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[**?**, **?**]. 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.

Here, we explore the decisions in detail.

3.1 Minimize Disruption

Dirty code is a pain to look at. Developers do not want to be distracted by unnecessary clutter in their code. Because our tool hooks into the copy-paste operation it is important that the additional content added be as small as possible since the operation occurs so frequently. One way we handle this is by keeping the added comment to one line. Instead of including the contextual information in the comment itself, we provide a hyperlink to a database entry with the information. Since the paste operation inserts standard co

agumenting the pasted content with comments, it is import

3.2 Intgerate into Workflow

- tooling: plugin can be made to work with any IDE (just a few lines of code) - Could be enforced by the establishment - Option to override standard paste or use a separate key binding - Once inserted, it is a standard comment so will work with any editor or code tool; extra support can be provided by IDE (e.g. showing the CodeStory next the code, or on hover)

3.3 Capture Context

- What information do you capture? (pilot survey suggests...) - Too much can be as bad as too little - Should augment/enchance what is already captured in the actual code (i.e. the result of the decisions: so should provide the rationale for the decision)

3.4 Present

- keep it close to the code (comment goes right above pasted content) - minimize interpretation work for developer (use natural langauge + sentences) - Make it viewable everywhere (webpage)

4. EVALUATION

Code reviews are an essential part of modern software engineering and we know that the quality of code reviews is a key factor to overall software quality. In order to evaluate whether CodeStory supports developers in program understanding tasks, we thus approached participants with code reviews with and without our tool (section 4.2) and analyzed whether the quality of the code reviews improved with our tool (section **??**). Before this study, however, we determined the general feasibility of CodeStory by performing a pilot survey among industry developers who perform code reviews on a daily basis (section 4.1). Threats to validity for our evaluation are described in section **??**.

4.1 Pilot Survey

At an early stage of this paper, we aimed to determine whether our assumption that it is useful to record developers' reasoning during coding tasks is true. Additionally, we wanted to collect information on what information may in fact be useful. We therefore approached 17 industry developers who regularly perform code reviews with a survey.

After providing a brief introduction of our research, we described the following scenario: *a developer has to choose a sort algorithm for a particular task. She googles 'sort array in JavaScript' and finds a code snippet on StackOverflow. She copies the snippet as a scaffold into her code.*

We then let participants rate the following question on a scale of 1 (not useful) to 5 (very useful): *How useful would the story above be during the code review?*

Finally, we collected opinions on the particular information of interest using the same scale from 1 to 5: *What elements from the above story would you consider useful?* The list of elements for the rating are shown in table 4.1.

4.2 Study Description

In order to determine the usefulness of CodeStory in practice, we approached developers with code reviews. We created a bootstrap repository on Github that contained a simple Maven Java-Project. We aimed to simulate a realistic scenario with two changes that are induced by a developer's research on StackOverflow and a subsequent copy/paste operation. We chose two questions on StackOverflow as scenarios for the study:

1. *Gson - convert from Json to a typed ArrayList<T>*[2]
2. *How do you compare two version Strings in Java?*[3]

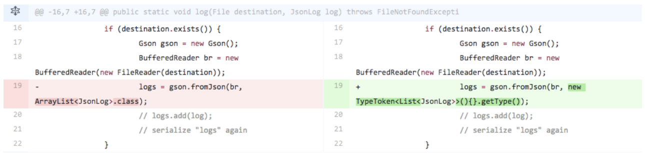
In (1), a developer was struggling with deserializing a JSON string to an instance of a custom Java class using the google-gson[4] library. The developer tries to pass a class object for the generic type `ArrayList<JsonLog>` by passing `ArrayList<JsonLog>.class` to `fromJson` (Gson's method for deserializing a JSON string), which is no valid Java code and does not compile (because class objects cannot be created from generics that way in Java). The first answer to the question, which is also the accepted answer for the question, describes how to solve this problem by using Gson's `TypeToken` class for creating the desired class object: `new TypeToken<List<JsonLog> >().getType()`. Figure 1 shows the diff of the change that we created for this scenario.

In (2), a developer was interested in how to compare version strings semantically. This question is associated with

[2]http://stackoverflow.com/questions/12384064/gson-convert-from-json-to-a-typed-arraylistt
[3]http://stackoverflow.com/questions/198431/how-do-you-compare-two-version-strings-in-java
[4]https://github.com/google/gson

Google search query leading to StackOverflow
StackOverflow question URL
StackOverflow question heading
StackOverflow question content
StackOverflow answer URL
Time of access of StackOverflow page
StackOverflow answer code snippet
Entire StackOverflow answer
StackOverflow answer rating
StackOverflow answer acceptance status
StackOverflow answer comments
Other StackOverflow answers

