Technical Report SOCS-TR-2007.3, School of Computer Science, McGill University, June 2007

An Empirical Study of the Concept Assignment Problem

Martin P. Robillard

School of Computer Science
McGill University
Montreal, QC, Canada
{martin}@cs.mcgill.ca

David Shepherd, Emily Hill, K. Vijay-Shanker, and Lori Pollock

Computer and Information Sciences
University of Delaware
Newark, DE, USA
{shepherd, hill, vijay, pollock}@cis.udel.edu

ABSTRACT

Concept assignment involves identifying the parts of the source code associated with the implementation of a high-level concept, such as a functional requirement. Although concept assignment is at the root of a number of software engineering activities (e.g., reverse engineering, requirement traceability), we know relatively little about the characteristics of the code that different developers map to a concept. For example, for a given concept, how much do the mappings produced by different developers vary? We designed and conducted an empirical study of concept assignment. Our study resulted in the collection of three distinct mappings for 16 different concepts (for a total of 48 mappings), and produced by 23 distinct subjects. This report describes our experimental procedure and the data we collected.

1. INTRODUCTION

Concept assignment involves recognizing high-level concepts in the application domain of a software system and associating these concepts with the artifacts that implement them [2]. Automated concept assignment techniques form the foundation for a number of software engineering techniques, including feature location [4] and documentation or requirements traceability [1, 3].

At a high level of abstraction, the notion of an application-domain concept can be well-defined. For example, developers discussing a requirement to save data to persistent storage might refer to the "Saving" concept without ambiguity. However, experience and research have shown that there is seldom a precise, universally-accepted association between high-level concepts and their implementation [5]. Instead, different developers, in different circumstances, are likely to have varying interpretations of the relevance of a given artifact to the implementation of a concept. Are the differences important? In other words, are concept assignments generally consistent across developers? Do they involve only small, accidental differences or can different developers' interpretations of a concept vary drastically? How can we account for variations in developers' perspective of a concept assignment when advancing the state of the art of software development environments?

To help answer these questions, we designed and conducted an extensive empirical study of the concept assignment problem. As part of this study, we identified 16 different concepts in four different software systems, and, for each concept, we asked three different subjects to produce a concept assignment. We were thus able to analyze a total of 48 assignments, and perform a detailed analysis of the nature of the assignments, their intrinsic characteristics, and their variability.

In the remainder of this report, we refer to high-level concepts as *concerns* because we believe that this term is now more common in the literature. Given a concern c_s defined on a software system s consisting of n implementation elements $\{e_1, ..., e_n\}$, and a subject d performing the mapping, we formulate the concept assignment as the generation of a *concern mapping* $m(c_s, d) \subseteq s$.

The contributions of this report include an experimental design for the study of concern mapping (concept assignment), including the description of 16 *high-level concerns* that can be used by others for the development of concern location techniques, and detailed data on the various mappings produced for each concern.

2. EXPERIMENTAL DESIGN

The main goal of our study was to gain a better understanding of the nature of the source code that different developers associate with a given concern. Given this general question, we designed our study in a way that would allow us to investigate a maximum number of concern mappings.

2.1 Methodology

Two of the authors identified 16 concerns in four different systems developed in Java. We describe the chosen systems in Section 2.2 and the concerns in Section 2.3. For each concern, we asked three different subjects to spend some time exploring the code and to map the elements of the code relevant to the concern. The subjects could complete their assigned mapping(s) on their own time and using their own environment, or they could use our laboratory facilities (for local subjects).

We prepared each of the target systems so that it could easily be imported as a project in the Eclipse[1] integrated development environment (IDE), and asked the subjects to import the required project and *explore the source code of the system to identify the code that implements the feature [...], using the features of Eclipse that [they were] comfortable with*.[2] The subjects were not provided any starting point or advice on the process or method to use except for the following guideline: *When deciding whether to include an element or not, use the following criterion: "it would be useful to know that the element is associated with the concern if I had to modify the implementation of the concern in the future, or if another developer had to modify the implementation of the concern"*. Finally, in an attempt to obtain comparable results, we asked sub-

[1] www.eclipse.org

[2] The text in italics in this section refers to excerpts from the instructions given to subjects (see Section 6)

jects not to include *much more than 20 elements*. If more elements seemed relevant, they were asked to choose the most relevant ones.

