How to Make the Hidden Visible – Code Clone Presentation Revisited

Sandro Schulze
University of Magdeburg
sanschul@ovgu.de

Niko Schwarz
University of Bern
schwarz@iam.unibe.ch

***Abstract*—Nowadays, a slew of clone detection approaches exists, producing a lot of clone data. This data needs to be analyzed, either manually or automatically. However, even after analysis, it is still not trivial to derive conclusions or actions from the analyzed data. In particular, we argue that it is often unclear how the cloning information should be presented to the user. We present our idea of task-oriented clone presentation based on use cases. We identify five use cases that have to be addressed by appropriate clone presentation techniques.**

I. Introduction

Intensive research has been performed on clone detection and evaluation—presentation is often left as an implementation detail to implementors. While there is a plethora of visualizations, current visualizations for code clones are limited [1], [2] in that they are too narrow to one use case, rather than supporting the multitude of use cases where cloning information is useful, *e.g.,* online clone reporting, quality assessment, and refactoring. They are rather directed to a certain task for which they are more or less appropriate.

To present code clones in a way that lets the user take immediate action, we need to understand the range of actions available. Therefore, we start our analysis by an overview over current use cases. We then proceed to show a mapping that shows for every use case which presentations is appropriate, and vice versa. While our overview is far from complete, the objective is to guide tool builders and give an overview over what is there and how it could be exploited.

II. Use Cases for Clone Presentation

Once code clones have been detected and analyzed, they must be accessible for further treatment. This step, called clone presentation, is not an obvious task. First of all, there might be different stakeholders such as software quality managers or software developers that need different views (including different degrees of granularity) on the clones. Second, these stakeholders want to perform different actions. In the following, we identify five use cases that encompass the different views and treatments of clones.

a) Quality assessment (QA): This use case mainly appears on the management level. For instance, the stakeholder wants to have an raw estimation on how the existing clones affect the overall system quality. Furthermore, the detection of hot spots, *i.e.,* parts of a system that contain a superior amount of clones, to define countermeasures is part of this use case.

b) Awareness (AW): This use case describes the fact that it is important for certain stakeholders, especially developers, to be aware of existing clones and their relations. During implementation, a developer has to know when he is changing a fragment that is part of a clone group, in order to prevent inconsistent updates.

c) Bug prediction (BP): If a bug has been found in a clone of a code snippet, then all other clones might be incorrect as well. Further, if a code snippet is copied from a source to a destination, a certain similarity between source and destination is implied. This could be exploited to predict bug occurrence.

d) Quality improvement (QI): This use case encompasses the removal of clones. This requires refactoring techniques and proper analysis of the differences between near-miss clones, in order to to select the correct refactoring.

e) Compliance (CO): This use case encompasses two issues: First, a stakeholder may be interested in whether or not code in the systems exists that has been copied from external sources (*e.g.,* third party libraries), in order to verify that no licensing terms are being violated. Second, there could be subsystems that contain secret code that is not allowed to be used outside pre-defined boundaries.

III. Putting the Pieces Together

Not all visualizations lend themselves equally well to all tasks. In the last section we have described use cases that have to be addressed by an appropriate clone presentation. However, due to the fact that different approaches are possible for clone representation and visualization, for each use case we select only some of the techniques and methods that could be mentioned. Our selection is the result of an intensive discussions at Dagstuhl seminar 12071 on clone detection, and could therefore be considered an expert consensus. In Table I, we show a compatibility matrix that relates use cases to clone presentation methods.

Not all visualization methods are mentioned. For a more comprehensive overview over these, we refer to existing surveys [4], [5].

