8層チェック表

自動化の現在地を塗ります。Day1 の最初と Day2 の最後の2回使います。

自動化の層を下から積み上げた8段です。Day1 の冒頭に現在地を1回塗り、Day2 の最後にもう一度塗ります。2回の差が2日間の成果です。

塗り方は3段階だけにしてください。細かく刻むと2回目の比較ができなくなります。

- 未: 置いていない
- 一部: 置いてあるが、定義どおりには効いていない
- 済: 効いていることを機械で確かめられる

「効いている」の判定は人の記憶ではなく、右の確認方法の欄に書いたコマンドか画面で決めます。D2 の core.md の語彙で言えば、[A] で守っているものは「一部」、[M] に移ったものが「済」です。

対象のハーネスは1人1式に絞ってください。D2 と smart3pm の両方を持っている方は、2日間で育てるほうを先に決めてから塗ります。

層	中身	成熟度	到達の目安	確認方法	Day 1	Day 2
1	非対話実行 (`claude -p`)	GA	人が画面を見ていない状態で1タスクが終了コード0で終わる	`claude -p "..." --output-format json` の終了コードと `.result` を見る。承認待ちで止まるなら未		
2	実行中の介入点 (hooks)	GA	PreToolUse / PostToolUse / Stop のどれかが、人の合図なしに作業を差し戻している	フックへ手で JSON を流して終了コード2が返る。`.claude/hooks/tests/` に回帰テストがある		
3	仕事の分割 (subagents / skills / plugins / MCP)	GA	観点ごと・規模ごとに別定義が走り分けている	`/tasks` にサブエージェント名とモデルと effort が出る		
4	承認の代わり (permission modes / sandbox)	GA	破壊的操作が deny とフックで止まり、止まったことが記録に残る	`permissions.deny` に巻き戻せない操作が並び、`/bin/rm` 経由もフックで止まる		
5	長時間化 (compaction)	GA	1タスクが文脈切れで終わらない。口頭の禁止事項がコンパクション後も効いている	`/compact` をまたいで制約が守られるか。守られないものは設定へ移す		

層	中身	成熟度	到達の目安	確認方法	Day1	Day2
6	並列化(Agent Teams / Workflows)	Agent Teams は experimental	複数の観点レビューが同時に走り、結果が1か所に集まる	同じ差分に3本のレビュアーを当てて、指摘の重なりと差が数えられる		
7	起動の自動化(CI / pipeline schedule)	GitLab 連携は beta	人が起動しなくてもパイプラインが動き、朝に結果が残っている	GitLab の Pipelines に schedule 起因の実行履歴がある		
8	プログラムからの呼び出し (Agent SDK)	GA	自社のツールから直接エージェントを呼べる	SDK 経由の実行ログがある		

成熟度の欄は 2026-09-10 時点の公式ドキュメントの表記です。GA は GitHub Actions 連携までを含み、GitLab CI/CD 連携は beta、Agent Teams は experimental、Routines は research preview と書かれています。Routines は claude.ai ログイン前提で、API キー運用では `/schedule` が出ません。スケジューラを GitLab 側に置くのはこのためです。

出典: https://code.claude.com/docs/en/headless / https://code.claude.com/docs/en/gitlab-ci-cd

塗り終えた後の記入欄

Day1 の記入が終わったら、次の2行だけ書いてください。Day2 の最後に同じ欄をもう一度書き、行が動いたかを見ます。

記入欄	Day1	Day2
「済」が付いた層の番号		
次に埋めると一番効く層と、その理由		

第7層の位置づけ

CI の導入は新機能の追加ではありません。いまは機械強制のタグが、エージェントの完了報告に載ることで担保されていると伺っています。第7層は、その担保をパイプラインの実行記録に置き直す作業です。第7層に印が付くまで、第1層から第6層の「済」はすべて自己申告の上に乗っています。

