SIGRAD 2012
A. Kerren and S. Seipel (Editors)

Visualization of Text Clones in Technical Documentation

Morgan Ericsson, Anna Wingkvist, and Welf Löwe[1]

[1]Linnaeus University, Sweden

Abstract

An initial study of how text clones can be detected and visualized in technical documentation, i.e., semi-structured text that describe a product, software, or service. The goal of the visualizations is to support human experts to assess and prioritize the clones, since certain clones can be either intentional or harmless. We study some existing visualizations designed for source code, and provide initial and limited adaption of these. A major difficulty in this adaptation is to manage the semi-structured technical documentation compared to structured source code.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications—

1. Introduction to Clones and Clone Detection

Duplicated and redundant text, so called (text) clones, can be a problem in technical documentation. Technical documentation is semi-structured text that describes a product, software, or service. We consider the text to be semi-structured, since it generally contains markup, such as XML, but this structure is defined by the authors of the document.

Clones increase the size of the documentation, which adds to the cost to store, translate, print, etc. while more effort is required to maintain the documentation, for example due to update anomalies. An update anomaly occurs when not all clones are changed to reflect an update to the content, e.g., an instruction is only changed in certain parts of the documentation. However, clones can also improve the quality of a documentation, by increasing readability and understandability. Duplicated text can help to create context and familiarity, and reduce the need for cross-references. The dual nature of clones is a problem to assess the quality of a documentation and there is a need to identify clones. Further, how human experts faced with this problem can be supported by visualizations. We ask the following question: how can visualization help to (quickly) identify clones?

Roy et al. [RCK09] introduce the foundations of clone detection. A text fragment F is any sequence of text characters. It can be uniquely identified, e.g., by its file, starting, and ending position. A fragment F_1 is a *clone* of another fragment F_2 if they are considered similar. We can define (different) function(s) f to determine similarity. If $f(F_1, F_2)$ is *true*, F_1 and F_2 are clones, and F_1 and F_2 form a *clone pair*. We assume f to define an equivalence relation. Hence, we can classify all text fragments using f and the fragments $F_1, F_2, \ldots, F_3$ of the same class form a so-called *clone set*.

Clone detection is automated extraction of similar text fragments. Approaches are in the domain of similarity function f, which could be based on plain text sequences, their syntactic structure, e.g., induced by XML markup, and their meaning (the latter is undecidable in general). For a given set of documents (files) and a given similarity function f, we can compute the clone sets. As the clone fragments of different clone sets may overlap or be included in each other, post-processing is necessary to deduce the actual document uniqueness or similarity. Moreover, to avoid noise, thresholds for too small text fragments and to disregard simple or automated text transformation, pre-processing could filter and abstract from the original text sequences, for example by replacing tabs and linefeeds with spaces.

Text cloning could happen when writing with haste, but also intentionally, e.g., when similar text fragments are required in different documents, or as a result of the writing process itself, e.g., when documents come in variants or versions. In any case, awareness of clones and understanding of their genesis is necessary for assuring the quality of documentations. It helps fixing typos and spelling errors consistently, uncovering plagiarism, understanding related document sets and their evolution, and identifying text fragment that may be generated automatically. Even awareness of absence of expected clones, for example a common header that should be present in all documents, can be helpful.

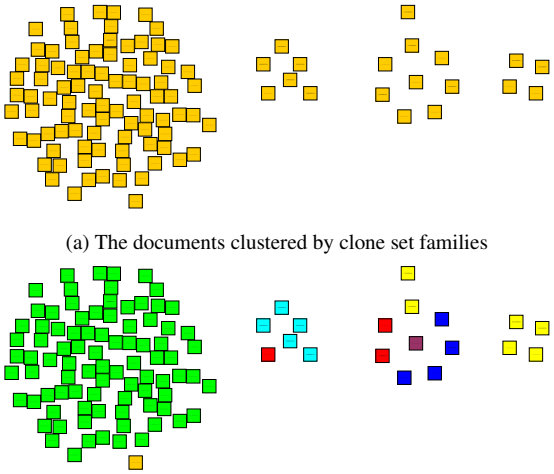
2. Visualization of Clones

There are plenty of research on how to visualize clones in source code. However, it is not clear how many of these visualizations are suitable for technical documentation. It is also not clear if the aim of these visualizations are to simply detect clones or also help to classify them. Here, we outline different kinds of visualizations and discuss how they can be adapted to technical documentation and if adapted versions provide enough information to help classify the clones. We base our discussion on the summary by Jiang et al. [JHH06]. They divided the visualizations on what relation they focus on; clone pairs or clone sets and since we want to classify clones we only focus on clone set visualizations.

