A Dataset for Evaluating Identifier Splitters

Dave Binkley†, Dawn Lawrie†, Lori Pollock*, Emily Hill‡, K. Vijay-Shanker*
†Loyola University Maryland, Baltimore, MD 21210
*University of Delaware, Newark, DE 19716
‡ Montclair State University, Montclair, NJ, 07043
{binkley, lawrie}@cs.loyola.edu, {pollock, vijay}@cis.udel.edu, hillem@mail.montclair.edu

***Abstract*—Software engineering and evolution techniques have recently started to exploit the natural language information in source code. A key step in doing so is splitting identifiers into their constituent words. While simple in concept, identifier splitting raises several challenging issues, leading to a range of splitting techniques. Consequently, the research community would benefit from a dataset (i.e., a *gold set*) that facilitates comparative studies of identifier splitting techniques. A gold set of 2,663 split identifiers was constructed from 8,522 individual human splitting judgements and can be obtained from www.cs.loyola.edu/~binkley/ludiso. This set's construction and observations aimed at its effective use are described.**

I. Motivation and Background

Program comprehension research has long been interested in the identifiers that appear in a program [1], [2], [4], [6], [12]. More recently, software tools have begun to leverage the natural language that appears in source code when addressing problems such as searching for code related to a particular topic or (re)establishing links between code and other software artifacts such as requirements and bug reports. The accuracy of these tools inherently relies on the accuracy of the underlying technique for splitting identifiers into their constituent words, abbreviations, and acronyms.

Although several techniques for identifier splitting have been developed [3], [5], [7], [8], [9], [10], [13], some forms of identifiers are challenging to all techniques. Past studies have shown that algorithms can perform well for identifiers with certain characteristics while being challenged by those with other characteristics [3], [5], [7]. For example, an algorithm might perform well on identifiers with digits but fail on multi-word identifiers of a single case (e.g., spongebob). Further evidence of the challenging nature of these problems can be found in identifiers that challenge even humans. For example, Figure 1 presents three such identifiers. Each of these examples was given to five independent human annotators, resulting in four to five distinct (i.e., unique) splittings. Given the clear challenge that splitting raises, the research community would benefit from a dataset that facilitates comparative studies of automated identifier splitting techniques.

Supporting such studies, especially those aimed at comparing a new algorithm with previous results, requires a *gold set*, specifically, a well crafted common test set of identifiers with human annotations identifying correct split locations. Diversity in the gold set leads to better external validity of the results derived using it. Thus, a balanced collection of different kinds of identifiers has value. Balance comes from including identifiers in the same proportion as found in real programs, while variety comes from including programs from several programming languages and a range of application domains.

This paper describes the construction of a dataset containing 2,733 identifiers constructed from 8,522 human splitting judgements. As with any subjective task, some identifiers are unequivocally split, while others cause differing opinions. For the latter, it is useful to have a measure of the annotator's confidence. Both agreement and confidence are taken into account in forming the gold set. While this set is primarily aimed at evaluating identifier splitting algorithms, it can be used more broadly in other mining tasks. For example, the splittings might be used to evaluate the types of identifiers that lead to the most uncertainty or they could by mined with the goal of learning how programmers understand the language contained within identifiers.

The remainder of this paper describes the methodology employed for gathering the dataset, the data's storage schema, characteristics and challenges using the dataset effectively, and lessons learned. The data can be obtained from www.cs.loyola.edu/~binkley/ludiso, which also provides programs written in Java, C, and C++ that support reading the data and extracting subsets based on different criteria (see Section IV).

II. Dataset Description

The dataset consists of two sets of split-identifiers. The first set, called the *gold set*, consists of 2,663 identifiers associated with one unique splitting. The second set, called the *raw data*, includes all the splits for each of the 2,733 identifiers. The size difference is due to the elimination of 70 identifiers where a single unique split could not be definitively determined.

