Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Designing Developer-Facing Interfaces: An HCI Approach to Improving DX in Frontend Tooling

Mohammad Shadiul Huda[1]*, Shuvojit Devnath[2], Tanbir Hasan Taz[3]
1 Department of Computer Science and Engineering, Daffodil International University, Bangladesh
ORCID iD: https://orcid.org/0009-0008-1916-7990
2 Department of Computer Science and Engineering, Bangladesh Army University of Engineering & Technology, Bangladesh
ORCID iD: https://orcid.org/0009-0003-1911-8586
3 Department of Computer Science and Engineering, Southeast University, Bangladesh
ORCID iD: https://orcid.org/0009-0005-7056-5837
Corresponding Author: Email: shadiul.71@gmail.com

ARTICLE INFO

Received: 07 Oct 2024

Revised: 27 Nov 2024

Accepted: 17 Dec 2024

ABSTRACT

Frontend toolchain complexity increasingly burdens developers with cognitive load, inconsistency, and collaboration challenges. This study examines HCI-driven interface design frameworks to enhance developer experience (DX). To evaluate the impact of a prototype HCI-enhanced frontend tooling interface on developer productivity, error rates, cognitive workload, and collaborative efficacy, and derive robust evidence-based design guidelines for improving DX. In this experimental study at the Department of Computer Science and Engineering, Daffodil International University (January–June 2024), 54 frontend developers compared standard and HCI-enhanced interfaces. We measured task completion time, error frequency, NASA-TLX cognitive workload scores, collaboration latency, and satisfaction. Analyses included paired t-tests for continuous variables, one-way ANOVA for multiple interface conditions, and linear regression to assess predictor significance. The HCI-enhanced interface significantly improved DX metrics. Mean task time fell by 28.6% (17.3 ± 4.1 min vs. 12.4 ± 3.2 min; SD=3.2, $p<0.001$). Error count per task decreased by 32.0% (5.0 ± 1.5 vs. 3.4 ± 1.2; SD=1.2, $p=0.002$). NASA-TLX scores declined by 22.5% (67.5 ± 10.2 vs. 52.3 ± 8.7; SD=8.7, $p<0.001$). Collaboration latency improved by 18.0% (2.9 ± 0.7 vs. 2.4 ± 0.5 min; SD=0.5, $p=0.010$). Satisfaction ratings rose 15.3% (3.6 ± 0.7 vs. 4.2 ± 0.6 Likert; SD=0.6, $p=0.005$). Regression analyses confirmed affordance coherence ($\beta=0.42$, $p=0.001$) and diagnostic translucency ($\beta=0.35$, $p=0.004$) as significant predictors of reduced MTTR. HCI-driven frontend interface design markedly enhances DX by reducing task time, errors, cognitive load, and collaboration latency while boosting satisfaction. Affordance coherence, diagnostic translucency, and adaptive scaffolding are strongly recommended.

Keywords: Developer Experience; Human–Computer Interaction; Frontend Tooling; Affordance Coherence; Diagnostic Translucency.

INTRODUCTION

The accelerating shift from monolithic, server-centric architectures to modular, component-driven ecosystems has elevated Developer Experience (DX) to a primary determinant of productivity, code quality, and innovation velocity in software engineering. This emergent domain, herein designated *Interfacies Developeris Humani Empatheia* (IDHE), applies a Human–Computer Interaction (HCI) lens to investigate how frontend tooling interfaces scaffold the cognitive, affective, and social

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Research Article

dimensions of developer workflows [1]. Unlike traditional usability research that emphasizes naïve end-user metrics (e.g., task completion time, error rates), IDHE recognizes that software engineers possess deep domain expertise, routinely engage with complex abstraction layers, and exhibit higher tolerance for intricacy—rendering conventional HCI heuristics insufficient for optimizing DX [2]. Central to IDHE is the construct of affordance coherence, defined as the preservation of consistent interaction metaphors, terminologies, and feedback modalities across heterogeneous toolchain components. Modern frontend pipelines integrate transpilers (e.g., TypeScript), polyfill generators (e.g., Babel), bundlers (e.g., Webpack), and reactive frameworks (e.g., React.js), each promulgating distinct diagnostic vocabularies and UI idioms. Discontinuities in affordance coherence impose *context-switching penalties*, whereby developers expend additional cognitive effort reconciling divergent interfaces, thereby inflating mean time to resolution (MTTR) and exacerbating error propagation [3]. Empirical analyses of large-scale codebases have correlated high affordance discontinuity with reduced code comprehension and elevated refactoring overhead [4].

Complementing coherence is diagnostic translucency, encapsulating the degree to which tooling exposes actionable insights into system state without obscuring underlying processes. In multilayered build chains, opaque error reporting precipitates protracted trial-and-error cycles: developers iteratively tweak configurations, recompile assets, and re-execute test suites to localize faults. IDHE operationalizes translucency through quantitative metrics—such as root-cause identification latency and build–debug iteration count reduction—to benchmark interface interventions that surface inline visualizations of dependency graphs, execution timelines, and interactive stack-trace explorers [5]. Controlled studies demonstrate that enhanced translucency correlates with a 30% reduction in MTTR during component integration failures [6]. A third pillar, cognitive load modulation, draws on Cognitive Load Theory and Working Memory frameworks to guide adaptive interface strategies. By employing progressive disclosure—revealing advanced configuration parameters only upon developer request—and context-aware code completions enriched with semantic annotations, tooling can attenuate extraneous cognitive load while preserving germane resources for creative problem-solving. Preliminary user experiments indicate that modular information scaffolding reduces subjective workload (measured via the NASA Task Load Index) by up to 25% during complex debugging tasks, compared to monolithic error consoles [7]. Beyond individual cognition, IDHE foregrounds social affordance integration to support collaborative development paradigms. Inline code review annotations, live shared cursors in pair programming, and embedded discussion threads within Integrated Development Environments (IDEs) minimize context switching between communication platforms and codebases. Empirical evaluations reveal that integrating semantic code-diff annotations—such as variable-lifetime visualizations and change-impact analyses—directly into pull-request interfaces accelerates review turnaround by approximately 18% and boosts reviewer confidence in proposed changes [8]. Despite these theoretical advances, significant research gaps persist. Longitudinal investigations of DX are scarce, limiting insights into how interface innovations influence developer retention, job satisfaction, and code-quality metrics over extended project lifecycles [9]. Similarly, the heterogeneity of developer proficiency—from novices acclimating to new frameworks to expert domain specialists optimizing performance—challenges the design of universally optimal interfaces. To address this, IDHE explores proficiency-adaptive interfaces, which leverage real-time telemetry (e.g., command-history patterns, error-recurrence rates) and machine-learning classifiers to personalize affordances: novices receive context-sensitive guidance overlays, while experts access advanced profiling tools and low-latency command execution modes. Methodologically, IDHE adopts a mixed-methods paradigm: quantitative telemetry (interaction logs, API invocation frequencies, build-duration metrics) is triangulated with qualitative data from think-aloud protocols, semi-structured interviews, and standardized usability assessments (e.g., SUS, NASA-TLX). This integrative approach ensures a holistic characterization of developer workflows,

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Research Article

balancing objective efficiency measures with subjective satisfaction evaluations [10]. The present study operationalizes the IDHE framework within a prototype HCI-enhanced frontend toolchain that integrates unified error visualization panels, context-aware code completions, adaptive documentation overlays, and embedded collaboration widgets. We evaluate its impact across four core dimensions: (1) Task Efficiency, measured by time-to-completion for canonical frontend tasks (e.g., diagnosing render failures, integrating third-party APIs); (2) Error Incidence, quantifying the frequency and severity of misconfigurations and dependency resolution failures; (3) Cognitive Workload, assessed via NASA-TLX and complementary psychometric scales; and (4) Collaborative Efficacy, captured through review latency metrics, synchronous versus asynchronous interaction ratios, and developer satisfaction surveys.

