Flow-State: Developer Well-being Plugin

Konstantina Anastasiadou
Zofia Rogacka-Trojak
Amy van der Meijden
Andriana Tzanidou
Jimmy Oei
Group 13
Delft University of Technology

ABSTRACT

Software engineering is inherently cognitively demanding, frequently pushing developers towards burnout through silent mental fatigue, a burden that is ultimately transferred onto teammates through complex code reviews. Existing developer productivity tools typically focus either on static code analysis or on tracking developer activity. However, no unified IDE solution currently addresses both the developer's individual cognitive load and the resulting burden placed on reviewers. In this paper, we present *Flow-State*, a VS Code IDE plugin designed to enhance developer productivity and flow, reduce cognitive load, and promote well-being. Our design draws inspiration from the SPACE and DevEx frameworks. The plugin consists of various features for monitoring individual cognitive load, such as the add-to-delete ratio and large code insertions. It additionally helps developers sustain their focus and prevent burnout by including features such as a Pomodoro timer and real-time feedback alerts. It offloads reviewer cognitive load by analyzing Pull Request characteristics such as lines of code, cognitive complexity, and unused dependencies before code is committed. We evaluate our proposed solution through Scenario-Based Design and Evaluation, by introducing realistic developer personas. Our results demonstrate that continuous background behavioral tracking can effectively diagnose and mitigate cognitive friction and, at the same time, tackle team-wide burnout.

1 INTRODUCTION

Software engineering (SE) is a cognitively demanding field that can lead to burnout, a work-related syndrome typically characterized by emotional exhaustion, depersonalization, and a diminished sense of personal accomplishment [1]. Several studies in the field have explored the causes and consequences of burnout and how it affects the productivity of SE professionals.

At the individual level, high cognitive load caused by reading software artifacts or executing cognitive processing tasks can affect developer productivity and software quality [2]. Such high cognitive demands on professionals may contribute to burnout symptoms [3]. At the social level, development practices such as submitting large, complex Pull Requests (PRs) transfer cognitive load to teammates. As a result, reviewers experience fatigue, code quality deteriorates, and collaboration becomes inefficient [4]. We want to protect developers from cognitive overload and prevent secondary cognitive load for teammates, promoting both individual and social sustainability in SE.

To achieve this, we propose *"Flow-State: Developer Well-being Plugin"*, a Visual Studio Code (VS Code) [5] IDE plugin designed to enhance developer productivity and flow, reduce cognitive load, and promote well-being. At the individual level, the plugin tracks a developer's focus over time, measures cognitive load, and provides insights to help maintain flow states. Flow state is a key part of the SPACE [6] framework and refers to the ability of the developer to complete work with minimal interruptions, delays, or friction. When the flow state is being disrupted, the plugin suggests real-time breaks and offers strategies to reduce cognitive load. At the social level, the plugging minimizes the cognitive load for reviewers by helping developers identify PRs that are difficult to review before they are submitted. It tracks modified lines of code, complexity of changed files, and unused dependencies. These metrics are collected in a dashboard, allowing developers and teams to monitor productivity levels and improve collaboration.

In this paper, in Section 2, we first present the background that inspired the development of our tool. In Section 3, we describe *Flow-State's* design. Section 4 validates our design through user scenarios, representing different personas. Finally, Section 5 discusses limitations and directions for future work.

2 BACKGROUND

This section presents the key concepts and prior research that motivate the design of *Flow-State*. We first examine the cognitive load of code authors, followed by the challenges for code reviewers, existing standards and tools, and the research gap in current developer productivity tooling.

To comprehensively define and evaluate developer productivity across these areas, our work is grounded in both the *SPACE framework* [6] and the *Developer Experience (DevEx) framework* [7]. SPACE proposes a multidimensional approach that measures Satisfaction and Well-being, Performance, Activity, Communication and Collaboration, and Efficiency and Flow. DevEx builds upon these dimensions by emphasizing the lived experience of the developer, specifically identifying cognitive load, uninterrupted flow, and feedback loops as the primary drivers of productivity. By moving beyond simple output metrics, these frameworks establish that preserving a developer's cognitive state and team interactions are vital to individual and social sustainability.

Author's Cognitive Load. While the individual cognitive demands of software engineering are well-known, identifying when

a developer is overwhelmed requires passive monitoring. A large-scale field study on program comprehension highlights that developers spend a significant portion of their time navigating and reading code rather than typing, which can lead to cognitive fixation or "tunnel vision" when stuck [8]. Research indicates that a developer's cognitive state can be inferred through behavioral signals; for instance, interaction data such as keystroke dynamics, typing rhythms, and frequent code revisions serve as indicators of programmer stress, frustration, and cognitive overload [9, 10].

Furthermore, modern workflows introduce new stressors. The rising reliance on AI coding assistants leads developers to insert large blocks of generated code into their editors. Recent studies show that comprehending and verifying LLM-generated code introduces significant cognitive load, creating a profound "false sense of progress" [11]. In a 2025 study, developers using AI assistants took longer to complete tasks due to the effort of reviewing generated code, while still perceiving increased productivity [12]. This results in what has been termed "comprehension debt": a gap between written and understood code [13].

Reviewer's Burden and Pull Request Latency. When an author's cognitive overload results in bloated code, that burden is transferred to the reviewer. A Microsoft research study found that the primary challenge of modern code review is understanding the code rather than detecting defects [14]. When PRs become too large, reviewers experience fatigue and delay starting the review [15]. This directly impacts development workflows, as large PR sizes are a major cause of delayed merges and bottlenecks [16].

Existing Standards. To mitigate reviewer fatigue, both the software industry and academic literature have established quantitative heuristics. Best practices recommend reviewing no more than 400 lines of code per session to maintain effectiveness [17]. Beyond size, structural complexity is critical. Campbell introduced the *Cognitive Complexity* metric to measure code understandability, proposing thresholds to flag overly complex methods [18]. Consequently, developers are increasingly urged to empathize with their reviewers by extracting complex logic and keeping PRs small and readable [19].

Limitations of Current Tools and Research Gap. Despite these heuristics, developer productivity tooling remains fragmented. Static analysis tools such as *SonarLint*[1] can flag complexity but do not capture developer behavior, while telemetry tools such as *WakaTime*[2] monitor activity without assessing code quality. Platforms that analyze PRs like *GitHub PR Analytics*[3] or *Code Climate*[4] typically operate after code submission, within CI/CD pipelines. This reactive approach places the burden on reviewers, leading to inefficient feedback loops. Additionally, traditional time-management methods like the Pomodoro technique [20] operate independently of the developer's actual context, often relying on rigid alarms that inadvertently disrupt flow. Consequently, there remains a lack of a unified IDE tool that simultaneously tracks the author's cognitive state via behavioral tracking and predicts the reviewer's burden using pre-commit code metrics.