The remainder of the dataset's description considers its schema and limitations from its construction. The description refers to substrings delimited by an underscore or separated by a transition from lower case to upper case as *hard words* (e.g., matchName includes the two hard words match and name). The term *soft word* refers to a word not so identified (e.g., matchname includes the two soft words match and name) [8].

A. Schema

Both identifier sets share the same schema. Each is provided in a space-separated file with all information for an identifier on a single line. The format of each line is as follows:

- the identifier's unique identification number,
- the original identifier as extracted from the source,

978-1-4673-2936-1/13 © 2013 IEEE

MSR 2013, San Francisco, CA, USA

Identifier:	defarcangpnt	wcspbrk	calcmandfpasmstart
Annotator Splittings	def arc ang pnt	wc sp brk	cal cmand fp asm start
	def arc ang pnt	wc spbr k	calc m and fp as m start
	def arcang pnt	wcsp brk	calc m and fp asm start
	defarcang pnt	wcspbrk	calc mand fp as m start
	defarcangpnt	wcspbrk	calcm and fpasm start
Correct Splitting	def arc ang pnt	w c s p brk	calc mand f p asm start
English Expansion	define arc angular point	wide character string pointer break	calculate Mandelbrot floating point assembler start

Fig. 1. Challenging identifiers with multiple unique splittings.

- the dominant language of the program that the identifier was extracted from,
- the program name from which it was extracted,
- the identifier as shown to annotators where a '-' separates hard words (i.e., a '-' denotes a hard split),
- the number of unique splittings (always 1 in the gold set, but not in the raw data),
- for each unique splitting, the identifier as split by annotators, followed by the number of annotators reporting this split (1 - 5) and then the confidence of each annotator.

B. Limitations

With the goal of providing increased external validity, the identifiers were randomly selected from open-source programs written in three different programming languages: C, C++, and Java. Thus, some caution is warranted when working with splitting algorithms designed for other languages. To obtain as many annotations as possible, annotators saw identifiers in isolation without their enclosing source-code context. Since the annotators were all volunteer programmers unlikely to have direct experience with any of the identifiers, the gold set reflects the splits of programmers new to a piece of source code. Finally, because inserting splits is subjective, at least three human judgements were obtained for each identifier. Greater detail on the collection and interpretation of the data is provided in Sections IV and V.

III. Methodology for Gathering Dataset

The subtasks for creating the dataset are identifying a set of identifiers, gathering *splitting judgements* for those identifiers, and finally, curating the data. The identifiers were extracted from a source code corpus of 2117 programs randomly selected from source forge and those used in prior studies [11]. The programs ranged in size from 1,423 to 3,087,545 LoC and covered many application domains (*e.g.*, aerospace, accounting, operating systems, program environments, games, etc.). In all, 434,392 unique C identifiers, 258,946 unique C++ identifiers, and 7,091,945 unique Java identifiers were extracted. For each language, 4,000 identifiers were randomly selected and then duplicates were removed. (This removal did not have a large impact on the sets. For example, only 5% of the C identifiers were found in the C++ collection.) Finally, the identifiers were placed in a random order.

TABLE I
Number of unique splittings per identifier.

Unique splittings	all data		non-zero confidence	
1	2,020	74%	2,071	76%
2	624	23%	598	22%
3	86	3%	64	2%
4	2	0%	0	0%
5	1	0%	0	0%

The second task is gathering splitting judgements for the identifiers. Each judgement is a description by an annotator of how to split an identifier into soft words and includes the annotator's self-rated confidence. Splitting judgements were gathered using a web-based Java applet that first provided brief instructions and then gathered the annotator's experience level. Preferring to probe annotator's untainted intuition and avoid bias, minimal instructions were given. The applet presented the annotator with a sequence of identifiers already split into hard words and asked the annotator to add or remove spaces to correctly split the identifier into its constituent soft words. Pre-splitting avoids tedium errors at the expense of potentially biasing the annotator. The annotator also rated their confidence on a scale from two for high confidence to zero for no confidence. While annotators could choose to stop at any time, they were shown at most 100 identifiers before the applet terminated. One goal in creating the oracle was to collect sufficient identifier splittings to support statistically significant conclusions. There is a tradeoff here as showing the identifier in isolation increases the number of split identifiers at the potential cost of annotation quality as the original source is not available.

