

# Code Story: Information Tracking for Program Comprehension

Felix Grund and Nick Bradley  
Department of Computer Science  
University of British Columbia  
Vancouver, BC, Canada  
ataraxie,nbrad11@cs.ubc.ca

## ABSTRACT

Software quality is strongly dependent on a developer's knowledge of the rationale that lead to code changes. Capturing this rationale would significantly improve program understanding tasks leading to higher software quality. Unfortunately, this is a hard problem because decisions are often made unconsciously and are not being tracked. This is partly due to the fact there is little tooling support and partly due to developers and managers putting little value in this information.

In this paper we describe a simple method for capturing developer reasoning during coding tasks. Our implementation, called CodeStory, integrates transparently into a developer's workflow by hooking into the standard copy-paste operation. When developers copy text or code snippets from StackOverflow, CodeStory will capture additional context and included it with the pasted content in their code. This information can later be seen by other developers giving them a better understanding of the author's rationale.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous;  
D.2.8 [Software Engineering]: Metrics—*complexity measures, performance measures*

## General Terms

Theory

## Keywords

ACM proceedings, L<sup>A</sup>T<sub>E</sub>X, text tagging

## 1. INTRODUCTION

A program is the result of decisions made based on information obtained from a variety of sources. Knowledge of these decisions can quickly evaporate if there is no process for capturing it. In practice, documenting knowledge

is often considered a resource-intensive process without tangible, short-term gains, so often it is skipped or performed inadequately[1, 2]. However, this information is useful in program comprehension tasks. In particular, we found there is strong industry support in code review.

approach \*\*introduce the information as a code story\*\*  
The contributions of this paper are as follows:

- A simple approach for capturing text-based collaborative reasoning among developers. This can include communications using email, instant messaging, websites and references.
- A prototype tool that implements this approach. It captures contextual information surrounding snippets copied from StackOverflow and stores it in a database using a Google Chrome extension. An Atom<sup>1</sup> package lets developers paste the snippet with a hyperlink to the code story.

Section 2 presents related work, ...

## 2. RELATED WORK

related-work.tex

## 3. APPROACH

approach.tex

## 4. EVALUATION

evaluation.tex

## 5. IMPLEMENTATION

CodeStory is implemented as three components: (1) a Chrome extension responsible for collecting contextual information surrounding text copied from StackOverflow, (2) a backend receiving this information and persisting it in a database, and (3) an Atom package that extends pasted snippets with a link to a page showing this information. Here we briefly describe the implementation details of each component and how they interact with each other.

### 5.1 Chrome Extension

The CodeStory Chrome extension mainly consists of a Content Script that is executed whenever a user visits a StackOverflow page. An event listener for the browser's *copy*

---

<sup>1</sup><https://atom.io>

**Table 1: Fields collected from StackOverflow upon copy**

Field Name	Description
copiedFrom	Whether the snippet was copied from a question or an answer.
type	Whether the copied snippet is code-only or a combination of code and text.
originalSelection	Exact text that was selected in the browser upon copy.
questionTitle	Title of the question of the current page.
questionUrl	URL of the question (i.e. the URL of the current page).
questionContent	Full content of the question as text.
questionContentWithHtml	Full content of the question preserving the HTML markup.
questionVotes	Number of votes the question received.
answerUrl*	URL of the answer.
answerContent*	Full content of the answer as text.
answerContentWithHtml*	Full content of the answer preserving the HTML markup.
answerVotes*	Number of votes the answer received.
accepted*	Whether the answer was accepted.
accessTime	Timestamp when the page was accessed.
fullCodeSnippet	The full code snippet (only available if the selected code is part of a code snippet).

\* Only available if copied from an answer.

*event* is added and called whenever content from the page is copied to the clipboard.

The listener first collects the required contextual information from the current page and saves it in a *storyData* object whose fields are shown in Table 1. This object is POSTed to the backend with a unique ID, the hash of the object combined with the current time, whenever a user copies content from StackOverflow. We ensure that the backend accepts cross-origin requests so that the Chrome extension can simply use *Ajax* calls.

In order for our Atom package to be able to link to the contextual information, we include the ID in the clipboard content using the JavaScript clipboard API<sup>2</sup>. The Atom package then uses the ID to form a URL which is included in the pasted content *\*\*Fig.\*\**.

## 5.2 Backend

Our backend consists of a web server implemented in Node.js using the *restify* package and a redis database. The database stores the *storyData* object using the ID mentioned in the previous section as the key. The web server provides three endpoints for creating and viewing the *storyData* object. In particular, it provides two REST endpoints, *POST /codestory/rest* and *GET /codestory/rest/:id* for creating and retrieving the object, respectively, and a *GET /* endpoint for serving html files. An example of a CodeStory web page is shown in *\*\*Fig.\*\**, which can be requested by using the URL in the CodeStory comment.

## 5.3 Atom Package

Atom is a popular, multi-platform, text editor that supports extending functionality with packages written in JavaScript. We created a simple package to provide enhanced pasting: if the clipboard content came from our Chrome extension then it will be inserted into the Atom editor with a comment above which includes a URL to the corresponding CodeStory web page. Otherwise, the standard paste operation is performed (i.e no comment is inserted). The correct commenting characters are used when pasting content into any file type supported by Atom. For example, *//*

*View code story:URL* would be inserted into a JavaScript file while *# View code story:URL* would be inserted into a Python file.

Maintaining expected paste behaviour was an important design choice that will affect developer adoption of the tool. We ens

## 6. DISCUSSION

discussion.tex

## 7. FUTURE WORK

future-work.tex

## 8. CONCLUSION

conclusion.tex

## 9. REFERENCES

- [1] O. Zimmermann, U. Zdun, T. Gschwind, and F. Ieymann, "Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method," in *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, WICSA '08, (Washington, DC, USA), pp. 157–166, IEEE Computer Society, 2008.
- [2] N. B. Harrison, P. Avgeriou, and U. Zdun, "Using patterns to capture architectural decisions," *IEEE Softw.*, vol. 24, pp. 38–45, July 2007.

<sup>2</sup><https://developer.mozilla.org/en-US/docs/Web/API/ClipboardEvent>