[1]https://www.sonarsource.com/products/sonarlint/
[2]https://wakatime.com/
[3]https://github.com/features/code-review
[4]https://codeclimate.com/

3 FLOW-STATE DESIGN

To address the gap described in Section 2, we developed *Flow-State*, a unified VS Code IDE plugin aimed at improving developer and reviewer productivity and well-being. *Flow-State* is designed as an advocate for the developer, rather than an instrument for workplace surveillance. All behavioral tracking and data processing occur locally on the developer's machine; no metrics are exported or stored on external servers. By guaranteeing complete data privacy, *Flow-State* fosters a safe, trust-based environment for self-reflection.

The plugin contains various features, and all can be enabled / disabled and customized from the settings of the IDE. Developers can manually adjust the sensitivity thresholds for all cognitive and behavioral alerts to suit their individual working styles. This design provides developers with control and flexibility over their workflow [21]. Each feature is designed to support one or more dimensions of the SPACE framework to capture developer productivity [6]. *Flow-State's* features are described below.

3.1 Cognitive Load

As mentioned in Section 2, cognitive load refers to the mental effort required to understand and work with information. *Flow-State* measures two types of cognitive load, one for developers and one for reviewers. The load for developers is about the effort it takes to create the code, while the load for reviewers is about reviewing and checking the code. The proposed features target the Efficiency and Flow dimension of the SPACE framework [6] by capturing the effort required to complete work with minimal interruptions, delays, or friction. They also indirectly support the Communication and Collaboration dimension by improving the code review process. Additionally, they indirectly capture Activity by providing behavioral signals and impact Performance [6] by reducing cognitive load and improving code quality.

The **cognitive load for developers** arises from understanding the code base, implementing changes, and ensuring its correctness. The load is assessed using the following metrics:

- **Read-to-Write Ratio:** Tracks reading activity inferred from scrolling behavior and writing activity detected through document changes (e.g., typing, editing). If it detects more reading than writing, then the developer is thinking intensely and deeply analyzing the code (high cognitive load). When the developer is actively scrolling and has not written any code for a predefined period of time, the system warns the developer by showing a warning in the status bar (Figure 1).
- **Add-to-Delete Ratio:** Tracks micro-churn (typing, deleting, and re-typing) to detect struggle. If it detects considerably more deletions than writes, it fires a temporary alert in the status bar (Figure 2).
- **Cognitive complexity:** Measured based on Campbell's guidelines [18]. It monitors the depth of nested loops and other complexity metrics while coding and assigns a complexity score. The score is incremented for each break in the linear flow of the code and when flow-breaking structures are nested. If the complexity score exceeds the set threshold

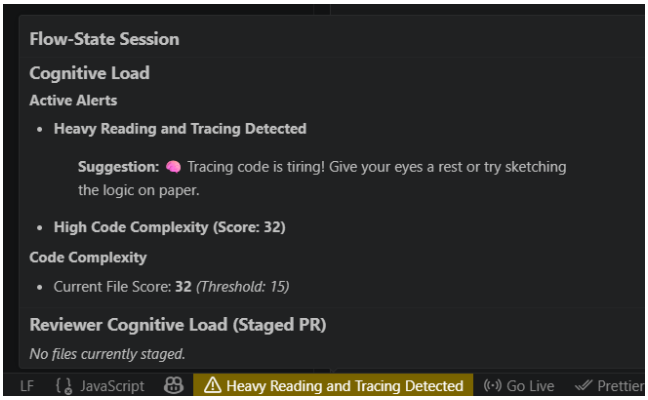

Figure 1: Flow-State Cognitive Load for Developer: Read-to-Write ratio warning.

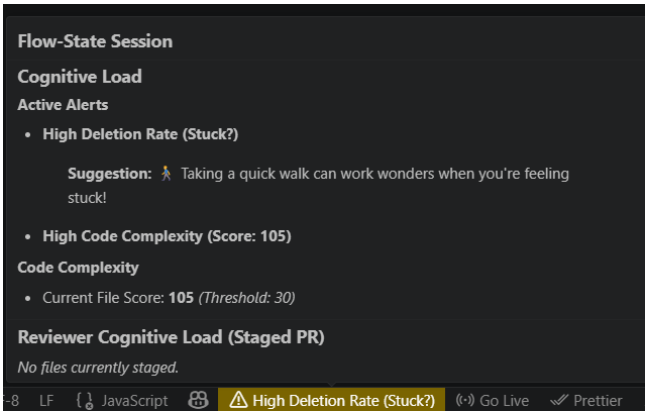

Figure 2: Flow-State Cognitive Load for Developer: Add-to-Delete ratio warning.

(e.g., 4+ nested loops), a temporary warning is displayed in the status bar (Figure 3). If complexity continues to rise, the alert is triggered again. When complexity is significantly reduced, an encouraging message is displayed on the status bar instead.

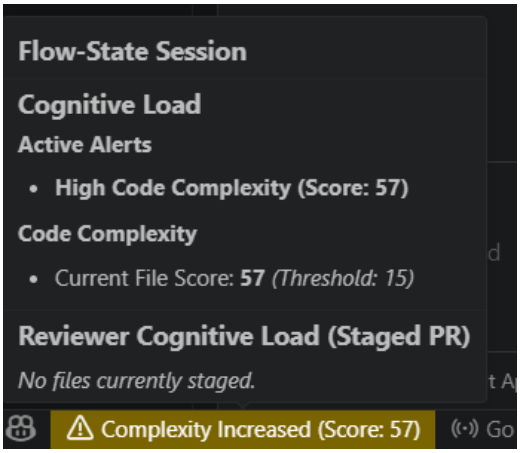

Figure 3: Flow-State Cognitive Load for Developer: Cognitive complexity warning.

- **Large Code Insertion:** Integrates the size of the last paste action from the activity tracker to detect high load when pasting big passages of code. When a paste of more than 600 characters occurs, it is classified as externally sourced code such as AI-generated code or code copied from online sources (e.g., Stack Overflow) and a temporary warning is shown in the status bar. This mechanism provides a way to measure the cognitive load associated with LLM-generated code (Figure 4).

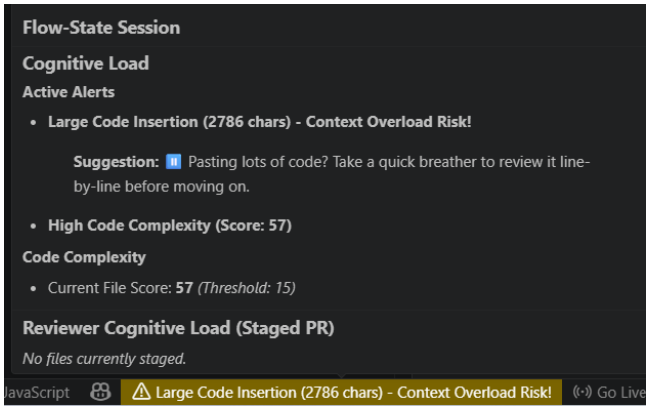

Figure 4: Flow-State Cognitive Load for Developer: Large Code Insertion.

The **cognitive load for reviewers** can be proactively assessed using the metrics described below. Rather than waiting until a PR is formally submitted, these metrics are computed dynamically every time a file gets staged.

