

AIのチームは、どこで失敗するか

失敗の分類（MAST）で、本番の開発チームの記録を読む

Where Does an Agent Team Fail? Applying the MAST Taxonomy to a Production Code-Generation Team

湯村 信司（Shinji Yumura） AQUA合同会社 2026年10月12日

この報告が当てはめた論文

M. Cemri ほか「Why Do Multi-Agent LLM Systems Fail?」 arXiv:2503.13657（2025年3月）

AIのチームの失敗の分類（MAST）を、本番の開発チームの記録192回に当てはめ、見つかった失敗の原因をコードまでたどった。

27 / 192

2つのAIの振り分けが、ともに「失敗あり」とした回。ほとんどが「確かめの誤り」だった

23 / 27

そのうち、AIではなく、チームの中のプログラムの確かめ（1つの不具合）が原因だった回

-13%

不具合を直したあとの料金。失敗ありの回は15回 → 3回に減り、出来（審査の合格）は変わらなかった

ABSTRACT

We applied the MAST failure taxonomy of Cemri et al. to 192 execution traces of the five-agent team (planner, three file-type specialists, reviewer) that builds React apps in YUA's Dev Mode, using two LLM annotators (Gemini 3.8 Flash and 3.7 Flash; agreement 98.4%, $\kappa = 0.94$). Both flagged failures in 27 runs, almost all labeled as incorrect verification (FM-3.3) and reasoning-action mismatch (FM-2.6) by the reviewer agent. Tracing the evidence into the code, we found that 23 of the 27 were caused not by the reviewer model but by a rule-based CSS consistency check that overwrote its scores: the check counted compared values such as `filter === 'all'` as undefined class names, forcing correct code to "fail" and the team into needless revision rounds. Failed runs used 2.3× the tokens yet passed an LLM judge as often as the rest (85% vs. 82%). After fixing the check, a

concurrent re-run (96 vs. 96 generations) cut runs with failures from 15 to 3 ($p = 0.005$) and cost by 13%, with no change in judged quality (79.2% vs. 77.1%, $p = 0.86$). Annotator instructions also mattered: without the team's rules, annotators labeled correct behavior as failures, and agreement fell to $\kappa = 0.61$. MAST located the symptom correctly but attributed it to an agent; in production systems, failures of the deterministic glue between agents can masquerade as agent failures.

要旨

AIのチームの失敗を14種類に分けた MAST (Cemri ほか) を、YUA の開発モードで React のアプリを作る5人のチームの記録 192回に当てはめた。振り分けは2つの AI (Gemini 3.8 Flash と 3.7 Flash) が独立に行い、一致は 98.4% ($\kappa = 0.94$) だった。

2つがともに失敗ありとしたのは27回で、ほぼすべてが見直し役の AI の「確かめの誤り」と「考えと行いの食い違い」とされた。ところが根拠をコードまでたどると、27回のうち23回は、AI ではなく、AI の点を上書きするプログラムの確かめが原因だった。この確かめは `filter === 'all'` の `'all'` のような比べている値をクラス名と数え、正しいコードを fail にして、むだな直しをくり返させていた。失敗ありの回はトークンを2.3倍使ったが、審査の合格は変わらなかった (85% と 82%)。

確かめを直して同じ時刻に測り直すと (各96回)、失敗ありの回は 15回から3回に減り ($p = 0.005$)、料金は13%下がり、出来は変わらなかった (79.2% と 77.1%、 $p = 0.86$)。また、振り分けの指示にチームの決まりを書かないと、正しい動きを失敗と数え、2つの振り分けの一致も $\kappa = 0.61$ に下がった。MAST は症状を正しくとらえたが、その原因を AI に帰していた。本番のチームでは、AI の間に入るプログラムの失敗が、AI の失敗に見えることがある。