The six visualizations of clone sets are: Metric Graphs [UKKI02], Hasse Diagrams [Joh94], Hyper-linked Web [Joh96], Linked Editing [TBG04], Duplication Aggregation Tree Map, and Clone Class Family Enumeration [RDL04]. Metric Graphs presents a set of metrics for each clone set, such as *%coverage* or *radius* (maximal distance from a common ancestor). Hasse Diagrams can be used to display a partial order based on the subset relation between files in clone sets. With positioning in a 2D space it shows relations between and properties of the clones, e.g., the degree of similarity. Hyper-linked Web can be considered a hypertext version of a Hasse diagram, where each clone set is displayed as a document with metrics such as size and links to super- and sub-clone sets as well as the actual files. Linked Editing shows clones in an editor view, where cloned parts are marked and can be edited simultaneously, i.e., a change can be made to all instances of a clone.

Duplication Aggregation Tree Map is modified to allow empty space. Each directory represent a box that contains smaller boxes (files). The size of the file boxes shows the number of internally (width) and externally (height) cloned lines. The size and shape of the directories mimic the amount and ratio of internal and external clones. Clone Class Family Enumeration groups clone sets if they contain the same set of files. In a 2D space, the clone class families and source files are positioned with respect to lines of cloned code, number of source files, lines of code and how many clone class families the source file is a member of, respectively.

Hyper-linked Web, Linked Editing and Hasse diagrams have interesting properties, but they are not suitable to quickly identify clones. Metric graphs rely on various metrics calculated on clones and [KKI02] put forward the need for more discussion on what metrics can be used to determine related clones. Duplication Tree Map Aggregation and Clone Class Family Enumeration provides new and improved ways to show dependencies between files, however at least Duplicate Tree Map Aggregation relies on properties of source code (package/class/directory structures, etc.). Based on our experience with documentation, it can be difficult to apply these structure properties to documentation, since the structure is not as controlled as it is in software where a method or a class have specific rules and purposes.



(a) The documents clustered by clone set families

(b) The clusters clustered by clone set with each document colored by which of the eight parts of the documentation it belongs to

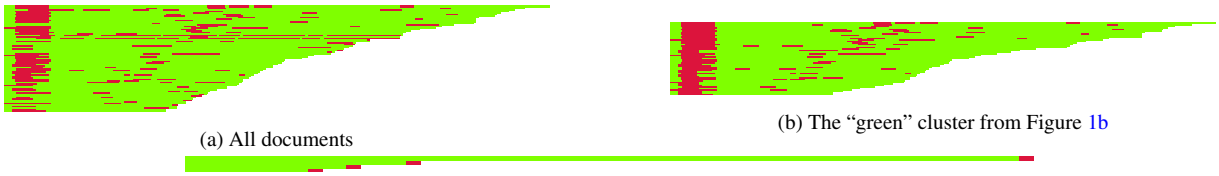
Figure 1: Visualizations of clusters of documents (square). Each cluster represents a group of document that are part of the same clone set family(ies). Nodes are not weighted and edges are removed to make the grapher clearer.

3. Experiment

We implemented a set of visualizations and applied them to the result of clone detection. The clone detection is performed on 300 documents from a real-world documentation of a telecom product that consists of about 600 documents. We define an information extractor that removes the XML markup from the documents, normalizes whitespace and converts all text to lower-case. We define a simple similarity function that recognizes identical sequences of words in sequences that are at least 100 words long. Text fragments that are identical form clone sets and are presented using a triple that consists of file name, line number of the first, and last line of the fragment, e.g., $(doc1, 4, 19)$. The total size of the 300 documents is approximately 90,000 lines.

The clone detection results in more than 500 clone sets, and approximately 25,600 lines (28%) of the documentation are clones of some other part. Compared to real-world software systems, where it is not uncommon that 7-20% of the source code are clones [RC08], this may seem high, but compared to our previous results for documentation (e.g., [WELL10], where we find in the range of 40-50% cloned text), this is low. On average, each clone set contains 7 document, and the largest contains 87.