- **Lines of Code Tracking:** Monitors all staged Git changes and filters out `.json`, `.lock`, and auto-generated files to capture the real footprint of changes. When more lines have been changed than a predefined threshold, a warning alert is shown in the status bar. This alert warns the developer that the code to be committed exceeds the scientific limit of human review capacity. The threshold is set at 400 lines of code (LOC) based on the SmartBear / Cisco study [22], which suggests that developers should review no more than 200–400 LOC at a time.
- **Complexity Analysis:** Detects the complexity of all staged files and throws a warning in the status bar when the complexity is above a predefined threshold. Complexity is measured based on Campbell [18]. High cognitive complexity might impact the cognitive effort of the reviewer to understand the code.
- **Zombie Dependencies:** Identifies dependencies listed in package.json that are installed but not actually used in the code of all staged/unstaged files. The extension recursively scans package.json files from the project root, analyzes imports in the source code, and flags any unused packages as zombie dependencies. Every time Git status changes, the extension re-verifies that every package listed in the package.json is still being used somewhere in the source code.

An overview of all Reviewer Cognitive metrics and associated warnings is illustrated in Figure 5.

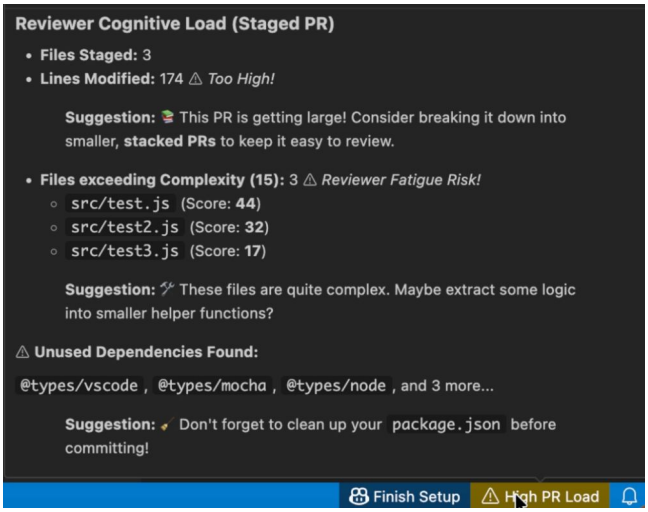

Figure 5: Flow-State status bar warning high cognitive load for reviewer.

3.2 Context Switch and Inactive Tabs

For tracking user focus and safeguarding developers' flow state, the plugin measures the frequency a developer switches between files as covered by the Efficiency and Flow dimension of SPACE [6]. When the frequency is high within a short period of time, a warning is displayed in the status bar (Figure 6). Switches to files in different folders are weighted more heavily, while switches within the same folder have a lower weight. In parallel, it monitors the number of inactive tabs a developer has open for an extended duration of time. If several tabs remain inactive beyond a predefined threshold, the user is notified by an alert in the status bar. Clicking the status bar opens a picker where the user can review inactive tabs and choose which ones to close (Figure 7). This feature helps developers keep their workplace focused and manageable and can thus positively influence Satisfaction and Well-being [6].

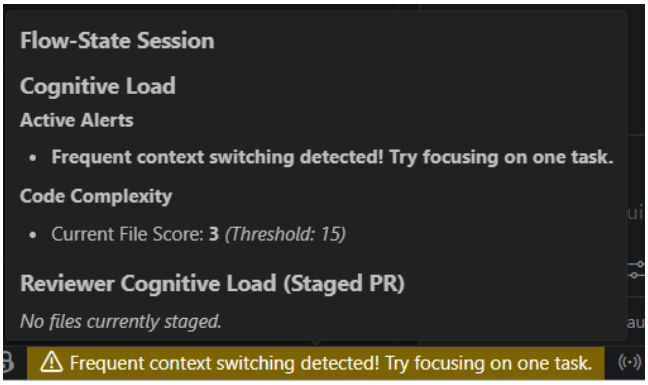

Figure 6: Flow-State high context switching warning.

3.3 Pomodoro Timer

A Pomodoro [20] timer (Figure 8) is implemented to help developers manage their focus by breaking work into dedicated intervals of deep-work (25 minutes), followed by short breaks (5 minutes) and long breaks (15 minutes). The developers can set the number of Pomodoro cycles and the period of short and long breaks, and

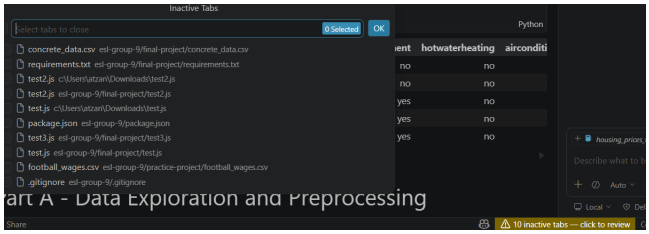

Figure 7: Flow-State inactive tabs warning.

start the timer. They can pause and resume at any moment. During breaks, the plugin provides suggestions to encourage healthy work habits and well-being, such as taking a coffee or stretching. Additionally, the timer **dynamically adjusts focus** duration based on the developer's cognitive complexity. If cognitive complexity is low, focus intervals are slightly extended to reward deep focus, but if code complexity is high, a proportional reduction is triggered to prevent burnout. The timer is accompanied by a to-do list where developers can plan, track, and organize their tasks during each focus session (Figure 9). These features directly target Efficiency and Flow as well as Satisfaction and Well-being [6].

Figure 8: Flow-State Pomodoro timer.

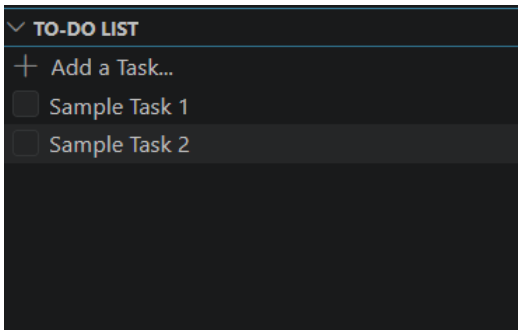

Figure 9: Flow-State to-do list.

3.4 Dashboard

The plugin is equipped with a dashboard that enables developers to track their performance. It displays statistics of all measures described above and any active warnings in real time. Within the SPACE framework [6], the dashboard captures Efficiency and Flow by enabling developers to detect disruptions in focus and cognitive load impact. It indirectly contributes to Activity by surfacing behavioral data. Additionally, it positively influences Satisfaction and Well-being by signaling burnout. An overview of the dashboard can be seen in Figure 10.

