Analyzing the Evolution of the Source Code Vocabulary

Surafel Lemma Abebe[1], Sonia Haiduc[2], Andrian Marcus[2], Paolo Tonella[1], Giuliano Antoniol[3]

[1]*FBK-irst*
38050 Povo, Trento, Italy

[2]*Dept. of Computer Science*
Wayne State University
Detroit, MI, USA

[3]*Dépt. de Génie Informatique*
École Polyt. de Montréal
Montréal, Quebec, Canada

surafel@fbk.eu; sonja@wayne.edu; amarcus@wayne.edu; tonella@fbk.eu; antoniol@ieee.org

Abstract

Source code is a mixed software artifact, containing information for both the compiler and the developers. While programming language grammar dictates how the source code is written, developers have a lot of freedom in writing identifiers and comments. These are intentional in nature and become means of communication between developers.

The goal of this paper is to analyze how the source code vocabulary changes during evolution, through an exploratory study of two software systems. Specifically, we collected data to answer a set of questions about the vocabulary evolution, such as: How does the size of the source code vocabulary evolve over time? What do most frequent terms refer to? Are new identifiers introducing new terms? Are there terms shared between different types of identifiers and comments? Are new and deleted terms in a type of identifiers mirrored in other types of identifiers or in comments?

1. Introduction

Source code is often the main software artifact used by developers during maintenance. It helps developers understand how the system works and what it does. The identifiers and the comments are the elements of the source code that give clues about the semantics and intent of source code entities (*i.e.*, what is their purpose).

The vocabulary of a software system is composed of all the words (or terms) used in constructing identifiers and writing comments or other documents. The importance of identifiers and comments in the source code have been recognized by previous research [2, 6-8, 11, 20, 21] Hardly any work addressed the evolution of the source code vocabulary and the findings so far are that "the evolution of the [software] lexicon is more limited and constrained than the evolution of the [software] structure" [3]. The goal of this work is to give a more detailed view of source code vocabulary evolution and expose facts about this process that were not previously studied. Specifically, we define the source code vocabulary as a union of smaller entities, *i.e.*, the identifiers vocabulary and the comments vocabulary. We study the evolution of each vocabulary with respect to each other and with respect to the system size.

The paper reports on an observational case study [1], which provides answers to four research questions:

RQ1: *How does the size of the source code vocabulary evolve over time?* In answering this question we investigated the relationship between the size of the vocabulary (*i.e.*, the number of all, new, and deleted terms) and the size of the source code.

RQ2: *What are the relationships between individual vocabularies that form the source code vocabulary?* We analyzed the commonalities between each vocabulary and whether the new and deleted terms are shared among them. The goal was to assess how individual vocabularies contribute to the overall evolution of the source code vocabulary.

RQ3: *Are new identifiers introducing new terms?* As individual vocabularies are not created the same way, we studied how the identifier vocabulary reflects the growth of the source code vocabulary.

RQ4: *What do the most frequent terms refer to?* Developers rarely have constraints in choosing what they should reflect in comments or identifiers (*e.g.*, domain concepts, design decisions, solution outlines, etc.). We analyzed the most frequent terms in each vocabulary and established how many are from the problem domain.

The main contribution of the paper is the knowledge we derived from the case study, which provides new insights into how the source code vocabulary evolves. Our long term objective is to develop tools that support developers managing and evolving the source code vocabulary in a consistent way, in order to make the vocabulary knowledge explicit and available to the other developers.

1534-5351/09 $25.00 © 2009 IEEE
DOI 10.1109/CSMR.2009.61

2. Case Study

In order to answer the four research questions, we designed an exploratory case study, following the guidelines described in [22]. Case studies are a type of empirical studies used also in software engineering, along with experiments, surveys, etc. [16]. They use primarily qualitative analysis to understand complex phenomena or test theories and are typically used when "*how*", "*why*", "*what*", etc. questions are being posed [22]. Case studies are particularly preferred over experiments in complex, real-life settings where there is little control over the variables.

In our case study, we analyzed the history of two open-source systems and we measured many attributes of the considered vocabularies. This section reports the most interesting findings, but it does not contain all the measurements and statistics we computed. The complete results are available in [1].

2.1. Objects of the study

The two software systems analyzed in our case study are ALICE (http://aliceinfo.cern.ch/) and WinMerge (http://www.winmerge.org/), both medium-sized software systems written in C++.

ALICE is an open-source software that provides functionalities for particle simulation and trajectory reconstruction and analysis, developed by the European Organization for Nuclear Research (CERN). The development of ALICE is being carried out in a geographically distributed environment. To make the code easily understandable and maintainable, a set of coding conventions was adopted; this is classified in four major categories: Naming, Coding rules, Style rules, and Guidelines. Every organization or institute delivering source code for the project is required to verify the code's compliance with the conventions. To facilitate this task a rule checker tool was implemented, which checks the code's compliance with the ALICE coding conventions and reports any violations.

The ALICE code is still evolving and it has currently reached version 4.15. In our case study, we focused on eight main packages: MUON, PHOS, STEER, PMD, RICH, TOF, TPC, and TRD. We used eight main releases of ALICE for the case study, namely: v3-05-01, v3-06-01, v3-07-01, v3-08-01, v3-09-01, v3-10-00, v4-01-00, and v4-02-00. The average time between the considered releases is approximately six months. The size of ALICE in lines of text for all the versions considered is shown in Table 1.