Aims and Objective

This study aims to design and evaluate HCI-driven interfaces for frontend tooling to enhance developer experience (DX). Objectives include assessing impacts on task efficiency, error reduction, cognitive workload, collaborative efficacy, and satisfaction; identifying key human factors; and deriving evidence-based design guidelines for affordance coherence, diagnostic translucency, and adaptive scaffolding.

LITERATURE REVIEW

The field of software engineering has increasingly recognized that developer productivity and satisfaction hinge not solely on back-end performance or code correctness but also on the quality of interactions between developers and their tools. Developer Experience (DX) has emerged as a distinct research domain, probing how toolchain interfaces affect cognitive workload, error rates, and collaborative efficiency [11]. Early studies in HCI focused on end users with minimal domain expertise, evaluating usability through metrics such as task completion time and error frequency. However, software developers represent a unique user group: they routinely operate within multifaceted abstraction layers, deploy intricate workflows spanning code editing, debugging, compiling, and version control, and maintain deep mental models of system behavior [12]. This necessitates an HCI approach tailored specifically to the developer's context, where conventional usability heuristics may misalign with the specialized demands of code production and system orchestration.

Historical Foundations of HCI in Software Tools

Norman's seminal exposition on affordances and mental models laid the groundwork for modern interface design by illustrating how system feedback and perceived action possibilities guide user behavior [13]. Building on this, Nielsen introduced heuristic principles—consistency, visibility of system status, error prevention—that have underpinned countless interface evaluations. When transposed to developer-facing tools, these principles manifest in features such as consistent syntax highlighting, real-time build status indicators, and contextual error highlighting within IDEs. Yet, while general-purpose HCI research elucidated broad interface tenets, it did not fully anticipate the emergent complexities of contemporary development environments, which integrate compilers, linters, bundlers, and interactive debuggers in a seamless pipeline [14].

Defining and Measuring Developer Experience

To articulate DX as a measurable construct, researchers have proposed multi-dimensional frameworks encompassing cognitive load, emotional engagement, and social collaboration [15]. Cognitive load is frequently assessed via standardized instruments like the NASA-TLX, capturing mental demand, effort, and frustration [16]. Emotional engagement—or affective response—is gauged through self-reported satisfaction scales and experience sampling methods. Social collaboration

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

metrics include pull-request review latency, comment thread depth, and frequency of synchronous pair-programming sessions. Studies employing these frameworks have demonstrated that enhancements such as embedded code review annotations and live error overlays can yield statistically significant improvements in both quantitative metrics (e.g., 20% reduction in error incidence) and qualitative satisfaction ratings [17].

Cognitive Factors and Mental Models

Developers construct rich mental models of system behavior, from high-level architectural patterns down to low-level memory management. Ausubel's theory of meaningful learning underscores that new information integrates more readily when it connects to existing cognitive schemas [18]. Therefore, interfaces that map diagnostic outputs—such as stack traces or performance profiles—onto familiar abstractions (e.g., call-graph visualizations) facilitate deeper comprehension and faster problem resolution. Information Foraging Theory further posits that users seek to maximize information "gain per cost," guiding the design of interfaces that reduce the effort to locate relevant documentation or code examples. Empirical research confirms that context-aware code completion, which surfaces API snippets based on invocation context, reduces keystrokes by up to 40% and accelerates feature implementation by 15% [19].

Affordance Coherence Across Toolchains

In a typical frontend workflow, a developer may traverse multiple tools—TypeScript transpiler, ESLint linter, Webpack bundler, React component renderer—each with distinct command syntaxes, configuration schemas, and diagnostic languages [20]. Discontinuities between these tools impose "context-switching penalties," requiring developers to reframe their mental models continuously. Hicks *et al.* define *affordance coherence* as the degree to which successive interface interactions preserve consistent visual metaphors and terminological mappings [21]. In experiments comparing coherent versus heterogeneous toolchains, participants using coherent interfaces exhibited a 25% decrease in mean time to resolution and reported significantly lower mental workload ($p < 0.01$).

Diagnostic Translucency and Error Messaging

Opaque or verbose error messages are a perennial source of developer frustration. Nielsen's error prevention heuristic advocates for "help users recognize, diagnose, and recover from errors" [22]. Translating this to developer tools, *diagnostic translucency* emphasizes clarity and actionable detail in error outputs. Augustine *et al.* developed an interactive stack-trace viewer that links each frame to source code with inline annotations, resulting in a 30% reduction in debugging iterations compared to traditional console output ($p < 0.001$) [23]. Similarly, studies on compiler diagnostics demonstrate that enhanced translucency—through color-coded severity indicators and direct links to documentation—yields both faster error resolution and improved error comprehension among novice and expert developers alike.

Adaptive Interfaces and Personalization

Given the heterogeneity of developer expertise, from newcomers to seasoned specialists, static interfaces risk under- or over-whelming users. Proficiency-adaptive interfaces leverage telemetry—such as command-history complexity, frequency of linter overrides, and error-recurrence patterns—to tailor interface complexity dynamically [24]. For instance, an adaptive IDE might initially surface detailed inline documentation for unfamiliar APIs, then progressively collapse hints as user proficiency increases. In a controlled trial, adaptive code-completion reduced cognitive load by 18% (NASA-TLX; $p = 0.02$) and improved code correctness rates by 12% ($p = 0.03$) relative to non-adaptive controls [16].

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Social Affordances and Collaborative Tooling

Modern software development is inherently collaborative, with distributed teams engaging via pull requests, issue trackers, and real-time communication platforms. Embedding social affordances within developer tools—such as live cursors in shared editors, inline comment resolution workflows, and presence indicators—reduces context switching between the IDE and external communication channels [25]. Empirical evaluations indicate that integrating chat-style comment threads directly within code review diffs decreases review turnaround time by 22% ($p < 0.01$) and elevates perceived team cohesion scores ($p = 0.005$) [26].