Givery

委譲判定シート

AI の指摘を4つに仕分けます。ハンズオン1とハンズオン2で使います。

AI が出した指摘を、4分類のどれに入れるかを決めるシートです。ハンズオン1(Issue #1)とハンズオン2(Issue #2・#3)で使います。

指摘を全部直すと、5行の修正に半日かかります。全部見送ると、レビューを回す意味がなくなります。この2つの間に線を引くために、指摘ごとに3軸を先に決めてから分類します。分類から入ると、その日の気分で線が動きます。

3軸

軸	値	見るところ
可逆性	戻せる / 戻しにくい / 戻せない	コードだけの変更は戻せます。DB のスキーマとデータ、外部連携、権限の変更は戻せません。コードを `revert` してもデータは戻りません
機械検証	CI で落ちる / テストを足せば落ちる / 人が読むしかない	今の CI(lint 8.3、lint 5.4、DB テスト、AI レビューのゲート)で落ちるかどうかです。「落ちるはず」ではなく、落ちたのを見たかどうかで書きます
影響範囲	1ファイル内 / 同一機能 / 共通処理と複数機能	`app/libraries/` と `app/core/` に触る指摘は、常に一番右です

4分類の決め方

3軸を書いてから、次の順で当てはめます。上から順に見て、最初に当たったところで止めます。

1. 可逆性が「戻せない」なら、機械検証の値に関わらず **要調査**。人が読んでから決めます
2. 機械検証が「CI で落ちる」なら **即修正**。人が採否を議論する対象ではありません。議論した時点で工数が二重になります
3. 影響範囲が「共通処理と複数機能」で、可逆性が「戻しにくい」なら **将来課題**。この MR では直しません
4. 指摘の前提がこのリポジトリの設計と食い違っているなら **意図した設計**。理由を1行書いて `known_issues.md` に K 番号で登録します
5. 残りは **即修正** か **将来課題** を、この MR の目的に照らして選びます

「意図した設計」に入れたものは、次から同じ指摘が出ないように記録するところまでが1セットです。記録しないと、同じ判断を毎回の改修で繰り返します。小さな改修ほど指摘対応に時間がかかる原因の一つがここにあります。

記入欄

No	出どころ	指摘の要約	可逆性	機械検証	影響範囲	分類	次の一手	記入者
1	同一セッション	`estimate_ctl.php` の `customer_name` が SQL 文字列に連結されていて、`' OR '1'='1` で全件返る	戻せる	テストを足せば落ちる	同一機能	即修正	`db_select` の第2引数へプレースホルダで渡し、検索のテストを1本足す	
2								
3								
4								
5								
6								
7								
8								
9								
10								

出どころは「同一セッション」「別セッション」「Codex」「CI」のどれかを書きます。同じ指摘が2か所から出たときは、両方に行を立てずに1行にまとめ、出どころを2つ並べてください。重なった数がそのまま、レビューを何本回すかの判断材料になります。

集計欄

項目	件数
指摘の総数	
即修正	
要調査	
将来課題	

項目	件数
意図した設計	
このうち CI で落ちたもの	

一番下の行が大きいほど、人が読む量を減らせます。人が読む量を減らす手は「基準表を厳しくする」ではなく「CI に落とせるものを CI へ移す」です。

計測表

自分の数字をその場で書きます。演習の区切りごとに手を止めて記入してください。

演習中に自分で数える欄です。あとから思い出して埋めることはできないので、その場で書いてください。

数字を並べる目的は、レビューの置き方を自分の数字で選べるようにすることです。論文の数字ではなく、自分の MR で何が起きたかを見ます。

Day1 3通りのレビュー

ハンズオン1で、同じ差分に3通りのレビューを当てます。順番は固定です。実装したセッションを最初に、履歴を切った側を後にします。

項目	(1) 実装したセッション	(2) `/clear` 後に同僚のコードとして	(3) 別ツール
指摘の総数			
うち P0 と P1 に相当するもの			
重大な指摘の有無(仕込み欠陥を当てたか)			
所要時間(分)			
使ったモデルと effort			