3.5 Status Bar

The status bar serves as the primary point of communication with the developer. It displays alerts in real time, and hovering over it

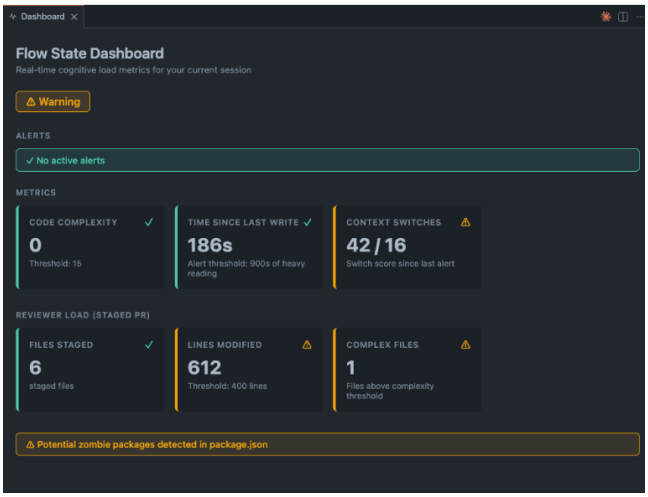

Figure 10: Flow-State dashboard displaying real-time developer performance statistics and active warnings.

reveals a tool tip summarizing active alerts, including the number of inactive tabs, a list of zombie dependencies, the current complexity score, and more. Whenever an alert is triggered, the status bar briefly flashes yellow to draw attention. Additionally, when the complexity score is low, the developer receives an encouraging message. The status bar is located at the bottom of the IDE (Figure 11), and clicking on it opens the dashboard for a more detailed overview. When no alerts are active, the status bar indicates that the system is in an optimal state.

3.6 Onboarding

When developers first install the plugin, they are presented with an onboarding window that explains the plugin and its features. During this walkthrough, they are guided through each feature and can choose to enable or disable any feature according to their preference (Figure 12).

3.7 Flow-State Implementation

Flow-State has been implemented as an extension for VS Code IDE. The repository containing the source code can be found **here**.

4 EVALUATION

Because *Flow-State* focuses on protecting developers' well-being rather than just tracking output, we need an evaluation method that captures the user's experience and context. To achieve this, we employ *Scenario-Based Design and Evaluation*, a methodology that provides a concrete narrative framework to set the stage for user reflection and action [23]. By guiding realistic developer personas through daily tasks, these scenarios allow us to evaluate how the tool assists users across different experience levels and working styles. Through these narratives, we demonstrate how *Flow-State* transforms abstract cognitive thresholds into friendly, actionable feedback that preserves both individual and social sustainability.

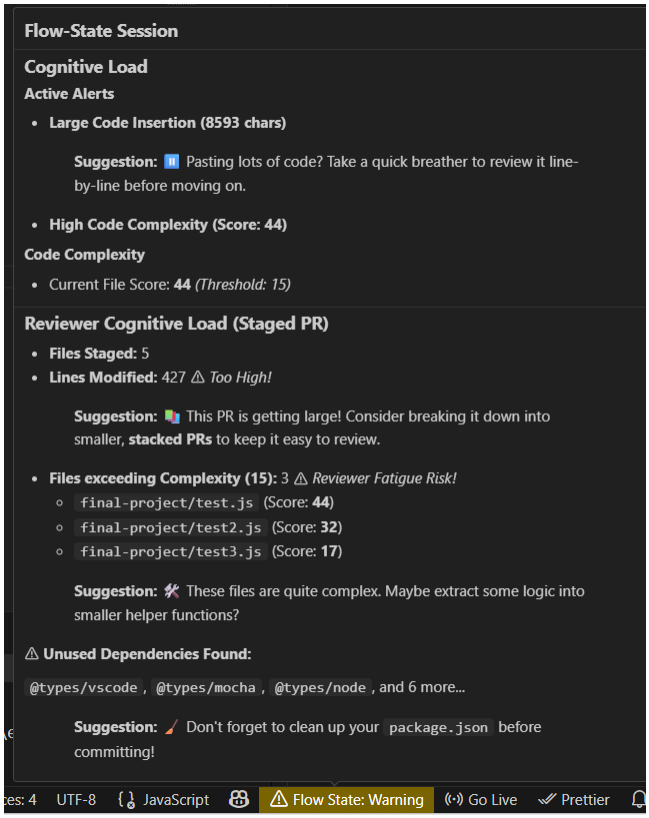

Figure 11: Flow-State status bar tooltip.

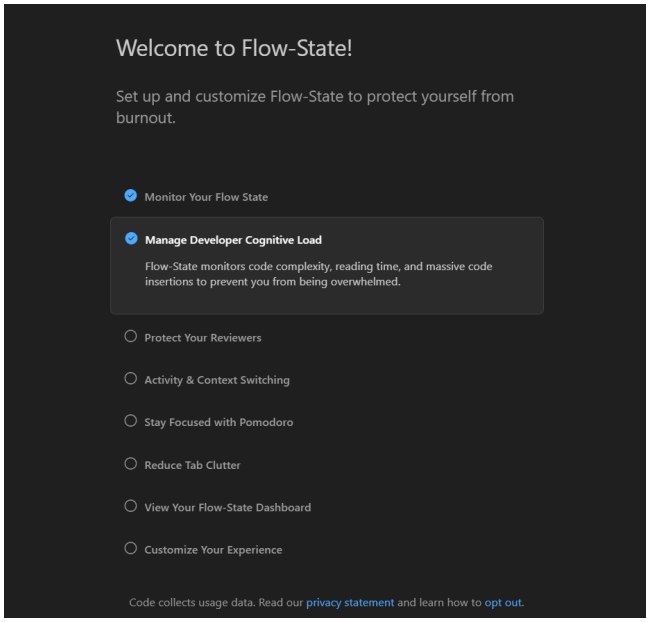

Figure 12: Flow-State onboarding window.

4.1 User Scenarios

We present three user scenarios to demonstrate how *Flow-State* works in realistic development contexts. These scenarios reflect

common challenges faced by developers across different experience levels and demonstrate how *Flow-State* provides support and actionable feedback.

4.1.1 Overcoming "Comprehension Debt" as a Junior Software Developer. Amy is a recently hired junior software developer in a large-scale software project. In a short period, she needs to juggle many responsibilities and tasks. Not only does she need to learn a new programming language, internal tooling, and how her team's code base works, she also has to start on her onboarding tasks. This steep learning curve is critical for her individual sustainability, which relies on maintaining her focus, skills, and well-being.

When she finally finds a part of the code base that she needs to change, she has no idea how to. She decides to copy a large block of AI-generated code to proceed quickly. However, as illustrated in Figure 13, *Flow-State's Developer Cognitive Load Tracker* detects a sudden insertion that exceeds the customizable `largeInsertionThresholdChars` limit (600 characters) and triggers a temporary yellow status bar `(warning) Large Code Insertion`, highlighting a high risk of context overload and comprehension debt [24]. Hovering over the status bar, Amy reviews the *Active Alerts* section and reads a friendly intervention: *"Suggestion: Pasting lots of code? Take a quick breather to review it line-by-line before moving on."*

Feeling the pressure to complete her onboarding task, Amy skips the advice to be methodical. Instead of a careful line-by-line review, she attempts to comprehend the entire massive block of code at once. After 15 minutes of frantically scrolling up and down the main file to mentally map the new logic, she is overwhelmed but lacks awareness of her own escalating cognitive fatigue, a major barrier to improving Developer Experience (DX) [25].