WinMerge is a differencing and merging tool for Windows which compares both folders and files, presenting the differences in a visual text format. It was first released in 2000 and currently has an active team of nine developers. The lead developer of WinMerge has been with the team since 2003 and has five times more commits than anyone else in the project; in consequence, the lead developer's habits have a great influence on how the source code is written and how identifiers are named and comments introduced in WinMerge. Also, the web page of the project lists a set of coding guidelines which are recommended (but not enforced) to all developers contributing to WinMerge. These guidelines include also instructions referring to comments; they recommend commenting every software entity, answering *why* and not *what* questions in comments and keeping the comments up-to-date.

WinMerge has reached a state of stable and mature codebase over the eight years since its release. For our case study, we used the entire source code of WinMerge and chose five versions: v2.2.0, v2.4.0, v2.6.0, v2.8.0, and v2.8.6, which are major stable releases of WinMerge and the latest available version at the time of the case study. The average time between the releases considered for the case study is eight months. The size of WinMerge in lines of text for all the versions considered is shown in Table 1.

2.2. Vocabulary definition

We consider the *source code vocabulary* (SV) as the union of five different vocabularies: the *class name vocabulary* (CV), the *attribute name vocabulary* (AV), the *function name vocabulary* (FV), the *parameter name vocabulary* (PV), and the *comment vocabulary* (CoV). Each of them represents, respectively, the set of unique terms that appear in class names, in attribute names, in function names, in parameter names, and in comments in the source code of a system. We refer to the elements of each vocabulary as *terms* (rather than *words*) since many of them are not proper words from a spoken language. We refer collectively to CV, AV, FV, and PV as *identifier vocabularies*.

2.3. Vocabulary construction and data collection

In order to extract the vocabularies and to compute the statistics used for answering the research questions,

Table 1. The number of lines of text in ALICE and WinMerge for the versions considered

ALICE	**v3-05-01**	**v3-06-01**	**v3-07-01**	**v3-08-01**	**v3-09-01**	**v3-10-00**	**v4-01-00**	**v4-02-00**
Lines of text	116,609	121,787	125,911	153,145	158,879	198,149	195,664	214,289
WinMerge	**v2.2.0**	**v2.4.0**	**v2.6.0**	**v2.8.0**	**v2.8.6**			
Lines of text	114,769	146,728	142,789	137,744	137,859			

we implemented a set of tools, which we describe below:

- ***diff***[1]**.** We used the *diff* file differencing tool to determine the set of new and deleted lines of code between two versions of the same system.
- ***src2srcml***[2]. src2srcml is a translator from source code to srcML [14], which tags each syntactic source code entity (*i.e.*, class, attribute, function, parameter, comment, etc.) using XML.
- ***Stemmer***. To obtain the stem of a term we used the Porter Stemmer [17].
- ***VocabularyExtractor***. We implemented this tool to add new tags to the srcML files, which contain information about each of the terms that are found in identifiers and in comments. The tool extracts identifiers and comments from the srcML files. For each identifier and comment it adds a new tag called `<vocabulary>`, which wraps information about all the tokens contained in that identifier or comment and their stems, using a tag called `<token>` (see Figure 1). The tokens are obtained by splitting the identifiers or comment words based on the use of camel casing and non-literals, such as underscores. During splitting, non-literals and numerals are discarded. CV, AV, FV, PV and CoV are constructed from the set of unique stems contained in the `<vocabulary>` tags associated with the corresponding identifier types and with comments, respectively. SV represents the union of all these vocabularies. The original identifiers and comment words are not included in these vocabularies. The srcML files modified as mentioned are used by the next tool in the chain.
- ***VocabularyDifferencer.*** We developed this tool to determine the new and deleted terms between two versions in each vocabulary (*i.e.*, CV, AV, FV, PV, CoV, and SV). The tool uses the modified srcML files provided by *VocabularyExtractor* for two versions *i* and *j* of the system and their identifier and comments vocabularies. For each vocabulary V, the tool computes V_i-V_j and V_j-V_i to determine new and deleted terms between the two versions. Then, it adds a new attribute called *dif* to the `<token>` tag introduced by *VocabularyExtractor* (see Figure 1). The value of the *dif* attribute can be *"New"*, *"Deleted"* or *"Kept"*.
- ***VocabularyProcessor.*** We built this tool to extract information from the `<vocabulary>` tags of all vocabularies in all versions. The tool populates an external database, which stores all the data about the vocabularies and is used to compute statistics across versions.
- ***IdentifierConstructor***. We developed this tool to help answer question RQ4. Based on the output provided by *VocabularyDifferencer,* this tool generates a summary of the number of new terms used in constructing identifiers for each identifier vocabulary.

```
<vocabulary>
<token name="set" stem="set" diff="New">
<token name="Values" stem="value" diff="Deleted">
</vocabulary>
```

Figure 1. The tags added by the *VocabularyExtractor* and the *VocabularyDifferencer* tools

2.4. Results

In order to answer the four research questions, we formulated several specific sub-questions, which each provide partial answers to the main research questions.

2.4.1. RQ1 - How does the size of the source code vocabulary evolve over time?

As software evolution is a complex phenomenon and several aspects of the system undergo evolution at the same time, we investigated if the size of the software vocabulary evolves in a similar manner to the size of the software system. We addressed the following sub-questions:

1. Are the evolution of the size of SV and the evolution of the system size similar?

2. Are the evolution of the number of new terms in SV and the evolution of the number of new lines of text in the system similar?

3. Are the evolution of the number of deleted terms from SV and the evolution of the number of deleted lines of text from the system similar?