(3) の別ツールは、Codex を持っている方は `codex review --base main "P0/P1 のみ、3件まで、コード提案不要"`、持っていない方は `git diff main | claude --bare -p --json-schema review.schema.json` です。

Codex CLI は P2 と P3 も出します。件数だけを比べると多く見えるので、2行目の「P0 と P1 に相当するもの」で比べてください。

書き終えてから答える欄

問い	記入欄
3つのうち、仕込み欠陥を当てたのはどれか	
同じ差分なのに指摘が変わった理由として、自分が納得できるもの	
自社で常用するなら、どの置き方にするか	
その置き方を選んだ理由(コストと再現性のどちらを優先したか)	

履歴を捨てた別セッションのほうが自己レビューより当たる、という測定結果があります(Cross-Context Review で F1 28.6 対 24.6、自己レビューの2回目は 21.7)。自分の数字がこれと違う向きに出たら、そのまま書いてください。1 件の MR で逆転することは普通に起きます。数字が一致しなかったことより、「履歴を切るだけで指摘が変わる」ほうが持ち帰る事実です。

Day1 委譲判定の集計

委譲判定シートの集計欄から転記します。

項目	件数
指摘の総数	
即修正	
要調査	
将来課題	
意図した設計	
CI で落ちたもの	
指摘の仕分けにかかった時間(分)	

Day2 検出率

ハンズオン2で、仕込み欠陥がいくつあるかは Issue に書いてあります。レビューがそのうち何件を当てたかを数えます。

項目	Issue #2	Issue #3
仕込み欠陥の件数		
レビューが当てた件数		
指摘の総数		
検出率(当てた件数 ÷ 仕込み欠陥の件数)		
ノイズ率(仕込み欠陥以外の指摘 ÷ 指摘の総数)		
REVIEW.md を書く前の指摘の総数		

項目	Issue #2	Issue #3
REVIEW.md を書いた後の指摘の総数		

下の2行が、この日で一番使う数字です。REVIEW.md を書いて指摘が減り、検出率が落ちていなければ、基準が効いています。両方とも減っていたら、Do not report に入れすぎです。

テスト有無の比較

同じリファクタを、機能テストが無い状態と有る状態の2回頼みます。

項目	テスト無し	テスト有り
CI の結果		
壊れたことに気づいた場所		
気づくまでにかかった時間(分)		

Day2 夜間ループ

ハンズオン3で、1回目がどこで止まったかを記録します。止まらずに完走したら、その旨を書いてください。

項目	記入欄
止まった箇所	
止まった理由	
どの層の設定で直すか	
直した内容(ファイル名とキー)	
再実行の結果	
ジョブの所要時間(分)	
使ったターン数(`--max-turns` の上限に当たったか)	

止まる箇所の候補です。当てはまるものに印を付けてから、上の欄を埋めてください。

- 権限で拒否されて、そこから先へ進まなかった
- PHP 5.4 で動かない構文が混入して、lint で落ちた
- テストが落ちて、そのまま終わった
- レビューの手順を飛ばして MR を作った

- 指摘を全部受け入れて、設計が元に戻った
- MR の作成が権限で通らなかった
- ターン数の上限に当たって、途中で終わった

最後に1枚書く欄

問い	記入欄
今のハーネスで夜間に回せる範囲	
人が残る3点	
年末までに埋める層の順番	

自社ハーネス診断シート

5層のどこが空いているかを、その場で点検します。

自分のハーネスが、5層のどこまで届いているかを1枚で見るための表です。点数を付ける表ではありません。空いている層を見つけて、次に何を足すかを決めるために使います。

対象は1人1式に絞ってください。複数のリポジトリを持っている方は、これから育てるほうを先に決めてから埋めます。

埋め方

判定は3つだけです。細かく刻むと、後で見返したときに比べられなくなります。

- **ある**: 効いていることを、右の欄の方法で確かめた
- **一部**: 置いてあるが、定義どおりには効いていない。または一部の環境でしか効かない
- **無い**: 置いていない

