Do Developers Read Type Information?

An Eye-Tracking Study on TypeScript



Samuel W. Flint[1] Robert Dyer[2] Bonita Sharif[2]

[1]Beacom College of Computer & Cyber Sciences
Dakota State University

[2]School of Computing
University of Nebraska–Lincoln

34th International Conference on Program Comprehension (ICPC 2026)

Do developers read type information?

Does the presence of type information change reading behavior in other ways?

This Study

- Reading patterns in TypeScript

This Study

- Reading patterns in TypeScript
- Through eye tracking

This Study

- Reading patterns in TypeScript
- Through eye tracking
- To determine if developers read type information

TypeScript Example

```
1  let i:  number
2  let output:  string
3
4  for (i = 1; i <= 100; i += 1) {
5    output = ""
6    let divBy3:  boolean = !(i % 3)
7    let divBy5:  boolean = !(i % 5)
8
9    if (divBy3) output += "Fizz"
10   if (divBy5) output += "Buzz"
11
12   // Converts to a string because output is defined as such
13   if (!(divBy3 || divBy5)) output = i.toString()
14
15   console.log(output)
16 }
```

TypeScript Example

```
let i:  number
let output:  string

for (i = 1; i <= 100; i += 1) {
  output = ""
  let divBy3:  boolean = !(i % 3)
  let divBy5:  boolean = !(i % 5)

  if (divBy3) output += "Fizz"
  if (divBy5) output += "Buzz"

  // Converts to a string because output is defined as such
  if (!(divBy3 || divBy5)) output = i.toString()

  console.log(output)
}
```

Previous Work

- Lubin and Chasins (2021) used grounded theory: developers use type annotations as documentation

Previous Work

- Lubin and Chasins (2021) used grounded theory: developers use type annotations as documentation
- Strengthened by Endrikat et al. (2014) (types better with documentation)

Previous Work

- Lubin and Chasins (2021) used grounded theory: developers use type annotations as documentation
- Strengthened by Endrikat et al. (2014) (types better with documentation)
- Hoeflich, Findler, and Serrano (2022) (unsound sometimes preferred)

Previous Work

- Lubin and Chasins (2021) used grounded theory: developers use type annotations as documentation
- Strengthened by Endrikat et al. (2014) (types better with documentation)
- Hoeflich, Findler, and Serrano (2022) (unsound sometimes preferred)
- Mayer et al. (2012) and Fischer and Hanenberg (2015): benefit of annotations

Previous Work

- Lubin and Chasins (2021) used grounded theory: developers use type annotations as documentation
- Strengthened by Endrikat et al. (2014) (types better with documentation)
- Hoeflich, Findler, and Serrano (2022) (unsound sometimes preferred)
- Mayer et al. (2012) and Fischer and Hanenberg (2015): benefit of annotations
- Annotations improve detectability of bugs in JavaScript (Gao, Bird, and Barr 2017)

Previous Work

- Lubin and Chasins (2021) used grounded theory: developers use type annotations as documentation
- Strengthened by Endrikat et al. (2014) (types better with documentation)
- Hoeflich, Findler, and Serrano (2022) (unsound sometimes preferred)
- Mayer et al. (2012) and Fischer and Hanenberg (2015): benefit of annotations
- Annotations improve detectability of bugs in JavaScript (Gao, Bird, and Barr 2017)
- Since submission, Alznauer et al. (2026) explored impact of type annotations on reading in Python

Research Questions

RQ1 Does type annotation presence change reading behavior?

RQ2 Is there a relationship between task correctness and reading behavior with respect to type annotation presence?

RQ3 Is there a correlation between reference behavior and working memory scores?

RQ4 Do developers prefer type annotations or find them to have utility?

Research Questions

RQ1 Does type annotation presence change reading behavior?

RQ2 Is there a relationship between task correctness and reading behavior with respect to type annotation presence?

RQ3 Is there a correlation between reference behavior and working memory scores?

RQ4 Do developers prefer type annotations or find them to have utility?

Research Questions

RQ1 Does type annotation presence change reading behavior?

RQ2 Is there a relationship between task correctness and reading behavior with respect to type annotation presence?

RQ3 Is there a correlation between reference behavior and working memory scores?

RQ4 Do developers prefer type annotations or find them to have utility?

Research Questions

RQ1 Does type annotation presence change reading behavior?