4. Which category of changes, i.e., new terms or deleted terms, drives the evolution of the size of SV?

For answering RQ1.1, we observed for each system the evolution of its size (computed as the number of non blank lines of text) and the evolution of the SV size, over the versions considered in the case study. Figure 2 depicts the evolution of these two measures in ALICE and WinMerge. The system size and the SV size are normalized following the linear scaling transform, which is a common normalization used when the minimum and the maximum values in a

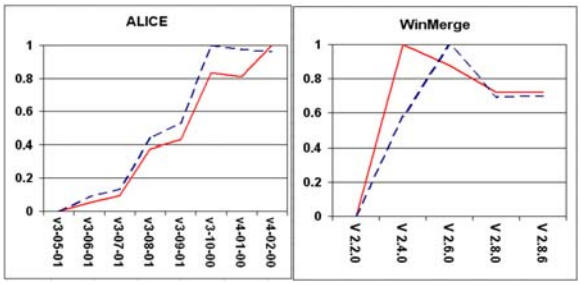

Figure 2. The evolution of the system size (solid) and the SV size (dashed) in ALICE and WinMerge

[1] http://www.gnu.org/software/diffutils/diffutils.html
[2] http://www.sdml.info/projects/srcml/

series are known. The actual number of new and deleted terms is shown in Table 2.

The data reveals that the two systems exhibit different behaviors with respect to the evolution in the size of SV and the system size (see Figure 2). We wanted to investigate the reasons for this discrepancy, so we analyzed each of the systems in detail and tried to explain the evolution of the measures by linking them to events which happened in the system.

In ALICE, the size of the SV and the system size evolve similarly in 7 out of 8 versions. This is confirmed also by the correlation between the two series, computed over the 8 versions considered in the case study (0.99 correlation with p-value < 0.05). The continuous steady increase in both measures over the first 6 versions may be explained by the fact that ALICE is still in the evolution phase. However, the sharp increase in v3-10-00, followed by the decrease in v4-01-00 in both the system size and the number of terms in SV represented an exception that needed to be investigated. We found that from v3-09-00 to v3-10-00, a major change occurred in the system, involving a new management system for files, which was developed in a different branch for almost one year and then merged into the main trunk. This explains the sharp increase between v3-09-00 and v3-10-00, and then the decrease in the following version, which is most likely due to the continued integration and refactoring of the new file management system.

In WinMerge, the two measures have a somewhat different evolution. The only commonalities are an increase between the first two versions and a tendency to stabilize over the last two. The sharp increase at the beginning is explained by the fact that a considerable amount of new functionality was introduced in the system between v2.2.0 and v2.4.0, which resulted in the introduction of new lines of text, and new terminology. From v2.4.0 to v2.6.0, new functionality was also added, which explains the increase in the number of terms; however, between these two versions, the package containing the source code files for the support of different languages in the GUI was refactored and the language support redesigned. This resulted in the deletion of a set of resource files containing a considerable number of lines of text, explaining the decrease in lines of text between the two versions. The decrease between v2.6.0 and v2.8.0 in both the size of the system and the size of SV is explained by the fact that an entire package, ExpatLib, containing files with a large number of lines of text, was deleted between the two versions. From v2.8.0 to v2.8.6 there are very few modifications, as v2.8.6 contains only minor bug fixes. The correlation between the size of the system and the SV size in WinMerge is 0.86, with a p-value of 0.06.

To understand better how the system evolves in its size in lines of text and in its vocabulary size, we answered RQ1.2 and RQ1.3 by analyzing the evolution of the number of new and deleted lines of text in the system size and the number of new and deleted terms from SV.

The increases and decreases in the system size are due to new and deleted lines of text in the system. We compute the set of new and deleted lines of text using the file comparison utility *diff*. Considering two versions i and j, and LT_i and LT_j the sets of lines of

Table 2. The initial vocabulary sizes and the number of new and deleted terms in each version of ALICE and WinMerge

ALICE	v3-05-01	v3-06-01		v3-07-01		v3-08-01		v3-09-01		v3-10-00		v4-01-00		v4-02-00	
	Initial Size	New	Del	New	Del	New	Del	New	Del	New	Del	New	Del	New	Del
CV	158	13	2	4	0	19	1	6	0	62	3	9	3	34	7
AV	946	29	0	31	1	142	24	34	4	212	20	30	29	225	64
PV	1,035	46	2	50	3	137	3	35	10	209	15	50	45	179	63
FV	839	42	0	13	1	128	2	26	7	235	14	55	27	176	133
CoV	5,615	296	58	205	73	940	127	317	58	1,536	264	649	741	1,297	1,440
SV	6,198	312	57	217	73	1,034	135	330	63	1,668	269	669	735	1,441	1,489

WinMerge	v2.2.0	v2.4.0		v2.6.0		v2.8.0		v2.8.6	
CV	163	30	3	12	1	15	8	0	2
AV	559	64	9	45	14	38	46	0	0
PV	997	155	11	64	13	39	101	1	3
FV	672	111	8	52	18	38	60	0	0
CoV	5,098	624	76	628	191	251	539	14	14
SV	5,393	696	77	644	199	257	582	14	13

text found in version i, respectively in version j, a line of text lt is considered *new* in LT_j if $lt \in LT_j - LT_i$ and *deleted* in LT_j if $lt \in LT_i - LT_j$. As diff compares files, if lt is moved from one file to another from version i to version j, it is considered both deleted and new, as it is deleted from the file where it appeared in version i and is new in the file where it appears in version j. If the line of text lt_i is modified to lt_j between the two versions, then lt_i is considered deleted and lt_j is considered new in LT_j. For the first version considered, the number of new and deleted lines of text can not be computed, as there is no data from a previous version.