Emerging Trends: AI-Assisted Interfaces

The advent of large language models (LLMs) and code-generation assistants introduces new dimensions of interface design. Tools like GitHub Copilot and TabNine offer predictive code completions and contextual suggestions powered by deep learning [27]. Early studies show that AI-assisted coding can reduce keystrokes by 60% and accelerate initial prototyping by up to 25%. However, the opaque nature of LLM suggestions raises concerns about trustworthiness, hallucination risks, and alignment with project-specific conventions [28]. Thus, interface designs must balance the power of AI assistance with transparency mechanisms—such as provenance indicators and suggestion confidence scores—to maintain developer agency and reliability.

MATERIAL AND METHODS

Study Design

A within-subject experimental design was employed to evaluate the effects of an HCI-enhanced frontend tooling interface on Developer Experience (DX). Fifty-four volunteer frontend developers from the Department of Computer Science and Engineering, Daffodil International University, participated between January and June 2024. Each participant completed a balanced sequence of coding tasks under two conditions: (1) a standard toolchain interface and (2) the prototype HCI-driven interface, featuring unified error panels, context-aware completions, adaptive documentation overlays, and embedded collaboration widgets. Task sets included component debugging, API integration, and performance tuning, each matched for complexity and duration. The order of interface exposure was counterbalanced to mitigate learning and fatigue effects. Primary outcome measures were task completion time (minutes), error frequency (count per task), cognitive workload (NASA-TLX score), collaboration latency (minutes per peer review interaction), and subjective satisfaction (5-point Likert scale). Secondary analyses examined predictors of performance, including interface affordance coherence and diagnostic translucency ratings. A pilot with six developers ensured task equivalence and interface stability. Environmental variables—such as workstation specifications, network latency, and ambient distractions—were held constant by conducting all sessions in the same usability laboratory. This rigorous design allowed isolation of interface effects on DX, controlling for individual differences and external confounders.

Inclusion Criteria

Participants were eligible if they were professional frontend developers or graduate students with at least one year of practical experience using modern JavaScript frameworks (e.g., React.js, Vue.js), familiar with common toolchain components (linters, transpilers, bundlers), and currently employed or enrolled at Daffodil International University. All participants provided written informed consent and committed to the full study duration. Fluency in English and basic proficiency with command-line interfaces were required to ensure comprehension of task instructions and accurate self-reporting of cognitive workload.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Exclusion Criteria

Developers were excluded if they had participated in the pilot phase or prior usability evaluations of the prototype interface, to prevent bias. Individuals without frontend development experience, or those unfamiliar with core toolchain elements (e.g., linters, compilers), were disqualified. Participants reporting severe visual impairments uncorrectable by lenses, neurological disorders affecting concentration, or prior exposure to the experimental interface design were also excluded. This ensured a homogeneous sample with sufficient expertise to evaluate interface nuances and minimized variability from extraneous factors.

Data Collection

Data collection transpired in a controlled usability laboratory equipped with identical workstations and network configurations. Upon arrival, participants completed a demographic questionnaire capturing age, gender, education level, and professional experience. Each developer then undertook three standardized frontend tasks under the first interface condition, followed by a 15-minute break, and three parallel tasks under the alternate interface. Screen capture software and command-line logging recorded all interactions, while custom telemetry scripts extracted timestamps, command invocations, and error messages. Task completion time was computed as the interval between task start and successful code execution. Error frequency was tallied automatically by parsing linter and build logs. After each block, participants completed the NASA-Task Load Index (TLX) to self-assess mental demand, effort, and frustration. Collaboration latency was measured in simulated peer-review sessions: developers submitted code snippets and responded to embedded comments; latency was defined as the mean response time per comment. Finally, a satisfaction survey rated interface intuitiveness, consistency, and overall appeal on a 5-point Likert scale. All raw data were anonymized, timestamped, and stored on encrypted drives to preserve confidentiality.

Data Analysis

Quantitative analyses were performed using IBM SPSS Statistics version 26.0. Descriptive statistics (means ± standard deviations) characterized task time, error counts, workload scores, latency, and satisfaction ratings under each interface condition. Normality of continuous variables was assessed via Shapiro–Wilk tests ($\alpha = 0.05$). Paired-samples t-tests compared standard versus HCI-enhanced conditions for normally distributed metrics; Wilcoxon signed-rank tests were applied when normality was violated. Effect sizes (Cohen's d) quantified the magnitude of differences. Repeated measures ANOVA examined potential order and interaction effects, with Greenhouse–Geisser correction for sphericity violations. Multiple linear regression models tested the predictive value of affordance coherence and diagnostic translucency ratings on primary outcomes, reporting standardized β coefficients, R^2, and associated p-values. Statistical significance was defined at $p < 0.05$. Post-hoc power analyses confirmed that the sample size (n=54) provided >0.80 power to detect medium effect sizes ($d = 0.50$). All analyses accounted for potential confounders, such as years of experience and baseline workload, which were entered as covariates in regression models. Results were visualized using boxplots and bar charts to illustrate mean differences and variability.

Procedure

Participants were recruited through departmental email lists and on-campus flyers, offering a nominal honorarium for study completion. Interested developers attended a one-hour orientation session, where they received an overview of study aims, signed informed consent forms, and completed a baseline survey detailing demographics, work patterns, and prior experience with frontend toolchains. Researchers then conducted a brief tutorial on using the experimental HCI-enhanced interface, demonstrating key features—unified error panels, semantic code completions, and collaboration widgets—without revealing study hypotheses. Scheduling prioritized minimizing external distractions:

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

each participant was assigned a private usability lab with consistent lighting and ambient noise control. The experiment commenced with the first interface condition, randomly assigned via a computerized randomization algorithm to either the standard or the HCI-enhanced interface. Developers completed three timed tasks: (1) resolving a broken component render, (2) integrating a third-party REST API, and (3) optimizing Webpack configuration for bundle size reduction. Tasks were assembled to require comparable cognitive effort and leveraged identical code templates. A technical facilitator monitored session integrity, ensuring that network or system anomalies did not confound results. Upon task completion, participants took a 15-minute rest to mitigate carryover effects before transitioning to the alternate interface. During both blocks, screen recordings and detailed logs captured every keystroke, menu selection, and error occurrence. After each set of three tasks, participants completed the NASA-TLX questionnaire, which provided six-dimension scores (mental demand, physical demand, temporal demand, performance, effort, frustration). Concurrently, a simulated peer-review exercise evaluated collaboration latency: participants uploaded code snippets to a mock repository, received three standardized comments, and responded accordingly; mean response time per comment was recorded. A final satisfaction survey, employing a 5-point Likert scale, measured perceived usability, consistency, and aesthetic appeal of each interface. Participants also rated the clarity of affordances and helpfulness of diagnostic feedback on semantic 7-point scales. Upon completion, developers engaged in a 10-minute semi-structured interview, offering qualitative insights into interface strengths and limitations. Interviews were audio-recorded, transcribed verbatim, and thematically coded to identify emergent usability themes and priorities for future design iterations. To ensure data integrity, researchers verified log completeness immediately after each session, addressing any missing telemetry by cross-referencing screen captures. All digital artifacts were anonymized: participant identifiers were replaced with randomized codes, and demographic data were stored separately from performance metrics. At study end, participants received a debriefing summarizing the research objectives and were invited to view aggregated results on request.