「ある」は、確かめた場合だけ付けてください。 置いた記憶があるだけのものは「一部」です。ハーネスは、効いていないことが画面に出ません。思い込みで「ある」が付くと、そこだけ確認が止まります。

先に書く欄

表を開く前に、予想を書いてください。埋め終わったら、予想と結果の差を見ます。差が出た場所が、いちばん危ない場所です。

予想	記入欄
自分のハーネスで、いちばん空いていそうな層	
「ある」が付きそうな項目の数(30 のうち)	

第1層 文脈

AI に読ませている決まりの層です。効き目は一番弱く、置き換えの対象を見つけるために点検します。

#	点検する項目	どうやって確かめるか	判定	次の一手
1	常時読み込む文書が1本に決まっている	入口のファイルを開き、他の文書への入口が1か所か数える	ある / 一部 / 無い	`template/AGENTS.md`
2	常時読み込む文書が80行以内	行数を数える	ある / 一部 / 無い	`template/AGENTS.md`
3	同じ決まりが2か所以上に書かれていない	決まりを3つ選び、他の文書に同じ記述が無いか探す	ある / 一部 / 無い	`設計の10原則.md` の8
4	禁止の行に代替が書かれている	禁止の行を5つ選び、代わりの道が同じ行にあるか見る	ある / 一部 / 無い	`template/AGENTS.md` の変更の進め方
5	ツールを乗り換えても残る部分が分かれている	特定のツール名を含む記述が、固有の配線ファイルに寄っているか見る	ある / 一部 / 無い	`template/CLAUDE.md` の冒頭
6	完了報告に必須項目がある	直近の作業報告を3件開き、同じ項目が並んでいるか見る	ある / 一部 / 無い	`template/CLAUDE.md` の作業の終わり方

第2層 権限

ツールの呼び出しを、通す・確認する・止めるの3つに仕分ける層です。

#	点検する項目	どうやって確かめるか	判定	次の一手
7	止める一覧が空でない	設定ファイルの止める欄を開き、行数を数える	ある / 一部 / 無い	`template/.claude/settings.json`
8	巻き戻せない操作が止める側に入っている	消す・履歴を書き換える・外へ送るの3つが並んでいるか見る	ある / 一部 / 無い	`template/.claude/settings.json`
9	秘密情報の読み取りが止まる	秘密情報のファイルを読ませてみて、止まるか見る	ある / 一部 / 無い	`template/.claude/rules.json` の `secret-file`
10	止める指定が4つの編集ツール全部に書かれている	止める欄の編集系の行を数え、ツール名が4つそろっているか見る	ある / 一部 / 無い	`template/.claude/settings.json`
11	確認を挟む操作が決まっている	確認の欄を開き、外へ送る操作が入っているか見る	ある / 一部 / 無い	`template/.claude/settings.json`
12	個人ごとの差が共有設定に混ざっていない	共有設定の中に、特定の人の環境でしか通らないパスが無いか探す	ある / 一部 / 無い	`50_環境別のフック設計` の5

