From Runnable Code to Shippable Applications: Test-Driven Development for Full-Stack Web Application Generation

Yuxuan Wan, Tingshuo Liang, Jiakai Xu, Jingyu Xiao, Yintong Huo, Michael R. Lyu
Computer Science and Engineering Department, The Chinese University of Hong Kong

1. Introduction

Coding agents can generate web applications from natural-language descriptions, yet a recent study shows that generated applications **fail to meet functional requirements in over 70% of cases** [1].

Test-driven development (TDD) can close this gap, but three challenges block automation for web apps:

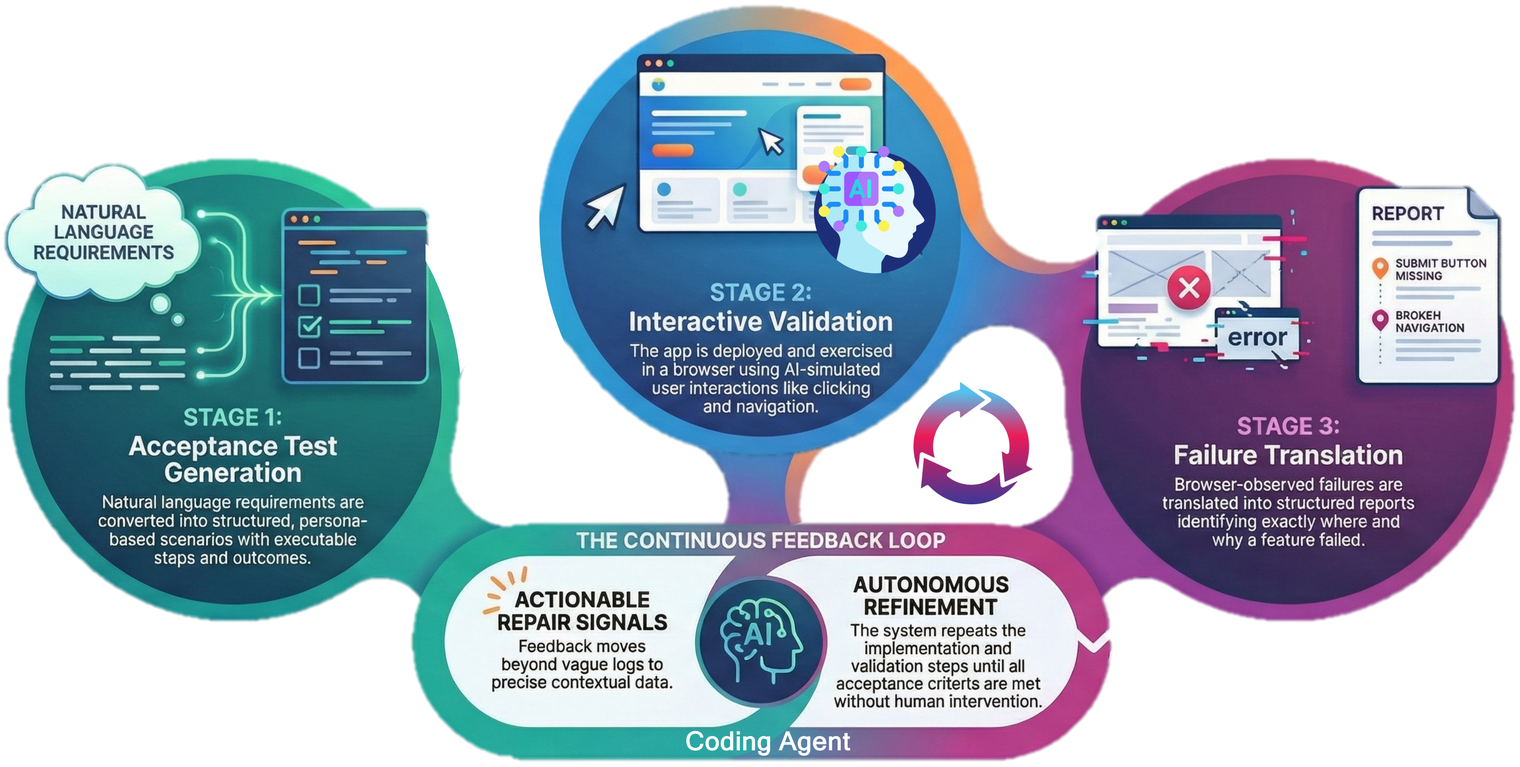
- **Requirement Concretization**: Vague NL descriptions must become executable acceptance tests.
- **Interactive Validation**: Correctness requires browser-based interaction, not just terminal output.
- **Failure Translation**: Browser-observed failures must be converted into actionable repair signals.

We present TDDev, a framework that automates all three steps, enabling coding agents to develop web apps in a closed TDD loop **with zero human intervention**.