The subjects were asked to record their mapping using ConcernMapper [6]. ConcernMapper is a simple Eclipse plug-in that allows users to associate fields or methods of classes by dragging and dropping them into a view, and to store the mapping in an XML file. The subjects were asked to provide the resulting XML file at the end of the code exploration session. The subjects were informed that they were not required to spend more than 60 minutes per mapping. The actual amount of time required to complete the mapping was ultimately left to the subjects' discretion, was not recorded, and does not constitute a variable of our study.

The participants were chosen through personal contacts of the authors. To qualify for the study, participants had to have Java programming experience and experience using the Eclipse platform. Two investigators prototyped the study by completing a mapping for the first concern. This mapping was then discarded. Any mapping produced as part of the study containing less than three, or more than forty, elements was judged invalid and rejected. Only three mappings were rejected in this way, and re-done with different subjects. Our final set of results consisted of 48 valid mappings completed by 23 distinct subjects who were not authors of this report.

Because the characteristics of the subjects did not constitute an independent variable in this study (see Section 2.4), we did not record such characteristics. We provide the distribution of subjects across concerns as part of Table 2. In this table, in each row, a concern identifier is followed by three subject identifiers starting with the letter "S". In this report, we preserved our actual subject identifiers, which do not form a complete sequence because different ranges were used by different investigators when recruiting subjects, and because of the subjects rejected as described above. The concern codes refer to the concerns described in Section 2.3. Given the coordination challenges of this large-scale study, subjects were assigned to tasks simply by assigning subjects to available tasks as they volunteered.

2.2 Target Systems

In looking for target systems for our study, we applied a number of criteria intended to facilitate the interpretation of the results and increase their validity. First, to avoid small systems that would not be representative of the software systems most developers tackle, we chose systems with a minimum size of 20,000 lines of code (LOC) and a minimum of 150 type declarations. We also looked for systems that met minimum requirements in reported bugs (150), downloads (70,000), and range of development time (2 years), so that our study would involve evolving systems with an active user community. Finally, we restricted our selection to systems that were registered with an open-source repository within the past six years to avoid antiquated code.

There exist several web sites that warehouse open-source code. We primarily searched Sourceforge[3] because it provides web interfaces to browse and search by statistics for each system. We used Sourceforge's search facility to identify all projects registered within the last six years and then used the interface's filtering capability to eliminate programs that did not meet our minimum requirements. After filtering by statistics we selected four GUI-driven applications that were reasonably easy to download and install. Based on these criteria, we selected the following four systems for our study.

[3]sourceforge.net

Table 1: Characteristics of Target Systems

Program	Version	NCLOC	Types	Methods
Gantt	2.0.2	43,246	555	3,991
Jajuk	1.2	30,676	227	1,867
JBidWatcher	1.0pre6	22,997	183	1,812
Freemind	0.8.0	70,341	617	5,388

GanttProject. An Eclipse Rich Client Platform-based application that allows users to plan projects using Gantt Charts.

Jajuk. A full-featured music player and organizer that supports a variety of audio formats such as MP3 or WAV.

jBidWatcher. A tool for bidding, sniping, and tracking bids on auction sites (for example eBay, Yahoo). Sniping refers to the action of entering a bid a very short time before the end of an auction in an attempt to prevent other bidders from re-bidding.

Freemind. An application allowing users to create mind maps (diagrams used to represent words, ideas, tasks or other items linked to and arranged radially around a central key word or idea).

The source code for the version of the four systems we used for this study is available on our research web site (see Section 6). Table 1 provides the main characteristics of our target systems, which we gathered using the Metrics plug-in for Eclipse [4] and Sourceforge.

2.3 Target Concerns

Two of the authors created a set of concerns by manually searching for high-level concepts in the bug database, user manual, and graphical user interface of the systems. We looked specifically for concerns in the application domain that subjects would have a good chance of being familiar with, that had a reasonably clear high-level definition, and that had a non-trivial mapping. Once identified by an investigator, every concern was then validated for appropriateness to the study by the other investigator. We briefly present a summary of the 16 concerns that we identified for our study. Concerns C1–C4 were defined on GanttProject, C5–C8 on Jajuk, C9–C12 on jBidWatcher, and C13–C16 on Freemind.

C1: Relationships. The feature allowing users to add a relationship between two tasks.

C2: Non-working days. The feature allowing users to specify the non-working days of the calendar (holidays and weekends) and taking these days into account when scheduling tasks.

C3: Completion. The task completion feature allowing users to specify how much of a task is completed.

C4: Undo. The mechanism allowing users to undo their actions.

C5: Play Song. The feature allowing users to play a song (or any file).