第3層 フック

実行の直前に割り込み、コマンドの中身まで見る層です。ここが一番厚くできます。

#	点検する項目	どうやって確かめるか	判定	次の一手
13	フックが1本以上動いている	止めたい入力を手で流し、終了コード2が返るか見る	ある / 一部 / 無い	`template/.claude/hooks/README.md`
14	破壊的な操作の判定が綴りの列挙ではない	同じ意味でオプションの並びを変えた形を3つ流し、全部止まるか見る	ある / 一部 / 無い	`template/.claude/rules.json`
15	包んだ形でも止まる	シェルを1枚かぶせた形を流し、止まるか見る	ある / 一部 / 無い	`template/.claude/hooks/guard.sh`
16	判定に使う道具が無いときに止まる	道具の場所を存在しないパスに差し替えて流し、止まるか見る	ある / 一部 / 無い	`template/.claude/hooks/guard.sh` の冒頭
17	フックにテストがある	テストを流し、件数と合否が出るか見る	ある / 一部 / 無い	`template/.claude/hooks/guard.test.sh`
18	テストに通す側の入力が入っている	テストの中身を開き、通す側の件数を数える	ある / 一部 / 無い	`template/.claude/hooks/guard.test.sh`
19	設定ファイルが改行コードで無効にならない	設定を CRLF で保存し直して流し、同じ判定になるか見る	ある / 一部 / 無い	`template/.claude/hooks/guard.sh`
20	止める対象がスクリプトの外にある	規則を1件足すときに触るファイルの数を数える	ある / 一部 / 無い	`template/.claude/rules.json`

第4層 CI

手元の設定と無関係に、提出のときに動く層です。手元を外した人にも効きます。

#	点検する項目	どうやって確かめるか	判定	次の一手
21	提出時に自動で動く検査がある	直近の提出を開き、検査の実行記録があるか見る	ある / 一部 / 無い	`template/ci/gate.sh`
22	落ちる条件が数値で決まっている	閾値がどこに書かれているかを指さす	ある / 一部 / 無い	`template/ci/gate.sh`
23	検査の結果が人の読む場所に出る	落ちたときの表示を、実際に落として見る	ある / 一部 / 無い	`template/ci/gate.sh`
24	文書に書いた数と実装の数を突き合わせている	文書の数値を1つ変えて検査を流し、落ちるか見る	ある / 一部 / 無い	`template/ci/verify-harness.sh`
25	レビューの出力形式が固定されている	出力の形を定めたファイルがあるか見る	ある / 一部 / 無い	`template/.claude/review.schema.json`
26	レビューの起動が人の気分に依存していない	直近の作業で、起動しなかった回数を数える	ある / 一部 / 無い	`template/.claude/agents/`

第5層 リポジトリ保護

手元の設定を全部外した人にも効く層です。

#	点検する項目	どうやって確かめるか	判定	次の一手
27	主要な枝に直接反映できない設定がある	ホスティングの設定画面を開き、保護の欄を見る	ある / 一部 / 無い	`README.md` の5層の表
28	履歴の書き換えが拒否される	設定画面で、強制的な反映の可否を見る	ある / 一部 / 無い	`README.md` の5層の表
29	受け入れ時にサーバ側で動く検査がある	サーバ側の検査の設定があるか見る	ある / 一部 / 無い	`50_環境別のフック設計` の落とし穴の表
30	戻す手順が配る手順と対になっている	配る手順書を開き、同じ粒度の戻す手順があるか見る	ある / 一部 / 無い	`template/docs/ROLLBACK.md`

埋め終わったら

3行だけ書いてください。ここが持ち帰るものです。

記入欄	書くこと
「無い」が一番多かった層	
予想と結果が食い違った項目の番号	
次の1週間で埋める項目を1つだけ	

選ぶのは1つだけにしてください。 5つ選んだ人は、1か月後にどれも終わっていません。1つ埋めて、効いていることを確かめてから次を選びます。

なお、第1層に「無い」が並んでいても、第3層と第4層が埋まっているなら危険な状態ではありません。逆に、第1層だけが埋まっていて他が空いている状態は、守っているつもりで何も止まっていない状態です。層の順番に埋める必要はありません。

持ち帰り台帳

作ったファイルの置き場所と担当を決めます。Day1 と Day2 の最後に使います。

2日間で作ったファイルを、自社ハーネスのどこに置くかを決めるための台帳です。Day1 の最後に7行、Day2 の最後に7行を埋めます。

演習で作ったものは演習リポジトリ(dl-training-app)の中にあります。そのままでは自社では動きません。この台帳の役目は、置き場所と担当と期日を1行ずつ決めて、研修が終わった翌週に手が止まらないようにすることです。

