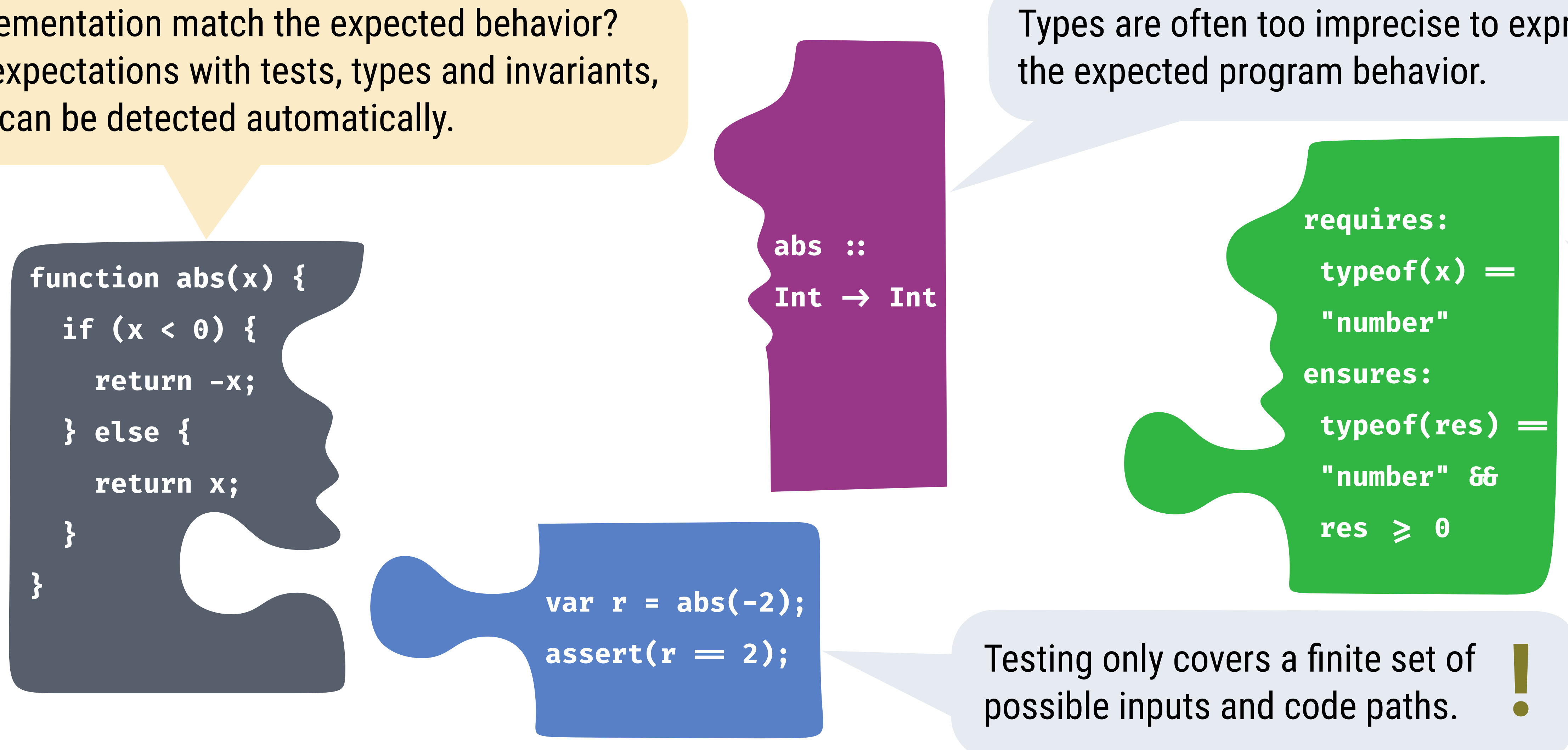
An Integrated Development and Verification Environment for JavaScript

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. !



```
function abs(x) {
  if (x < 0) {
    return -x;
  } else {
    return x;
  }
}
```

```
abs ::
Int → Int
```

```
requires:
  typeof(x) ==
  "number"
ensures:
  typeof(res) ==
  "number" &&
  res ≥ 0
```

```
var r = abs(-2);
assert(r == 2);
```

Testing only covers a finite set of possible inputs and code paths. !