RQ2 Is there a relationship between task correctness and reading behavior with respect to type annotation presence?

RQ3 Is there a correlation between reference behavior and working memory scores?

RQ4 Do developers prefer type annotations or find them to have utility?

Research Questions

RQ1 Does type annotation presence change reading behavior?

RQ2 Is there a relationship between task correctness and reading behavior with respect to type annotation presence?

RQ3 Is there a correlation between reference behavior and working memory scores?

RQ4 Do developers prefer type annotations or find them to have utility?

Tasks

Comprehension

- Four items, 56–95 lines
- Three Gists
- One from AWS DynamoDB on GitHub

Tasks

Comprehension

- Four items, 56–95 lines
- Three Gists
- One from AWS DynamoDB on GitHub

Bug Localization

- 51–594 Lines
- Two from VSCode extensions (Word Count, Julia Debugger)
- Bug descriptions from GitHub Issues/PRs
- Planted bugs in: task from Rosetta Code and custom task

Eye Tracking Tools

- iTrace with the Atom Editor
- Tobii Spectrum @ 120 Hz
- Fixations computed with iTrace Toolkit
  (IDT Filter, duration 10 ms, dispersion 125 px)

Measurements

- Experience, Education, Views Around Type Annotations (Survey)
- Time on Task
- Partial Credit Load (Partial Recall weighted by Load, Working Memory Capacity)
- Correctness (Line, Description: judged in combination with LLM)
- Fixation-Derived Metrics (Time in AOI, Fixations on AOI, Regressions to AOI)

Areas of Interest

```
let i:  number
let output:  string

for (i = 1; i <= 100; i += 1) {
  output = ""
  let divBy3:  boolean = !(i % 3)
  let divBy5:  boolean = !(i % 5)

  if (divBy3) output += "Fizz"
  if (divBy5) output += "Buzz"

  // Converts to a string because output is defined as such
  if (!(divBy3 || divBy5)) output = i.toString()

  console.log(output)
}
```

Areas of Interest

```
1  let i:  number
2  let output:  string
3
4  for (i = 1; i <= 100; i += 1) {
5    output = ""
6    let divBy3:  boolean = !(i % 3)
7    let divBy5:  boolean = !(i % 5)
8
9    if (divBy3) output += "Fizz"
10   if (divBy5) output += "Buzz"
11
12   // Converts to a string because output is defined as such
13   if (!(divBy3 || divBy5)) output = i.toString()
14
15   console.log(output)
16 }
```

Participants

- 26 participants registered
- 4 suppressed; 5 gave incomplete data

$\Rightarrow$ Total $n = 22$

- Average 4.57 ± 2.31 years programming
- Most had at least some college
- $20 \geq$ somewhat experienced with JavaScript
- $12 \geq$ somewhat experienced with TypeScript