Increases and decreases in the SV size are determined by the new and deleted terms from the vocabulary. Considering two versions, i and j, and a vocabulary V as one of the vocabularies CV, AV, FV, PV, or CoV, a term t is considered *new* in V_j, if $t \in V_j - V_i$, and *deleted* in V_j if $t \in V_i - V_j$. We must point out here that we do not account for term renaming. If a term is renamed from t_i to t_j between V_i and V_j, t_i is considered deleted and t_j is considered new. We do not compute the number of new and deleted terms in the first version in neither of the two systems, as we do not have data from a previous version to compare to. This explains the fact that in Figure 3 and Figure 4 the number of versions represented is smaller than the total number of versions by one.

Figure 3 shows the evolution of the number of new lines of text added to the system and the number of new terms added to SV in ALICE and WinMerge. The measures are normalized using the linear scaling transformation.

In ALICE, we observe a behavior followed by both the number of new lines added and the number of new terms added, which is defined by a sharp increase, followed by a sharp decrease (see Figure 3). In WinMerge, the two measures evolve differently. However, the maximum number of new lines and terms added is achieved for both measures in v2.4.0, which is most likely due to the introduction of numerous new features. The minimum number of new lines and terms added is observed in v2.8.6, as between the last two analyzed versions only minor bug fixes were performed.

As for the number of deleted lines of text from the system and the number of deleted terms from SV (see Figure 4), we observed that in ALICE, the two measures evolve similarly. Both measures experience oscillations between the first few versions and then know a sharp increase from V3-09-01 until the last version. In WinMerge, however, the number of deleted lines of text in the system and the number of deleted terms from SV do not evolve at the same rate, but both increase until v2.8.0 and then from v2.8.0 to v2.8.6 they both decrease sharply. This decrease is explained by the small number of modifications made between the last two versions, mostly minor bug fixes.

In order to get a better understanding of how each type of change (new/deleted terms) contributes to the evolution of the SV size (RQ1.4), we analyzed in more detail the number of new and deleted terms in each version. We observed that most of the times, the number of new terms is significantly larger than the number of deleted terms, with few exceptions.

In ALICE, the changes to the way files are managed, which occurred in v3-10-00, is reflected in the high number of new terms in this version, which is the highest in the observed history of ALICE. The high number of new and deleted terms from SV in the last versions of ALICE is also a consequence of this major change.

In WinMerge, the number of new terms added to SV is significantly larger than the number of deleted terms until v2.8.0 where the package ExpatLib was removed from the system. This is due to the addition of new functionality in the system, which introduces new concepts and in consequence new terminology. From v2.8.0 to v2.8.6, minor modifications were made, which explains the low number of both new and deleted terms.

The vocabulary, which dictates the number of

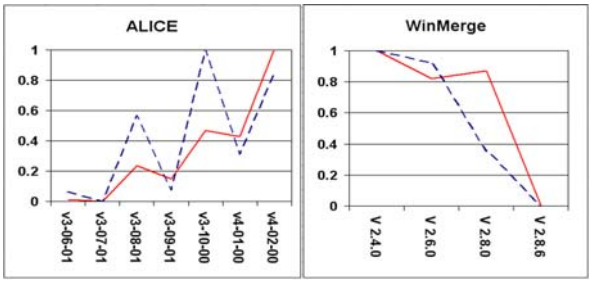

Figure 3. The evolution of the number of new lines of text in the system (solid) and of the number of new terms to SV (dashed) in ALICE and WinMerge

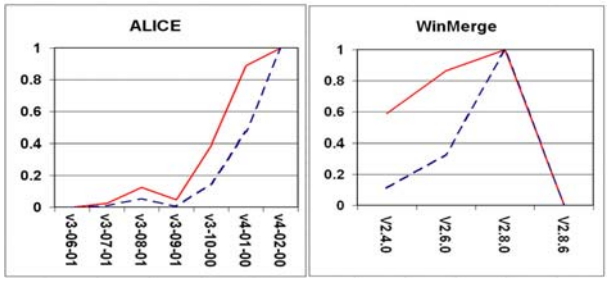

Figure 4. The evolution of the number of deleted lines of text from the system (solid) and of the number of deleted terms from SV (dashed) in ALICE and WinMerge

terms and the changes that affect SV, is CoV in both systems, as it contains the most terms in SV and the most number of new and deleted terms. An interesting fact is that both the number of lines of text in the system and the number of terms in the source code vocabularies are comparable in size in the two systems. The number of terms per line of text is also very close between the two systems: 49 terms per 1,000 lines of text in ALICE and 44 terms in SV per 1,000 lines of text in WinMerge.

2.4.2. RQ2 - What relationships exist between the individual vocabularies in the system?

In order to understand better which vocabularies drive the evolution of the SV, we analyzed the relationships between the individual vocabularies of the source code, i.e., CV, AV, FV, PV, and CoV. Details about the construction of each of these vocabularies can be found in Section 2.3. We focused on the following sub-questions to answer RQ2:

1. Which is the preponderant vocabulary in the system?

2. Are there any terms shared between vocabularies?

3. Are there new and deleted terms shared between vocabularies?

For answering RQ2.1, we analyzed the number of terms in each of the source code vocabularies, based on the data shown in Table 2. In both systems, CoV is significantly larger than all the other identifier vocabularies and its size is on average double the sum of the sizes of the identifier lexicons, across versions. In ALICE, CoV is followed in size, in this order, by FV, AV, and PV. In WinMerge, the rank of AV and PV changes, so CoV is followed by FV, PV, and AV. The smallest vocabulary in both systems is CV.

