Draft Reflections

Ullman (2017, p. 164) identifies six key levels of reflection ranging from the lowest level, the mid-level , to the deepest level:
i) Recount/Description of an experience: 'This category captures the subject matter of the reflective writing'

ii) React /Feelings: 'Often, the feeling of being concerned, having doubts, feeling uncertain about something, or frustration are reasons for a reflective thought process. However, feelings such as surprise or excitement [may] also [be] mentioned'

iii) Personal beliefs: Reflection is often personal. 'This is about one's assumptions, beliefs, the development of a personal perspective, and the knowledge of self'

iv) Recognising difficulties: 'Expressing an alert, critical mindset is an important part of reflective writing. A critical stance involves being aware of problems and being able to identify or diagnose such problems'

v) Perspective: Relate/research: 'The writer considers other perspectives. For example, the perspectives of someone else, theory, the social, historical, ethical, moral, or political context'

vi) Lessons learned and future intentions: 'Retrospective outcomes: Descriptions of lessons learned, better understanding of the situation or context, new insights, a change of perspective or behaviour, and awareness about one's way of thinking. Prospective outcomes: an intention to do something, and planning for the future'

Principles of sustainability to create viable systems

My four-month internship at EdFlow.ai I was able to full grasp the concept around sustainability in software development. At first, I thought sustainability was just a technical concern, reducing load times, optimising API calls, and writing clean code. However, through hands-on experience and mentorship, I realised that true sustainability in tech encompasses environmental, economic, and operational longevity.

One of my earliest tasks involved optimising an ai chatbot used by educators and students. The initial implementation suffered from performance bottlenecks, excessive re-rendering in React and redundant API polling. My first though was to apply quick fixes, like memoisation and request throttling. However, my senior developers pushed me to think beyond immediate performance gains and consider long-term maintainability. We refactored the component into smaller, reusable hooks and adopted a state management strategy that minimised unnecessary data fetching. This not only improved efficiency but also made the system easier to debug and extend in the future.

Beyond the code itself, I observed how team practices influenced sustainability. Agile methodologies, when poorly managed, can lead to technical debt accumulation and rushed features, undocumented

workarounds, and fragile integrations. At EdFlow, we mitigated this by implementing clean CI/CD procedures. This practice ensured that short-term deadlines didn't compromise long-term viability. I began to see the great impact of modular design and continuous iteration preventing systemic obsolescence.

The most profound lesson came when I was taught how to migrate UI styling from custom CSS to Tailwind. Initially, I was unsure and naïve to what tailwind even was. To better learn I did some research for the benefits, comparing:

- Development speed (Tailwind's utility-first approach reduced styling conflicts)
- Onboarding efficiency (new hires could contribute faster without learning proprietary CSS)
- Performance gains (smaller bundle sizes due to purged unused styles)

The learning process took me two weeks but reduced my styling-related bugs by a highly significant amount and accelerated future feature development. This experience taught me that sustainability often requires upfront investment, a principle that applies not just to code, but to business decisions, team structures, and even hardware choices.

Looking ahead, I now prioritise "architectural sustainability", designing systems that remain adaptable as requirements evolve. Whether through microservices, reusable components, or automated testing through playwright, I aim to build software that doesn't just function today, but works years later.

Professional Practice within intercultural and global contexts.

My four-month internship at EdFlow.ai allowed me to fully grasp the complexities of working in a global, intercultural tech team. I used to think that effective collaboration was just about clear communication and technical skills. However, through hands-on experience, I realised that cultural intelligence, adaptability, and empathy are equally critical to success in a distributed work environment.

One of my earliest projects involved collaborating with a frontend developer in the Philippines to build an admin dashboard for educators/teachers. During a code review, I provided direct feedback: "This implementation won't scale; it needs a rewrite." While I intended to be efficient, my Philippines-based colleague later explained that my phrasing felt abrupt and demotivating. In his work culture, feedback is often framed more diplomatically, such as:

- "Could we explore a more scalable approach here?"
- "I see what you're aiming for, but I reckon we can refine this together."

This was a pivotal moment. I learned that communication styles vary widely across cultures, and what's considered "productive" in Sydney might be perceived as abrasive elsewhere. To adapt, I began:

- Softening critiques by pairing suggestions with acknowledgments ("Great effort on ____. Let's optimise the logic for _____.").
- Using asynchronous tools (Loom recordings, detailed Jira comments) to bridge time zones and language barriers.
- We also began doing weekly scrum meetings including all developers from Australia, and out of Australia.

Initially, I was tasked with learning some various skills to be up to date with the projects requirements. At first, I was mentored through a task-oriented approach, being assigned tickets with minimal guidance, assuming independence. However, my I rarely asked questions, leading to delays. After a

1:1 chat, I was able to discuss my needs and requirements. As a junior, I often waited for explicit permission to speak up or challenge ideas, and had my mentors change their style by:

1. Holding weekly lunch chats to discuss not just work, but career goals and challenges.
2. Structuring feedback with clear examples (screen-sharing while explaining code reviews).
3. Building a sense of friendship and connection through playing games such as table tennis.

Within weeks, my confidence and productivity improved dramatically. This taught me to better communicate about my issues, rather than staying silent in times of uncertainty.

Broader Lessons and Future Applications

This experience mirrored large global team strategies, where inclusivity and adaptability are prioritised. Key takeaways:

- Efficiency isn't just speed, it's ensuring everyone is understood and motivated.
- Psychological safety accelerates productivity. Small adjustments (written summaries after meetings) can prevent misunderstandings.
- Global careers require active learning. I have looked into frameworks like Hofstede's cultural dimensions to anticipate differences.

Looking ahead, I'll prioritise:

- Seeking roles in global teams to deepen my intercultural fluency.
- Advocating for bonding times such as team lunches, extra-curricular bonding time.
- Documenting processes visually through means such as videos to overcome linguistic barriers.

This internship didn't just improve my coding skills, it taught me how to thrive in a borderless tech industry. Just as sustainable code requires foresight, effective collaboration demands wilfulness, humility, and a willingness to adapt. I now see these not as soft skills, but as core skills for contemporary engineers.