RQ1: Changes to Reading Behavior

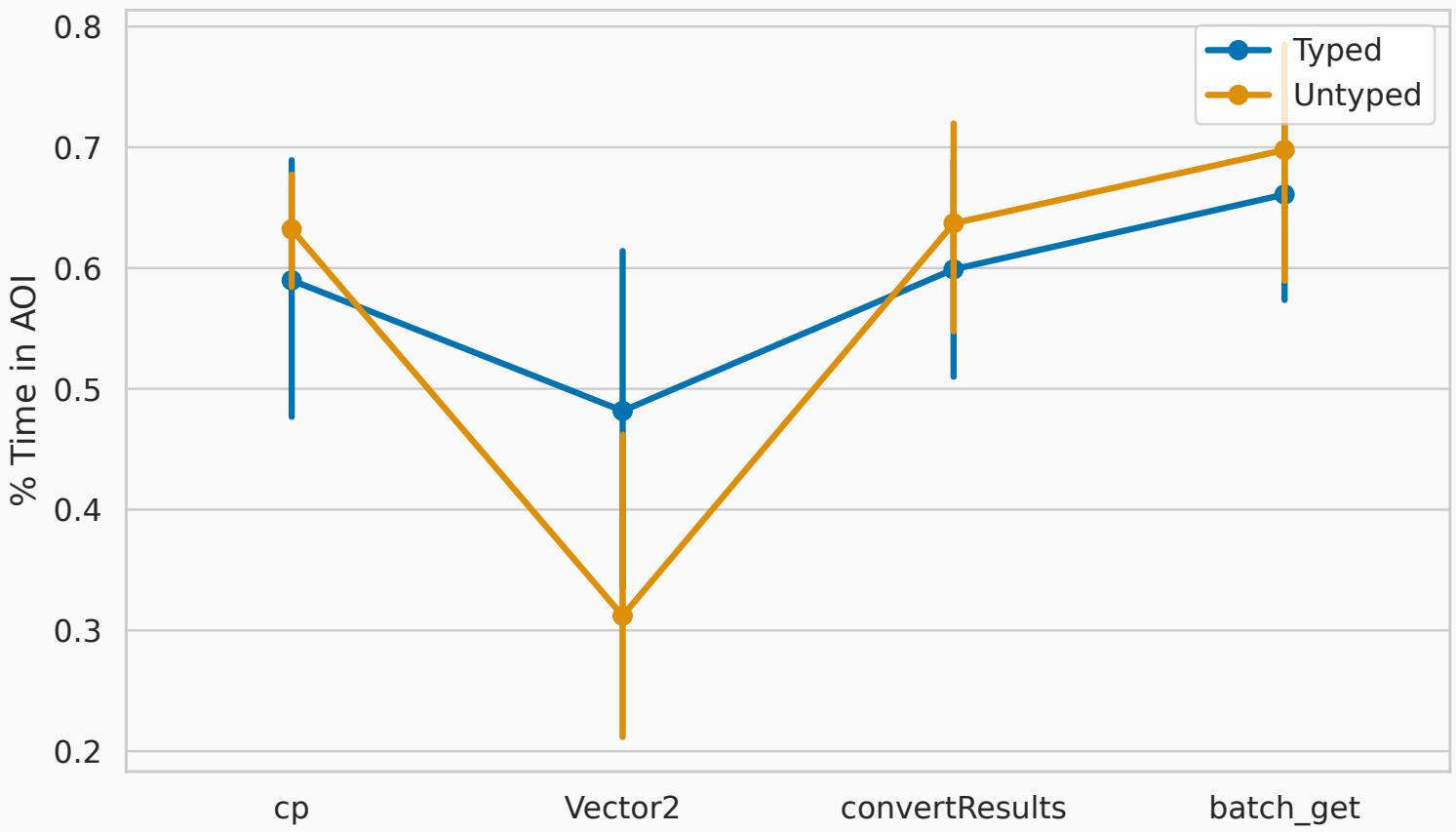

Figure 1: Differences between time spent on declaring lines in comprehension tasks. Differences are driven primarily by task, not annotation status.

RQ2: Task Correctness and Reading Behavior

Table 1: Participant correctness for localization tasks; again, the primary driver was task, not annotation status. $l/d/t$ represents l participants who selected the correct line, d who provided an accurate description of the cause of the bug, and t who saw that stimulus.

	Annotated	**Unannotated**
avl-tree	0 / 0 / 5	1 / 1 / 17
hash-table	8 / 8 / 12	5 / 6 / 10
juliaDebug	0 / 3 / 17	0 / 0 / 5
vscode-wordcount	1 / 5 / 10	6 / 8 / 12

Participants performed best on
`hash-table` and `vscode-wordcount`.

RQ3: Reading Behavior and Working Memory

- When types are shown, higher working memory capacity negates impact of types on reading behavior
- In comprehension tasks, effect is through a slight increase in time looking at declaring lines
- For debugging, the opposite holds: effect comes from a decrease in time looking at declaring lines
- However, none of the impacts are significant