1 はじめに

AIのチーム (複数の AI に役を分けて仕事をさせる仕組み) は、なぜ失敗するのか。Cemri ほか [1] は、7つの仕組みの実行の記録を人が読み、失敗を3つの分類・14種類にまとめた (MAST)。分類は「仕組みの設計の問題」「AIどうしのすれ違い」「確かめの問題」の3つである。MAST は、AI に記録を読ませて振り分ける方法も示しており、失敗を調べる共通のものさしとして広く使われはじめている。

本稿では、YUA の開発モードで React のアプリを作っている5人のチームの記録 192回を、MAST で振り分けた。そのうえで、見つかった失敗の原因を、コードまでさかのぼって確かめ、直して測り直した。わかっ

たのは、分類がさし示した「AI の確かめの誤り」の多くが、AI ではなく、チームに組み込まれた小さなプログラムの不具合から来ていたことである。

2 材料と振り分け方

- **チーム**：計画（Yua）→ 見た目（Noa）→ 型と道具（Toa）→ 画面の部品（Kaito）→ 見直し（Ren）。Ren の点が75点以上で、しかも「fail」のファイルがなければ終わる。届かなければ直しを頼む（3回まで）。作るAIは Gemini 3.5 Flash-Lite。
- **材料**：前回までの測定（003）の5人のチームの記録 192回（24題 × 2回 × 4回の測定）。頼みごと・計画・役どうしのやりとり・見直しの結果・できたファイル（全文）を、1回ずつの記録にまとめた。
- **振り分け**：MAST の14種類の定義と、チームの決まり（担当するファイルがない役は呼ばれない、終える条件）を渡し、`gemini-3.8-flash` と `gemini-3.7-flash` にそれぞれ独立に振り分けさせた。根拠のあるものだけを、根拠の文とともに挙げさせた。審査役の点は渡していない。

3 振り分けの結果

失敗の種類（MAST）	3.8-flash	3.7-flash
失敗ありの回（192回中）	28	29
FM-3.3 確かめの誤り（問題を見逃す・ないものを指摘する）	27	27
FM-2.6 考えと行いの食い違い	22	20
FM-1.2 役割を守らない	1	1
FM-1.3 終わった手順のくり返し	1	1
そのほかの10種類	0	0

表1 種類ごとの、その失敗が挙げられた回数（1回に複数挙がることもある）。2つの振り分けの「失敗あり／なし」の一致は 98.4%（ $\kappa = 0.94$ ）。

失敗は「確かめの問題」（FC3）と「すれ違い」（FC2）に集まり、「設計の問題」（FC1）はほとんどなかった。MAST の元の論文では、3つの分類の割合は FC1 が 43.8%、FC2 が 32.2%、FC3 が 24.1% で、私たちのチームとは大きく違う。根拠の文を読むと、ほぼすべてが見直し役の Ren に向けられていた。

- FM-3.3：「直っている App.tsx に、未定義の CSS クラスがあると誤って判定している」

- FM-2.6：「まとめでは『前回の指摘は適切に直っている』と書きながら、App.tsx を 40点（fail）のままにして直しを頼んでいる」

失敗ありの回（2つが一致した27回）は、なしの回に比べて、Ren を呼んだ回数が 2.9回（なし 1.0回）、全体の呼び出しが 9.6回（5.8回）、トークンが 43,360（18,877）、時間が 50秒（28秒）だった。一方、審査役の合格は 23/27（85%）で、なしの回の 133/162（82%）と変わらなかった。失敗は、出来を下げるのではなく、むだな直しのくり返しとして料金と時間に表れていた。

4 原因をたどる

「まとめでは直ったと言いながら、40点の fail のまま」という食い違いは、どの回もほぼ同じ形だった。そこでコードを読むと、Ren の答えのあとに、プログラムによる確かめが入っていた。CSS で定義されたクラスと、画面の部品で使われているクラスを突き合わせ、合わないものがあれば、その部品の点を 40点・fail に書きかえる仕組みである。

この確かめは、`className={...}` の中にある文字列を、すべてクラス名として数えていた。そのため、次のような正しいコードの `'all'` を「未定義のクラス」とみなしていた。

```
<button className={filter === 'all' ? 'tab active' : 'tab'}>
```