書き方

- 置き場所はパスで書きます。「ハーネスに入れる」では翌週の自分が探せません
- D2 と smart3pm のどちらか片方しか使わない行は、使わないほうに「対象外」と書きます。空欄のままにしないでください
- 担当は名字を書きます。「チーム」は担当ではありません
- 反映日は日付を入れます。未定のものは、いつ決めるかの日付を入れます

移すときの分岐

D2 は PowerShell 5.1 で、コードと文字列を ASCII のみに保つ約束があります。smart3pm は bash が正です。同じ目的のガードが2言語で二重に実装されている状態で、統一の時期はまだ決まっていないと伺っています。

演習では ps1 と sh の両方を配っています。片方だけを持ち帰るか、両方を置いて片方を正とするかは、この台帳を埋めるときに決めてください。決めた結果を「置き場所」の欄に残すのが一番速い記録です。

Day1

番号	持ち帰り物	D2 での置き場所	smart3pm での置き場所	担当	反映日
D1-1	規模別レビュアー2本(`review-light.md` / `review-full.md`)				
D1-2	settings.json のモデルと effort(`model` / `effortLevel` / `maxEffortLevel`)				
D1-3	deny 5本(巻き戻せない操作)				
D1-4	CLAUDE.md の「戻し方」節				

番号	持ち帰り物	D2 での置き場所	smart3pm での置き場所	担当	反映日
D1-5	deny 3本(`.env` と認証情報の読み取り)				
D1-6	PreToolUse フック `block-destructive` (ps1 / sh)と `hooks/tests/cases.json` のケース				
D1-7	実装とレビューを分ける定義1本(`skills/codex-review/SKILL.md` か `agents/review-other.md`)と MR テンプレの「Codex レビュー結果」欄				

D1-2 の `maxEffortLevel` は v2.1.267 で追加されたキーです。持ち帰り先の Claude Code が古いと無視されます。反映日を決める前に `claude --version` を確認してください。

Day2

番号	持ち帰り物	D2 での置き場所	smart3pm での置き場所	担当	反映日
D2-1	CLAUDE.md と AGENTS.md の実行環境制約節(代替の書き方の表)				
D2-2	PostToolUse フック `check-phpver` (ps1 / sh)と `cases.json` の追加ケース				
D2-3	`REVIEW.md` (自社版初版)と AGENTS.md の `## Code Review Rules`				
D2-4	観点別サブエージェント3本と、コードレビュー側へ移した `known_issues.md` / `patterns.md`				
D2-5	`review.schema.json` と `gate.ps1` / `gate.sh` 、レビューを2ジョブに分けた `.gitlab-ci.yml`				
D2-6	Stop フック `stop-gate` (ps1 / sh)				
D2-7	夜間ジョブ一式(`nightly_implement` 、pipeline schedule、 `skills/implement-issue` 、止め金を書いた settings.json)				

D2-4 の known_issues と patterns は、リリースレビュー側で既に動いているものをコードレビュー側へ写す作業です。新しく作る話ではありません。

研修後の宿題

自社ハーネスの側に残っていて、2日間では触らなかった箇所です。担当と期日だけ決めてから解散してください。

項目	担当	期日
フックのテストがある側とない側の非対称の解消(D2 は回帰テストあり、smart3pm は4本のみ)		
フック自身の ASCII 制約の機械検査、または pwsh 7 への移行		
PowerShell と bash の二重運用をいつ片方に寄せるかの決定		
作業指示書の Notion への反映		

Givery

社内展開ガイド A4雛形

Day2 の終盤に、A4 1枚をその場で書き上げます。

Day2 の終わりに、この雛形の上で A4 1枚を書きます。読む相手は、2日間の研修に出ていない運用メンバーです。

書けたかどうかの判定は1つです。作成者以外の1名が、このガイドだけを見て設定を1つ入れ直せること。入れ直せなければ、粒度が足りていません。

各項目に記入例を付けています。記入例の行は消して、自分の内容に置き換えてください。

---

1. 何を

追加したものを、ファイル名で書きます。機能名では探せません。

ファイル	何をするもの
記入例: `.claude/hooks/check-phpver.ps1`	記入例: PHP ファイルを編集した直後に走り、5.4 で動かない構文があれば AI に差し戻す