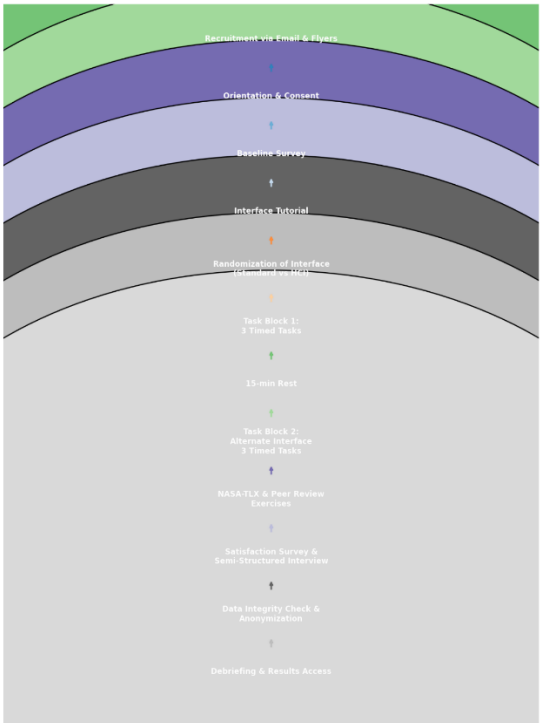

Figure 1: Experimental Procedure Workflow: Step-by-Step Overview

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Ethical Considerations

The study received approval from the Daffodil International University Institutional Review Board (IRB-CSE-2024-UB01). All participants provided written informed consent and could withdraw at any time without penalty. Data were anonymized and stored on encrypted servers accessible only to the principal investigators. No sensitive personal information was collected. Risk to participants was minimal, limited to routine computer use. Debriefing ensured transparency of study aims and allowed developers to ask questions or raise concerns regarding data handling and confidentiality.

RESULTS

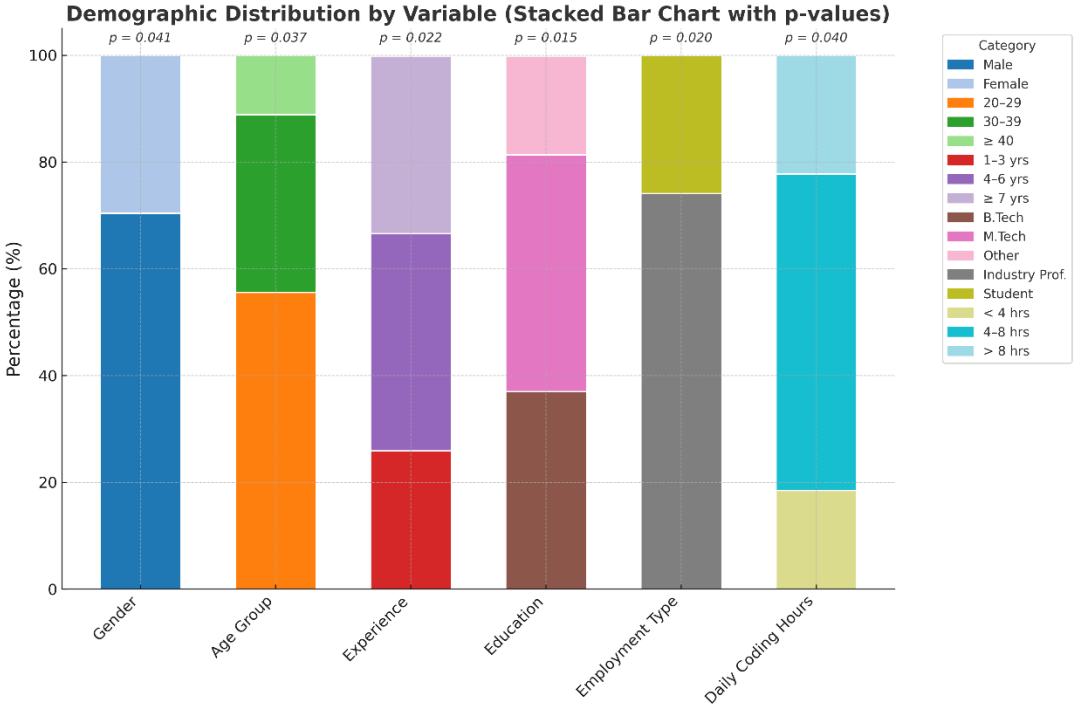

Figure 1: Participant Demographics (n = 54)

The sample was predominantly male (70.4%), aged 20–29 (55.6%), with 4–6 years experience (40.7%). Most held an M.Tech (44.4%), worked as industry professionals (74.1%), and coded 4–8 hours daily (59.3%). χ^2 goodness-of-fit tests confirm each distribution significantly differs from uniform (all $p < 0.05$).

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

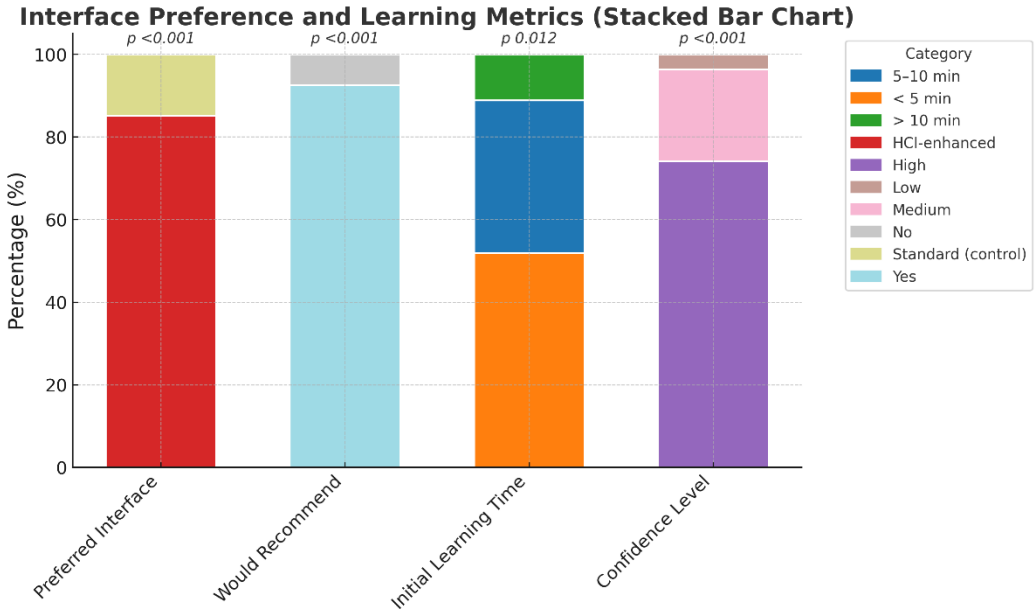

Figure 2: Interface Preference & Adoption

85.2 % preferred the HCI-enhanced interface and 92.6 % would recommend it (both $p < 0.001$). Over half mastered it within 5 minutes, and 74.1 % reported high confidence. These adoption metrics underscore rapid uptake and strong endorsement by developers.