To overcome this lack of visibility, *Flow-State's Activity Tracker* calculates the time passed since her last write and recognizes her continuous, chaotic scrolling. It immediately fires a `(warning) Heavy Reading Detected` warning to indicate high cognitive load. This timely warning prompts Amy to check the *Active Alerts*, which suggests: *"Suggestion: Reading and checking code is tiring! Give your eyes a rest or try sketching the logic on paper."* Following this advice, she steps back, rests her eyes, and eventually breaks the code into smaller, more comprehensible functions that will be easier to test, rather than attempting to process the entire code at once. By intervening, the plugin directly supports the core DX dimensions of managing cognitive load and maintaining flow [7], preventing "tunnel vision" and preserving Amy's well-being.

4.1.2 Turning Procrastination into Effective Work. Andrea is a mid-level developer who often struggles with heavy context switching and task initiation, a common barrier to workflow continuity. However, once he overcomes this initial mental friction, he tends to "hyper-focus". Traditional time-management techniques, like strict 25-minute *Pomodoro* timers, often interrupt him precisely when he reaches peak cognitive engagement, aggressively disrupting his flow state. Frequent context switches and rigid interruptions have been shown to severely fragment focus time, directly reducing a developer's overall productivity and satisfaction [26].

To overcome his initial procrastination, Andrea starts a standard 25-minute focus session using *Flow-State's integrated Pomodoro timer*. As mapped out in Figure 14, after some minutes of warming up, he successfully enters deep focus, writing clean, functional code with a consistently low complexity score. Instead of a rigid alarm interrupting him 25 minutes later, the plugin's *ActivityTracker* detects his sustained typing rate and optimal cognitive load. Recognizing his high efficiency, the tool dynamically adapts by applying a 1.2x time multiplier. It silently displays a subtle `Good Focus! +5 min bonus` notification. This dynamic adaptation directly supports the *Efficiency and Flow* dimension of the SPACE framework [6], allowing him to capitalize on his momentum without artificial interruptions.

Later that afternoon, Andrea encounters a stubborn bug. While trying to trace the root cause across the architecture, he rapidly clicks through several different files across multiple folders in a short period. *Flow-State* detects this high-frequency navigation and triggers a `(warning) High Context Switching` alert, gently advising him to focus on one component at a time. Additionally, the plugin detects a graveyard of inactive tabs he left open during his search, prompting a quick cleanup to restore order to his workspace and minimize visual cognitive load.

Frustrated by the bug, he returns to the main file, where he repeatedly writes new logic and immediately deletes it. The plugin monitors this behavior, and once his activity falls below the customizable `addDeleteRatioThreshold` (0.3), it triggers a `(warning) High Deletion Rate (Feeling Stuck?)` warning in the status bar. Hovering over the alert, Andrea reads the friendly intervention: *"Suggestion: Taking a quick walk can work wonders when you're feeling stuck!"*

Caught in a moment of cognitive fixation, Andrea delays taking a break and attempts a brute-force workaround, heavily nesting loops and conditionals. This causes the file's cognitive complexity to spike above his configured `complexityThreshold`. *Flow-State* recognizes these compounding indicators of cognitive friction: a high deletion rate paired with sudden, severe code complexity. Concluding that the current focus session has become counterproductive, the dynamic *Pomodoro timer* automatically shortens his remaining time. By gracefully triggering his break early, the tool proactively intervenes to prevent burnout. This intervention explicitly safeguards the *Satisfaction and Well-being* dimension of the SPACE framework [6], actively encouraging him to step away and reset before the frustration compounds.

4.1.3 Preventing PR Reviewer Fatigue and the "Curse of Knowledge". Kona is a senior lead engineer finalizing a massive feature, after a week of intense, isolated deep work. While they have successfully built a deep understanding of the new architecture, they are vulnerable to the "curse of knowledge", forgetting how overwhelming this large, highly contextual change set will be for a peer reviewer seeing it for the first time [14].

Preparing to open a Pull Request, Kona stages their entire workspace. As visualized in Figure 15, *Flow-State's ReviewerTracker* instantly analyzes the staged diff and changes the status bar background to yellow, displaying a brief `(warning) High PR Load` alert.

Curious, Kona hovers over the status bar to view the *Reviewer Cognitive Load* dashboard. The tool provides three distinct, quantitative insights into the friction their PR will cause: First, it flags that their change-set heavily exceeds the configured `reviewerLocThreshold` of 400 lines, triggering a friendly intervention: *"Suggestion: This PR is getting large! Consider breaking it down into smaller,*

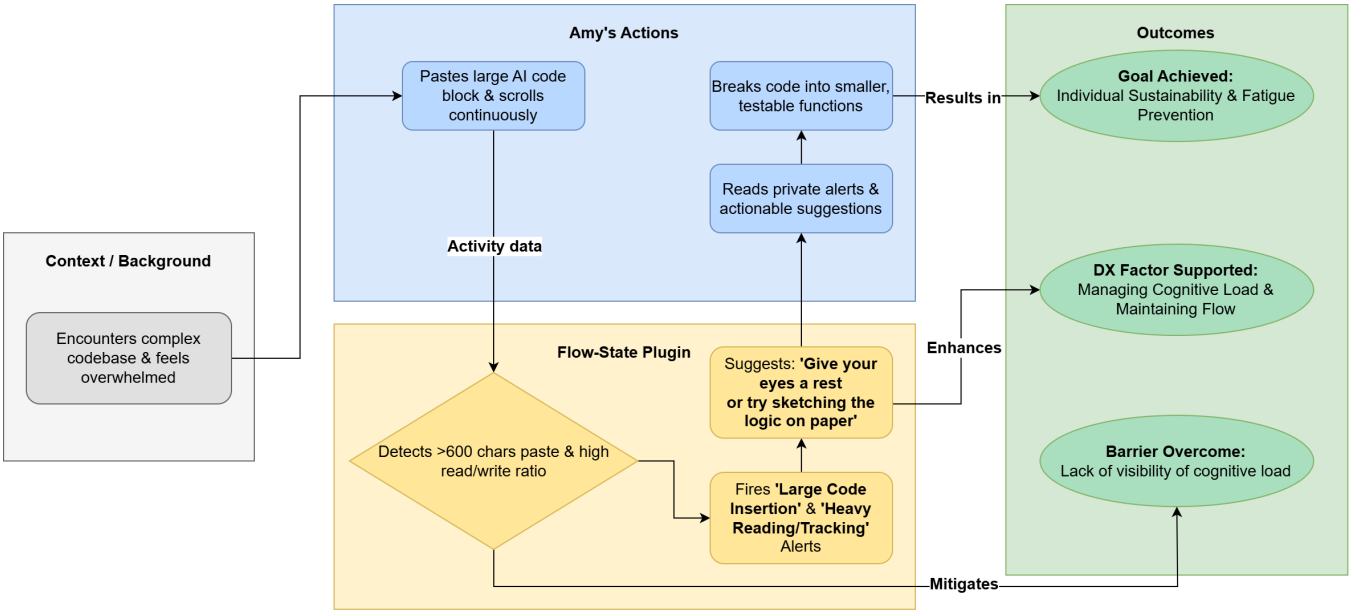

Figure 13: Flow representing interactions between Amy and Flow-State.

