

Claude Code で 600 テストケースを 書いて得た知見

2025/07/31
LINEDC Claude Code 実践 Meetup

概要

- Claude Code をフル活用してテストが無かったバックエンドのプロジェクトに自動テストを導入
- 最終的に**全自動**で全コントローラのテストケースを書けるようになった
- つまづいたポイントと解決方法を紹介

前提

- 技術スタック
 - Cloud Functions for Firebase
 - Cloud SQL
 - TypeScript
 - Prisma

課題: 每回の承認が面倒

- CC に張り付いてコマンドを承認するのが面倒
- ポチポチしているうちに `rm -rf` を許可してしまうリスク

解決方法

公式の DevContainer を導入して承認をスキップ

- 3 ファイルをコピペして VSCode 開くだけで OK
- 放置できるだけで体験が全然違う！おすすめ！

DevContainer 導入手順

- VS Code と Remote - Containers 拡張機能をインストール
- Claude Code 本体の .devcontainer/ を自分のプロジェクトにコピペ
- VS Code でリポジトリを開く
- 「Reopen in Container」をクリック

参考: [公式ドキュメント](#)

課題: 途中で作業を勝手に止める

- 長時間の作業中に処理が中断される
- 突然の 本日の作業はここまでとします。

解決方法

- TODO.md を作成し作業の状態を保持する
- シェルスクリプトでループ実行する

```
for i in {1..10}; do claude -p "/fill-todo"; done
```

課題: 余計な実装が無限増殖する

- 要求していない機能や設計を勝手に追加する
- CLAUDE.md に実装方針を書いても全然無視される

解決方法

- linter ルールで CC がやりがちな実装を禁止する
- エラーメッセージで修正方法を明記するのが効果的

ディレクトリごとに eslint ルールを設定する

```
test/
  └── controllers/
    ├── eslintrc.js
    ├── getUser.test.ts
    ├── updateUser.test.ts
    └── ...
  └── factory/
    ├── eslintrc.js
    ├── userFactory.ts
    └── ...
└── eslintrc.js
```

例: test/controllers/eslintrc.js

```
module.exports = {
  extends: ["../../.eslintrc.js", "../.eslintrc.js"],
  rules: {
    "no-restricted-syntax": [
      "error",
      {
        selector: [
          "CallExpression[callee.object.object.name='prisma'][callee.property.name='create']"
          ].join(''),
        message: "テストデータの作成は、Factory で実装してください",
      },
      {
        selector: [
          "TSNonNullExpression MemberExpression[property.name='authId']",
          "TSNonNullExpression OptionalMemberExpression[property.name='authId']",
        ].join(', '),
        message: '.authId! の代わりに .authId as string を使用してください',
      },
    ],
  },
};
```

課題: コード生成と修正の効率が悪い

- 初手で型エラー や linter エラーが多い状態でコードが生成される
- エラーを含めた修正に時間がかかる
- エラーが残った状態でタスクが終了する

解決方法

hooks で即時フィードバック

- 即時フォーマット
- 即時型エラー通知

例: .claude/settings.json

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "
              jq -r '.tool_input.command | test(\"rm -rf|dd if=|:((){ :|:& };;:)\""
              then {\"decision\": \"block\", \"reason\": \"危険なコマンドは実行できません。別の方を検討してください。\"} else empty end'
            "
          }
        ]
      }
    ],
    "PostToolUse": [
      {
        "matcher": "Write>Edit|MultiEdit",
        "hooks": [
          {
            "type": "command",
            "command": "
              jq -r '.tool_input.file_path | select(contains(\"/gcp-infra/functions/\"))
              and (endswith(\".js\") or endswith(\".ts\")))' | xargs -r sh -c 'cd /workspace/gcp-infra/functions && npx eslint --fix \"$@\"'
            "
          },
          {
            "type": "command",
            "command": "
              jq -r '.tool_input.file_path | select(contains(\"/gcp-infra/functions/\") and (endswith(\".js\") or endswith(\".ts\")))' |
              xargs -r sh -c 'cd /workspace/gcp-infra/functions && if ! npm run typecheck >/dev/null 2>&1;
              then echo \"型エラーが検出されました。修正してください。\" >&2; exit 2; fi'
            "
          }
        ]
      }
    ]
  }
}
```

まとめ

Claude Code を自走させるために大切なこと

- 何度も実行すれば作業が確実に完了するようにコマンドを設計する
- ルール < 既存のコード << プロンプト <<<< 静的解析
- 静的解析で自分の代わりに即時フィードバックしてくれる状態を作る

学び

現在のプログラミングは、

- 自動テストのループ
- 品質を担保するガード

が書ければ終わるという感覚を得た。

参考:

<https://aoai-ai-coding.mizchi.workers.dev/#5>

学び

逆説的に

- 静的解析で品質を担保しやすい設計にする
- あるいは静的解析で担保できる程度の品質を受け入れる

ことでAIエージェントによる開発のスケールが実現できそう。

学び

Agentic Coding はやってみて分かることがめちゃくちゃ多い！！
みんな手を動かそう！！

番外編: 最近よく使うテクニック

Kiro を再現したコマンドで実装計画を立てる

参考: [kiroを参考にして作成したCLAUDE.md](#)

- 要件定義 → 設計 → タスク分割 → 実行 のステップを強制する
- それぞれのフェーズで人間がレビューする
- 余計なことをやろうとしているのを事前に防げる
- CLAUDE.md だと指示に従わないことがあるのでカスタムコマンドとして定義するのを推奨

git worktree で並行開発 & PR 作成

```
git worktree add ../project-feature-a feature-a  
git worktree add ../project-feature-b feature-b
```

```
cd ../project-feature-a && claude -p "/issue-workflow #120"  
cd ../project-feature-b && claude -p "/issue-workflow #121"
```

- Issue から PR を作成するコマンドを定義しておく
- GitHub 上で PR をレビューする
- レビューコメントを拾って修正するコマンドを実行

```
claude -p "/review-comment-fix"
```

ご清聴ありがとうございました