Supporting Coding Activity by Associating Web Bookmarks With Source Code Features

Ken Nakayama
Institute for Mathematics and Computer Science
Tsuda College
2-1-1 Tsuda-machi, Kodaira-shi
Tokyo, Japan
e-mail: ken@tsuda.ac.jp

Eko Sakai
Department of Humane Informatics
Faculty of Letters
Otani University
Koyama-Kamifusacho, Kita-ku
Kyoto, Japan
e-mail: echo@res.otani.ac.jp

Yoshihisa Nitta
Department of Computer Science
Faculty of Liberal Arts
Tsuda College
2-1-1 Tsuda-machi, Kodaira-shi
Tokyo, Japan
e-mail: nitta@tsuda.ac.jp

***Abstract*— Computing tools are often provided as various kinds of software libraries. To enjoy the benefit of the latest technologies, a user has to continuously learn their usage. During such learning activity, referring to various web articles is a common practice for programmers. Although a variety of information can be found on the web, specific information of interest tends to exist fragmented and scattered on many web pages. Therefore, bookmarking them in a well-organized way is inevitable for later use. However, manually organizing bookmarks requires extra effort for a user. To overcome the problem, a semi-automatic bookmark manager for coding related web pages is presented. A prototype system is implemented as a plug-in of an integrated development environment. The system observes bookmarking activity by a user on a web browser, and associates each bookmark with (1) features of the source code being edited, and (2) features of the source code editing in current session. The target language of prototype system is Java. User experience of the prototype is presented as preliminary evaluation.**

Keywords—programming; web bookmark; source code feature

I. INTRODUCTION

For programmers, user-contributed contents on the web, such as blogs, are major source of information about coding. It is common to use an Integrated Development Environment (IDE) and a web browser together. The coding and web referencing activities are indivisible. There exists various useful knowledge which is not included in reference manuals. For example, early adopters of some software library may report bugs on the latest version, or senior programmers may post programming tips for beginners, and so on. The fact that a large number of users continue to contribute a wide variety of articles at all times, is the source of the strength of those articles. As a result, user-contributed contents cover wide range of topics as a whole.

The fact, however, is also the cause of drawbacks. Since such articles tend to be short, unorganized, and possibly incorrect, almost all of them are not suitable for a programmer's specific situation and purpose. Therefore, the programmer has to search and gather useful web pages with effort from his own viewpoint.

As web pages that have been judged valuable are likely to be viewed again by the programmer in the future, they should be bookmarked to reduce efforts of searching them again. But in practice, coding-related bookmarking is not so easy for a programmer. Commonly used conventional tree-structured bookmarking seems to be too simple to express semantically mutually related pages. Moreover, a programmer may want to classify web pages from a viewpoint of a specific problem solved. This makes the semantic structure more complex. In other words, the semantic relation and the viewpoint of the problem solved are not necessarily orthogonal.

Although more sophisticated bookmarking methods such as the use of tags, for example [1], (this is equivalent to putting one bookmark into two or more folders) may be able to express the structure, they often need a cumbersome operation to use in practical programming. There is an attempt [2] to integrate Stack Overflow (via [3]) crowd knowledge in the IDE, but it is desirable that arbitrary web pages can be registered to bookmark.

In this paper, semi-automatic tag-based bookmarking system for coding activity is presented. Observing both an IDE and a web browser, the system extracts features (tags) from the source code being edited when a new bookmark is registered. There are some literatures, for example [4], that attempt to analyze programmer's action on IDE, but still little is known. Thus, the prototype system presented here adopted a simple static method. The bookmark is added to the bookmark table in the system together with features. Later, a programmer can retrieve recommended bookmarks from the tagged bookmarks using another source code as a query.

A prototype system has been implemented as a plug-in of IDE eclipse (via [5]). The target language is Java. The primary user interface consists of three extra buttons (`start`, `end`, and `search`) added to the IDE. The feature used in current prototype system is identifier (e.g. class name, method name) which appears in the source code.

In the next section, to introduce our motivation, coding-related bookmarks "badly" organized by a nonprofessional programmer are reviewed. We believe that most of ordinary programmers have the similar problem in handling bookmarks. Section III describes a prototype system [6] from a programmer's point of view. In Section IV, the model of bookmark registration with source code feature and edit feature is presented, then the mechanism of bookmark retrieval is explained in Section V. There are some implementation issues

 ISBN: 978-1-61208-290-5

in Section VI. Since the evaluation of the system is still under way, preliminary user experience and discussion is presented in Section VII. Finally, Section VIII concludes this paper.