An identifier was *promoted* from receiving judgements when it received three judgements with non-zero confidence or five total judgements. By the end of the data collection, 8,522 judgements of 2,733 promoted identifiers were collected during 112 different sessions. Annotator experience ranged from second year students to practicing professionals with almost fifty years of experience. The average was 13.1 years of experience with a median of 7 years. Most judgements (86.2%) were of high confidence with only 3.8% being of zero confidence. As a result, 90.2% (2,466 of the 2,733 identifiers) required only three judgements to promote, with 7.7% (211) receiving four judgements and 2.1% (56) five judgements.

The third subtask, curating the dataset, involves coping with identifiers that were split in multiple ways. Table I presents a breakdown of the number of unique splittings each identifier received. For example, the first row includes identifiers for which all judgements agreed while the second row includes those identifiers that received two distinct splittings. The central columns labeled "all data" includes all confidence levels while the right columns labeled "non-zero confidence" exclude judgements with zero confidence.

There are several possible ways to arrive at the canonical split for the gold set: from selecting the highest vote getter

where each annotator gets one vote, to selecting the split with the highest total or average confidence. The final choice for the gold set is a confidence-weighted majority score where each vote (annotator's choice) is weighted by its confidence. This score has two main advantages: (1) it helps break ties that exist when considering confidence or vote alone, and (2) it implicitly ignores splits where annotator confidence is zero. The 70 identifiers with multiple splits tying for the highest score were dropped from the gold set.

The resulting gold set provides the research community with a definitive dataset to be used to compare splitting algorithms. However, to support alternate comparisons including algorithms that consider multiple correct splittings, the raw data is provided in addition to the gold set.

IV. Dataset Characteristics & Usage Challenges

This section takes a deeper look at the gathered data. It begins by identifying three subsets of the gold set that have increased levels of confidence. These are used to investigate six characteristics of the dataset: language balance, the need for splitting, the impact of programming language, average confidence, the relation between style and confidence, and finally split removals. Each of these investigations concludes with a *usage challenge* highlighting how aspects of the dataset can be exploited to investigate key challenges for identifier splitting algorithms.

The dataset has sufficient size to consider three key subsets: *2UniqueSplits*–identifiers with two unique splittings where the evaluation might consider either of the two answers as correct (598 identifiers), *UniqueSplit*–identifiers with one unique splitting (2,071 identifiers), and *HC-core*–the highest-confidence core of UniqueSplit (1,758 identifiers), which contains those identifiers with one unique splitting where all judgements received the highest confidence score.

Beginning with language balance, one of the goals of the gold set's construction was a balance between the three languages C, C++, and Java. Statistically, balance was achieved in the overall data. Formally a χ^2 proportions test finds no difference in the proportion of identifiers from each of the three languages. This is also true in UniqueSplit and its high-confidence core, HC-Core. However, it is not true of 2UniqueSplits where there are fewer C++ identifiers ($p < 0.0001$). Thus, care should be taken in drawing conclusions about the uniformity of the C++ identifiers when using 2UniqueSplits.

Having attained reasonable balance across the three languages, the next question deals with the need to split identifiers beyond hard words. The results are summarized in Table II, which presents counts and percentages of the number of identifiers with annotator-created soft words. The counts are included to provide some measure of the practical significance. From the last row in the table, the overall percentage of identifiers that required further splitting is 28% for all identifiers, 22% for UniqueSplit, and 59% for 2UniqueSplits. Note that each identifier from 2UniqueSplits is counted twice (once for each unique split). One of the two must differ from the original, so the percentage modified will always be greater than 50%. This makes comparison with UniqueSplit meaningless. The comparison of HC-Core's 13% having splits added (not shown in Table II) with UniqueSplit's 22% and all identifiers 28% supports the observation that greater splitting comes with lower confidence.