Under these circumstances, we expected that the evolution of CoV would have the most impact on the evolution of the overall system vocabulary SV. In order to assess whether this is true, we compared the evolution of each of the vocabularies and we computed the Pearson correlation between the SV and each of the individual vocabularies. We observed that in ALICE all the individual vocabularies had a high correlation with SV, with CoV having the highest correlation, 0.99. This observation, along with the fact that all the vocabularies follow the same direction of change (see Figure 5) over all versions, with the exception of CoV and SV, indicate that our assumption was true and that CoV is indeed the vocabulary which affects the evolution of SV the most in ALICE.

In WinMerge, all the vocabularies except for CV follow the same direction of change (see Figure 5). When computing the correlation between SV and each individual vocabulary, we found that all the vocabularies had a very high correlation with SV (over 0.96), with the exception of CV, which had 0.88. However, the low number of terms in CV makes that its evolution does not impact the evolution of SV, which follows the common direction dictated by the other vocabularies (see Figure 2). In WinMerge, the vocabulary that has the highest correlation with SV is again CoV, with a correlation of 0.99.

For answering RQ2.2, we computed the number of terms in the pair-wise intersection of all identifier vocabularies and in the intersection of all vocabularies and the percentage of shared terms between vocabularies (see Table 3 on the next page).

The results reveal that, on average, almost all terms in CV are present in CoV (96% in ALICE and 100% in WinMerge). Most terms in CV are also found in FV (99.7% in ALICE and 98% in WinMerge). This means that almost all words used in class identifiers are also used in function names and in comments.

In both systems, there is a high percentage of shared terms between the identifier vocabularies and CoV. In the case of AV, 75% of the terms are shared with CoV in ALICE and 90% in WinMerge. FV shared 83% of the terms with CoV in ALICE and 85% in WinMerge and PV shared 71% of its terms with

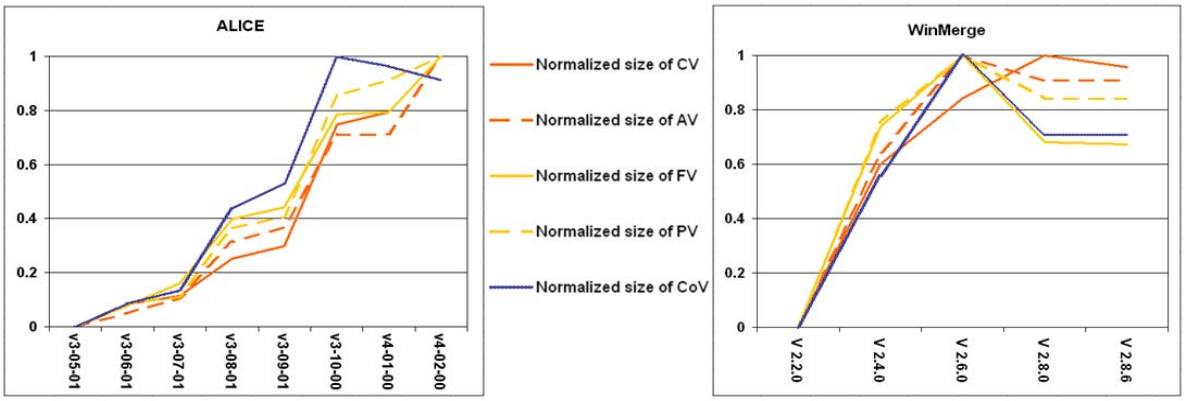

Figure 5. The evolution of the sizes of CV, AV, FV, PV, CoV in ALICE and WinMerge

Table 3. The number of terms in the pair-wise intersection of vocabularies in ALICE and WinMerge

		ALICE	WinMerge			ALICE	WinMerge
CV∩AV	Size	126	147	**AV∩PV**	Size	510	371
	% from CV	59%	76%		% from AV	43%	60%
	% from AV	11%	24%		% from PV	48%	48%
CV∩FV	Size	211	191	**AV∩CoV**	Size	878	555
	% from CV	99.7%	98%		% from AV	75%	90%
	% from FV	16%	17%		% from CoV	13%	10%
CV∩PV	Size	112	124	**FV∩PV**	Size	492	443
	% from CV	53%	64%		% from FV	38%	58%
	% from PV	10%	16%		% from PV	46%	40%
CV∩CoV	Size	203	194	**FV∩CoV**	Size	1,076	955
	% from CV	96%	100%		% from FV	83%	85%
	% from CoV	3%	3%		% from CoV	15%	17%
AV∩FV	Size	749	466	**PV∩CoV**	Size	757	627
	% from AV	64%	75%		% from PV	71%	82%
	% from FV	58%	42%		% from CoV	11%	11%

CoV in ALICE and 82% in WinMerge.

There are on average 99 terms (1% of SV) shared by all vocabularies in ALICE and 111 terms (2% of SV) shared by all vocabularies in WinMerge, across all versions.

For RQ2.3, we computed the intersection between the sets of new terms in each vocabulary. We also identified the intersection between the sets of deleted terms in each vocabulary.

The data indicates that no terms were deleted from all vocabularies at the same time neither in ALICE nor in WinMerge.