II. CODING-RELATED BOOKMARKS

A. Original Bookmark List

We interviewed one nonprofessional programmer and have analyzed his bookmarks. The purpose is to grasp how the bookmark of the Web page seen during coding is classified. There were 870 programming-related unique bookmarks in the bookmarks added during the period of about two years and three months .

These were only saved in order of the addition. That is, there was no subfolder. The programmer explained the reason as follows.

- Most of these bookmarks were added during coding. During coding, he had to be concentrated on the coding activities itself. And he felt it troublesome to classify bookmarks after coding. Anyway, the bookmark was not classified.
- If bookmarks were arranged in the added order, he thought that the relevance between bookmarks could be guessed by the nearness of a time stamp. However, when the bookmark list became larger than a screen, manual search became difficult gradually. Since only the title of the Web page was saved in the bookmark list, word search was useless.

After all, he did not use the bookmark list effectively.

B. Session Segmetation

It can be considered that the bookmarks with near addition time are for the same or similar purpose. In this analysis, a series of bookmarks added within 3 hours after the last registration were defined as one session. As a result, 870 bookmarks were divided into 332 sessions. That is, it is assumed that these bookmarks were added for 332 individual purposes. During one session, 2.6 bookmarks were added on the average. The maximum was 24 bookmarks per session.

C. Free Tagging Experiment

Next, he was asked to give tags freely in order to know how the programmer recognizes each bookmark. He was instructed to give tags completely in order to make it easy to look for a bookmark for himself in the future. He was allowed to give two or more tags like ”Java / IDE / Eclipse/JDT” to the single bookmark. As a result, 63 kinds of unique tags were used. It means that 2.8 tags are contained on the average in one session (this corresponds to one purpose). The maximum was nine tags per session. Many of relations between tags were a main classification / ”sub classification” types, such as ”programming language/Scala”, for example. However, there were some relations between the tags which lack consistency. For example, although ”Java” is programming language, ”programming language” tag is omitted.

The following comments were obtained from the programmer through the interview.

- ”Java” appeared repeatedly and it was obvious that it was ”programming language.”
- Based on the estimated number of the search results by tags, he decided the criteria ”how detailed tags should be given.” The number of the bookmarks which he can look through easily is 20-30. He thought that the number of search results should not exceed this.
- He wanted to search bookmarks, using an error message, a method name, or a class name as a query. However, it was difficult to realize this manually.

III. PROTOTYPE SYSTEM

A. User Interface

From a programmer’s point of view, the system has two phases: **coding and bookmark addition** and **bookmark reference**.

The primary user interface is three extra buttons (`start`, `end`, and `search`) added to an IDE as shown in the left window of Fig. 1 (buttons are shown together with a lot of other buttons). `Start` and `end` buttons are used in the former phase to let the system know when a bookmark session starts and ends. On the other hand, `search` button is used to view recommended bookmarks in the latter phase.

B. Coding and Bookmark Registration

This phase is the same with ordinary coding and bookmarking activity except that the programmer explicitly indicates the semantic sections of editing and bookmarking activities using ”`start`” and ”`end`” buttons to the system. The system assumes a source code is being opened on IDE. A programmer is supposed to click `start` button before he starts editing with some specific intention. Once it is done, he clicks `end` button. Activities between `start` and `end` is a bookmark session. The granularity of a bookmark session is up to the programmer.

When a programmer finds some Web pages useful, he can add bookmarks to the web browser as usual. When the bookmark session ends, all bookmarks added during the session are associated with features of the source code and editing activity during the bookmark session.

Current implementation of the system requires clicking `end` button on completion of one bookmark session even if there is no bookmark registered during the session. By clicking `start` button again, the programmer may start the next bookmark session.

C. Bookmark Retrieval

When a programmer clicks the `search` button with a source code opened on the IDE, the system recommends bookmarks as shown in Fig. 2, if any, based on the code. Recommended bookmarks are listed together with “Changed Methods” and “Changed Classes” that represent edit features. The model for each phase is detailed in the following.