The third issue investigated is the impact of programming language on the number of soft words. For 2UniqueSplits, there is no statistical difference in the number of identifiers requiring additional splits between languages. However, UniqueSplit hard words from C identifiers receive marginally more splittings than C++ or Java ($p = 0.044$ and $p = 0.0045$). In addition, hard words from C identifiers received more splittings than C++ ($p = 0.048$, but not Java, $p = 0.104$). Given the age and history of C programming, this pattern is to be expected [11].

Equally interesting is the average confidence of identifiers that went unchanged compared to those into which annotators inserted one or more splits. Here an unchanged splitting always received a higher average confidence. This difference is statistically significant for all the data and those identifiers receiving one or two unique splittings ($p < 0.0001$). It is not significant for the 86 identifiers with three unique splittings or the three identifiers having four or five splittings. The lower confidence when inserting a split may indicate a hesitancy of annotators to make changes. If this is the case, the gold set underestimates the need for splitting.

Another difference in confidence comes from the style (camel case or underscore) of the identifier. Of the 2733 identifiers retired 1158 (42%) include mixed case but no underscore (and are thus assumed to use the camel-case style), while 915 (33%) include a single case and at least one underscore. The average confidence of the 1158 is 1.87 while that of the 915 is 1.84. Ignoring judgements with zero confidence these averages rise to 1.92 and 1.90, respectively. One implication here is that camel-case identifiers are accompanied by higher overall splitting-judgement confidence. This suggests that same-case identifiers are harder to split. Further evidence for this observation comes from the two identifiers receiving four unique splittings and the one receiving five unique splittings, which are all lower case with no underscores.

Finally, considering the splits *removed* by annotators, only 73 identifiers (2.7%) had a hard split removed. Just over half (41) of these identifiers involved digits. The hard splitting rule for digits separates strings of digits from surrounding letters. This correctly splits an identifier such as err2string into the three hard words err 2 string, which corresponds to the natural language phrase *error to string*. Other cases should only be split on one side such as play3DMovie (break before) and mpeg4player (break after). Finally, some require no spits such as the V4L2 in V4L2_CAP_TIMEPERFRAME. For identifiers not involving digits, the annotator's removal contradicted the hard word rules. For example, the underscore in CTL_HOME leads to the two hard words CTL and HOME, which were joined together by an annotator to produce CTLHOME. In summary, the removed splits suggest a need for splitting algorithms that focus on splits in and around digits [3].

TABLE II
NUMBER OF SPLITS INTRODUCED OVERALL, BY LANGUAGE, AND BY LEVEL OF UNIQUENESS.

	All Identifiers (Gold Set)			One Unique Splitting			Two Unique Splittings		
Language	total	split added		total	split added		total	split added	
C	916	281	(31%)	682	173	(25%)	406	234	(58%)
C++	906	240	(26%)	706	147	(21%)	364	220	(60%)
Java	911	248	(27%)	683	142	(21%)	426	250	(59%)
All	2,733	769	(28%)	2,071	462	(22%)	1,196	704	(59%)

V. LESSONS LEARNED

In gathering the identifier splitting data and using it to compare identifier splitting techniques, four key observations can be made. They relate to working with a random sample, facilitating annotations, collecting annotator confidence, and determining canonical splits for the gold set.

Random Sampling. Randomly sampling C, C++, and Java programs allows the distribution of the dataset to mirror that of real programs and thus provides better external validity. However, some types of identifiers occurred too infrequently to draw statistically significant results. In the future, one might augment the gold set with more instances of three types of identifiers: those with digits, those requiring no splits, and those that contain splits between two letters of the same case. Interestingly, the same-case splits appear to be especially challenging for existing identifier splitting techniques.

