**Windsurf Report**

**Introduction:**

Windsurf is an AI powered app builder it provides development environment that can be imported from **VS code or cursor** and it can also be add as a plugin in **VS code, Jet Brains, Neovim, Visual Studio, Vim, Jupiter, Chrome, Eclipse and Xcode.** It is designed to accelerate workflow of teams and individual’s application. Windsurf integrates agentic coding workflows, context memory, and intelligent automation to help developers move from idea to a working application. It **currently supports over 70 programming languages**, placing it among the most inclusive AI-driven IDEs available today.

**Features of Windsurf:**

**Cascade Flows for Multi-File Edits and Agentic Coding:**

Cascade is Windsurf’s “engine room.” It doesn’t just edit one file at a time it can reason across multiple files, coordinate dependencies, and even update configuration files automatically. Think of it as an AI workflow layer that plans changes step by step, applies them intelligently, and keeps the whole project in sync.

**Memories and Rules:**

In Windsurf, **Memories and Rules** work hand in hand to make the AI feel less like a disposable assistant and more like a persistent teammate. Memories allow the system to retain project context, coding preferences, and past decisions across sessions, so you don’t have to repeat instructions every time you return to a task. Rules act as explicit guardrails such as “always use async SQLAlchemy” or “prefer functional React components” that guide how code is generated, refactored, and maintained. Together, they create a development flow where the AI consistently adapts to your style and team standards, ensuring that suggestions remain aligned with long-term goals and project conventions.

**Inline AI for Precise Refactoring and Docstring Generation:**

Instead of broad, sweeping changes, Inline AI works like a scalpel. You can highlight a section of code and ask for a docstring, refactor to async/await, or optimize a loop. The changes apply exactly where you need them, reducing the risk of unintended edits elsewhere in the codebase.

**Dead Code Cleanup and Auto-linter Integration:**

Windsurf helps keep your codebase lean. It scans for unused imports, redundant files, and outdated variables, removing them automatically. Coupled with auto-linting, it ensures your code follows best practices, fixing style and formatting issues without you having to touch a linter manually.

**Tab-to-Import and Smart Suggestions for Faster Development:**

When typing new functions or classes, windsurf automatically suggests the necessary imports. Pressing Tab pulls in the right dependency instantly. This eliminates the need to manually search and import libraries, and also reduces the chance of import errors.

**Image-to-Code and Live Web Preview for Design Integration:**

Front-end developers get a big boost here. You can drop in a screenshot, wireframe, or design mockup, and Windsurf generates the corresponding HTML/CSS/JavaScript structure. The live preview feature lets you see changes as you go, with clickable UI elements that can be directly edited via AI.

**Built-in AI Terminal and Deployment Tools:**

Windsurf isn’t just about writing code it can suggest and run terminal commands too. From installing dependencies to running migrations or starting servers, the AI terminal reduces context-switching. The built-in deployment pipeline lets you push and preview apps quickly, often with a single click.

**Planning Mode:**

Windsurf is designed to handle complex development tasks by breaking them down into structured steps before writing any code. Instead of immediately generating files or edits, the AI first creates a high-level plan outlining what changes are needed, which files will be touched, and how the pieces fit together. This helps developers review, refine, and adjust the approach before implementation begins. Planning Mode is especially useful for multi-file features, architectural updates, or workflows that could otherwise become messy if tackled directly. By separating thinking from doing, windsurf ensures that development remains transparent, predictable, and easy to align with team goals.

**Feature that worked well:**

**Memories and Rules:**

Memories and Rules is windsurf’s powerfull weapon when used well it can build some pretty good application that has potential to become a production ready application. Windsurf uses context awareness it knows what we are trying to implement by adding memories and rules to it provides a bonus in development.

**Tab to import and Smart Suggestions for Faster Development:**

Tab to import is consistent in finding and import the packages that are needed. Likewise, smart suggestions also use **tab to jump** for recommendation like giving suggestions for implementing hooks, component usage, endpoint suggestions with basic function, fill in the middle of a function and can anticipate your next cursor position. By using context awareness, it consistent most of the time.

**Auto-linter Integration:**

When windsurfer generates code with error it marks the file so it can revisit the file again fix it according to the current codebase.

**Plaining Mode:**

Plaining mode give the developer upper hand in understanding what cascade (AI-powered agent system) is trying to do. It provides a work plan or to-do list for implementing a function. It gives a good to-do for migrating from one framework to another framework.

**Issues Faced:**

Most of my issues I faced are around **Model’s Context Size**, **Not Following Rules** **and Memories**, **False Completion** and **Terminal Command Execution**.