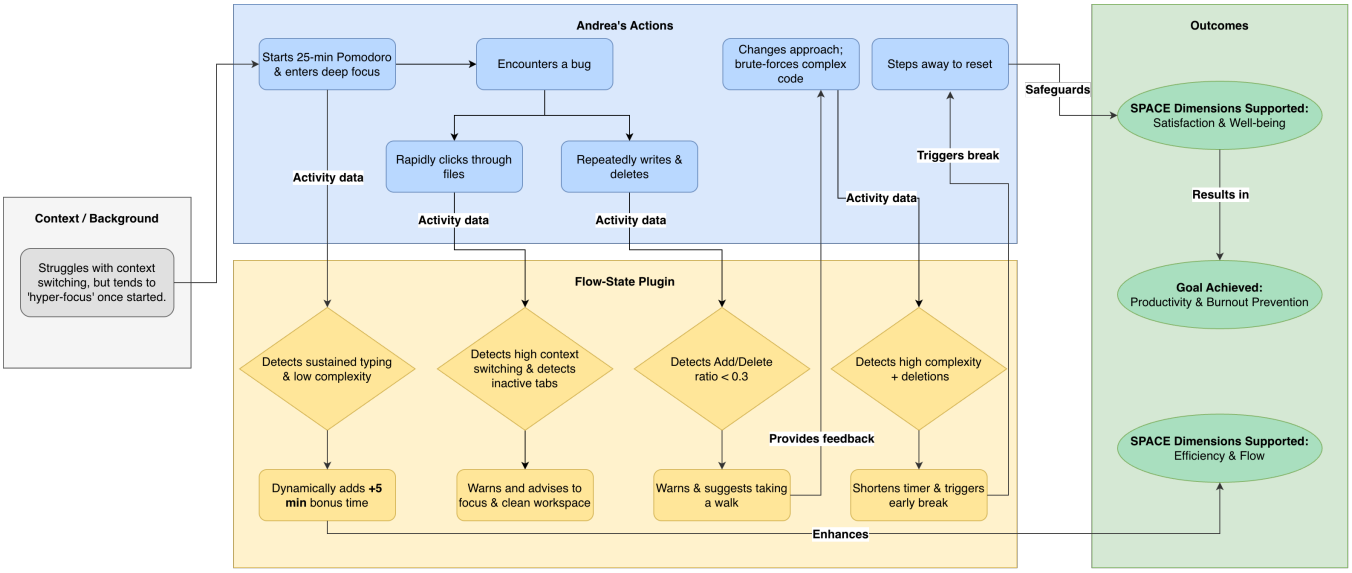

Figure 14: Flow representing interactions between Andrea and Flow-State.

stacked PRs to keep it easy to review." Second, the dashboard explicitly lists the names and scores of two specific staged files that breach the complexity threshold, suggesting: *"Suggestion: These files are quite complex. Maybe extract some logic into smaller helper functions?"* Finally, the tool detects that they installed libraries that they ultimately did not use. It lists the exact names of these "zombie dependencies" directly in the tool tip, reminding them: *"Suggestion: Don't forget to clean up your package.json before committing!"*

Prompted by these contextual nudges, Kona unstages their changes. They delete the unused libraries, refine the deepest loops in the flagged files, and successfully split their massive feature branch into three separate, logically grouped "stacked" PRs. This proactive decomposition directly supports the *Collaboration and Communication* dimension of the SPACE framework [6], ensuring that knowledge transfer between teammates remains efficient rather than burdensome. Furthermore, by shifting the evaluation of code review friction to the authoring phase, *Flow-State* optimizes developer *Feedback Loops*—a core pillar of Developer Experience (DevEx) [7]. Ultimately, the tool prevents reviewer fatigue before the PR is even opened, enabling more thorough peer reviews and faster integration cycles [14].

5 DISCUSSION

This section discusses our scenario-based evaluation of *Flow-State*. While background tracking effectively monitors developer activity,

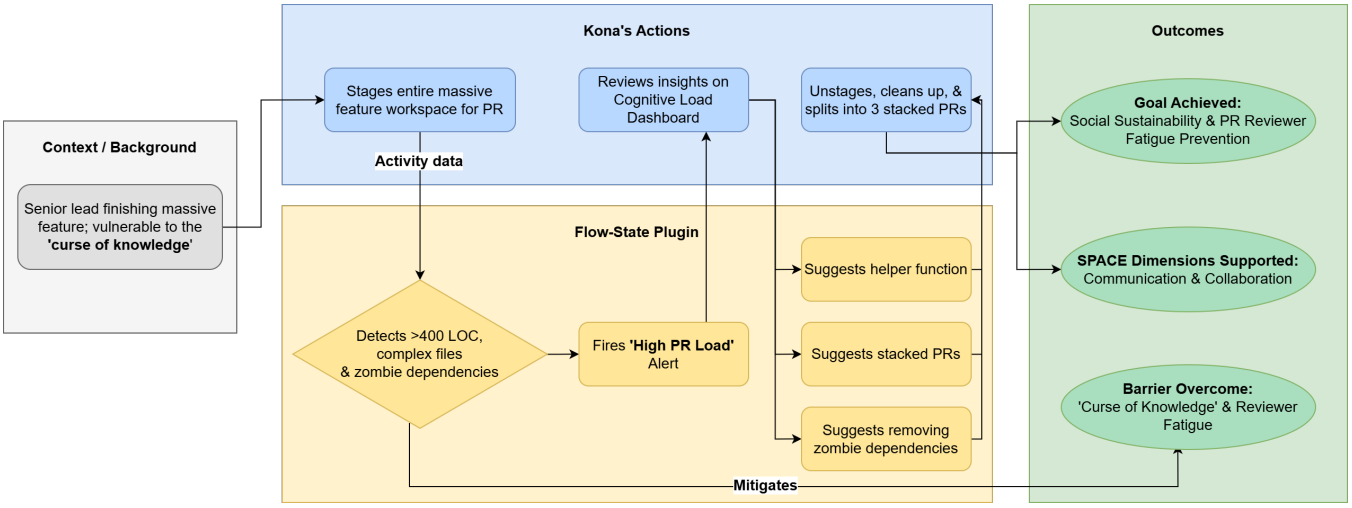

Figure 15: Flow representing interactions between Kona and Flow-State.

translating qualitative experiences like cognitive fatigue into strict numeric rules highlighted the inherent difficulty of quantifying human behavior. We interpret our findings through the SPACE and DevEx frameworks to demonstrate how the tool supports both individual and social sustainability, before concluding with our limitations and directions for future work.