Figure 1a shows document similarity: each node is a document grouped by similarity. Each of the four clusters depicts

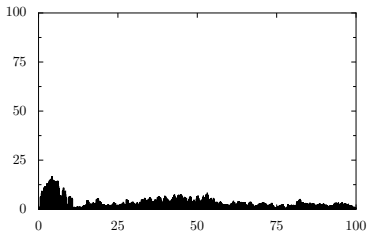


(a) All documents

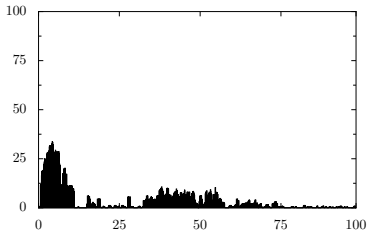
(b) The "green" cluster from Figure 1b

(c) The four node "yellow" cluster from Figure 1b

Figure 2: Pixelmap representations of three different clone sets. Each pixel represents a line, red pixels represent clones and green pixels represent non-clones. Each pixelmap is restricted to documents between 100-400 lines for clearer figures.



(a) All documents



(b) The "green" cluster from Figure 1b

Figure 3: Histograms depicting the percentage of clone sets (y-axis) that a line (x-axis) is part of. The documents are normalized to the length of the largest document. Clones are positioned relatively, but their length is not normalized. The pattern in (b) is more distinct than in (a) due to the common structure of the documents in (b).

a Clone Class Family. It illustrates where the clones exist, which is generally not enough information to gain an understanding and prioritize which clones to remove. For example, pair-wise comparison of 87 documents using a diff-tool or further visualizations is a cumbersome task. We applied a coloring based on which documentation purpose a document belongs to (extracted from file name conventions), and two clusters shows a strong correlation between purpose and similarity (cf. 1b).

The largest cluster contains documents describing a function or command. The document similarity indicates that a common template is used and clones are intentional. If we consider a pixelmap (cf. Figure 2) of a subset of the documents (documents between 100 and 400 lines, to help reduce the image size), we find that the clones (red pixels) are placed in the beginning of the documentation and in a band across the middle that mirrors the length of the document. If we investigate the cluster with four documents, where documents have another purpose, we can see that each of them shares a common postscript.

To gain more structure insights than the pixel maps provide, we project the clone sets w.r.t. one document and calculate a hierarchy based on textual containment; Figure 4 shows such a hierarchy (tree). Each tree shows the structure of clones in respective document, and each direct child of the root represents a red sequence of pixels in Figure 2 If we project and overlay the clone sets w.r.t. several documents and represent the tree nodes using different colors per document (and appropriate color mixtures for nodes representing clone sets of several documents), we can see similarity in the clone structure between the documents. This idea is similar to Jiang's et al. visualization of super clones, i.e., combinations of clones, using code execution structure (basic-block graphs) to relate different source files. Since we do not have such structure in documentation, we have to rely on simpler methods to relate different documents. One approach is to normalize the length of a document in order to make it comparable. Figure 3 shows the relative number of clone sets a line is part of. So, if the first line of all the documents are part of 10 of a total of 100 clone sets, the position 1 would have a value of 10%.

4. Discussion and Conclusions

Our main finding is that even though we rely on clone detection and extraction for source code, most of the visualizations are not a good fit for documentation. A major challenge is that documentation lacks some of the structure that is available in code. Our initial attempts suggest that further work is needed on both extraction and visualization of text clones. Most of the documentation we analyze contains text and markup, even if most of the discussion here is focused on text. We can extract certain structural properties from the markup, such as bullet lists, headers, etc., but they are not as well defined nor used as purposefully as structure and (static) semantics information of a programming language. The company that produce the documentation often provide structure, rules, and guidelines, and these could be used to provide additional semantic information. For exam-

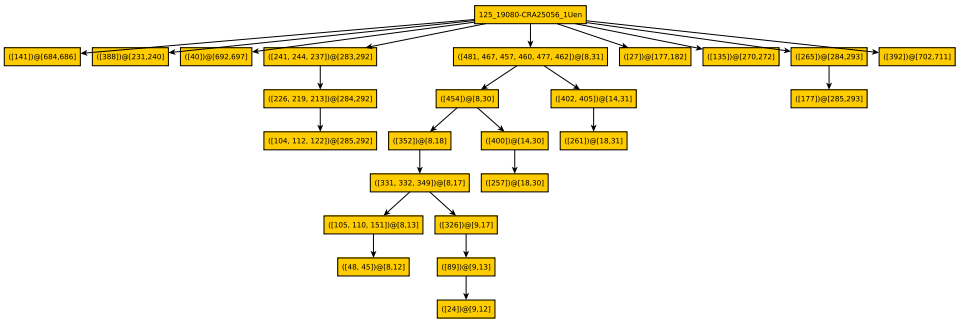


Figure 4: A tree that depicts clone sets and subsets that are present in a single document. Each box shows the clone sets the clone is part of and the line numbers in the document. The tree is built using subset containment, i.e., a parent contains the subsets of all its children. Each red sequence in a single document in Figure 2 is a direct child of the root.

