Code Story: Information Tracking for Program Comprehension

Felix Grund and Nick Bradley
Department of Computer Science
University of British Colombia
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's 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 include a reference to 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, LaTeX, text tagging

1. INTRODUCTION

Software is the result of decisions made based on information obtained from many sources. While software implicitly captures the results of these decisions, the rationale behind them evaporates unless some other method is used to explictly capture it. In practice, documenting knowlegde is often considered a resource-intensive process without tangible, short-term gains, so often it is skipped or performed inadaquately[1, 2]. However, as our pilot survey indicates, it is exactly this rationale that developers require to provide effective code reviews and to confidently make changes to even simple codebases.

Developers do not program in a vacuum: they use resources on the internet and colloborate with other people to accomplish a programming task. It is thhe boundary between people where there is a rich source of information as it is packaged to move from one person to another. One common instance of this is StackOverflow. Developers post technical questions about a problem they have encountered and then, after reviewing serval possible solutions contributed by other developers, they choose one that best fits their requirements. In this case, the StackOverflow web page acts as the boundary where knowledge must be packaged and its structure makes it easy to capture the rationale. Other possiblities include email, instant messaging and Google searches to look up reference material or to ask a question.

To exploit this, we take a simple approach of capturing a small amount of information every time developer copies external text into their program from a supported source. This information tells a story about the code: where did it come from, what was the developer looking, why was the developer looking (in the case of stackoverflow, the question would indicate why), how (google search, email), when. We refer to this information colloquially as a code story.

The contributions of this paper are as follows:

- **Transfer of knowledge across a bouandry*** A simple approach for capturing text-based collaborative rationale 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[1] package lets developers paste the snippet with a hyperlink to the code story.

Section 2 presents related work, ...

2. RELATED WORK

[1]https://atom.io

related-work.tex

3. APPROACH

Anything that does not directly result in usable code is an extra cost to developers. However, capturing decision rationale is important to improving future development. Thus, we want to minimize the upfront cost of capture while maximizing the future benefits of the captured information. Taking these constraints into consideration we decided on four key design decision crieteria for CodeStory:

- Minimize disruption to code flow by including only a hyperlink to the comment,
- Integrate capturing into the workflow without obstructing it
- Capture contextual information
- Present the caputured information in a meaningful way to increase its usefulness and value

3.1 Minimize Disruption

3.2 Intgerate into Workflow

3.3 Capture Context

3.4 Present

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 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 `POST`ed 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[2]. The Atom package then uses the ID to form a URL which is included in the pasted content **Fig.**.

[2]https://developer.mozilla.org/en-US/docs/Web/API/ClipboardEvent

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 retreiving 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 ensured that only clipboard content ending with the special tag, `CodeStory:<`*`HASH`*`>`, would alter the default paste behaviour. Further, we set the default key-binding to be CTRL+SHIFT+V but the user can change the binding to override the default paste command by setting the binding to CTRL+V. Alternatively, the paste operation can be invoked from the packages menu in Atom.

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. leymann, "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.

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 snipped 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.