Program verification enables the programmer to specify and statically check the expected behavior based on annotations such as postconditions and invariants. ✓
However, verification issues can become very difficult to understand and resolve! ✗

Proposed Solution: Executable Counterexamples and Interactive Verification Tools

Understandable and predictable verification algorithm

- 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

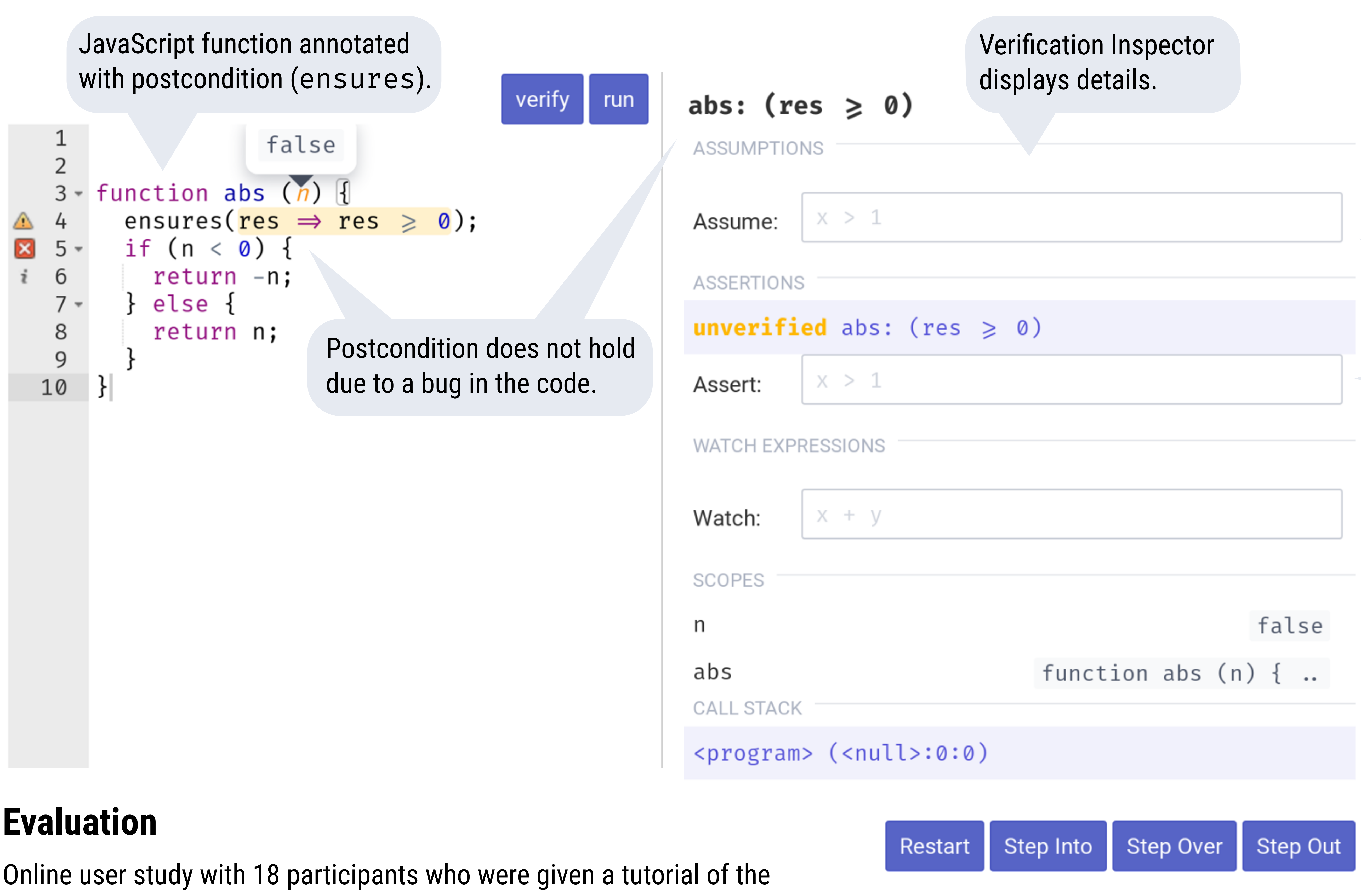
Step-by-step debugging based on generated testcases

- Enable traditional debugging experience for verification issues

JavaScript function annotated with postcondition (ensures).

Verification Inspector displays details.