 ISBN: 978-1-61208-290-5

Fig. 1: `Start`, `end`, and `search` buttons on an IDE.

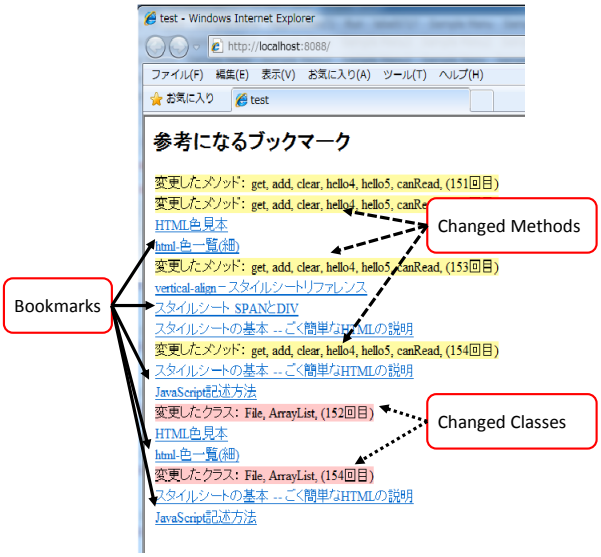

Fig. 2: Recommended bookmarks (shown on the Web browser).

IV. Bookmark Registration Using Features of Code and Coding Activity

A. Bookmark Session

A programmer usually breaks down their task into semantically coherent coding activities. “Coherent” in this context means that each activity reflects programmer’s specific editing intention. If a bookmark can be associated with a coding activity, and if features of such activity are obtained, these features can be used for bookmark classification.

To split a series of programmer’s coding interactions into semantically coherent coding activities, we ask a programmer explicitly indicate the beginning and the end of each activity. We call the period between them **bookmark session** (regardless of whether any bookmark is registered or not during the period). Bookmarks that are registered between them are associate with the corresponding coding activity.

Fig. 3 illustrates the bookmark registration in coding and bookmark addition phase. The horizontal right arrow depicts the operation time flow. Interaction and bookmark of a Web browser (Google Chrome) are shown above the arrow, while interaction and stored features of an IDE are shown below. The time period explicitly indicated by a programmer using `start` and `end` buttons is a bookmark session.

B. Source Code Feature and Edit Feature

To make such classification useful, the followings are essential: (a) the definition of features, (b) the method to infer features from a coding activity, (c) the definition of query types, and (d) the method to get matching bookmarks with a query.

Since the source code being edited may erroneous, even syntactically incomplete, features should be obtained without requiring running it. Query should be as simple as possible for a programmer in order not to disturb coding activity.

We first define **source code feature** which reflects the source code being edited or browsed. The source code feature is not directly used when registering a bookmark. Rather, we define **edit feature** as the difference of source code features at

 ISBN: 978-1-61208-290-5

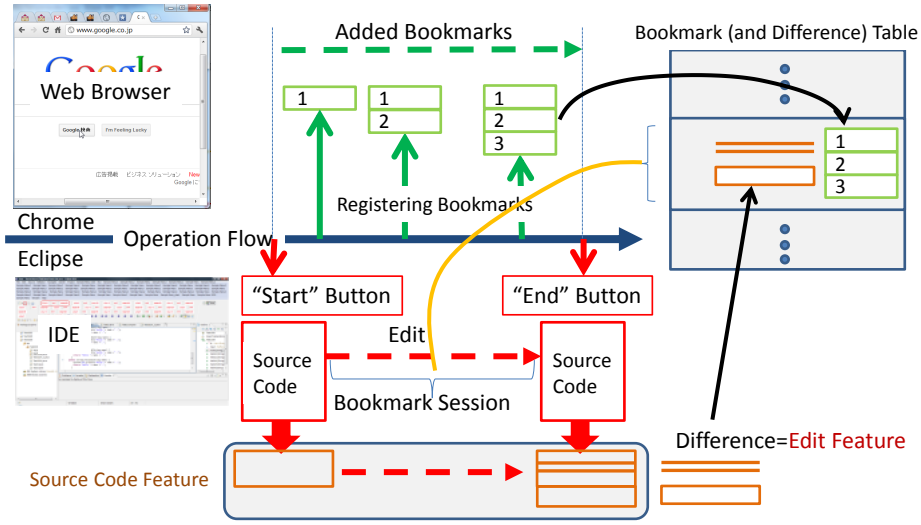

Fig. 3: The overview of the propsed system (coding and bookmark addition).

the beginning and the end of a bookmark session. We expect the edit feature reflects a programmer's coding activity.