比べている値（`'all'`、`'work'`、`'neon'`、`'新宿店'` など）は、クラスではない。Ren が何度「直った」と判断しても、プログラムが同じ指摘を足して fail に戻すので、直しは上限の3回までくり返された。

失敗ありの回（2つが一致した27回）の中身	回
プログラムの確かめの誤り（比べている値をクラスと数えた）	23
Ren の確かめの誤り（使っていない <code>substr()</code> を指摘、正しい当たり判定を誤りとする など）	3
画面の部品の担当が、頼まれていない型のファイルを書きかえた（FM-1.2）	1

表2 失敗の原因。最後の版のファイルにこの確かめをかけると、22回で「未定義のクラス」が出たが、比べている値を外すように直すと 0回になった。失敗なしの165回で同じ指摘が出たのは1回だけだった。

つまり、MAST の分類（と AI の振り分け）は、症状を正しくとらえていた。確かめが誤っており、言っていることとしていることが食い違っていた。しかし、その責任を AI の Ren に帰していた。実際に誤っていたのは、Ren の答えを上書きするプログラムだった。記録には Ren の答えと、上書きされたあとの点が区別されずに出ていたため、振り分けからは見分けられなかった。

5 直して測り直す

比べている値を外すように確かめを直し、直す前と直した後のチームを、別々の場所で立ち上げて同じ時刻に測った（各96回、24題 × 2回 × 2回）。採点と振り分けは、ここまでと同じ方法で行った。

指標	直す前	直した後
失敗ありの回（2つの振り分けがともに挙げた）	15/96	3/96
直しが上限の3回までまわった回	9	1
Ren を呼んだ回数（1題あたり）	1.25	1.08
AIを呼んだ回数（1題あたり）	6.30	5.91
トークン（1題あたり）	22,391	19,436
料金（100題あたり）	\$2.31	\$2.02
仕上がるまで（中央値）	43.0秒	42.5秒
動いた	96/96	96/96
審査役の合格（2点）	76/96（79.2%）	74/96（77.1%）

表3 直す前と直した後（同じ時刻に測定）。失敗ありの回の差は $p = 0.005$ 、3回までまわった回の差は $p = 0.018$ 、合格の差は $p = 0.86$ （フィッシャーの正確確率検定）。2つの振り分けの一致は $\kappa = 0.96$ と 1.00 。

むだな直しが減り、料金は13%下がった。出来は変わらなかった。不具合は大半の回では起きず、起きた回だけで料金を倍にしていたので、仕上がりの時間の中央値はほとんど動いていない。

直した後に残った3回のうち、1回は同じ確かめの別の不具合だった。 `on ? '' : 'active'` のような空の文字列があると引用符の組がずれ、 `' : '` をクラス名と数えていた。これも直した。1回は Ren の誤り（元の配列を壊していない `sort()` を誤りとした）、もう1回は振り分けの誤りだった。画面がエラーを見つけて自動で直した2回目の生成を、「終える条件に気づかない（FM-1.5）」と数えていた。2つの不具合を直した確かめを、これまでの5人のチームの記録 480回の最後のファイルにかけると、誤った指摘は 50回から 0回になった。

6 振り分けの指示による揺れ

AI に振り分けさせるとき、チームの決まりを指示に書くかどうかで、結果は大きく変わった。

振り分けの指示	結果
v1：チームの決まりを書かない	失敗ありが 37 回。担当のファイルがないので呼ばれなかった Toa を「役割を守らない (FM-1.2)」と 5 回数えた
v2：終える条件を誤って書いた（「75 点以上で終える」とだけ書き、fail が残れば続けることを書かなかった）	正しく続けた回を「終える条件に気づかない (FM-1.5)」と数えた（3.8-flash で 23 回、pro で 12 回）。失敗ありは 33 回と 59 回、一致は $\kappa = 0.61$
v3：正しい決まりを書いた（本稿の結果）	FM-1.5 は 0 回。一致は $\kappa = 0.94$