As for new terms, there are very few new terms shared between all vocabularies. In ALICE only three out of the eight versions have common new terms in all vocabularies, one common term in two of the versions and three new common terms in one version. In WinMerge, there is only one version which has one common new term between all vocabularies.

We were particularly interested in the intersections between the new and deleted terms in each of the identifier vocabularies and the CoV, to see if new or deleted terms in identifiers are also reflected in new or deleted terms in comments, respectively. For the average intersection between new identifier terms and new comment terms across all versions, we found a maximum of 38% of new identifier terms reflected also in new comments for ALICE and 23% for WinMerge. The list of identifier vocabularies, ordered by the average percentage of new terms also found in new comment terms is CV, PV, FV and AV for ALICE. For WinMerge, the order is the same except for CV and FV, which switch places.

In the case of deleted terms, in ALICE, in some of the versions considered, there is a particularly large number of deleted identifier terms also found among the deleted comment terms. For example, in version v3-06-01 all the deleted attribute terms are also deleted comment terms. In addition, in version v3-05-01, 50% of the deleted attribute terms are also comment deleted terms. At the same time, there are some versions where the identifier vocabularies have no deleted terms, and in consequence, the percentage of identifier deleted terms common with the comment deleted terms can not be computed.

In WinMerge, across all versions, the percentage of deleted identifier terms found also in deleted comment terms does not exceed 30% and this maximum is found in FV in v2.6.0.

2.4.3. RQ3 - Are new identifiers introducing new terms?

To answer this research question, we analyzed the number of new identifiers (*i.e.*, class, attribute, function, and parameter) constructed using only terms existing in their corresponding vocabulary (*i.e.*, CV, AV, FV, or PV), the number of new identifiers which introduce one new term in their vocabulary, and the number of new identifiers which introduce more than one new term in their corresponding vocabulary. The results are presented in Figure 6.

We found that in ALICE, on average across all versions, 56% of the new identifiers are constructed using only terms already existing in their vocabulary. In WinMerge, this percentage is 70% of all new identifiers.

As for the new identifiers that introduce new terms to their corresponding vocabulary, most of them introduce only one new term. In ALICE, on average, 85% of the identifiers that contribute with new terms (representing 37% of all the new identifiers) introduced only one new term. In WinMerge, the percentage is 90%, representing 26% of all new identifiers.

The rest of the new identifiers introduce two or more new terms, and they represent 7% of all the new

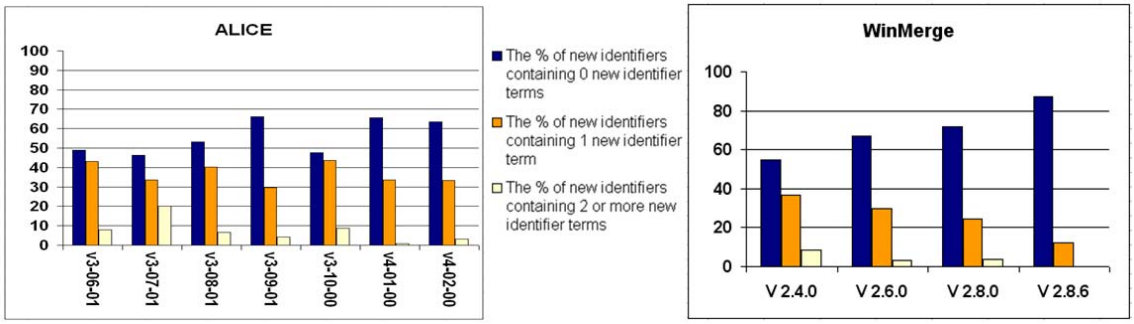

Figure 6. The percentage of new identifiers that introduce no new term, one new term, and two or more new terms in CV, AV, FV, PV, CoV in ALICE and WinMerge

identifiers in ALICE and 4% in WinMerge. The maximum number of new terms that a new identifier introduces is 4 (found in 2 versions out of 7 containing new identifiers) in ALICE and 3 in WinMerge.

In conclusion, the majority of new identifiers do not introduce new terms in their corresponding vocabulary and from the ones that do, almost all introduce only one new term.

2.4.4. RQ4 - What do the most frequent terms refer to?

For RQ4, we computed the frequency of the terms in each vocabulary, for each version considered and we analyzed the 15 most frequent terms in all the vocabularies, and manually identified problem domain terms found in each of these lists. The authors determined if a term is a domain term or not based on their prior knowledge, on the analysis of the system's functionality and source code and on the systems' documentation.

In ALICE, the vocabulary with the most domain terms on average among the 15 most frequent terms in the vocabulary is CV, with 13 domain terms, followed by FV, with 10, then by AV with 8 terms, CoV with 6 terms and finally by PV with 5 domain terms on average. The domain terms we identified among the most frequent terms in all versions are the stems of: *hit, track, pad, digit, cluster, chamber, pho, muon, segment, rec, event, tr, rich, tpc, trigger, tof, mp, tp, phosrp, cpv, emc, ppsd, energy, ev, sector, rh.*

In WinMerge, the vocabulary with the highest number of problem domain terms on average in the list of 15 most frequent terms is FV with 13 terms, then AV, also with 13 terms, followed by CV with 10, CoV with 8 and finally PV with 4. In WinMerge, the list of problem domain terms we found is the list of stems of the following terms: *line, file, string, text, char, match, name, data, update, edit, end, version, modify, dir, merge, filter, status, diff, prop, file, str, left, right.*

In both systems, the vocabularies with the least problem domain terms among the most frequent terms are CoV and PV. One possible explanation for this could be that, as seen before in the results of RQ2, there are many more terms in comments than in identifiers, and there are also numerous solution domain terms found in comments. For example, the stems of the terms *number*, *copyright, parameter* are among the most frequent stems in ALICE and the terms *psz, afx, gnu* are among the most frequent ones in WinMerge. One possible explanation of the low number of domain terms in parameter names can be explained by the fact that formal parameters are often extreme contractions (even single letters), and the semantic (linguistic) often information is carried by their type, not their name. On the other hand, the most frequent terms in class names, attribute names, and function names refer to domain concepts.