2. TDDev Framework

TDDev automates the TDD loop through **three stages**:

- **Stage 1 — Acceptance Test Generation:** Converts NL requirements into structured test cases via persona-driven soap opera testing.
- **Stage 2 — Interactive Validation:** Deploys the app and validates it through a lightweight LLM-backed browser testing agent using Playwright.
- **Stage 3 — Failure Translation:** Converts browser interaction trajectories into structured repair reports for the coding agent.



3. Protocols & Setup

We define four protocols along the **enforcement axis**:

Protocol	Description
Incremental	One feature at a time; bounded deploy–test–repair loop per feature. *High enforcement.*
Whole-Project	Full implementation first; bounded deploy–test–repair over all tests. *Medium enforcement.*
Agentic	Agent has TDD tools but decides when/how to use them. *Low enforcement.*
Non-TDD	No TDD tools; single-pass generation. *Baseline.*

#	Agent	Model	Benchmark
1	ClaudeSDK	Sonnet 4.6	WebGen-Bench
2	ClaudeSDK	Qwen-3.5	WebGen-Bench
3	OpenCode	Qwen-3.5	WebGen-Bench
4	ClaudeSDK	Sonnet 4.6	ArtifactsBench

Metric: $\text{acc@}k = \frac{N_{\text{Pass}} + 0.5 \times N_{\text{Partial}}}{N_{\text{Total}}} \times 100\%$
Each combination runs all four protocols. Attempt budget K=5.

4. Main Results

Combination	Incremental	Whole-Proj.	Agentic	Baseline
ClaudeSDK + Sonnet 4.6	31.5	49.1	**65.8**	31.3
ClaudeSDK + Qwen-3.5	**71.4**	51.4	41.0	23.3
OpenCode + Qwen-3.5	45.7	**50.7**	27.3	11.7
ClaudeSDK + Sonnet (Art.)	81.4	**86.2**	82.9	78.6

TDD infrastructure consistently improves quality: **+34–48 percentage points** over no-TDD baseline across all WebGen-Bench combinations.

5. RQ Answers

- **RQ1 (Module Reliability):** Test generation covers **91.9%** of ground-truth features. Testing agent catches **100%** of defects with zero false positives; residual errors are conservative false negatives.
- **RQ2 (TDD Benefit):** TDD improves quality by **34–48 pp** over baseline across all WebGen-Bench combinations. Benefit scales with task complexity (+7.6 pp on simpler ArtifactsBench).
- **RQ3 (Enforcement Level):** Optimal level depends on generation style. Holistic models → **agentic**; incremental enforcement yields **no improvement**. Conservative models → **incremental**.
- **RQ4 (Feedback Rounds):** Whole-Project doubles accuracy by k=5 (most gains in first two rounds). Incremental stagnates for holistic models but improves for conservative ones (59.7%→71.4%). A single agentic attempt (65.8%) outperforms five enforced rounds for Sonnet.

6. Key Findings

- **Finding 1:** When protocol matches the model's style, TDD **more than doubles** accuracy (Sonnet+agentic: 65.8% vs. 31.3%; Qwen+incremental: 71.4% vs. 23.3%). **Enforcement level is as consequential as model choice.**
- **Finding 2:** Matched configs deliver the largest gains at the lowest cost; mismatched ones produce worst accuracy *and* highest token spend (up to 25×). **Protocol choice is the single largest lever on quality and cost.**
- **Finding 3:** TDDev eliminates all manual intervention, while baseline requires repeated developer attention. **The right metric is unattended fraction, not total time.**

7. User Study & Cost

Three developers used TDDev vs. Bolt.diy:

Method	Manual	Interv.	Prompt
Bolt.diy	4.7 / 15.2 min	3.0	74 words
TDDev	**0.0** / 18.7 min	**0.0**	**0 words**

"It removes the most frustrating part — opening the browser, figuring out what is wrong, and trying to explain it." **— Developer**

Cost efficiency:

Config	Tok. (M)	Δpp
Sonnet + Agentic	5.9	**+36.7**
Sonnet + Increm.	9.7	+0.0
Qwen + Increm.	108.7	**+46.7**
Qwen + Agentic	9.9	+16.7

Conclusions

- TDDev automates TDD for web app generation, closing the runnable–shippable gap.
- TDD improves quality by **34–48 pp** over baseline.
- Optimal protocol depends on model style: holistic → agentic; conservative → incremental.
- Protocol mismatch wastes the entire retry budget (**25**× token cost, 0 pp gain).
- User study: TDDev reduces manual intervention to **zero**.

References

[1] Z. Lu et al. WebGen-Bench: Evaluating LLMs on generating interactive websites from scratch. *arXiv:2505.03733*, 2025.