表4 振り分けの指示の違い（同じ192回）。v2 の2つ目の振り分けは `gemini-3.1-pro-preview`。v3 では、1日の利用の上限に達したため `gemini-3.7-flash` に替えた。

決まりを知らない振り分けは、仕組みどおりの正しい動きを失敗に数える。MAST を自分のチームに当てはめるときは、まず「何が正しい動きか」を振り分けに教える必要がある。そうしないと、失敗の数は、チームではなく指示の書き方を測ってしまう。

7 考察

7.1 「AIのチームの失敗」は、AIの失敗とは限らない

本番のチームは、AI だけでできているわけではない。点の書きかえ、終える条件、ファイルの割り振りなど、プログラムの決まりが AI の間に入っている。MAST は AI どうしのふるまいを分類するので、プログラムが起こした失敗も、いちばん近くにいる AI の失敗として数えられる。今回、確かめの誤りの大半は、AI の判断ではなく、AI の判断を上書きするプログラムから来ていた。

7.2 分類は、原因ではなく「どこを見るか」を教える

それでも、MAST は役に立った。失敗が「確かめ」と「食い違い」に集まり、根拠の文が同じ形をくり返していたことが、コードを読む手がかりになった。分類の数をそのまま「AI の弱点」と読むのではなく、どこを調べるかの地図として使うのがよい。

7.3 失敗は、出来ではなく料金に出る

失敗ありの回でも、最後の出来は下がっていなかった。最後の版は正しく、むだな直しがまわっていただけだからである。出来だけを測っていると、この失敗は見えない。呼び出しの回数とトークンを、回ごとに記録しておくことが、見つけるきっかけになった。

7.4 前回の報告への影響

報告 004 の5人のチームの料金（100題あたり \$2.16）にも、この不具合のむだが含まれている（96回中9回）。この9回を除くと \$1.92 になり、1人の形との料金の差は 004 の表より大きい。出来への影響は小さいと考えられる（不具合が起きた回でも、審査の合格は下がっていなかった）。

8 限界

- **振り分けは AI**：2つの振り分けはどちらも Gemini の系統で、人による振り分けはしていない。ただし、本稿の主な結論（プログラムの不具合）は、振り分けではなく、コードと記録で確かめ、直して測り直した。
- **規模**：1つのチーム、24題（日本語の小さなアプリ）。振り分けは192回、測り直しは各96回。
- **審査**：作るAIと審査役が同じ系統（Gemini）で、人による評価はない。審査の合格は回によって十数ポイント揺れる（002）ので、比べたのは同じ時刻に測ったものどうしである。
- **記録の見え方**：記録には、プログラムが書きかえたあとの点しか残っていなかった。AI の答えそのものを残していれば、振り分けでも見分けられた可能性がある。
- **MAST の割合との比べ**：元の論文は、別の種類の仕事（数学・ソフトウェア開発など）と、人による振り分けの結果である。割合の違いをそのまま比べることはできない。

9 まとめ

本番の5人のチームの記録に MAST を当てはめ、失敗の原因をコードまでたどった。分類は、失敗が「確かめ」と「食い違い」に集まることを正しく示したが、その多くを AI の見直し役のせいにしていた。実際の原因は、AI の答えを上書きするプログラムの小さな不具合だった。直すと、失敗ありの回は 15回から3回に減り、料金は13%下がり、出来は変わらなかった。AIのチームの失敗を調べるときは、AI だけでなく、AI の間に入るプログラムも疑う必要がある。そして、振り分けを任せる AI には、まずチームの決まりを教える必要がある。

参考文献

1. M. Cemri, M. Z. Pan, S. Yang, et al. “Why Do Multi-Agent LLM Systems Fail?” arXiv:2503.13657, 2025.
2. 湯村 信司. 「1人のAIで十分か」. AQUA Research Technical Report 004, 2026.
3. 湯村 信司. 「AIの開発チームは何人がいいか」. AQUA Research Technical Report 003, 2026.
4. 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.