C6: Shuffle Mode. The feature allowing users to toggle between listening to tracks in sequential order or in random order.

C7: Add Song. The feature allowing users to add a song to the playlist by dragging and dropping.

C8: Sort Collection. The mechanism allowing users to sort their entire music collection according to different parameters (e.g., genre, artist, etc.).

C9: Updating Auctions. The mechanism that constantly updates the information about auctions of interest (e.g., time left).

[4]metrics.sourceforge.net

C10: Acquire Thumbnail Image. The mechanism to download and cache the thumbnail image for each auction of interest for quick display when a user mouses over that auction.

C11: Execute Bid. The feature allowing users to bid on items of interest via a dialog box.

C12: Delete Auction. The feature allowing users to delete an auction from the list of auctions of interest.

C13: Fold Node. The feature allowing users to fold nodes in a mind map so that their children are/are not displayed.

C14: Zoom Functionality. The feature allowing users to zoom in or out when viewing a mind map.

C15: Undo Create Child. The feature allowing users to undo the creation of a child node, thus deleting the new node.

C16: Autosave. The mechanism that automatically saves the mind map the user is currently modifying.

2.4 Variables and Measures

Independent Variables

Given the exploratory nature of our questions, we strove for simplicity in our experimental design to facilitate the interpretation of the results. For this reason, we chose to manipulate only two independent variables: *concerns* and *subjects*. The *concern* variable takes as value one of the 16 different concerns identified by the investigators (Section 2.3). The *subject* variable identifies which one of 23 different subjects recruited for the study completed a given mapping (see Section 2.1).

Dependent Variable and Measures

Given a concern c and a subject d, our only dependent variable is the mapping $m(c, d)$, which consists of a number of implementation elements (fields and methods). However, to facilitate our analysis of the data, we designed a number of measures that abstract specific characteristics of a mapping or of a set of mappings. The measures pertain to inter-subject agreement.

Mapping Cardinality. The number of elements in $m(c, d)$.

Union Cardinality. Given three mappings m_i defined on a given concern, the union cardinality is the cardinality of the union of all of the mappings ($|\bigcup_{i=1}^{3} m_i|$). This measure represents the total number of distinct elements identified as associated with a concern by any subject.

Intersection2 Cardinality. Given three mappings m_1, m_2, and m_3 defined on a given concern, the intersection2 cardinality is the cardinality of the set containing every element that is in at least two of the three mappings. This measure represents the number of distinct elements identified as associated with a concern by at least two of three subjects.

Intersection3 Cardinality. Given three mappings m_1, m_2, and m_3 defined on a given concern, the intersection3 cardinality is the cardinality of the set containing every element that is in all three of the mappings ($|m_1 \cap m_2 \cap m_3|$). This measure represents the number of distinct elements identified as associated with a concern by all three subjects.

Agreement. Given two mappings m_1, and m_2 defined on the same concern, the agreement is the ratio of common elements over the average of the number of elements in a mapping ($2|m_1 \cap m_2|/(|m_1| + |m_2|)$). This ratio, expressed as a percentage, represents the degree with which two different subjects agree on a mapping.

3. RESULTS

In this section, terms in *italics* refer to measures defined in Section 2.4. Table 2 presents the main characteristics of each of the 48 mappings we have collected. For each concern, we first list the identifier for each of the three subjects who produced a mapping for the concern (the three columns under the Subjects/Size header). Below each subject identifier, we present the *mapping cardinality* for the subject's mapping. For example, for concern C1 subject S3 produced a mapping containing 21 elements.

For each concern, we also provide the average *mapping cardinality* for all three subjects (Avg.), the *union cardinality* ($\cup$), the *intersection3 cardinality* ($\cap 3$), and the *intersection2 cardinality* ($\cap 2$). For example, for C2, the average cardinality of a mapping is $(24+17+30)/3 = 23.7$) and the union cardinality is 63. Only one element was mapped by all three subjects and seven elements were mapped by at least two subjects. The last column (A.A.) presents the average *agreement* for the concern. This value is obtained as the average of the *agreement* measure for all three possible pairs of subjects. For example, the subjects who produced mappings for C3 agreed on average on about 31% of their elements.

Finally, the bottom row of the table lists the average *mapping cardinality* across all 48 mappings, the average *union cardinality* across all concerns, the average *intersection2 cardinality* and *intersection3 cardinality* across all concerns, and the average *agreement* across all pairs of developers for a same concern, respectively.