Inherently, the best way should be determined experimentally. Various methods exist for this purpose. For example, the system may be track the cursor position or the editing point which a programmer operates. However, such real-time tracking may be difficult to realize efficiently, and may also have a problem of execution performance. Therefore, as a starting point, we have chosen simplest way for the prototype system.

C. Features Used in The Prototype

Examples of source code feature and edit feature are shown in Fig. 4.

Source code feature $F_S(c)$:
A set of the following names (identifiers of Java programming language) of a Java source file c . (1) class names and method names defined, (2) instanciated class names, and (3) method names used. The occurrence frequency (the number of appearance in the code) of each name is not used. The system statically (syntactically) analyze the code to gather these names.

Edit feature $F_E(c_1, c_2)$:
Set difference of the source code features before and after a bookmark session, that is the sum-set of both "the added name" and "the deleted name"

$$F_E(c_1, c_2) = (F_S(c_1) \cup F_S(c_2)) - (F_S(c_1) \cap F_S(c_2)) \quad (1)$$

where c_1 and c_2 are the content of the source file at the begininng and the end of a bookmark session, respectively.

D. Registering Bookmarks With Features

At the end of a bookmark session, bookmarks registered during the session are associated with edit feature, and this is added to the bookmark table (Fig. 3).

E. Behavior of The Prototype

When the `start` button is clicked, with the source code content c_1 being opened, the system performs the followings:

1) The current source code feature $F_S(c_1)$ is recorded.
2) In order to detect the bookmarks added during the editing, the bookmark list of the time is recorded. That is, already registered bookmarks are not taken into account. Such bookmarks are not shown in Fig. 3 for simplicity.

After that, a programmer may edit the source code. When a programmer thinks that he completed one semantic section of an editing, he clicks the `end` button. Suppose that the content of the source code is now c_2. Activities from `start` through `end` is a bookmark session. The system performs the followings:

1) The current source code feature $F_S(c_2)$ is recorded.
2) The edit feature $F_E(c_1, c_2)$ is calculated using recorded $F_S(c_1)$ and $F_S(c_2)$.
3) Let B be a set of bookmarks that are registered during this bookmark session. This set may be empty. The pair
$$S = (F_E(c_1, c_2),\ B) \quad (2)$$
which represents bookmarks tagged with features, where S corresponds to a bookmark session, is added

