自動修復が8回走り、すべて失敗に終わった。そのうちピアノは、写真で見るかぎり正しく描かれていた。判定は、色・枠・背景のある箱も「見えている」と数えるように直した。

前回の報告（001）で、唯一の自動修復の例として挙げたアナログ時計も、この誤りによるものだった。001 には訂正を入れた。同じ誤りは、本番のチャットにも入っていた（直したものは、開発モードの改善とあわせて反映する）。

P3 測る道具が、返事の日本語を化けさせた

測る道具は当初、流れてくる返事をブラウザの外から読んでいた。返事の形式（text/event-stream）には文字の種類の指定がなく、日本語が化けていた。そのまま進めると、審査役は化けたコードを読んで採点することになる。返事をページの中で、文字として読むように直した。直す前の途中までの測定は捨てた。

P4 流れてくる返事の「届き始め」を「終わり」と取り違えた

ブラウザの「返事が来た」の知らせは、流れてくる返事では「届き始め」に出る。これを「終わり」として扱うと、生成の途中で写真を撮ってしまう。「届き終わった」の知らせを使うように直した。

P5 作らずに聞き返した

「カフェのおしゃれな紹介ページを作って。メニューとアクセスも入れて」は、A でも B でも、10回すべてで「どんな雰囲気がいいですか？」と聞き返された。これは判定の役が「相談が要る」と分けたためで、画面の側の改善では防げない。判定の指示を変えると（3.6）、毎回作るようになった。

P6 審査役の合格率は、同じ条件でも大きく揺れる

何も変えていない A は、3回で 14.6%・16.7%・31.3% だった（図1）。3つの条件を同じ時刻に測っても、回が違うだけで3つそろって十数ポイント動いた（表2）。原因は、作るAIの答えの揺れと、審査役の採点の揺れの両方である。48回という規模では、この揺れが条件のあいだの差より大きい。**1回の測定で「良くなった」「悪くなった」とは言えない。**

> AIによる審査の合格率だけを見ていたら、本稿の結論は「変えても出来は変わらない」になっていた。実際には、狙った性質（本物の部品・日本語の画面・動作）はどれも大きく、はっきりと動いていた。審査役は、細かい仕様の好み（「チェックボックスがない」「単位の書き方」など）を広く拾う。そのぶん、狙った性質の変化が埋もれやすい。

---

7 考察と限界

7.1 審査役の採点は、機械的な指標と組で使う

狙った性質を機械的に数えられる指標（部品を import したか、画面の文字の言葉は何か、動いたか）は、少ない回数でもはっきりした差を出した。審査役の採点は広く見られる反面、揺れが大きい。改善の効果を主張するときは、次の3つを勧める。

- 狙った性質ごとに、機械的な指標を先に決めておく
- 比べる条件を同じ時刻に測り、それを何回かくり返す
- 何も変えていない条件をくり返し測り、揺れの大きさ（再現性）を一緒に報告する

7.2 エージェントの「思い込み」

P1 の説明の誤りは、コードの不具合（3.4）と組になっていた。不具合だけ直しても、エージェントは古い説明を信じて、使えるはずの部品を避け続ける。指示の中の環境の説明は、仕組みを変えるたびに確かめ直す必要がある。

7.3 限界

- **規模**：24題×2回を、A は3回、B・C は2回。審査役の合格率の差は 5% 水準ではっきりしない。
- **1回目と2回目で中身が違う**：1回目の B・C には P2 の誤りがあり、1回目の C には「まず作る」が入っていない。表1はすべての回を合わせているため、C の実力をやや低めに見積もっている。

- **日本語の判定は簡単なもの**：JSX の中の文字の過半が日本語かを数えた。文字のない画面は分母から外した。
- **作るAIと審査役が同じ系統**（どちらも Gemini）で、人による評価はない。
- **コードの上の指標は、狙った性質だけを見る**。全体の出来の良さを表すものではない。

---

8 関連研究

複数のAIで役割を分けてソフトウェアを作る試みには、MetaGPT [1] や ChatDev [2] がある。本稿の開発モードも同じ形だが、ブラウザの中のプレビューで利用者が直接動かす点と、その実行環境の説明がエージェントの判断を左右した点を扱った。

AIによる採点の偏りや、人の評価との一致は [3][4][5] で調べられている。本稿は、同じシステムをくり返し測ったときの採点の揺れ（再現性）を、本番の改善の評価という場面で実測した。実行結果から誤りを直す試み [6] に対しては、自動修復の作動が見張りの誤検知に左右されうることを示した。

---

9 まとめ

コードを書くエージェントの改善を、AIの審査役だけで測ると、実際に起きた大きな変化を見落とすことがある。今回は、実行環境の説明を正すだけで本物のグラフ部品の使用が 40% から 100% になり、短い決まりで日本語の画面が 45% から 94% になった。動作成功も上がった。しかし、審査役の合格率は同じ条件でも十数ポイント揺れ、これらを見分けられなかった。

評価の側にも不具合は入り込む。「真っ白」の誤判定は、前回の報告の結論の一部を誤らせていた。測る道具を、作る道具と同じように点検すること。狙った性質の機械的な指標と、くり返しの測定を組み合わせること。この2つを、今後の改善の標準の手順とする。

A 付録：条件と環境

項目	内容
お題	前回（001 の付録）と同じ24題。各2回
測った回	A：変更前の1回＋同時測定の2回／B・C：同時測定の2回
作るAI	gemini-3.1-flash-lite（計画・書く・見直す、すべて）

項目	内容
審査役	gemini-3.8-flash（3回採点の中央値。1回目は温度0、2・3回目は0.7）
プレビュー	React 18.3.1（開発用の版）、Babel 7.26.4、npm の部品は esm.sh
ブラウザ	Google Chrome（ヘッドレス、1280×900）
測定日	2026年10月11日

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. L. Zheng, W.-L. Chiang, Y. Sheng, et al. “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.” *NeurIPS Datasets and Benchmarks*, 2023. arXiv:2306.05685.
4. C.-H. Chiang, H.-y. Lee. “Can Large Language Models Be an Alternative to Human Evaluations?” *ACL*, 2023. arXiv:2305.01937.
5. P. Wang, L. Li, L. Chen, et al. “Large Language Models are not Fair Evaluators.” *ACL*, 2024. arXiv:2305.17926.
6. X. Chen, M. Lin, N. Schärli, D. Zhou. “Teaching Large Language Models to Self-Debug.” *ICLR*, 2024. arXiv:2304.05128.