2. どこに

置き場所と、効かせるための登録先です。ファイルを置くだけでは動かないものが混ざっています。

ファイル	置き場所	登録先
記入例: `check-phpver.ps1`	記入例: `D2/.claude/hooks/`	記入例: `.claude/settings.json` の `hooks.PostToolUse`、matcher は `Edit

ASCII の制約に触れる場合は、ここに書いてください。

記入例: このフックは PowerShell 5.1 で動きます。コードとメッセージは ASCII のみです。日本語のメッセージを入れると、CP932 環境で実行時の出力が壊れます。

3. なぜ

1行ずつ、入れなかったときに何が起きるかを書きます。「品質向上のため」は理由になりません。半年後に外す判断ができません。

ファイル	入れなかったときに起きること
記入例: `check-phpver.ps1`	記入例: CLAUDE.md に書いた制約は数ターンで薄れ、`??` が混入する。CI の `lint-phpver` は `php:7.4-cli` なので、`::class` と `finally` はそこも通り抜けて本番で落ちる

4. 壊れた時の戻し方

止まったときの見分け方と、戻す手順です。ここが書けていないと、最初のトラブルで全部が外されます。

症状	見分け方	戻し方
記入例: 編集のたびに差し戻される	記入例: `[check-phpver]` で始まるメッセージが出ている。行番号と構文名が付いている	記入例: 指摘された行を代替の書き方に直す。誤検出のときは phpcs を入れる。緊急で止めるなら settings.json の `hooks.PostToolUse` の該当ブロックだけを外す。ファイルは消さない

止め方を「ファイルを消す」にしないでください。消すと、次に入れ直す人が最初から作ります。

5. 担当と連絡先

項目	記入欄
このガイドの担当	
設定を変えるときの相談先	
変更の記録場所(MR かドキュメント)	
次の見直し日	

6. まだ入れていないもの

入れなかったものを書いておくと、次に足す人が同じ検討を繰り返しません。

項目	入れなかった理由
記入例: OS のサンドボックス	記入例: Windows では Seatbelt も bubblewrap も動かない。守りは deny とフックの2層で、3層目は無い

書き終えた後の確認

- 作成者以外の1名が、このガイドだけを見て設定を1つ入れ直せた
- ファイル名がすべて実在する。研修中の仮の名前が残っていない
- 「なぜ」の欄に、起きることが具体的に書いてある
- 戻し方が、ファイルを消す以外の手順になっている

Givery

効果測定の4指標表

研修後に測る指標と、研修前との突き合わせ方です。

レビュー工数を1つの数字で追うと、うまくいきません。工数は測る人の記憶に依存し、改修の大きさでばらつき、比較にならない数字が残ります。

代わりに、GitLab の MR データから機械的に取れる4つに限定します。人が入力する欄を1つだけ残し、残り3つはデータから取ります。入力欄が増えるほど、3か月後には誰も書かなくなります。

4指標

No	指標	定義	取り方	人の入力
1	MR あたりの AI 指摘件数と採用率	1つの MR で AI が出した指摘の件数と、そのうち対応した件数の割合	指摘件数は AI レビュージョブの `findings.json` を artifact に残して数えます。採用率は委譲判定シートの「即修正」の件数を分子にします	採否だけ
2	人が採否を判断した件数	指摘のうち、CI で自動的に落ちたものを除いた件数	指摘の総数から、ゲートジョブが落とした P0 と P1 の件数を引きます	なし
3	ゲートで止まった MR の割合	ゲートジョブが失敗した MR の数 ÷ 全 MR の数	パイプラインのジョブ結果から数えます	なし
4	マージまでの時間と手戻り MR 率	MR 作成からマージまでの時間の中央値と、マージ後に同じ箇所を直す MR が出た割合	`created_at` と `merged_at` の差。手戻りは同じファイルに触る後続 MR を数えます	なし