 ISBN: 978-1-61208-290-5

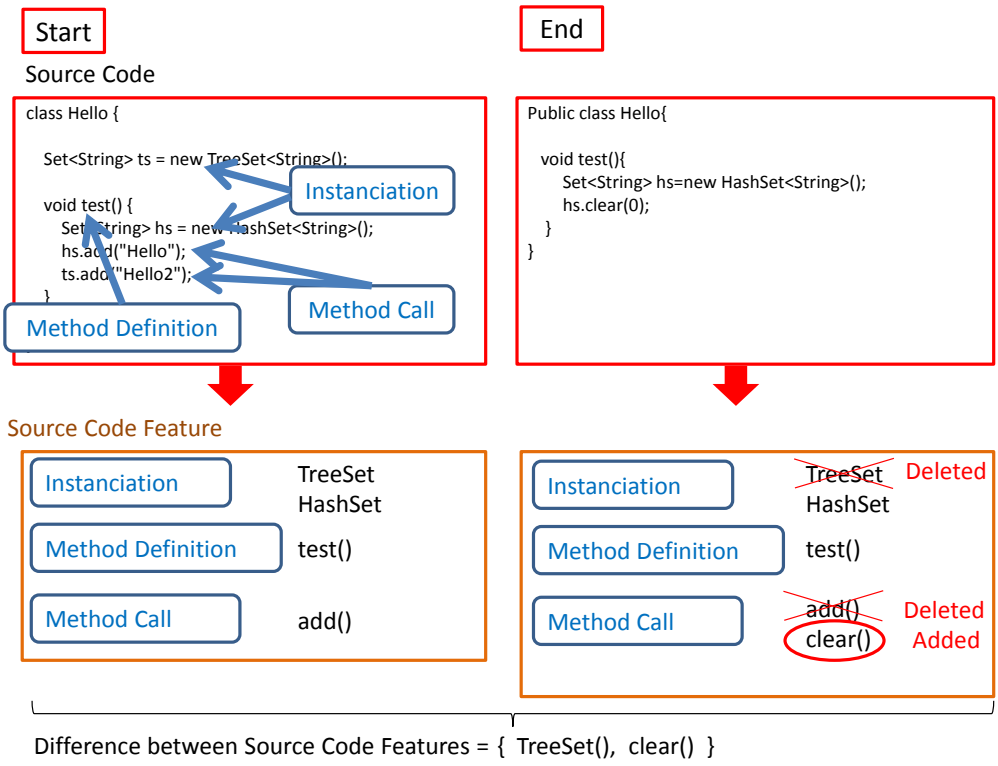

Fig. 4: Souce code feature and edit feature for the prototype.

to the bookmark table. Currently, the source code features at the time of start and end are not recorded.

V. Bookmark Retrieval Using Features

When `search` button is clicked, the recommended bookmarks are retrieved using a source code feature, not a edit feature, as a query. This is a bookmark reference phase. Suppose that N bookmark sessions

$$S_r = (F_{Er}, B_r)\ (r \in 1, \cdots, N) \quad (3$$

are stored in the bookmark table, and the current content of the source code in IDE is c. The source code feature $F_S(c)$ is compared to all stored edit features $F_{Er}\ (r \in 1, \cdots, N)$ to find the maximum match $S_k = (F_{Ek}, B_k)\ (k \in 1, \cdots, N)$ then it is presented to a programmer through a web browser (Fig. 2). This is illustrated in Fig. 5.

In the current prototype system, the similarity metric between features `match` is defined as the number of common elements as follows:

$$\text{match}(F_1, F_2) = |F_1 \cap F_2| \quad (4)$$

where F_1 and F_2 are both features.

VI. Implementation

This prototype system is implemented using Java as a plug-in of Eclipse IDE (via [5]). SQLite relational database (via [7])

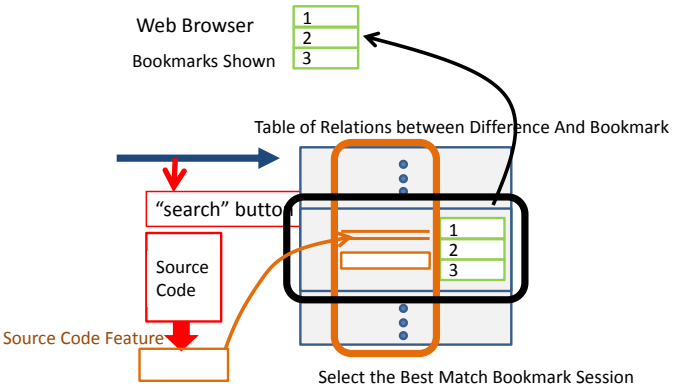

Fig. 5: The overview of the propsed system (bookmark search phase).

is used for the bookmark table. The system reads the bookmark file of Chrome (via [8]) (web browser) directly from a file system. The bookmark file is represented in JSON format (via [9]). For presenting the recommended bookmarks to a programmer, the system has a simple HTTP server as a part of the IDE plug-in.

 ISBN: 978-1-61208-290-5

The source code feature is extracted from the abstract syntax tree (AST) which represents the syntactic structure of a source code. The node of AST recursively represents syntax elements, such as method declaration or a method call.

The system uses AST generated by Java Development Tools (via [10]) (JDT) in eclipse. In JDT, each AST node is an instance of a `ASTNode` class. By defining the subclass of the `ASTVisitor` class, all the `ASTNode` can be scanned in order (or "visited"). In the subclass, `visit` methods with various AST node parameter types can be defined. A desired process can be described to the AST node class by which a `visit` method is defined.

VII. Discussion

A. Edit Feature and Source Code Feature

In the prototype system, a source code feature is used as a query for bookmark reference, while an edit feature is recorded on a bookmark addition. In this sense, the system is asymmetric.