Table 1: Elements of interest for survey rating



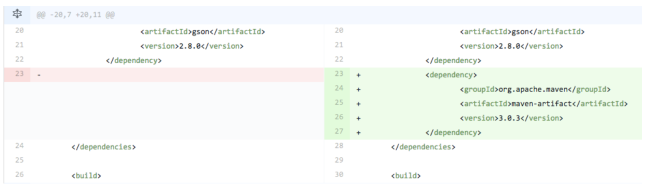
```
@@ -16,7 +16,7 @@ public static void log(File destination, JsonLog log) throws FileNotFoundExcepti
```

```
                if (destination.exists()) {
                        Gson gson = new Gson();
                        BufferedReader br = new
BufferedReader(new FileReader(destination));
-                       logs = gson.fromJson(br,
ArrayList<JsonLog>.class);
                        // logs.add(log);
                        // serialize "logs" again
                }
```

```
                if (destination.exists()) {
                        Gson gson = new Gson();
                        BufferedReader br = new
BufferedReader(new FileReader(destination));
+                       logs = gson.fromJson(br, new
TypeToken<List<JsonLog>>(){}.getType());
                        // logs.add(log);
                        // serialize "logs" again
                }
```

Figure 1: Diff view for study scenario 1

semantic versioning[5] which defines what versions should be considered higher than others, i.e. *1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-beta < 1.0.0-rc.1 < 1.0.0*. More specifically, the question was: *given two version strings a and b, which one is higher?* For this scenario, we chose a lower ranked solution[6] as a base version to be replaced by a higher ranked solution[7] as the improved version. Both solutions are answers on the same given question on StackOverflow but in this case neither of the two is the accepted answer. Figures 2 and 3 show the diff of the change that we created for this scenario.

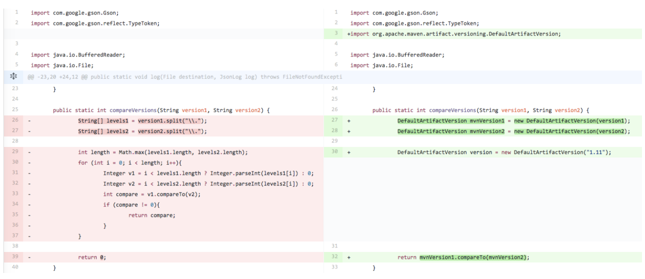


```
@@ -20,7 +20,11 @@
```

```
                        <artifactId>gson</artifactId>
                        <version>2.8.0</version>
                </dependency>
-
        </dependencies>

        <build>
```

```
                        <artifactId>gson</artifactId>
                        <version>2.8.0</version>
                </dependency>
+               <dependency>
+                       <groupId>org.apache.maven</groupId>
+                       <artifactId>maven-artifact</artifactId>
+                       <version>3.0.3</version>
+               </dependency>
        </dependencies>

        <build>
```

Figure 2: Diff view for study scenario 2 (part 1)



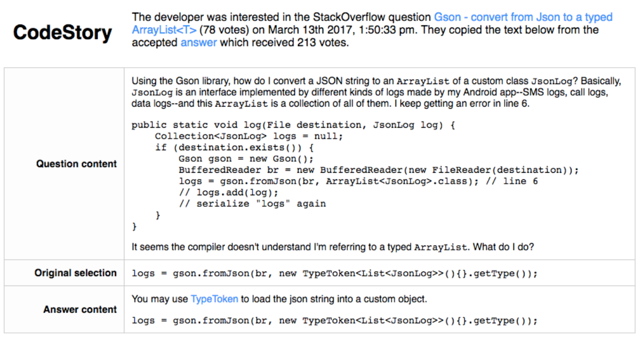
```
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.io.BufferedReader;
import java.io.File;
@@ -23,20 +24,12 @@ public static void log(File destination, JsonLog log) throws FileNotFoundExcepti
        }

        public static int compareVersions(String version1, String version2) {
-               String[] levels1 = version1.split("\\.");
-               String[] levels2 = version2.split("\\.");

-               int length = Math.max(levels1.length, levels2.length);
-               for (int i = 0; i < length; i++){
-                       Integer v1 = i < levels1.length ? Integer.parseInt(levels1[i]) : 0;
-                       Integer v2 = i < levels2.length ? Integer.parseInt(levels2[i]) : 0;
-                       int compare = v1.compareTo(v2);
-                       if (compare != 0){
-                               return compare;
-                       }
-               }

-               return 0;
        }
```

```
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
+import org.apache.maven.artifact.versioning.DefaultArtifactVersion;

import java.io.BufferedReader;
import java.io.File;

        }

        public static int compareVersions(String version1, String version2) {
+               DefaultArtifactVersion mvnVersion1 = new DefaultArtifactVersion(version1);
+               DefaultArtifactVersion mvnVersion2 = new DefaultArtifactVersion(version2);

+               DefaultArtifactVersion version = new DefaultArtifactVersion("1.11");

+               return mvnVersion1.compareTo(mvnVersion2);
        }
```

Figure 3: Diff view for study scenario 2 (part 2)

[5]http://semver.org/
[6]http://stackoverflow.com/a/27891752/1105907
[7]http://stackoverflow.com/a/6640972/1105907

For both scenarios we now added CodeStory to the developer workflow. The resulting CodeStory pages are shown in Figures 4 and 5.



CodeStory

The developer was interested in the StackOverflow question Gson - convert from Json to a typed ArrayList<T> (78 votes) on March 13th 2017, 1:50:33 pm. They copied the text below from the accepted answer which received 213 votes.