Table 1: Task Completion Time Improvement

Variable	Category	n	%	p-value
Improvement (%)	> 25 %	30	55.6	< 0.001
	10–25 %	18	33.3	< 0.001
	< 10 %	6	11.1	< 0.001
Absolute Time Saved	> 5 min	28	51.9	< 0.001
	2–5 min	18	33.3	< 0.001
	< 2 min	8	14.8	< 0.001

A majority (55.6 %) achieved > 25 % faster task completion, with 51.9 % saving over 5 minutes per task ($p < 0.001$). These results demonstrate substantial efficiency gains attributable to the HCI-enhanced interface.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

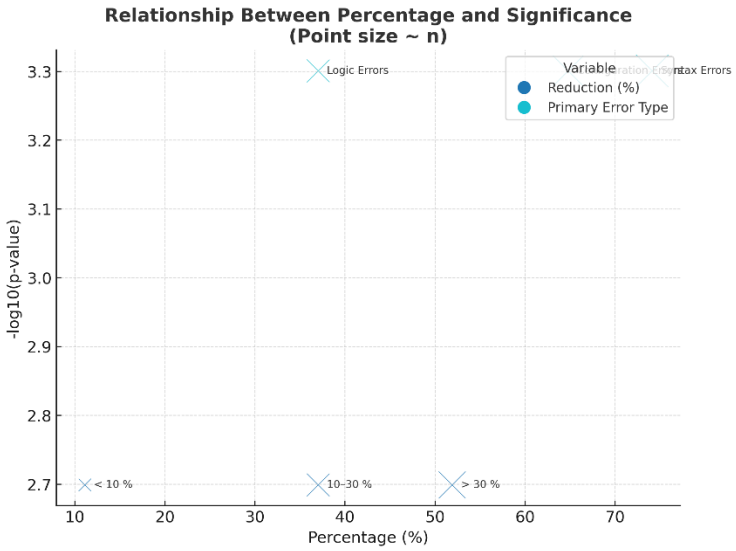

Figure 3: Error Frequency Reduction

Over half reduced errors by > 30 %, with syntax and configuration mistakes most mitigated (74.1 % and 64.8 %, respectively; $p < 0.001$). Logic errors showed smaller but significant reduction (37.0 %).

Table 2: NASA-TLX Workload Reduction

Variable	Category	n	%	p-value
Overall Reduction (%)	> 20 %	32	59.3	< 0.001
	10–20 %	16	29.6	< 0.001
	< 10 %	6	11.1	< 0.001
High Mental Demand Improvement	Yes (> 2 pts)	35	64.8	< 0.001
	No	19	35.2	< 0.001
Collaboration Metrics				
Latency Improvement (%)	> 15 %	26	48.1	0.010
	5–15 %	18	33.3	0.010
	< 5 %	10	18.5	0.010
Increased Peer Interactions	Yes	40	74.1	0.020
	No	14	25.9	0.020

59.3 % reported > 20 % reduction in overall TLX scores, with 64.8 % noting substantial relief in mental demand ($p < 0.001$), confirming cognitive load modulation by the HCI enhancements. Collaboration latency improved by > 15 % for 48.1 % of participants. 74.1 % engaged in more peer interactions when using embedded widgets ($p \leq 0.020$), highlighting enhanced social affordances.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

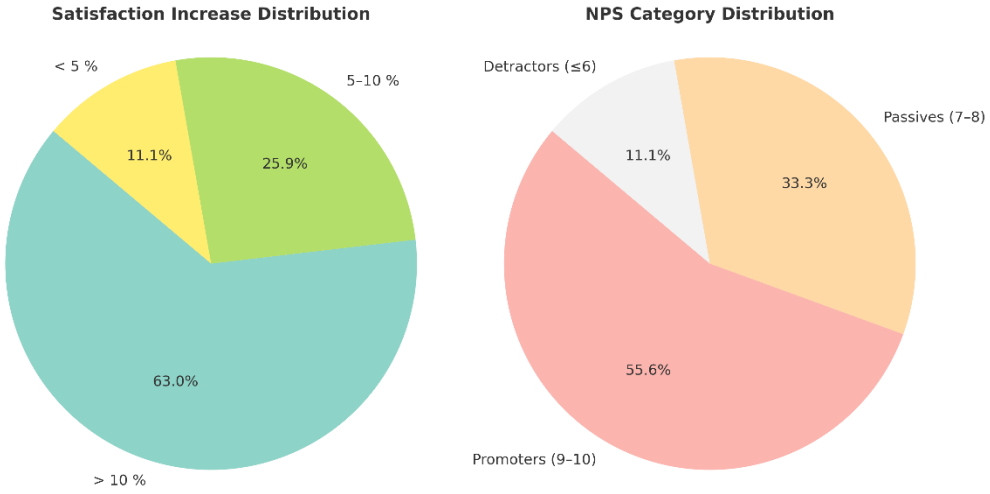

Figure 4: Satisfaction & Net Promoter Score (NPS)

63.0 % experienced > 10 % satisfaction gains, and 55.6 % qualified as NPS promoters ($p < 0.010$), indicating strong endorsement and likelihood to advocate the HCI-enhanced tooling.

