CSC 510 SOFTWARE ENGINEERING

PROJ 1D1

Group 17 Team Members:

1. **Pradyumna Chacham,pchacha2**
2. **Sai Mahathi Suryadevara, ssuryad6**
3. **Sai Sumedh Kaveti, skaveti**
4. **Sadana Ragoor, sragoor**

REFLECTION DOCUMENT

LLMS Used:

1. ChatGPT

2. Claude

3. NotebookLM - 1b1 and 1c1 - RAG equivalent

Pain Points in Using LLMs

Working with ChatGPT, NotebookLM, and Claude across these projects taught us that consistency is the biggest challenge. Zero-shot prompts almost always produced vague outputs that missed stakeholder details and realistic flows. Context window limitations hit us especially hard with Claude - when we tried to feed in all 30 use cases from project 1b1, it couldn't handle the load and we had to fragment our work into smaller chunks.

Another major frustration was scope creep by AI. The models, particularly Claude, kept suggesting futuristic but impractical ideas like blockchain integration for our food delivery app. These suggestions sounded impressive but were completely irrelevant to actually getting someone a sandwich. All three models also lacked strategic judgment - they could expand requirements endlessly but couldn't make the hard decisions about what mattered most for an MVP.

Surprises

We were genuinely surprised by how differently each model approached the same tasks. NotebookLM was methodical and practical but struggled with larger data loads and felt less engaging. Claude was creative and holistic but often over-engineered solutions, suggesting flashy but irrelevant features. ChatGPT balanced coverage and creativity but sometimes became verbose and repetitive.

The transformation through pre and post-processing completely caught us off guard. Raw outputs were rarely submission-ready, but with proper templates, examples, and filtering, the results became genuinely workable. The models also helped us catch blind spots we didn't know we had - like initially missing delivery drivers as stakeholders in our food delivery system.

What Worked Best

Template-driven prompting was the breakthrough. Providing exact structure with preconditions, main flow, subflows, and alternative flows created consistent, well-formatted use cases that followed software engineering conventions. Few-shot examples made an enormous difference - showing models concrete examples from our lectures helped them understand what quality work looked like.

Role-based prompting became my go-to strategy. Instead of asking "what can this system do," We learned to ask "what would a customer need? What challenges would restaurant staff face?" This approach ensured broader stakeholder coverage and generated more realistic, user-centered requirements. Filtering outputs through our MVP lens using make-or-break and core value tests helped refine chaotic suggestions into focused, meaningful features.

We also had good response from the LLMs when we prompted along different lines, basically asking LLMs what are the existing flaws in the use-cases, etc., and then shoring up those points to make sure that we covered a wider range of viable use cases.

What Worked Worst

Zero-shot prompts were consistently disappointing. Asking open-ended questions like "list 30 use cases" without structure produced generic, repetitive lists that were essentially useless. Unbounded brainstorming sessions just overwhelmed me with noisy, unrealistic suggestions where models confused quantity with quality.

Large single prompts consistently overloaded the systems, especially Claude's smaller context window. Trying to get fully polished, submission-ready reports in one go yielded verbose but shallow content that lacked the depth needed for our assignments. Most frustratingly, none of the models could handle prioritization and trade-offs - when we needed to cut from 30 use cases to 10 for our MVP, they were completely useless at making strategic decisions.

Pre- and Post-Processing Strategies

Success required significant preparation work. We learned to break complex prompts into smaller, manageable tasks and anchor every request with examples from lectures or earlier assignments. The models needed concrete patterns to follow rather than expecting them to understand software engineering conventions intuitively. We split our task into first focusing on 10 use cases at a time, and then asking the LLMs for 10 more successively, asking it explicitly each time to remember previous 10 use cases to retain context better.

Post-processing was equally time-consuming but essential. We had to merge duplicate suggestions, filter everything against our critical dependency tests, and categorize outputs into core versus supporting features. Cross-referencing outputs from multiple models became valuable for filling gaps - NotebookLM might miss creative angles while Claude suggested impractical features, but together they provided comprehensive coverage.

Best/Worst Prompting Strategies

The most effective strategy was few-shot prompting with concrete examples. Rather than expecting AI to understand what we wanted from descriptions, we showed them exactly what good use cases looked like from our course materials. Role-based and scenario-based prompts grounded in real system context worked much better than abstract requests. The "what could go wrong?" approach became particularly valuable for uncovering edge cases and alternative flows.

The worst strategies were overly open-ended requests without structure or boundaries. Expecting LLMs to handle condensation or prioritization tasks was consistently disappointing - they're much better at expansion than strategic decision-making. Finally, expecting polished, submission-ready answers without significant human filtering was a recipe for disappointment across all models.

Through this experience, we came to see LLMs as powerful creative assistants rather than autonomous requirements engineers. They excel at helping humans think through complex systems and generate possibilities, but they need human judgment for strategic decisions, prioritization, and quality control.