ple, our coloring of the document similarity graph, based on purpose (file name conventions) helps to explain one of the large clusters of clones. However, such semantic information is based on written guidelines or convention, which might not be used consistently. Consider, e.g., the four documents with the same postscript; these all have the same purpose, but there are 15 other documents with the same purpose that are not part of the cluster.

The syntactic and semantic structure of source code is also a benefit when you want to visualize super clones, i.e. a combination of (all) clone sets, because these provide a structure that can be used to normalize the different clone sets. To achieve the same effect on documentation/text, we need to find ways to normalize the different file lengths and structures. So to visualize super clone information, we scale the file length, and position the clones relative to the start of the file. The scaling works well, if the positions of the clones are relative to the file (e.g., pre- and postscript). But scaling can affect the positioning of a clone, which can mask actual patterns and create non-existing ones. The pixelmaps are an alternative to super clone visualization. However, we still experience problems with file lengths, since patterns that are not relative are again difficult to represent.

Since we currently only consider the text when we visualize the position of the clone sets, we plan to map these back to the XML representation and see if we can use this information to improve the normalization of the documents. This might affect both the super clone and pixelmap representations. We also plan to investigate more advanced data normalizations techniques and determine if these are applicable. Another, related track is to investigate if relative positioning is exact enough, if we improve the resolution.

We will look into other types of visualizations, where we do not focus on position within the document or clusters of clone sets: for example graphs that focus on the order of the clone sets within the files, and individual differences between clone sets that are present in the same range of files. Here, the visualizations we adapt focused on source code clones, but we will study other types of software and information visualizations. Even if our experiment shows that properties of the documentation, such as types of documents, etc. can be a risk to include when clustering documents that are similar, we see if we can find a better set of properties, for example author, creation date, etc. Also, we need to improve our tools and the quality of the visualizations produced.

References

[JHH06] JIANG Z. M., HASSAN A. E., HOLT R. C.: Visualizing clone cohesion and coupling. In *APSEC* (2006), pp. 467–476. 80

[Joh94] JOHNSON J. H.: Visualizing textual redundancy in legacy source. In *Proc of the 1994 conf of the Centre for Adv Studies on Collaborative research* (1994), pp. 9–18. 80

[Joh96] JOHNSON J. H.: Navigating the textual redundancy web in legacy source. In *Proc of the 1996 conf of the Centre for Adv Studies on Collaborative research* (1996), pp. 7–16. 80

[KKI02] KAMIYA T., KUSUMOTO S., INOUE K.: CCFinder:a multilinguistic token-based code clone detection system for large scale source code. *IEEE Tran Soft Eng 28*, 7 (2002), 654–670. 80

[RC08] ROY C. K., CORDY J. R.: An empirical study of function clones in open source software. In *WCRE* (2008), Hassan A. E., Zaidman A., Penta M. D., (Eds.), IEEE, pp. 81–90. 80

[RCK09] ROY C. K., CORDY J. R., KOSCHKE R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Sci. Com. Prog. 74*, 7 (2009), 470–495. 79

[RDL04] RIEGER M., DUCASSE S., LANZA M.: Insights into system-wide code duplication. In *Proc of the 11th Working Conf on Reverse Eng* (2004), IEEE Computer Society, pp. 100–109. 80

[TBG04] TOOMIM M., BEGEL A., GRAHAM S. L.: Managing duplicated code with linked editing. In *Proc of IEEE Symp on Visual Languages-Human Centric Comp* (2004), pp. 173–180. 80

[UKKI02] UEDA Y., KAMIYA T., KUSUMOTO S., INOUE K.: Gemini: Maintenance support environment based on code clone analysis. In *Proc of the 8th Int Symp on Software Metrics* (2002), IEEE Computer Society, pp. 67–76. 80

[WELL10] WINGKVIST A., ERICSSON M., LÖWE W., LINCKE R.: A metrics-based approach to technical documentation quality. In *Proc of the 7th Int Conf on the Quality of Info and Com Tech* (2010), pp. 476–481. 80