**Improving FocusBuddy: From Script to Smart Assistant**

**Introduction**

In the first two blogs, I shared why I built FocusBuddy and how it works as a terminal-based assistant for managing focus sessions. The tool is simple: it prompts a task, starts a 25-minute Pomodoro session, logs the data, and reminds you to take a break. But once I had a working version, I didn’t stop there.

Improvement is part of every good developer’s journey. FocusBuddy started as a basic script but quickly became a sandbox for exploring new features, experimenting with ADHD-friendly interfaces, and asking one key question: *how can this tool make my life even easier?*

In this post, I’ll walk through the ways I improved FocusBuddy after the initial build—from technical upgrades to design refinements—and share the lessons I learned along the way.

**Reassessing the Core Experience**

The first version of FocusBuddy worked well, but it lacked flexibility. It always ran 25-minute sessions and 5-minute breaks. It didn’t track task priorities or let me review session history easily. Worst of all, if I accidentally closed the terminal mid-session, everything was lost.

So I began rewriting and improving it incrementally. Each change made the tool smarter, more forgiving, and more usable.

**Improvement #1: Configurable Session Lengths**

Different tasks need different focus durations. Sometimes I only have 15 minutes. Other times, I want to do a full 50-minute deep work block.

I added environment variables that let users override the defaults:

FOCUS\_MINUTES=50

BREAK\_MINUTES=10

Then in the script:

focus\_duration=$((FOCUS\_MINUTES \* 60))

break\_duration=$((BREAK\_MINUTES \* 60))

Now I can run:

FOCUS\_MINUTES=15 ./focusbuddy.sh

and get a shorter cycle.

**Improvement #2: Interrupt Handling (Ctrl+C Safety)**

Accidentally hitting Ctrl+C used to cancel everything, losing all session data. That was frustrating.

To fix this, I used Bash traps to catch termination signals:

trap ctrl\_c INT

function ctrl\_c() {

echo "\nFocus session interrupted. Logging partial session..."

log\_session "$task" "$start\_time" "INCOMPLETE"

exit 1

}

Now even if I quit early, the session is logged as incomplete. This helped reinforce that *some effort still counts*—important for ADHD users who struggle with perfectionism.

**Improvement #3: Task Priority and Tagging**

To better organize my work, I introduced simple task tagging:

read -p "Task (e.g. [urgent] Finish report): " task

Later, I could filter logs by tag:

grep "\[urgent\]" sessions.csv

This added a layer of planning while still keeping the interface simple.

**Improvement #4: Visual Countdown**

Originally, the session ran silently in the background. But I often found myself wondering how much time was left.

So I added a visual countdown timer using while loops and tput:

for (( i=$focus\_duration; i>0; i-- )); do

printf "\rTime left: %02d:%02d" $((i/60)) $((i%60))

sleep 1

done

This small addition made a big difference in motivation. I could see the time ticking, like a digital hourglass.

**Improvement #5: Daily Summary Log**

I created a focus\_log/YYYY-MM-DD.txt file that stores each day’s tasks. This lets me quickly review what I did without parsing CSVs:

echo "[$(date +%H:%M)] $task ($duration mins)" >> focus\_log/$(date +%F).txt

At the end of the week, I could look at all my files and see how many sessions I completed and what themes emerged.

**Improvement #6: GUI Version (Experimental)**

Not everyone likes the terminal. I started working on a lightweight GUI version using Zenity:

zenity --entry --text="What will you focus on?"

This allowed me to:

* Add pop-up prompts
* Show timer progress bars
* Give break reminders visually

**Improvement #7: Mood Tracker**

Since ADHD often involves emotional dysregulation, I wanted to track my mood across sessions. I added this prompt:

read -p "Mood (happy, tired, anxious, etc): " mood

And appended it to the session log:

echo "$task,$start\_time,$end\_time,$duration,$mood" >> sessions.csv

Over time, I could begin to correlate mood with productivity. This helped me understand how sleep, nutrition, and stress were affecting my focus.

**Improvement #8: Break Reminders with Self-Care Prompts**

During breaks, FocusBuddy now gives random suggestions:

breaks=("Stretch your legs" "Drink water" "Look outside" "Breathe deeply")

message=${breaks[$RANDOM % ${#breaks[@]}]}

echo "$message"

This made breaks feel intentional, not just idle time.

**What I Learned From Iterating**

Improving FocusBuddy taught me a lot about development, design, and my own mind:

* **User experience is everything**: Technical accuracy isn’t enough—tools must *feel* good to use.
* **Small details matter**: Adding timers, summaries, or color made the tool more useful and friendly.
* **Iterate continuously**: I didn’t build this all at once. Each feature came from real-world frustration.
* **Track impact**: Logging data gave me motivation and proof of progress.

Every improvement was inspired by a pain point—something that annoyed me, confused me, or made me avoid using the tool. Fixing those made FocusBuddy more empowering.

**Future Ideas**

I’m still excited about features I haven’t built yet:

* **Pomodoro streak tracking**
* **Cloud sync for logs**
* **Auto-suggest tasks based on history**
* **Graphs of productivity over time**
* **Integrate with mindfulness prompts or music APIs**

**Closing Thoughts**

Improving FocusBuddy was more than a coding exercise. It was a journey in self-awareness, problem-solving, and accessibility. It taught me that automation isn’t just about writing less code—it’s about *removing mental roadblocks*.

Every time I made the script easier to use, I made my life easier. Every time I logged a session, I reminded myself that I was capable of staying focused—even if only for 25 minutes at a time.

In the next blog, I’ll reflect on the full journey: what I’ve learned about ADHD, automation, and designing tools that truly serve the people who use them.

Thanks for reading. And remember—your tools should work *for* your brain, not against it.