This reflects our naive intuition in the early stage of our design. The edit feature is more specific than the source code feature. At the time of the addition of a bookmark, the more specific feature is desirable. On the other hand, to refer to the bookmark, the more robust feature is desirable. By using source code feature as a query, a programmer can query bookmarks even before editing anything, for instance, just after opening a souce file. This of course can be extended to use edit feature as a query.

Our experience shows that the definition of current edit feature seems to be too specific, because only deleted or newly created methods or classes are recognized as an edit feature. That is, modifications within the definition of existing method or class are not taken into account. The better source code feature and edit feature should be explored.

B. When Should Start and End Be Clicked?

Ideally, the bookmark session should reflect a programmer's subjective semantic chunk of an editing activity. Relying on explicit clicks of `start` and `end` seems to be working well to some degree. However, always forcing a programmer to click these buttons is an extra burden. The means to infer a bookmark session should be explored. At the same time, some way to explicitly control the recognition of a bookmark session should be provided.

C. Problems of Current Implementation

- The class name and method name used as the source code feature are not always a fully-qualified name (FQN) like `java.util.String`. If JDT provides FQN, the system uses it. If FQN is not available, a simple name (without qualification) is used. This may introduce name confusion if the same name is used in more than one context. For example, if the methods of the same name are defined in two or more classes, the system cannot distinguish them to each other.
- When extracting a feature, the order or the frequency of occurrences of a class name or a method name are not taken into consideration. This could be improved.
- In the current system, both the added method names and the deleted method names are included in the edit feature. Distinguishing these sets of names might be informative.

VIII. Conclusion and Future Work

This paper has presented a semi-automatic tag-based bookmarking system for coding activity. A prototype system has been implemented as a plug-in of Eclipse IDE. The target language is Java. Used with a web browser, the system offers a way to register and retrieve personal bookmarks of a programmer. Observing both an IDE and a web browser, the system extracts features (tags) from the source code being edited when a new bookmark is registered. The bookmark is added to the bookmark table in the system together with features. Later, a programmer can retrieve recommended bookmarks from the tagged bookmarks using another source code as a query.

The problem of inferring programmer's intention by observing coding activity is quite difficult. The current prototype system can infer the intention very roughly using simple definition of source code feature and edit feature. However, experience with the system suggested that even such rough intention might be useful for bookmarking. The better features should be explored.

Another direction of improvement is multiuser collaboration. The prototype system assumes the use of a single programmer, but the bookmark table may be shared and collaboratively used by multiple users. For example, collaborative code reading is a promising situation. Suitable method and effectiveness evaluation should be further explored.

References

[1] C. Marlow, M. Naaman, D. Boyd, and M. Davis, "HT06, tagging paper, taxonomy, Flickr, academic article, to read," in *Proceedings of the seventeenth conference on Hypertext and hypermedia*. ACM, 2006, pp. 31–40.

[2] L. Ponzanelli, "Exploiting crowd knowledge in the IDE," Ph.D. dissertation, Master's thesis, University of Lugano, 2012.

[3] Stack Overflow, "Stack Overflow," [retrieved: Jul 28, 2013]. [Online]. Available: http://stackoverflow.com/

[4] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Transactions on Software Engineering*, vol. 32, no. 12, p. 971, 2006.

[5] The Eclipse Foundation, "Eclipse integrated development environment," [retrieved: Jul 04, 2013]. [Online]. Available: http://www.eclipse.org/

[6] C. Tanaka, K. Nakayama, Y. Nitta, and E. Sakai, "Programming assistance by associating web bookmarks with source code difference," vol. 111, no. 470, Mar. 8–9, 2012, Technical Report of IEICE LOIS2011-111, pp. 231–235, (in Japanese).

[7] SQLite Consortium, "SQLite," [retrieved: Jul 04, 2013]. [Online]. Available: http://www.sqlite.org/

[8] Google, "Chrome," [retrieved: Jul 04, 2013]. [Online]. Available: http://www.google.com/chrome/

[9] JSON.org, "Introducing JSON," [retrieved: Jul 04, 2013]. [Online]. Available: http://www.json.org/

[10] The Eclipse Foundation, "Eclipse Java development tools (JDT)," [retrieved: Jul 04, 2013]. [Online]. Available: http://eclipse.org/jdt/

 ISBN: 978-1-61208-290-5