指標2が、回答書に書かれた課題に一番近い数字です。「品質は上がったが工数削減は限定的」は、指標2が減らないまま指標1が増えている状態です。指標2を減らす手は、基準表を厳しくすることではなく、機械で判定できる指摘を CI 側へ移すことです。指標3が上がって指標2が下がれば、狙いどおりに動いています。

取り方

GitLab の REST API から取ります。エンドポイント名は下のとおりですが、自社インスタンスの版で使えるかは確認してから使ってください(要確認)。研修の調査で一次情報を取得できているのは、pipeline schedules とプロジェクトアクセストークンと push options の3つだけです。

取りたいもの	エンドポイント	主なクエリ
MR の一覧と作成・マージ日時	`GET /projects/:id/merge_requests`	`state=merged`、`created_after`、`created_before`、`per_page`

取りたいもの	エンドポイント	主なクエリ
MR ごとのパイプライン	`GET /projects/:id/merge_requests/:iid/pipelines`	
ジョブの成否	`GET /projects/:id/pipelines/:pipeline_id/jobs`	`scope=failed`
ジョブの artifact(`findings.json`)	`GET /projects/:id/jobs/:job_id/artifacts/:artifact_path`	
MR で変更したファイル	`GET /projects/:id/merge_requests/:iid/changes`	手戻りの判定に使います

認証は `api` スコープのプロジェクトアクセストークンです。masked variable に置きます。

集計の単位は週です。日次にすると改修の粒度のばらつきに埋もれます。月次にすると、施策を入れてから効果を見るまでが遅すぎます。

ビフォー値との突き合わせ

ビフォーは研修前の8週分です。突き合わせの前に、次の3つを揃えてください。揃っていない数字を並べると、施策の効果ではなく期間の性質を見ることになります。

揃えるもの	決め方
対象リポジトリ	ビフォーと同じリポジトリだけを数えます。研修で使った演習リポジトリは混ぜません
対象 MR の絞り込み	Draft のまま閉じた MR、リリース用のマージ MR、設定ファイルだけの MR を外すか入れるかを最初に決めて、両側で同じにします
指摘件数の数え方	ビフォー側は現行の AI レビュージョブが MR コメントに投稿した件数です。アフター側は `findings.json` の配列長です。数え方が違う期間は、切り替えた週に印を付けます

比較する形です。

指標	ビフォー8週(研修前)	アフター4週(研修後)	差
1 MR あたりの AI 指摘件数(中央値)			
指摘の採用率			
人が採否を判断した件数(1 MR あたり)			

指標	ビフォー8週(研修前)	アフター4週(研修後)	差
ゲートで止まった MR の割合			
マージまでの時間(中央値)			
手戻り MR 率			

ビフォー側にゲートジョブは存在しないので、4行目のビフォーは 0% です。0% から上がることが悪い変化ではありません。止まった分だけ、人の目を通らずに済んだということです。

アフターの初回値

ハンズオン2で数える「仕込み欠陥の検出数 ÷ 指摘の総数」を、アフター側の初回値として置きます。既知の欠陥が入ったコードに対して、レビューが当たるかを測る形です。過去の改修10本を使ったバックテストの代わりになります。

`patterns.md` のタグ集計は補助です。タグを付けるのは人なので、主指標には置きません。

数字が揃わない場合

ビフォー値を正確に取るのは難しいという話が先方から出ています。8週分が揃わないときは、4指標の表だけを渡して、ハンズオン2の検出率を唯一の実測値として扱ってください。揃わない数字を埋めた表を作るより、実測が1つある表のほうが後で使えます。

出典: https://docs.gitlab.com/ci/pipelines/schedules/ / https://docs.gitlab.com/user/project/settings/project_access_tokens/

Givery