Maximizing LLM Code Generation Quality: A Systematic Approach

What This Is About

The Challenge:

- Easy to have LLMs generate anything—music, images, slides, or code
- Hard to do it consistently with high quality
- This presentation demonstrates maximizing code quality while minimizing interaction overhead

Git Repositories:

- https://github.com/farmisen/todoish
- https://github.com/farmisen/scrapinator

Tooling Stack

Core Tools:

- **Linear** - Issue tracking and project management
- **GitHub** - Git repository hosting
- **Claude Dev** - AI-driven development process orchestration

Overall Process

Workflow Overview:

1. Start with formal software specification document
2. Connect Claude to Linear using official Linear MCP server
3. Claude analyzes spec and creates Linear tickets
4. Review and amend tickets as necessary
5. Prompt Claude to implement individual tickets
6. Claude follows workflow defined in `claude.md`
7. **End result:** Pull Request posted to GitHub

Evolution of claude.md

Version 1 - First Iteration

- Created by running `/init`
- No automation
- Manual prompting required

```
## Commands
- Build: `pnpm build` or `npm run build`
- Dev: `pnpm dev` or `npm run dev`
- Lint: `pnpm lint` or `npm run lint`
```

Version 2 - Linear Integration

Key Improvements:

- Integrated Linear MCP server for ticket CRUD operations
- Enabled ticket creation from Claude
- **Discovery:** Claude has poor branch management skills

```
## Linear Project

This codebase is associated with the **YOUR PROJECT NAME** project in Linear.
When creating, fetching, listing, or updating issues, always use this project context.
```

Reference: https://linear.app/docs/mcp

Version 3 - Implementation Recipes

Approach:

- More explicit implementation procedures
- **Results:** Mixed success—Claude frequently forgets instructions

```
### Branch Management
- **Branch Creation**: create new branch before starting work
- **Pull Latest Changes**: pull latest from `main`
- **Fetch Ticket Details**: get latest description
```

Version 4 - Being More Assertive

Strategy:

- Used emphatic wording: ALWAYS, NEVER, IMPORTANT
- **Results:** Better compliance, but still insufficient

```
- **Branch Creation**: ALWAYS create new branch
- **Pull Latest Changes**: ALWAYS pull from `main`
- **Title Format**: ALWAYS use `[<ticket-id>] Title`
```

Version 5 - Explicit Workflow Steps

Implementation:

- Explicitly listed development workflow steps
- **Results:** ~60% compliance, frequent drift

```
1. **Fetch Ticket Details**: ALWAYS fetch the latest ticket description before starting implementation.
2. **Create a new branch**: ALWAYS create a new branch from main for the ticket.
3. **Search Documentation**: ALWAYS check the doc folder for relevant information.
4. **Plan**: ALWAYS suggest a plan before starting the actual implementation
...
10. **Create a pull request**: ALWAYS create a pull request.
11. **Self Review the pull request**: NEVER skip this step.
12. **Address the comments**: ALWAYS address all the comments left open.
```

Version 6 - Internal Planning Control

Breakthrough Approach:

- Require Claude to create todo list before starting
- **Results:** >90% compliance rate
- Simple reminders redirect missed steps

```
**IMPORTANT**: When starting work on a ticket,
IMMEDIATELY create a TodoWrite list with all
these steps as individual todos. Use the template
bellow and mark each as completed as you progress.
This ensures no steps are missed.
```

Version 6 - Todo List Format

```
1. [ ] Fetch latest [TICKET-ID] details from Linear
2. [ ] Create new branch from main
3. [ ] Search /doc folder for relevant information
4. [ ] Write implementation plan and get approval
5. [ ] Implement the feature/fix
6. [ ] Run quality checks
7. [ ] Run tests and maintain coverage
8. [ ] Update documentation if needed
9. [ ] Commit changes with proper message
10. [ ] Create PR with standard format
11. [ ] Self-review PR and add comments
12. [ ] Address review comments
```

Key Insights

Communication Principles:

- Similar to effective human-to-human communication
- Avoid ambiguities
- Be assertive about requirements and methodology

Next Evolution - Multiplying with Worktrees

The Scaling Challenge

Current Limitation:

- Minimal interaction enables multiple Claude instances
- **Problem:** Git allows only one branch per folder
- **Solution:** `git worktree add <path> <branch-name>`

Implementation: mkwt

Concept: Create different worktree env per Claude instance

Challenge: Projects need more than git-tracked files

- Environment variables/secrets
- Dependencies
- Virtual environments

Solution: `./bin/mkwt ticket-id`

mkwt Script Functionality

1. Fetch ticket description from Linear
2. Build branch name
3. Create worktree
4. Symlink environment files from main branch
5. Create virtual environment (Python projects)
6. Install dependencies
7. Navigate to worktree folder
8. Launch Claude interactive session with ticket context

Future Development

Immediate Goals

- Convert worktree scripts into standalone tool
- Create project context generator for claude.md

Extensibility Goals

- Ticket provider agnostic
- Stack agnostic
- Model agnostic

Enhanced Feedback

- Notion MCP server integration
- Puppeteer for visual feedback loops

Current Limitations

Anthropic Rate Limiting Update

> *"Next month, we're introducing new weekly rate limits for Claude subscribers, affecting less than 5% of users... advanced usage patterns like running Claude 24/7 in the background—that are impacting system capacity"*

Impact: Advanced usage patterns will face new restrictions

Discussion: https://news.ycombinator.com/item?id=42713757

Questions & Discussion

Thank you for your attention!

Resources:

- https://github.com/farmisen/todoish
- https://github.com/farmisen/scrapinator
- https://linear.app/docs/mcp
- https://git-scm.com/docs/git-worktree