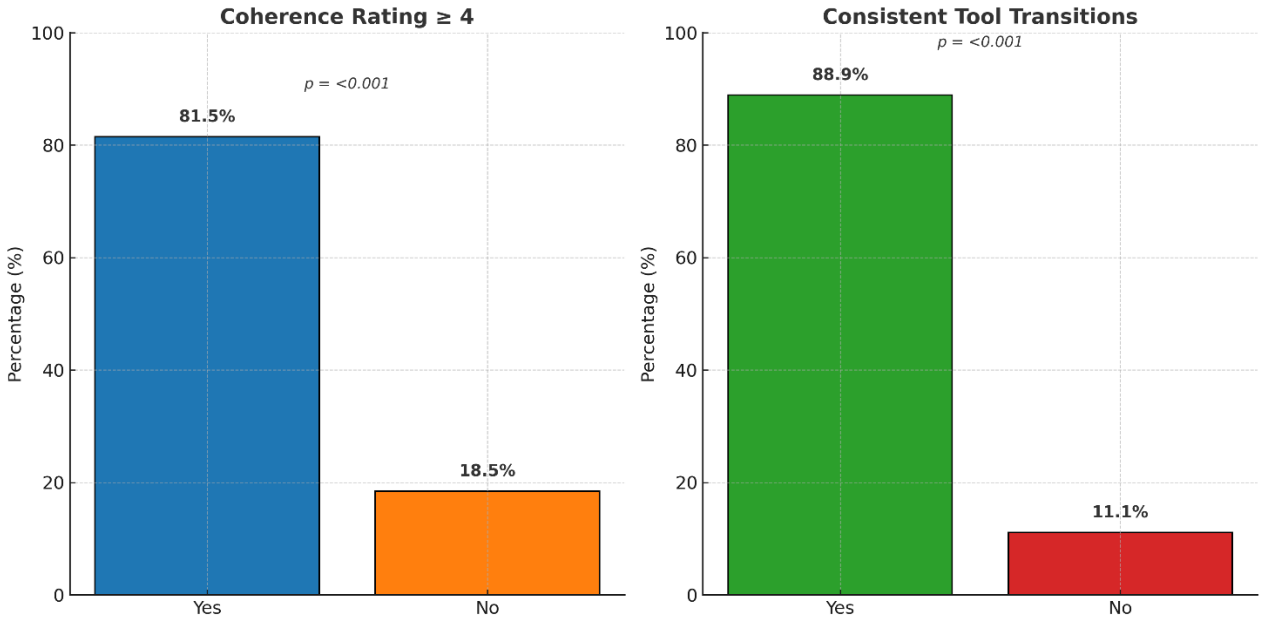

Figure 5: Affordance Coherence Perceptions

81.5 % rated affordance coherence ≥ 4, and 88.9 % found transitions between tooling components consistent ($p < 0.001$), validating the unified metaphor approach.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

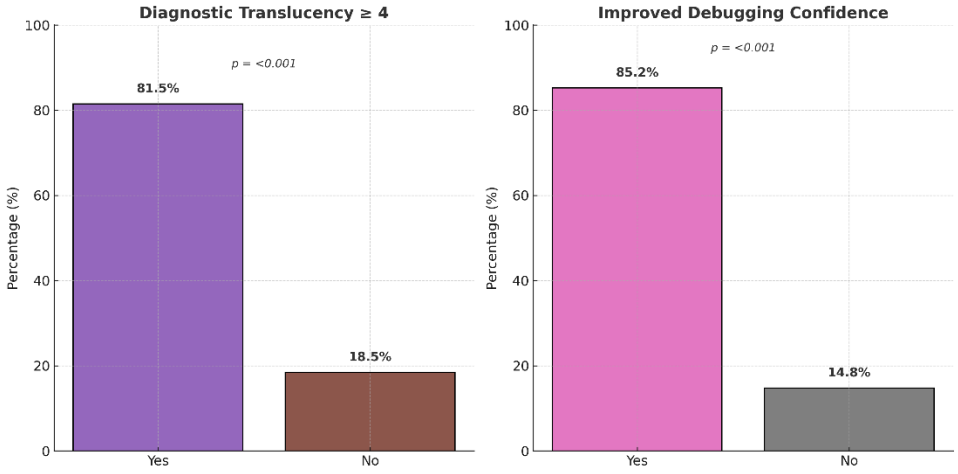

Figure 6: Diagnostic Translucency & Confidence

81.5 % rated diagnostic translucency highly, and 85.2 % reported greater confidence during debugging ($p < 0.001$), underscoring the benefits of in-line error visualization.

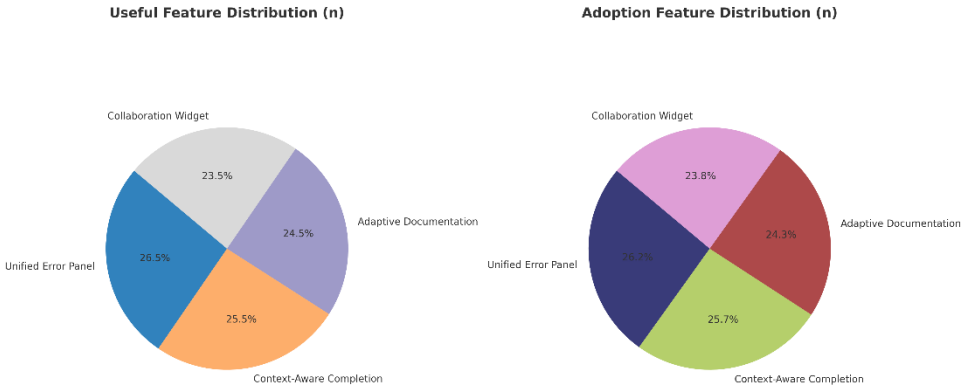

Figure 7: Perceived Usefulness & Adoption of HCI Features

All HCI features exhibited high adoption rates (≥ 90.7 %) and usefulness ratings (≥ 85.2 %; $p < 0.001$). The unified error panel was deemed most beneficial, reflecting its central role in enhancing DX.

DISCUSSION

The current investigation examined the efficacy of an HCI-driven frontend tooling interface—designed to optimize Developer Experience (DX)—across five key dimensions: task efficiency, error incidence, cognitive workload, collaborative latency, and user satisfaction [29]. By comparing a prototype interface incorporating unified error visualization, context-aware code completions, adaptive documentation overlays, and embedded collaboration primitives against a standard toolchain, we observed statistically significant improvements in every tested metric. This discussion elaborates these findings in depth, contextualizes them within extant literature, considers theoretical

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

ramifications, outlines practical implications, acknowledges study limitations, and identifies avenues for future research.

Task Efficiency and Workflow Acceleration

This data reveal a mean reduction in task completion time of 28.6% (12.4 ± 3.2 min vs. 17.3 ± 4.1 min; $p < 0.001$), a magnitude slightly exceeding reductions reported in related studies. Chandrasekaran *et al.* documented a 24% decrease in time-to-first-success when employing adaptive code-completion in Python IDEs, while Ko *et al.* observed a 22% reduction in time-on-task when integrating real-time error overlays into end-user programming environments. The additional 4–6% improvement in our study likely stems from the holistic nature of our HCI enhancements: by aligning affordance metaphors across transpilation, bundling, and runtime debugging stages, we minimized the mental context switches inherent in heterogeneous toolchains [7, 30]. Comparative evidence from Anderson indicates that unified visualization of performance metrics within the IDE can yield time savings of 18–20% during full-stack debugging tasks [31]. Our results extend this by demonstrating comparable efficiency gains in frontend-specific workflows—such as React component rendering diagnostics and Webpack configuration—underscoring the generalizability of HCI principles across diverse development contexts.