**Model’s context size:**

Windsurf is semi-good at remembering what it is doing, when creating a production level application, it needs to remember about most of the applications functionality even if application is divided into smaller parts. When lower reasoning models are used context size shrinks so it sometime drops the rules and memories we provide to it and doesn’t follow it. If session is larger it overwrites the rules and memories.

**Not Following Rules and Memories:**

These are the likely reasons that may cause windsurf to not follow rules and memories

**Context capacity shrinks:**

The smaller model can’t “hold” as much history in its working memory. That means your rules + memories + current files + instructions may not all fit, so some drop out or get compressed.

**Weaker adherence to system priorities:**

High-end models usually weigh instructions like “follow rules/memories strictly” more strongly. Lower models sometimes treat them more like suggestions than constraints.

**Serialization trade-offs:**

Windsurf has to decide what to stuff into the limited context window: recent edits, Cascade steps, or your persistent rules. Depending on that ordering, rules/memories might get deprioritized.

**Interpretation ability:**

Even if the rules make it in, a lighter model might not apply them consistently (e.g., missing subtle style rules or cross-file architectural guidelines).

**False Completion:**

These are the likely reasons that may cause windsurf to give false completion issue

**Context-limited awareness:**

It usually only “sees” the open files, the last Cascade step, and whatever it can load within the model’s token budget. Once it finishes a chunk (say, a module), it assumes the migration goal is satisfied because it has no certainty about the rest of the codebase unless you keep feeding it.

**Completion heuristics:**

Some models err on the side of saying “done” once the local task looks complete. If it successfully rewrites one module to Angular syntax, it interprets that as “task achieved,” unless explicitly reminded that the entire project still needs migration.

**Migration ambiguity:**

React to Angular isn’t a 1-to-1 mapping. There’s routing, services, DI, module structure, etc. If the model doesn’t “see” those pieces in its current context, it assumes nothing else is pending.

**Terminal Command Execution:**

It hallucinates when executing commands like

* Trying to execute command wrong directory.
* Trying to execute terminal commands that doesn’t work in PowerShell.
* Gives continues command without waiting for the result or before old command execution is finished.
* Based on the model terminal command execution is differs, more intelligent model understand the error and modify the command.
* Gets stuck when executing terminal command and have to stop the whole execution.
* Creates a runner files to run the main file when its command fails.

**What I liked most about the product:**

Windsurf is a solid AI powered app builder for developer, the thing I really liked about the product are **Credit Consuming Rate**, **Memories/Rules** and **Importing from Code Editor/IDE**.

**Credit Consuming Rate:**

Credits in windsurf is based on the no of prompts that developer can use and they also give a free pro trail for 14 days with 100 credits to a new user. But one slight downside is some premium model (Claude sonnet 4, Claude Opus 4.1) require from 2 to 20 credit for a prompt.

**Memories/Rules:**

Like I said previously memories and rules are powerful, I developed my backend (python with fastapi) with a single prompt with the help of memories and rules.

**Importing from Code Editor/IDE:**

Unlike other AI app builders windsurf can be installed locally, it can be used like a regular vs code with all its extensions too. After my credit got exhausted I can still open and develop like a regular code editor.

**Ease of adding new features:**

Adding new features are really easy it can understand and modify the necessary codes to implement. It uses auto-linter to resolve if an error occurs while adding new feature. Unless it is a full migration from a one framework to another framework or one component library to another component library it gets confused and mess up the code.

**How it responds to natural language queries:**

Windsurf is good at understand natural language in paragraph and in points too based on what we provide and what we are try to develop it create a to-do to proceed with implementation and strong in step by step instructions like add components, generate CRUD endpoints. But it requires extra clarification for complex architecture.

**Issues found and how you resolved them:**

**Model’s context size:**

Mention the file what I want to update mention it relevant memories if it has any and specify the functionality clearly in prompt.

**Not Following Rules and Memories:**

Whenever moving to new feature use a prompt to re-remember the rules and memories related the new feature I am going to implement.

**False Completion:**

Clearly inspect current implementation and specify what needed to update in prompt.

**Terminal Command Execution:**

Commands can be edited before executing so I edit the command before executing in the terminal.

**Conclusion:**

Windsurf shows real promise as an AI-powered app builder that blends coding assistance, planning, and automation into a single development environment. Its strengths lie in persistent **Memories and Rules**, smart coding aids like **Tab-to-Import**, and a developer-friendly setup that integrates seamlessly with existing editors. While issues such as limited context size, rule adherence gaps, false completion, and inconsistent terminal command execution highlight areas where it still needs refinement, most of these can be worked around with careful prompting and oversight. For everyday development tasks, windsurf speeds up workflows, reduces repetitive effort, and provides a structured path from idea to implementation. With further improvements to model consistency and terminal reliability, it has the potential to become a truly indispensable tool for building production-ready applications.