Facilitating Annotations. The goal in creating the gold set was to collect sufficient identifier splittings to support statistically significant conclusions. To collect a sufficient volume of split identifiers requires reducing annotator workload to the extent possible. Thus, annotators saw an isolated identifier, without its source code context. In addition, the identifier was shown pre-split into hard words. The intent was to reduce the annotator's workload so that they could focus on identifying the more challenging splits and help avoid mistakes due to boredom. However, pre-splitting hard words can potentially bias annotators in favor of default hard word splits. Furthermore, showing the identifier in isolation potentially reduces annotation quality. This trade-off between dataset size, annotator workload, and overall annotation quality represents a classic tradeoff when building any such gold set.

Collecting Annotator Confidence. Utilizing self-reported confidence information helps choose between different splits. However, since the confidence is self-reported, the scale is dependent on personal opinion and thus varies among subjects. In the future, it would be interesting to study the inter-annotator consistency of confidence scores. This information could be mined from the raw data. Future collections might include some kind of confidence control to make confidence more comparable across subjects.

Determining Canonical Splits. After data collection, it was noticed that some identifiers would have benefited from additional annotations aimed at attaining a definitive splitting. Recall that 70 of the 2,663 identifiers in the raw data were dropped from the gold set due to insufficient annotator confidence and agreement. In hindsight, more than five judgements per identifier could have been collected for these difficult cases. Such data is of particular interest as these identifiers are some of the most interesting in exposing differences between identifier splitting techniques.

VI. SUMMARY

In summary, the presented dataset, including the raw data and the gold set, form a valuable resource for comparing and designing tools and techniques that investigate and exploit the natural language found in program identifiers.

VII. ACKNOWLEDGMENTS

Thanks to the annotators and statistician Chris Morrell. Support provided by NSF CCF 0916081 and NSF CCF 0915803.

REFERENCES

[1] Nicolas Anquetil and Timothy Lethbridge. Assessing the relevance of identifier names in a legacy software system. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 4. IBM Press, 1998.

[2] D. Binkley, D. Lawrie, S. Maex, and C. Morrell. Identifier length and limited programmer memory. *Sci. Comput. Program.*, 74(7):430–445, 2009.

[3] Simon Butler, Michel Wermelinger, Yijun Yu, and Helen Sharp. Improving the tokenisation of identifier names. In *Proceedings of the 25th European conference on Object-oriented programming*, ECOOP'11, pages 130–154, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] Bruno Caprile and Paolo Tonella. Nomen est omen: Analyzing the language of function identifiers. In *WCRE '99: Proceedings of the 6th Working Conference on Reverse Engineering*, pages 112–122, 1999.

[5] A. Corazza, S. Di Martino, and V. Maggio. Linsen: An approach to split identifiers and expand abbreviations with linear complexity. In *IEEE International Conference on Software Maintenance*, ICSM '12, Washington, DC, USA, 2012. IEEE Computer Society.

[6] Florian Deissenboeck and Markus Pizka. Concise and consistent naming. *Software Quality Control*, 14(3):261–282, 2006.

[7] Eric Enslen, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009*, 0:71–80, 2009.

[8] Henry Feild, David Binkley, and Dawn Lawrie. An empirical comparison of techniques for extracting concept abbreviations from identifiers. In *Proceedings of IASTED International Conference on Software Engineering and Applications (SEA'06)*, November 2006.

[9] L. Guerrouj, M. Di Penta, G. Antoniol, and Y. Guéhéneuc. Tidier: an identifier splitting approach using speech recognition techniques. *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

[10] D. Lawrie, D. Binkley, and C. Morrell. Normalizing source code vocabulary. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 3 –12, oct. 2010.

[11] D. Lawrie, H. Feild, and D. Binkley. Quantifying identifier quality: An analysis of trends. *Journal of Empirical Software Engineering*, 12(4), 2007.

[12] Ben Liblit, Andrew Begel, and Eve Sweetser. Cognitive perspectives on the role of naming in computer programs. In *Proceedings of the 18th Annual Psychology of Programming Workshop*, 2006.

[13] N. Madani, L. Guerrouj, M. Di Penta, Y. Gueheneuc, and G. Antoniol. Recognizing words from source code identifiers using speech recognition techniques. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 68 –77, march 2010.