4. THREATS TO VALIDITY

Several factors must be considered when analyzing the data reported. We discuss the most important of these factors in terms of four common dimensions of empirical validity [7].

Construct Validity. Our analysis assumes that the mappings produced by our subjects correctly reflect their true interpretation of the mapping between a concern and source code. In practice, for various reasons, subjects may have included elements they did not really think were relevant, or omitted elements that they thought were relevant.

Internal Validity. The internal validity reflects whether the results truly represent a causal relationship, as distinguished from spurious relationships. The most important factor potentially impacting the internal validity of our study is our decision not to consider different classes of subjects, but instead to consider all subjects as equally able to perform the task. This decision was instrumental for us to obtain a large number of mappings, but raises the possibility that characteristics of the subjects (e.g., programming experience) produce spurious effects. This possibility is mitigated by the fact that we obtained three different mappings per concern, and that many developers performed multiple tasks

External Validity. The nature of mappings between concerns and source code can potentially be impacted by a number of characteristics of the system, including the application domain (are the high-level concepts easy to understand?), the system architecture (is it well-designed?), the programming language used (object-oriented or procedural?), or the programming culture (is the code commented?). In our study, we limited our target systems to medium-size, GUI-based Java systems. Although this decision allowed us to obtain more consistent results that could be interpreted in a homogeneous context, additional investigation will be necessary to determine if our findings generalize to other classes of systems.

Table 2: Mapping Size and Agreement Measures

Crn.	Subjects/Size			Avg.	∪	∩3	∩2	A.A.
C1	S3 21	S7 14	S14 22	19.0	51	0	5	9%
C2	S3 24	S4 17	S6 30	23.7	63	1	7	12%
C3	S4 11	S7 18	S8 26	18.3	39	2	14	31%
C4	S9 7	S11 12	S40 13	10.7	25	1	6	25%
C5	S4 14	S5 9	S11 11	11.3	27	0	7	19%
C6	S4 10	S5 7	S8 18	11.7	25	1	9	32%
C7	S5 17	S8 11	S10 7	11.7	27	2	6	30%
C8	S10 8	S12 10	S13 11	9.7	20	2	5	35%
C9	S10 14	S12 20	S13 9	14.3	29	3	11	35%
C10	S14 20	S21 11	S23 9	13.3	23	6	11	61%
C11	S23 10	S24 8	S25 8	8.7	15	3	8	53%
C12	S7 17	S23 19	S26 9	15.0	27	6	12	53%
C13	S21 15	S27 13	S28 18	15.3	25	6	15	58%
C14	S21 18	S27 9	S28 28	18.3	32	3	20	45%
C15	S29 16	S31 3	S32 9	8.3	28	0	0	0%
C16	S26 10	S31 3	S32 14	9.0	15	2	10	47%
Aggregation				13.5	29.4	2.4	9.1	34%

5. CONCLUSIONS

Concept assignment, which we refer to as *concern mapping*, involves identifying the parts of the source code associated with the implementation of high-level concerns. This process is a central activity in many maintenance tasks, and numerous research projects have focused on the development of tools and techniques intended to facilitate it.

The contributions of this report include an experimental design for the study of concept assignment, the description of benchmark concerns, and their corresponding, empirically-determined mappings. Although the mappings we collected should not be considered as the "official" version of a mapping for a concern, our experimental setup nevertheless allowed us to triangulate the mappings of multiple subjects: The intersection of mappings produced by different subjects for a given concern thus provides a very good insight into what three independent developers considered relevant to a concern.

6. ON-LINE APPENDIX

Our source code, experimental instructions, and raw data can be downloaded from http://www.cs.mcgill.ca/~martin/concerns

7. REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[2] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster. Program understanding and the concept assignment problem. *Communications of the ACM*, 37(5):72–82, 1994.

[3] A. Egyed and P. Grünbacher. Automating requirements traceability: Beyond the record & replay paradigm. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, pages 163–171, 2002.

[4] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, 2003.

[5] G. C. Murphy, W. G. Griswold, M. P. Robillard, J. Hannemann, and W. Leong. Design recommendations for concern elaboration tools. In *Aspect-oriented Software Development*. Addison-Wesley, 2004.

[6] M. P. Robillard and F. Weigand-Warr. ConcernMapper: simple view-based separation of scattered concerns. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse technology eXchange*, pages 65–69, 2005.

[7] R. K. Yin. *Case Study Research: Design and Methods*, volume 5 of *Applied Social Research Methods Series*. Sage Publications Ltd., 2nd edition, 1989.