The detailed results obtained, for each version and for each vocabulary, are available in [1].

2.5. Discussion

Summarizing our findings, we have observed that:

- **RQ1:** System vocabulary and system size often exhibit a parallel evolution trend (although in WinMerge the two trends sometimes diverge). Addition of new features affect the vocabulary used by programmers, by increasing it. Similarly, important changes of existing functionalities (*e.g.*, file management in ALICE) is usually reflected in major vocabulary changes.

- **RQ2:** The vocabulary used to build class identifiers has the largest number of terms in common with other vocabularies, which may descend from the OO "good" design principle, prescribing that classes represent the core domain concepts. On the other hand, identifier vocabulary changes are only marginally reflected in the comment vocabulary, possibly indicating a tendency of comments to become obsolete and misaligned with the code.

- **RQ3:** New identifiers usually introduce no or at most one new term. This indicates that the vocabulary used by developers is reused whenever possible and extended with parsimony. This is a somewhat

surprising result, since such a consistent use of terms is left to the ability of programmers and receives (currently) no dedicated tool support. Maybe it is an indicator of the quality of the development teams involved in our two case studies.

• **RQ4:** Frequent terms are associated with core domain concepts. This result suggests a very simple heuristics to extract domain knowledge from a system, based on simple frequency analysis.

Overall, our results indicate that the vocabulary used by programmers is subject to an evolution pressure similar to that affecting code evolution. As a consequence, we expect that keeping vocabulary evolution under control is a major challenge, as it is happening with the code. The quality of the vocabulary affects how identifiers are constructed, how core concepts are recognized and named, and in general how difficult it is for a programmer to understand and evolve the system. Hence, tools are needed to help programmers maintain and improve the quality of vocabularies, especially when performing major changes or introducing new identifiers. From our study, we can envision a central role of class vocabulary terms and of frequent terms in recovering and structuring the knowledge conveyed by identifiers.

2.6. Threats to validity

Even for case studies where the results are not generalized, there are usually factors that represent threats to certain aspects of the validity of the results [22]. In our case study, we considered only class names, attribute names, function names, parameter names, and comments. Even though these are the most important software entities in an object-oriented software system, we have little knowledge about how the vocabulary of other entities such as local variables, constants, types, etc. evolves. In our future work, we plan to address these vocabularies as well. Also, to extract and collect the information needed, we mostly relied on consolidated theory and tools, but we cannot be sure that a different tool chain would not collect slightly different data. Since the entire processing was performed with the same tool chain on the two systems and for all versions, errors, if any, would be systematic and most likely would not affect our findings substantially.

In order to obtain more generalizable results, we need to investigate a larger set of programs, with a richer set of versions, developed in different environments, using different programming languages and by different development teams. For example, we only considered two systems in our case study and for WinMerge we only had five versions in our analysis, which made it difficult to observe patterns in the evolution of the source code vocabulary and to draw conclusions on the reasons for this evolution.

3. Related Work

The source code vocabulary was studied in several papers [2-9, 11-13, 15, 19, 21], as researchers have long acknowledged the important role that the information stored in identifiers and comments plays in the comprehension and maintenance of software systems.

Previous work focused mainly on the role of identifiers and comments in facilitating program comprehension [2, 6-8, 21]. In [18] an empirical study is conducted to identify the type of knowledge used by software developers during maintenance. Other work has addressed the way domain concepts are reflected and used in source code [4, 11, 15, 19] Further work has focused on identifier quality and proposed methods for improving it. The analysis of function identifier structure was performed in [6] and identifier refactoring to improve meaningfulness was studied in [5]. In the same direction, [13] and [21] investigated how different naming styles (*i.e.*, single letters, abbreviations, and full words) affect comprehension and both studies came to the conclusion that full word identifiers provide better comprehension. In [7], the conciseness and consistency of identifiers is analyzed based on the relationship between concepts and their names. Comments have also been studied in [9], where the authors found that the language of comments in a software is a sub-language of English and in [10], where the authors studied the co-evolution of identifiers and comments.

Closest to our research is [3], which studied the stability in the evolution of the source code lexicon and the evolution of the structure of a software system. The stability metrics for a software entity between two versions were computed as the cosine between two vectors, representing the entity in the two versions. In the case of the structural stability metric, the vectors contained the values of several structural metrics and in the case of the lexical stability metric, the vectors contained the frequencies of the words found in the entity in the two versions. Our work is different from [3] by the fact that it does not focus on measuring stability; rather, it analyzes the evolution of the structure and of the vocabulary of a software system and focuses on the changes that drive this evolution at a fine level of granularity. Our work provides also insights into *how* the software vocabulary evolves.

