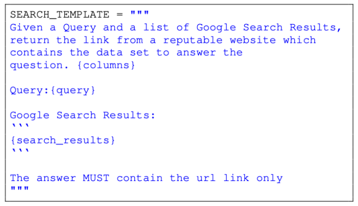
Prompting in the Wild: An Empirical Study of Prompt Evolution in Software Repositories

CS 846

Presented by Brian Do - 2025/11/04

Motivation/Problem being solved

- LLMs are now integrated into real software systems.
- Prompts are key artifacts in LLM-based software.
- Yet, little is known about how prompts evolve over time.
- Research gap: What are real-world patterns of "prompt engineering"?



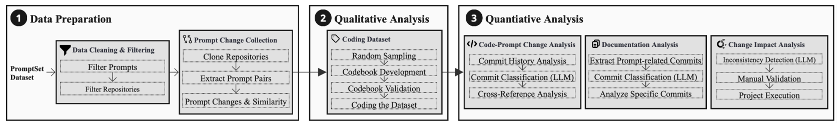
~~~
SEARCH_TEMPLATE = """
Given a Query and a list of Google Search Results,
return the link from a reputable website which
contains the data set to answer the
question. {columns}

Query:{query}

Google Search Results:
```
{search_results}
```

The answer MUST contain the url link only
"""
~~~

Example of developer-written prompt

Research Goal & Question

- Understand how prompts evolve in software development
- Four Themes of Research Questions:
  1. What types and patterns of prompt changes occur?
  2. How do prompt changes co-occur with code changes?
  3. How do developers document and describe prompt changes?
  4. What is the impact of prompt changes on prompt consistency and LLM output?

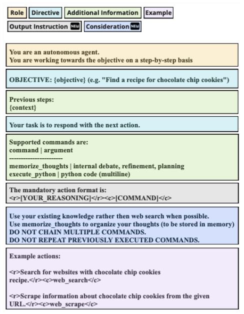
Method Overview



243 repositories, 1262 prompt changes

Key Findings

- **Prompt changes are focused and localized** (Obs. 1)
  - Most changes involve adding or modifying specific parts rather than restructuring the whole prompt.
  - Structural or presentation changes are rarer and mainly aim for clarity improvements.
- **Developers refine "Considerations" the most** (Obs. 2)
  - These guide how the model behaves; context and formatting come next, while core instructions stay stable.


Role
Directive
Additional Information
Example
Output Instruction NEW
Consideration NEW

You are an autonomous agent.
You are working towards the objective on a step-by-step basis

OBJECTIVE: {objective} (e.g. "Find a recipe for chocolate chip cookies")

Previous steps:
{context}

Your task is to respond with the next action.

Supported commands are:
command | argument
----
memorize_thoughts | internal debate, refinement, planning
execute_python | python code (multiline)

The mandatory action format is:
<r>[YOUR_REASONING]</r><c>[COMMAND]</c>

Use your existing knowledge rather then web search when possible.
Use memorize_thoughts to organize your thoughts (to be stored in memory)
DO NOT CHAIN MULTIPLE COMMANDS.
DO NOT REPEAT PREVIOUSLY EXECUTED COMMANDS.

Example actions:

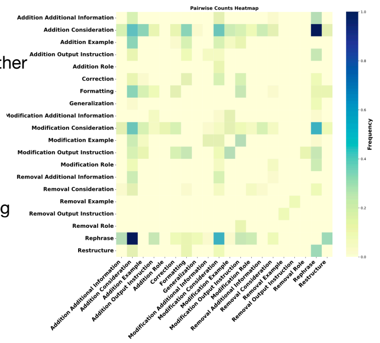
<r>Search for websites with chocolate chip cookies recipe.</r><c>web_search</c>

<r>Scrape information about chocolate chip cookies from the given URL.</r><c>web_scrape</c>

Example of refined prompt components

Key Findings

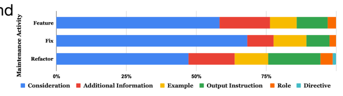
- **Prompt changes are focused and localized** (Obs. 1)
  - Most changes involve adding or modifying specific parts rather than restructuring the whole prompt.
  - Structural or presentation edits are rarer and mainly aim for clarity improvements.
- **Developers refine "Considerations" the most** (Obs. 2)
  - These guide how the model behaves; context and formatting come next, while core instructions stay stable.
- **Rephrasing supports conceptual changes** (Obs. 3)
  - When "Considerations" change, they're often paired with rephrasing.



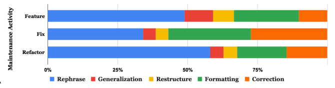
Proportional Distribution of Changes Across Prompt Components

Key Findings

- **Prompt evolution follows software workflows** (Obs. 4–5)
  - Most prompt changes happen during feature development; bug fixes and refactoring are less common.
  - Feature changes add new instructions, bug fixes modify behaviors, and refactors mainly rephrase for clarity.
- **Documentation is sparse and vague** (Obs. 6–7)
  - Only ~21.9% of prompt changes are documented; most commit messages are abstract or non-specific.
- **Prompt changes can cause inconsistencies** (Obs. 8-9)
  - Changes sometimes break instruction alignment or structural coherence.
  - And even when changes are intentional, they don't always produce the desired model behavior.



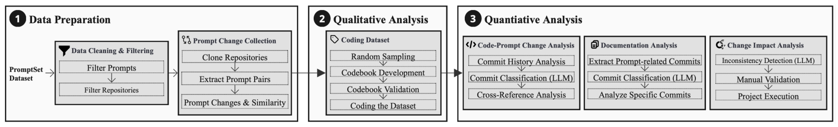
Distribution of prompt component changes over software maintenance activity types



Distribution of component-independent changes over software maintenance activity types

Positive Points

- Systematic, replicable approach: mining GitHub repositories and tracking prompt evolution.
- Novel empirical angle on LLM usage: connects prompt engineering with software evolution.
- Valuable dataset foundation for future prompt research.



Negative Points

- No developer interviews - limited insight into intent, rationale, and barriers.
- Dataset bias - focused on open-source Python repos, excludes shorter prompts (<15 words).
- Small behavioral evaluation subset - 7 projects only; conclusions are suggestive, not definitive.

Future Work

Conduct a **developer-centered study** to understand why prompt changes are rarely documented.

- Use the paper's dataset to identify and contact maintainers of LLM-integrated repositories.
- Combine surveys + semi-structured interviews to uncover real-world barriers (e.g., time pressure, lack of standards or tooling).
- Prototype lightweight tools such as an IDE plugin or Git commit template that detect prompt changes and encourage short documentation of intent or expected behavior.
- Goal: Make prompt maintenance more transparent and reliable in real-world LLM-integrated software.

Rating

4/5

- Timely and methodical first look at prompt evolution.
- But, limited in scope and behavioral validation.

Discussion Points

- Should prompts be version-controlled and reviewed like code?
  - Versioning, testing, linting?
- Can we measure "good" prompt evolution empirically?
  - Is it shorter, clearer, or simply more effective?
- How can we mitigate unpredictable effects of prompt changes?
  - Is the unpredictability inherent or is it due to prompt construction choices?

Thank you for listening and participating!