RQ4: Developer Preference

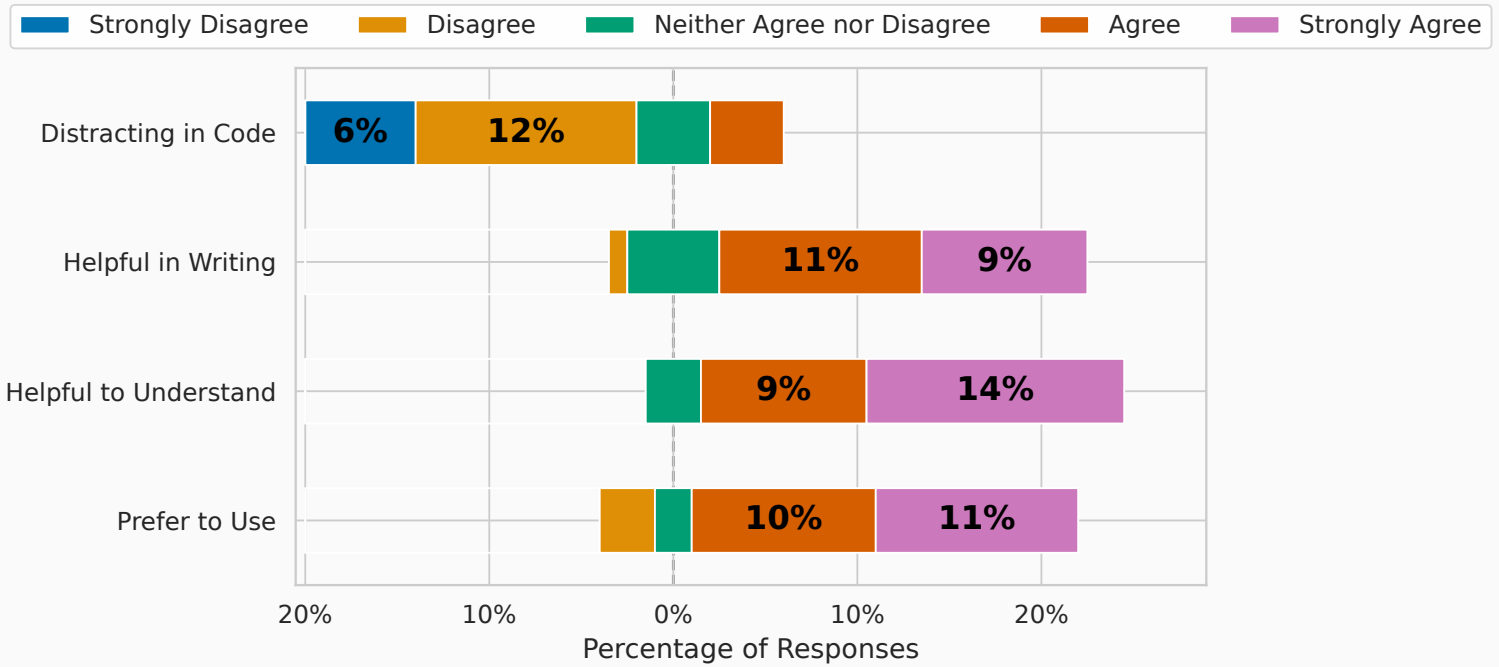

Figure 2: Summary of developer preferences. As previously noted, developers generally find type annotations useful, and not distracting.

Implications

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks
- **Developer Preference:**
  - Developers' stated preferences reinforce types as documentation, even if we cannot see it

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks
- **Developer Preference:**
  - Developers' stated preferences reinforce types as documentation, even if we cannot see it

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks
- **Developer Preference:**
  - Developers' stated preferences reinforce types as documentation, even if we cannot see it
- **For Education:**
  - Type information is implicit in languages like TypeScript or Python, it may be worth teaching students to use them early on

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks
- **Developer Preference:**
  - Developers' stated preferences reinforce types as documentation, even if we cannot see it
- **For Education:**
  - Type information is implicit in languages like TypeScript or Python, it may be worth teaching students to use them early on

Implications

- **Reference Behavior:**
  - Type annotations may not add enough additional info to impact reading
  - But, they may still act as documentation in other tasks
- **Developer Preference:**
  - Developers' stated preferences reinforce types as documentation, even if we cannot see it
- **For Education:**
  - Type information is implicit in languages like TypeScript or Python, it may be worth teaching students to use them early on

Further study remains necessary!

Summary

- Eye Tracking study on TypeScript
- Impact of Type Annotations on Reading Behavior
- Impact of Working Memory Capacity
- Developer Preferences around Types

Major Findings

1. Type annotations not frequently read; minimal impact on task correctness
2. Working memory has minimal impact on reference behavior
3. Annotations viewed positively

Scan for more information ⇒

Real World Code

- Less experimental control, but more realistic tasks
- Names *not* modified

$\Rightarrow$ Tasks did not generally require use of type info or the type hierarchy

- Tasks which require significant use of type information or relations that can be completed in a short time are hard to find