4. Conclusions and Future Work

In this paper we analyzed the evolution of the source code vocabulary of two open-source software

systems. Among the most important findings are the fact that the vocabulary and the size of a software system tend to evolve the same way, with some exceptions, the fact that the comment vocabulary is the major responsible for the evolution of the source code vocabulary and that the comment vocabulary also contains more than ¾ of the terms found in any identifier vocabulary. Also, we found that most of the new identifiers do not introduce new terms in their vocabulary, but rather are constructed using existing terms. At the same time, the most frequent terms in class names, attribute names, and function names are problem domain terms. Overall, the results indicate that the evolution of the source code vocabulary does not follow a trivial pattern and more research is needed to fully understand it.

Our future work will focus on addressing more aspects of the evolution of the source code vocabulary, such as the evolution of other types of identifiers, like local variables, constants, types, etc. Also, we will include more systems in our case studies, found in various stages of development and we will consider numerous versions for each system, including minor and major releases of the systems. At the same time, we will also address other research questions regarding the spread of the terms across the system and across vocabularies. We plan also to include the vocabulary of other software artifacts in the analysis.

5. Acknowledgements

Sonia Haiduc and Andrian Marcus were supported in part by a grant from the US National Science Foundation (CCF- 0820133).

6. References

[1] Abebe, S. L., Haiduc, S., Marcus, A., Tonella, P., and Antoniol, G., "Analyzing the Evolution of the Source Code Vocabulary", Technical Report, WSU, CS, 2008, Online at http://www.cs.wayne.edu/ ~severe/TechnicalReport-WSU-2008-01

[2] Anquetil, N. and Lethbridge, T., "Assessing the Relevance of Identifier Names in a Legacy Software System", in *Proceedings Annual IBM Centers for Advanced Studies Conference (CASCON'98)*, 1998, pp. 213-222.

[3] Antoniol, G., Gueheneuc, Y.-G., Merlo, E., and Tonella, P., "Mining the Lexicon Used by Programmers during Software Evolution", in *Proc. Int'l Conf. on Software Maintenance*, 2007, pp. 14-23.

[4] Biggerstaff, T., Mitbander, B., and Webster, D., "The Concept Assignment Problem in Program Understanding", in *Proc. Int'l Conf. on Soft. Engineering,* 1994, pp. 482-498.

[5] Caprile, B. and Tonella, P., "Restructuring Program Identifier Names", in *Proc. Int'l Conf. on Software Maintenance,* 2000, pp. 97-107.

[6] Caprile, B. and Tonella, P., "Nomen Est Omen Analyzing the Language of Function Identifiers", in *Proc. Working Conf. on Reverse Engineering*, 1999, pp. 112-122.

[7] Deissenboeck, F. and Pizka , M., "Concise and Consistent Naming", *Software Quality Journal*, 14, 3, 2006, pp. 261-282

[8] Etzkorn, L., Bowen, L., and Davis, C., "An Approach to Program Understanding by Natural Language Understanding", *Natural Language Engineering*, 5, 03, 1999, pp. 219-236.

[9] Etzkorn, L., Davis, C., and Bowen, L., "The Language of Comments in Computer Software: A Sublanguage of English",*Journal of Pragmatics*, 33, 11, 2001, pp.1731-1756.

[10] Fluri, B., Wursch, M., and Gall, H. C., "Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes", in *Proc. Working Conf. on Reverse Engineering*, 2007, pp. 70-79.

[11] Haiduc, S. and Marcus, A., "On the Use of Domain Terms in Source Code", in *Proc. Int'l Conf. on Program Comprehension,* 2008, pp. 113-122.

[12] Høst, E. and Østvold, B., "The Programmer's Lexicon, Volume I: The Verbs", in *Proc. Int'l Working Conf. on Source Code Analysis and Manipulation*, 2007, pp.193-202

[13] Lawrie, D., Morrell, C., Feild, H., and Binkley, D., "What's in a Name? A Study of Identifiers", in *Proc. Int'l Conf. on Program Comprehension*, 2006, pp. 3-12.

[14] Maletic, J. I., Collard, M. L., and Marcus, A., "Source Code Files as Structured Documents", in *Proc. Int'l Workshop on Program Comprehension,* 2002, pp. 289-292.

[15] Ohba, M. and Gondow, K., "Toward Mining Concept Keywords from Identifiers in Large Software Projects", in *Proc. Int'l Workshop on Mining Soft. Repositories,* 2005, pp. 1-5.

[16] Perry, D. E., Porter, A. A., and Votta, L. G., "Empirical studies of software engineering: a roadmap", in *Proc. of the Conf. on The Future of Software Eng.*, 2000, pp. 345-355.

[17] Porter, M., "An Algorithm for Suffix Stripping", *Program*, 14, 3, July 1980, pp. 130-137.

[18] Ramal, M. F., de Moura Meneses, R., and Anquetil, N., "A Disturbing Result on the Knowledge Used during Software Maintenance", in *Proc. of the Working Conf. on Reverse Engineering*, 2002, pp. 277-286.

[19] Ratiu, D. and Deissenboeck, F., "From Reality to Programs and (Not Quite) Back Again", in *Proc. Int'l Conf. on Program Comprehension*, 2007, pp. 91-102.

[20] Sim, S. and Holt, R., "The ramp-up problem in software projects a case study of how software immigrants naturalize", in *Proc. Int'l Conf. on Software Eng.*, 1998, pp. 361-370.

[21] Takang, A., Grubb, P., and Macredie, R., "The Effects of Comments and Identifier Names on Program Comprehensibility: An Experimental Investigation", *Journal of Programming Languages*, 4, 3, 1996, pp. 143-167.

[22] Yin, R. K., *Case Study Research: Design and Methods*, Sage Publications Inc, 2003.