5.1 Key Findings

Breaking down the developer journey into specific scenarios yielded the following insights regarding workflow efficiency and fatigue prevention within software engineering teams.

5.1.1 Detecting Cognitive Fatigue Automatically. A major challenge for developer well-being is identifying when a developer is overwhelmed by a large volume of information without requiring them to manually report their mental state. Traditional well-being tools often rely on disruptive pop-up surveys asking for the user's mood, which inherently breaks their focus [10]. Our evaluation illustrates that within the tested scenarios, background activity trackers—such as measuring Andrea's Add/Delete ratio during moments of frustration, or Amy's extended Read/Write times—can serve as silent indicators of developer "tunnel vision" and fatigue [8]. This aligns with existing literature showing that behavioral signals, such as frequent code deletions and typing rhythms, are strong indicators of programmer frustration and stress [9, 10]. By continuously monitoring these statistics, *Flow-State* successfully captures the *Activity* dimension of the SPACE framework [6] to balance well-being and productivity without causing secondary interruptions.

5.1.2 Mitigating Internet-Induced Cognitive Overload. Modern software development heavily relies on external code generation, whether through AI coding assistants or copying solutions from forums like Stack Overflow. While these tools increase writing speed, they shift a heavy cognitive burden to reading and verifying code. Recent studies warn that this creates a "false sense of progress" [11] and results in "comprehension debt" [13]. Our evaluation highlights how combining two specific metrics—detecting large, sudden code insertions followed immediately by extended periods of scrolling without typing (high Read/Write ratios)-enabled *Flow-State* to diagnose this comprehension debt for junior developers like Amy. By warning developers when they paste too much unverified code and prompting them to take micro-breaks, the tool effectively manages cognitive load, which is a core dimension of improving the overall Developer Experience (DevEx) [7].

5.1.3 Adapting Focus Timers to Protect Flow State. Time management strategies, such as the Pomodoro technique [20], present a paradox for developers: they are highly effective for overcoming initial procrastination, but their rigid alarms often interrupt developers precisely when they reach peak focus. Research shows that rigid interruptions and frequent context switches severely fragment focus time, directly reducing overall productivity and developer satisfaction [26]. Our evaluation suggests that integrating dynamic behavioral metrics into a timer can resolve this contradiction. As seen in Andrea's scenario, by applying a time multiplier when code complexity is low and typing is steady, *Flow-State* protects deep work and enhances the *Efficiency and Flow* dimension of the SPACE framework [6]. Conversely, by shortening the timer when Andrea's Add/Delete ratio indicates frustration, the tool proactively prevents burnout, illustrating that developer timers can benefit from adapting to the user rather than being strictly static.

5.1.4 Minimizing Context Switching and Workspace Clutter. Beyond writing code, navigating complex software architectures imposes a high environmental cognitive load on developers. Our evaluation demonstrates that for user personas like Andrea, unmanaged workspaces—characterized by frantic file-hopping and a buildup of inactive tabs—can exacerbate mental fatigue. By actively measuring the frequency of file switches, particularly across different directories, *Flow-State* was able to successfully identify when Andrea was lost or context-thrashing. Furthermore, by prompting developers to review and close inactive tabs, the tool minimizes visual noise. This proactive workspace management directly supports the *Satisfaction and Well-being* dimension of the SPACE framework [6] by maintaining a focused, manageable IDE environment.

5.1.5 Fostering Developer Agency Through Data Visibility and Customization. While background tracking enables automated interventions, our evaluation suggests that exposing this data directly to the user can be equally valuable for developer well-being. By centralizing real-time statistics into a localized Dashboard, developers can track their own performance metrics-such as file context switches, time since the last write, and cognitive complexity-enabling the type of self-reflection on work habits that positively impacts perceived productivity [26]. This visibility transforms *Flow-State* into a tool for self-reflection, overcoming the lack of awareness that often hinders Developer Experience (DevEx) [25] and allowing users to recognize their own disruptive patterns before burnout occurs [1]. Furthermore, to ensure the tool adapts to individual working styles rather than dictating them, every feature can be completely customized, enabled, or disabled directly from the IDE settings. Beginning with an initial onboarding walkthrough, developers have the freedom to choose which interventions suit their specific preferences. By guaranteeing that all tracking remains strictly private and that the user retains ultimate control over the tool's functionality, the plugin provides essential control and flexibility-factors increasingly recognized as vital components of modern software development [21]. This empowers developers to take ownership of their own *Activity* and workflows [6] without the pressure of rigid obligations or external surveillance.

5.1.6 Shifting Code Review Friction "Left". Typically, reviewer fatigue is only identified after a PR is submitted, resulting in delayed feedback and bottlenecked workflows [16]. Literature confirms that understanding the code, rather than finding defects, is the primary challenge in modern code review [14], and keeping PRs under 400 lines of code is crucial for review effectiveness [17]. By shifting the evaluation of code review metrics (Lines of Code, per-file complexity, and zombie dependencies) to the author's local staging phase, *Flow-State* encourages experienced developers like Kona to confront the "curse of knowledge" before sharing their work [19]. This proactive approach directly supports the *Communication and Collaboration* dimension of the SPACE framework [6] and optimizes developer *Feedback Loops*, a key DevEx factor [7]. Ultimately, by intercepting overly complex PRs at the source, *Flow-State* actively fosters social sustainability [4], ensuring that one developer's output does not become someone else's burnout.

5.2 Limitations & Future Work

While *Flow-State* demonstrates significant potential in managing developer cognitive load, our evaluation revealed certain constraints inherent to heuristic-based behavioral tracking. Addressing these limitations is crucial for refining the tool's accuracy and user experience, providing clear directions for future work.

5.2.1 Heuristic Inaccuracies and False Positives. The plugin relies on rigid (albeit highly configurable) quantitative thresholds to infer complex, qualitative mental states. For example, while a user can manually adjust the `addDeleteRatioThreshold`, a fixed numeric value struggles to capture the nuance of human intent, reflecting the broader difficulty of sensing developer emotions via interaction data [9, 10]. A severe drop in this ratio triggers a "Frustration/Stuck" warning; however, a developer conducting a massive, highly satisfying code refactor might trigger this same threshold while feeling highly productive. In such cases, the tool's intervention could be perceived as an annoying false positive. Future iterations should aim to make these heuristics context-aware rather than purely numeric. This could involve integrating lightweight algorithms that dynamically learn a developer's unique baseline behaviors over time, or enabling the plugin to recognize IDE-specific refactoring commands to temporarily suppress frustration alerts during intentional code deletion.

5.2.2 "Surveillance" Anxiety and Privacy Concerns. Tracking behavioral signals like keystrokes, scrolling time, and deletion rates inherently borders on workplace surveillance. Even if the data remains strictly local, the mere presence of behavioral monitoring can induce anxiety, which directly contradicts the SPACE framework's explicit warning against using productivity metrics for individual performance evaluation [6]. If developers feel that these metrics could be exported or used by management to evaluate their performance, they will likely uninstall the tool, as developer autonomy is a critical requirement of modern engineering culture [21]. The success of *Flow-State* relies entirely on developers trusting that the tracking is strictly local, private, and designed solely for their personal well-being. Future work should focus on transparent data governance. This includes explicitly visualizing within the UI that no external network requests are made, and potentially adding verifiable "data destruction" features that allow users to wipe their behavioral history at the end of each session to guarantee complete privacy.

