An Integrated Development and Veriﬁcation Environment for JavaScript
Christopher Schuster, Cormac Flanagan — University of California, Santa Cruz

There are many different ways to check whether a program is correct, such as testing, typechecking and static veriﬁcation.

Does this implementation match the expected behavior?
By specifying expectations with tests, types and invariants, any mismatch can be detected automatically.

Types are often too imprecise to express the expected program behavior.

integrated debugger
- Avoid brittle heuristics and automatic inference
- Display concrete counterexamples for free variables
  - Use model from SMT solver for failed veriﬁcation conditions
  - Simple values can be shown as popups
  - Complex values (such as functions) need to be synthesized

Interactive Veriﬁcation Inspector
- Show details for veriﬁcation conditions (such as assumptions and assertions)
- Enable programmers to add, remove and manipulate assumptions as part of an interactive, exploratory environment

Step-by-step debugging based on generated testcases
- Enable traditional debugging experience for veriﬁcation issues

Proposed Solution: Executable Counterexamples and Interactive Veriﬁcation Tools
Understandable and predictable veriﬁcation algorithm
- Avoid brittle heuristics and automatic inference

Interactive Veriﬁcation Inspector
- Show details for veriﬁcation conditions (such as assumptions and assertions)
- Enable programmers to add, remove and manipulate assumptions as part of an interactive, exploratory environment

Step-by-step debugging based on generated testcases
- Enable traditional debugging experience for veriﬁcation issues

Evaluate
Online user study with 18 participants who were given a tutorial of the integrated development and veriﬁcation environment followed by a series of small programming and veriﬁcation tasks and an online survey.

Response in Survey | Verification Inspector | Counterexamples | Integrated Debugger |
--- | --- | --- | --- |
Helpful | 33% | 55% | 44% |
UI Issues | 55% | 39% | 44% |
Not useful | 6% | 6% | 6% |
Impairs Development | 11% | 0% | 6% |

Implementation
Veriﬁcation conditions are translated to SMT logic. If the SMT solver ﬁnds a counter-example, it is used for automatic test generation. Finally, the veriﬁcation inspector shown above lets users interactively manipulate veriﬁcation conditions.

Try it out yourself!
esverify.org/idve