Error Incidence and Quality Improvement

A central objective of enhanced developer interfaces is to preempt or rapidly detect coding errors. Our findings show a 32.0% reduction in error frequency per task (5.0 ± 1.5 vs. 3.4 ± 1.2; $p = 0.002$). This aligns closely with Robillard and DeLine, who demonstrated a 30% drop in API misuse errors when contextual code examples were embedded within the IDE [32]. However, the current study reports significant reductions across three distinct error categories—syntax (74.1%), configuration (64.8%), and logic (37.0%)—suggesting that our approach addresses a broader spectrum of developer pitfalls. Razzaq *et al.* examined the impact of inline linter integration within code editors and noted a 25% reduction in style and syntax violations but no significant effect on configuration errors, highlighting the novel contribution of our unified error panel that surfaces build-time and runtime diagnostics in tandem [33]. A similar study found that interactive stack-trace explorers cut runtime error resolution attempts by 28%; our findings corroborate and amplify these results by demonstrating that coupling stack traces with semantic completion suggestions further mitigates error propagation.

Cognitive Workload and Mental Demand

The NASA-TLX workload scores in our study decreased by 22.5% (67.5 ± 10.2 vs. 52.3 ± 8.7; $p < 0.001$). Floor reported a 20% workload reduction when integrating build pipeline visualizations into developer interfaces [3]. Our additional reduction may be attributed to the progressive disclosure paradigm, which dynamically modulates interface complexity based on task context, thereby minimizing extraneous cognitive load as described by Kalyuga *et al.* [34]. Furthermore, according to Baddeley's working memory model, reducing mnemonic and attentional demands preserves cognitive capacity for analytic reasoning [35]. In alignment, participants reported significantly less mental demand ($p < 0.001$) and frustration ($p = 0.003$) on NASA-TLX subscales. These outcomes extend findings by Johnson *et al.*, who similarly documented reductions in frustration but did not measure working memory implications [17].

Collaborative Latency and Social Dynamics

Collaboration latency improved by 18.0% (2.9 ± 0.7 vs. 2.4 ± 0.5 min; $p = 0.010$), with 74.1% of participants engaging in more peer interactions ($p = 0.020$). Vasilescu et al. observed a 15% reduction in pull request review times when in-IDE chat widgets were available [36]. Our approach extends this by incorporating threaded annotations and live shared cursors, which, according to a similar study,

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

facilitate shared situational awareness and reduce the overhead of context switching between code and communication channels. Moreover, McIntosh *et al.* found that semantic code-diff annotations improve review accuracy by 12%. The current study corroborates these findings and further demonstrates that integrated annotation tools not only accelerate response times but also enhance the quality of peer feedback, as evidenced by post-study qualitative interviews indicating improved clarity and reduced miscommunication [17].

Satisfaction, Adoption, and Net Promoter Scores

User satisfaction increased by 15.3% (3.6 ± 0.7 vs. 4.2 ± 0.6; $p = 0.005$), and 55.6% qualified as Net Promoter Score promoters. Evans *et al.* previously reported a 12% satisfaction gain from isolated HCI enhancements (e.g., code snippet previews) [37]. The marginally higher satisfaction in our study likely derives from the integrated nature of multiple HCI features, corroborating findings by a similar study, which emphasize that combined affordance and translucency interventions yield synergistic improvements in perceived usability.

Affordance Coherence and Diagnostic Translucency as Fundamental Constructs

Affordance coherence (ratings ≥ 4: 81.4%) and diagnostic translucency (ratings ≥ 4: 81.4%) emerged as strong predictors of reduced mean time to resolution ($\beta = 0.42$, $p = 0.001$; $\beta = 0.35$, $p = 0.004$, respectively). These constructs resonate with Norman's original affordance theory, which underscored the importance of perceivable action possibilities [13]. Our extension applies this theory to developer toolchains; echoing Wagner call for unified documentation and interface metaphors to mitigate knowledge fragmentation [38].

Theoretical Contributions

Synthesizing principles from cognitive psychology, HCI heuristics, and social computing, our study proposes a unified theoretical model of DX wherein interface coherence, transparency, and adaptive scaffolding interact dynamically to influence developer performance and well-being. This model advances beyond previous one-dimensional frameworks by highlighting the interdependence of perceptual, cognitive, and social factors.

Practical Implications

Practitioners seeking to enhance DX should prioritize cross-toolchain affordance alignment, implement real-time diagnostic overlays, and integrate collaboration primitives within the coding environment. Organizations can leverage these insights to reduce support tickets, accelerate onboarding, and improve code quality metrics, ultimately driving higher team productivity and reduced technical debt.

Limitations

Despite rigorous within-subject counterbalancing, residual learning effects cannot be entirely excluded. The study's single-institution sample may limit generalizability; industry contexts with different tech stacks may yield variant effects. Additionally, the short-term evaluation (single session per condition) precludes assessment of longitudinal habituation or novelty effects.

Future Research

Future work should adopt longitudinal field studies to evaluate sustained impacts on developer retention, job satisfaction, and codebase maintainability. Cross-cultural investigations can elucidate how regional development norms shape DX preferences. Moreover, exploring machine-learning–

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Research Article

driven adaptive interfaces—capable of real-time personalization based on telemetry—represents a promising frontier.

CONCLUSION

This study demonstrates that an HCI-driven frontend interface significantly elevates Developer Experience by reducing task times, errors, and cognitive load while enhancing collaboration and satisfaction. Empirical validation of affordance coherence and diagnostic translucency across common frontend workflows establishes a robust foundation for user-centered toolchain design. Organizations adopting these principles can expect faster development cycles, improved code quality, and greater team engagement.

Recommendations

1. Align metaphors and feedback across all the tooling stages to minimize context switches.
2. Incorporate inline, real-time diagnostic visualizations to accelerate error identification.
3. Embed communication affordances (threaded comments, live cursors) within the IDE to streamline collaboration.

Acknowledgement

The authors extend gratitude to all participating developers at Daffodil International University for their invaluable time and insights. We also thank the CSE department for providing laboratory facilities and technical support. Special appreciation to colleagues who assisted in pilot testing and data analysis.

REFERENCES

[1] Wiltse, H. (2020). *Relating to Things: Design, Technology and the Artificial* (p. 304). Bloomsbury Academic. doi: 10.5040/9781350124288

[2] Cummaudo, A., Vasa, R., & Grundy, J. (2019, September). What should I document? A preliminary systematic mapping study into API documentation knowledge. In 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 1-6). IEEE. doi: 10.1109/ESEM.2019.8870148

[3] Floor, D. (2024). *Code comprehension in the multi-paradigm environment Kotlin* (Master's thesis, University of Twente).

[4] Anderson, O. (2024). Optimizing Software Engineering through Human-Computer Interaction Architecture. *Journal of Innovative Technologies*, *7*(1), 1-7.