5.2.3 Alert Fatigue and the Configuration Burden. While *Flow-State* empowers developers to fully customize their alert thresholds, finding the perfect balance places a cognitive configuration burden on the user. If these thresholds are configured too strictly, or if a user simply leaves them at overly sensitive default settings, the status bar will constantly flash warning colors. Ironically, a tool designed to reduce cognitive load could inadvertently increase it by bombarding the developer with continuous visual distractions. Rather than preserving the flow state, these constant notifications act as micro-interruptions that severely fragment focus time and reduce overall productivity [26]. Ultimately, this continuous overstimulation leads to "alert fatigue", causing users to simply tune out and ignore the tool's critical warnings altogether. To combat this without forcing users to constantly tweak their settings, future development should refine the intervention strategy. Implementing an exponential back-off strategy for notifications would ensure warnings remain meaningful without being overwhelming. Furthermore, future versions could aggregate low-priority alerts into a post-session summary on the dashboard, reserving real-time status bar flashes only for critical, flow-breaking issues.

6 CONCLUSION

Software engineering is inherently cognitively demanding, frequently pushing developers towards burnout through silent mental fatigue, a burden that is ultimately transferred onto teammates through complex code reviews. To address this dual threat, this project introduced *Flow-State*, an IDE plugin designed to promote

both individual and social sustainability within software engineering teams.

Through our scenario-based evaluation of distinct developer personalities, we showcased that continuous, background behavioral tracking can effectively diagnose and mitigate cognitive friction without requiring disruptive self-reporting. At the individual level, *Flow-State's* interventions addressed the unique needs of different working styles-from helping a junior developer navigate AI-induced "comprehension debt" to assisting a hyper-focused mid-level engineer in managing workspace clutter and frustration. By dynamically adapting focus timers based on real-time typing rhythms and code complexity, the tool illustrates how productivity aids can protect a developer's flow state rather than rigidly interrupting it.

At the social level, by evaluating the workflow of a senior engineer vulnerable to the "curse of knowledge", *Flow-State* highlighted its ability to tackle team-wide burnout by shifting the evaluation of Pull Request friction to the author's local workspace. By warning developers about massive file sizes, excessive complexity, and unused dependencies before they commit, the tool actively prevents reviewer fatigue and fosters empathetic collaboration. Ultimately, while heuristic-based tracking presents future design challenges regarding user trust and alert fatigue, *Flow-State* suggests that modern developer tooling can evolve beyond simply tracking code output, serving instead as a vital guardian of a healthy, sustainable software engineering culture.

REFERENCES

[1] Tulili, T. R.; Capiluppi, A.; Rastogi, A. Burnout in software engineering: A systematic mapping study. 2023; p 107116.

[2] Gonçales, L. J.; Farias, K.; da Silva, B. C. Measuring the cognitive load of software developers: An extended systematic mapping study. 2021; p 106563.

[3] Chandrasekaran, S. Enhancing Developer Experience by Reducing Cognitive Load: A Focus on Minimization Strategies. 2024; pp 104–108.

[4] di Biase, M.; Bruntink, M.; van Deursen, A.; Bacchelli, A. The effects of change decomposition on code review—a controlled experiment. 2018; PeerJ Computer Science, 5:e193 (2019), DOI:10.7717/peerj-cs.193.

[5] Microsoft Visual Studio Code. 2025.

[6] Forsgren, N.; Storey, M.-A.; Maddila, C.; Zimmermann, T.; Houck, B.; Butler, J. The SPACE of Developer Productivity: There's more to it than you think. 2021; pp 20–48.

[7] Forsgren, N.; Storey, M. A.; Maddila, C.; Zimmermann, T.; Houck, B.; Butler, J. DevEx: What actually drives productivity. 2024; pp 26–33, Originally published in ACM Queue in 2023.

[8] Xia, X.; Bao, L.; Lo, D.; Xing, Z.; Hassan, A. E.; Li, S. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. 2018; pp 951–976.

[9] Kołakowska, A. Towards Detecting Programmers' Stress on the Basis of Keystroke Dynamics. Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS). 2016; pp 1621–1626.

[10] Müller, S. C.; Fritz, T. Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress. Proceedings of the 37th International Conference on Software Engineering (ICSE). 2015; pp 688–699.

[11] Khan, M. F. A.; Feri, L. E.; Nguyen, H.; Karimi, H. Student-Perceived Cognitive Load of LLM-Generated Programming Exercises. Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA). 2025; pp 1–12.

[12] METR (Model Evaluation & Threat Research) *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*; Research Report, 2025.

[13] Osmani, A. Comprehension Debt - the hidden cost of AI generated code. 2026; Accessed: 2026-03-17.

[14] Bacchelli, A.; Bird, C. Expectations, Outcomes, and Challenges of Modern Code Review. Proceedings of the International Conference on Software Engineering (ICSE). 2013.

[15] Graphite How large PRs slow down development. 2023; Accessed: 2026-03-17.

[16] Zhang, Y.; Wang, Y.; Wang, H.; Yin, G. Pull Request Latency Explained: An Empirical Overview. 2022; pp 1–38.

[17] SmartBear Best Practices for Peer Code Review. 2024.

[18] Campbell, G. A. Cognitive complexity: A new way of measuring understandability. Proceedings of the 2018 International Conference on Technical Debt (TechDebt). 2018; pp 38–39.

[19] Ismail Cognitive load in code reviews: How to write PRs your teammates can actually read. 2023; Accessed: 2026-03-17.

[20] Cirillo, F. The Pomodoro Technique. 2026; Accessed: 2026-03-29.

[21] Krohn, R. Why Greater Autonomy is the Future of Software Development: State of the Developer 2022. 2022; Atlassian report based on survey of over 2,000 developers, Accessed: 2026-03-17.

[22] Cohen, J. The Largest Case Study of Peer Code Review—Ever: Cisco Systems Code Review Metrics. 2006; SmartBear / Cisco case study analyzing 2,500 reviews of 3.2M LOC showing optimal review effectiveness for 200–400 LOC per session.

[23] Bødker, S. Scenarios in user-centred design—setting the stage for reflection and action. 2000; pp 61–75.

[24] Vaithilingam, P.; Zhang, T.; Glassman, E. L. Expectation vs.nbsp;Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems. New York, NY, USA, 2022; Accessed: 2026-03-17.

[25] Greiler, M.; others An Actionable Framework for Understanding and Improving Developer Experience. 2024.

[26] Meyer, A.; Barton, L.; Murphy, G.; Zimmermann, T.; Fritz, T. The Work Life of Developers: Activities, Switches and Perceived Productivity. 2017; pp 1–1.