Question content

Using the Gson library, how do I convert a JSON string to an `ArrayList` of a custom class `JsonLog`? Basically, `JsonLog` is an interface implemented by different kinds of logs made by my Android app--SMS logs, call logs, data logs--and this `ArrayList` is a collection of all of them. I keep getting an error in line 6.

```
public static void log(File destination, JsonLog log) {
    Collection<JsonLog> logs = null;
    if (destination.exists()) {
        Gson gson = new Gson();
        BufferedReader br = new BufferedReader(new FileReader(destination));
        logs = gson.fromJson(br, ArrayList<JsonLog>.class); // line 6
        // logs.add(log);
        // serialize "logs" again
    }
}
```

It seems the compiler doesn't understand I'm referring to a typed `ArrayList`. What do I do?

Original selection

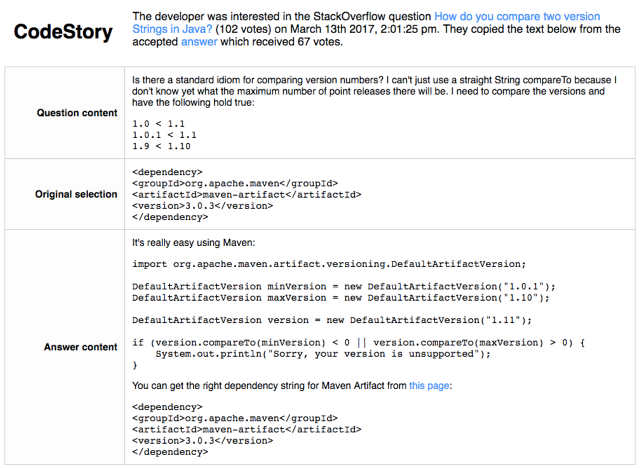
```
logs = gson.fromJson(br, new TypeToken<List<JsonLog>>(){}.getType());
```

Answer content

You may use TypeToken to load the json string into a custom object.

```
logs = gson.fromJson(br, new TypeToken<List<JsonLog>>(){}.getType());
```

Figure 4: CodeStory page for scenario 1



CodeStory

The developer was interested in the StackOverflow question How do you compare two version Strings in Java? (102 votes) on March 13th 2017, 2:01:25 pm. They copied the text below from the accepted answer which received 67 votes.

Question content

Is there a standard idiom for comparing version numbers? I can't just use a straight String compareTo because I don't know yet what the maximum number of point releases there will be. I need to compare the versions and have the following hold true:

```
1.0 < 1.1
1.0.1 < 1.1
1.9 < 1.10
```

Original selection

```
<dependency>
<groupId>org.apache.maven</groupId>
<artifactId>maven-artifact</artifactId>
<version>3.0.3</version>
</dependency>
```

Answer content

It's really easy using Maven:

```
import org.apache.maven.artifact.versioning.DefaultArtifactVersion;

DefaultArtifactVersion minVersion = new DefaultArtifactVersion("1.0.1");
DefaultArtifactVersion maxVersion = new DefaultArtifactVersion("1.10");

DefaultArtifactVersion version = new DefaultArtifactVersion("1.11");

if (version.compareTo(minVersion) < 0 || version.compareTo(maxVersion) > 0) {
    System.out.println("Sorry, your version is unsupported");
}
```

You can get the right dependency string for Maven Artifact from this page:

```
<dependency>
<groupId>org.apache.maven</groupId>
<artifactId>maven-artifact</artifactId>
<version>3.0.3</version>
</dependency>
```

Figure 5: CodeStory page for scenario 2

For the changes described with scenario 1 and 2 we now created another version for each that includes a link to the respective CodeStory page as URL in a comment above the pasted content. Since scenario 2 contained two paste operations (one in an XML file and one in a Java file), two such comments were added respectively. This constellation gave us four treatments for our study:

- **T1:** Scenario 1 without annotation

- **T2:** Scenario 2 without annotation
- **T3:** Scenario 1 with annotation
- **T4:** Scenario 2 with annotation

The study was performed with 8 developers among which 4 participants had an academic background (graduate students) and 4 had an industry background (> 4 years of practical experience with Java). With participants from different backgrounds we ensured generalizability of our study for both academia and industry. Due to time constraints we decided to assign each participant 2 code reviews (we assumed 10min for each code review). To avoid learning bias for the given scenarios and treatments, we decided to let each participant do one code review for each scenario, one with annotation and one without annotation. To further avoid learning bias, we ensured that 4 participants start with a code review with annotation and the other 4 start with a code review without annotation. Table **??** shows the outline of treatments and participants.

4.3 Impact for Code Review Tasks

4.4 Threads to Validity

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 **??**. 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[8]. 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 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

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

Participant	Treatment	Background*
P1	T3 + T2	i
P2	T3 + T2	a
P3	T1 + T4	i
P4	T1 + T4	a
P5	T4 + T1	i
P6	T4 + T1	a
P7	T2 + T3	i
P8	T2 + T3	a

* a = academia, i = industry

Table 2: Study Outline

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.

Table 3: Fields collected from StackOverflow upon copy