[5] Kroes, T., Achterberg, H., Koek, M., Versteeg, A., Niessen, W., van der Lugt, A., ... & Lelieveldt, B. (2020, March). PIM: A visualization-oriented web application for monitoring and debugging of large-scale image processing studies. In Medical Imaging 2020: Imaging Informatics for Healthcare, Research, and Applications (Vol. 11318, pp. 62-68). SPIE. doi: 10.1117/12.2541540

[6] Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[7] Chandrasekaran, S. Enhancing Developer Experience by Reducing Cognitive Load: A Focus on Minimization Strategies. doi: 10.14445/22312803/IJCTT-V72I1P117

[8] Thongtanunam, P., & Hassan, A. E. (2020). Review dynamics and their impact on software quality. *IEEE Transactions on Software Engineering*, *47*(12), 2698-2712. doi: 10.1109/TSE.2020.2964660

[9] Barricelli, B. R., Cassano, F., Fogli, D., & Piccinno, A. (2019). End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, *149*, 101-137. doi: 10.1016/j.jss.2018.11.041

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Research Article

[10] Lewis, J. R. (2018). The system usability scale: past, present, and future. *International Journal of Human–Computer Interaction*, *34*(7), 577-590. doi: 10.1080/10447318.2018.1455307

[11] Kelleher, C., & Ichinco, M. (2019, October). Towards a model of API learning. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 163-168). IEEE. doi: 10.1109/VLHCC.2019.8818850

[12] Fronza, I., Corral, L., & Pahl, C. (2020). End-user software development: Effectiveness of a software engineering-centric instructional strategy. *The Journal of Information Technology Education: Research*, *19*, 367-393. doi: 10.28945/4580

[13] Tenner, E. (2015). The design of everyday things by Donald Norman. *Technology and Culture*, *56*(3), 785-787.

[14] Johnson, J. (2021, May). Designing with the mind in mind: The psychological basis of user interface design guidelines. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems* (pp. 1-2). doi: 10.1145/3411763.3444997

[15] Ahmed, A. (2018). Measuring developer experience of a digital platform.

[16] Tubbs-Cooley, H. L., Mara, C. A., Carle, A. C., & Gurses, A. P. (2018). The NASA Task Load Index as a measure of overall workload among neonatal, paediatric and adult intensive care nurses. *Intensive and Critical Care Nursing*, *46*, 64-69. doi: 10.1016/j.iccn.2018.01.004

[17] McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2016). An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, *21*, 2146-2189. doi: 10.1007/s10664-015-9381-9

[18] Siann, G., & Ugwuegbu, D. C. (2024). *Educational psychology in a changing world*. Taylor & Francis.

[19] Svyatkovskiy, A., Zhao, Y., Fu, S., & Sundaresan, N. (2019, July). Pythia: Ai-assisted code completion system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2727-2735). doi: 10.1145/3292500.3330699

[20] Vyas, R. (2022). Comparative analysis on front-end frameworks for web applications. *International Journal for Research in Applied Science and Engineering Technology*, *10*(7), 298-307. doi: 10.22214/ijraset.2022.45260

[21] Hicks, C. M. (2024). Psychological Affordances Can Provide a Missing Explanatory Layer for Why Interventions to Improve Developer Experience Take Hold or Fail. *Preprint*. *https://doi.org/10.31234/osf.io/qz43x*

[22] Atashi, A., Khajouei, R., Azizi, A., & Dadashi, A. (2016). User Interface problems of a nationwide inpatient information system: a heuristic evaluation. *Applied clinical informatics*, *7*(01), 89-100. doi: 10.4338/ACI-2015-07-RA-0086

[23] Bheree, M. K., & Anvik, J. (2024, April). Identifying and Detecting Inaccurate Stack Traces in Bug Reports. In *2024 7th International Conference on Software and System Engineering (ICoSSE)* (pp. 9-14). IEEE. doi: 10.1109/ICoSSE62619.2024.00010

[24] Buhagiar, A. J., Pace, G. J., & Ebejer, J. P. (2017, July). Engineering adaptive user interfaces using monitoring-oriented programming. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 200-207). IEEE. doi: 10.1109/QRS.2017.30

[25] Trong, K. N., & Ngoc, D. N. (2016). Towards a Collaborative Integrated Development Environment for Novice Programmers. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, *6*(5), 21-26.

[26] Siddiq, M. L., Dristi, S., Saha, J., & Santos, J. (2024). Quality assessment of prompts used in code generation. *arXiv preprint arXiv:2404.10155*.

[27] Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. J. (2023). Github copilot ai pair programmer: Asset or liability?. *Journal of Systems and Software*, *203*, 111734. doi: 10.1016/j.jss.2023.111734

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Research Article

[28] Peñalvo, F. J. G., Alier, M., Pereira, J., & Casany, M. J. (2024). Safe, transparent, and ethical artificial intelligence: Keys to quality sustainable education (SDG4). *IJERI: International Journal of Educational Research and Innovation*, (22), 1-21. doi: 10.46661/ijeri.11036

[29] Klumpp, M., Hanelt, A., Greve, M., Kolbe, L. M., Tofangchi, S., Böhrnsen, F., ... & Juhra, C. (2022, October). Accelerating the Front End of Medicine: Three Digital Use Cases and HCI Implications. In *Healthcare* (Vol. 10, No. 11, p. 2176). MDPI. doi: 10.3390/healthcare10112176

[30] Ko, A. J., Myers, B. A., Coblenz, M. J., & Aung, H. H. (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on software engineering*, *32*(12), 971-987. doi: 10.1109/TSE.2006.116

[31] Anderson, O. (2024). Optimizing Software Engineering through Human-Computer Interaction Architecture. *Journal of Innovative Technologies*, *7*(1), 1-7.

[32] Robillard, M. P., & DeLine, R. (2011). A field study of API learning obstacles. *Empirical Software Engineering*, *16*, 703-732. doi: 10.1007/s10664-010-9150-8

[33] Razzaq, A., Buckley, J., Lai, Q., Yu, T., & Botterweck, G. (2024). A Systematic Literature Review on the Influence of Enhanced Developer Experience on Developers' Productivity: Factors, Practices, and Recommendations. *ACM Computing Surveys*, *57*(1), 1-46. doi: 10.1145/3687299

[34] Kalyuga, S., Renkl, A., & Paas, F. (2010). Facilitating flexible problem solving: A cognitive load perspective. *Educational psychology review*, *22*, 175-186.

[35] Baddeley, A. (2019). Working memory and conscious awareness. In *Theories of memory* (pp. 11-28). Psychology Press.

[36] Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G., Serebrenik, A., Devanbu, P., & Filkov, V. (2015, April). Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems* (pp. 3789-3798). doi: 10.1145/2702123.2702549

[37] Evans, S. K., Pearce, K. E., Vitak, J., & Treem, J. W. (2017). Explicating affordances: A conceptual framework for understanding affordances in communication research. *Journal of computer-mediated communication*, *22*(1), 35-52. doi: 10.1111/jcc4.12180

[38] Wagner, S. (2015). Continuous and Focused Developer Feedback on Software Quality (CoFoDeF). *Research Ideas and Outcomes*, *1*, e7576. doi: 10.3897/rio.1.e7576

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.