Try it out yourself!
esverify.org/idve



```
function abs (n) {
  ensures(res => res ≥ 0);
  if (n < 0) {
    return -n;
  } else {
    return n;
  }
}
```

Postcondition does not hold due to a bug in the code.

abs: (res ≥ 0)

ASSUMPTIONS

Assume: x > 1

ASSERTIONS

unverified abs: (res ≥ 0)

Assert: x > 1

WATCH EXPRESSIONS

Watch: x + y

SCOPES

n false

abs function abs (n) { ..

CALL STACK

<program> (<null>:0:0)

The programmer can enter additional assumptions and assertions as boolean expressions to resolve this verification issues with interactive experimentation.

For every verification issue, an automatically generated test can be debugged using standard debugging controls such as watch expressions and step-by-step execution.

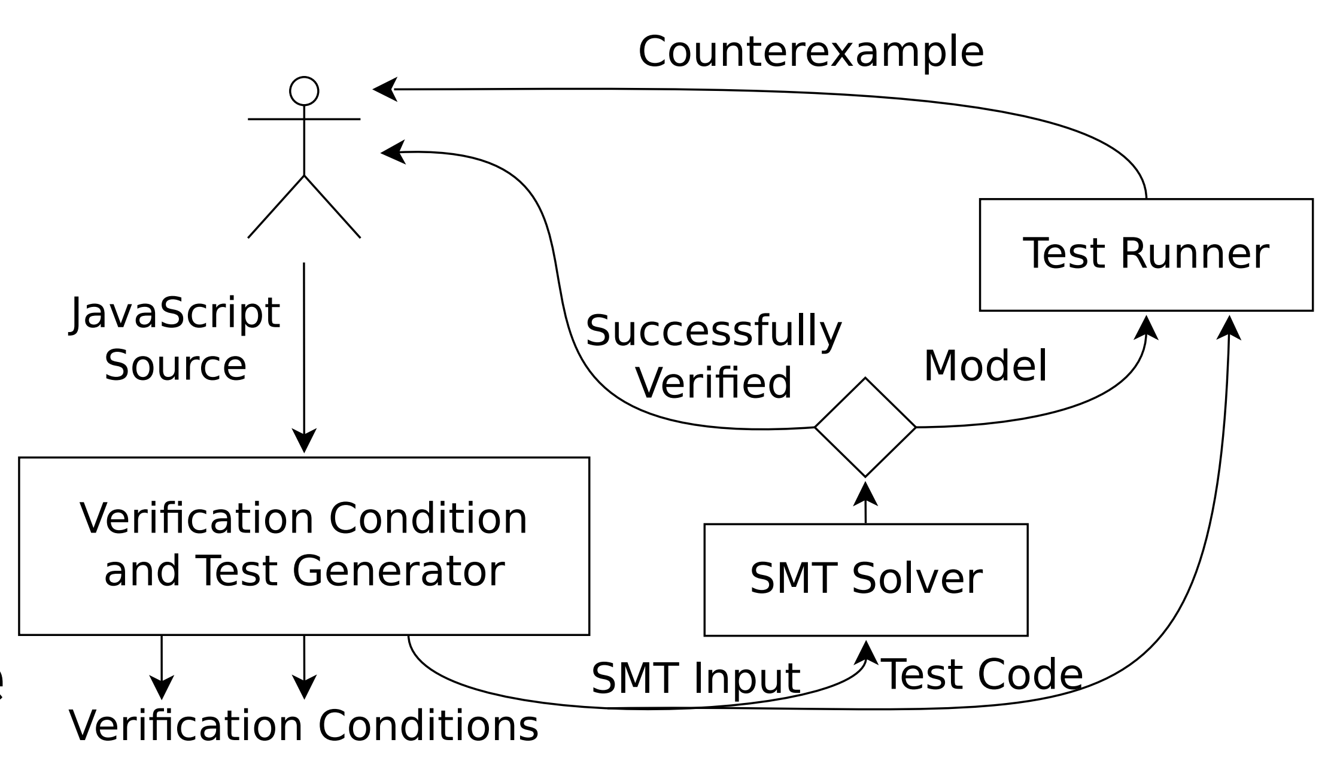
Evaluation

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	Counter-examples	Integrated Debugger
✓	Helpful	33%	55%	44%
!	UI Issues	55%	39%	44%
!	Not useful	6%	6%	6%
✗	Impairs Development	11%	0%	6%

Implementation

Verificaiton 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.



Counterexample
Test Runner
JavaScript Source
Successfully Verified
Model
Verification Condition and Test Generator
SMT Solver
SMT Input
Test Code
Verification Conditions