We argue that clone visualizations such as SeeSoft views or TreeMaps are helpful to provide a big picture of the clones in the system and thus support the use cases QA, AW, and CO. To this end, a SeeSoft view (cf. Figure 1 (b)) represents each file as rectangle and each clone as a bar within this rectangle, indicating its size and position. Additionally, code

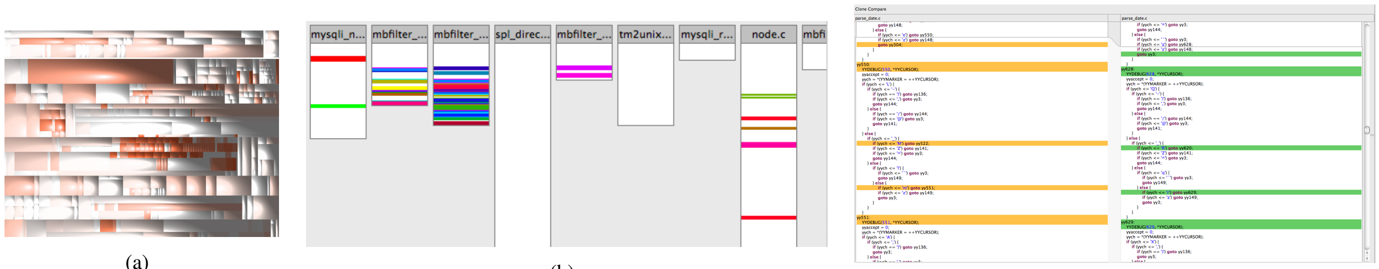
(a) (b) (c)

Figure 1. Examples for (a) a tree map, (b) a seesoft view, and (c) a compare view from the clone detection and report part of ConQAT [3]

Table I
MATRIX SHOWING WHICH CLONE PRESENTATION FEATURE CAN BE USED FOR WHICH USE CASE.

	QA	AW	BP	QI	CO
SeeSoft View [6]	X	X	?		X
TreeMap [7]	X	X			X
Source code view		X	X	X	
Compare view			X	X	
Links		X			X
Dashboard	X	?			X
Filtering/querying/zooming	X	X			
User-generated meta-data				X	
Revision history	X		X		

clones that belong to the same clone set have the same color. As a result, the stakeholder receives an overview of clones and how they are scattered throughout the system. Similarly, a TreeMap (cf. Figure 1 (a)) represents each file as rectangle with information on size and position, relatively to the whole system. Furthermore, the color indicates whether such a file contains many clones or not, which enables an easy detection of so-called *hot spots*.

In contrast to the previously mentioned visualizations, a developer requires methods for clone presentation that are seamlessly integrated in their development process. We propose that the source code view (as provided by common IDEs) and a compare view (cf. Figure 1 (c)), providing a face-to-face comparison of two code clones, are appropriate to fulfill these demands and thus to support the use cases BP and QI. For the source code view, we suggest to integrate even more sophisticated approaches such as linking between corresponding clones. Then, the developer could receive information on corresponding clones in case he changed a cloned code fragment. Furthermore, they could be provided with means to change the corresponding clones consistently. Beyond that, the compare view can provide even more fine-grained information such as highlighting the differences of two code clones.

Finally, the aforementioned approaches can be complemented by further presentation techniques. For instance, the revision history can be exploited to provide evolutionary information about the clones while user-generated meta-data (*e.g.*, by *tagging the clones*) can provide useful insights about the developer's view on certain clones.

IV. SUMMARY

We have summarized the most common use cases of clone detectors and mapped them to visualizations that can display them to the user. While we do not claim completeness, we want to stimulate discussion on our categorization of use cases and the respective clone visualization approaches.

V. ACKNOWLEDGMENT

This paper is the result of a working group at a Dagstuhl seminar on clone management[1]. We would like to thank Hamit Abdul Basit, Saman Bazrafshan, Daniel M. German, Nils Göde, and Martin P. Robillard for fruitful and inspiring discussions at the Dagstuhl seminar, leading to this position paper.

REFERENCES

[1] R. Tairas, J. Gray, and I. Baxter, "Visualization of Clone Detection Results," in *Eclipse technology eXchange*. ACM, 2006, pp. 50–54.
[2] J. Cordy, "Exploring Large-Scale System Similarity Using Incremental Clone Detection and Live Scatterplots," in *ICPC*, 2011, pp. 151–160.
[3] E. Juergens, F. Deissenboeck, and B. Hummel, "CloneDetective - A Workbench for Clone Detection Research," in *ICSE*, 2009, pp. 603–606.
[4] S. Diehl, *Software Visualization*. Springer, 2007.
[5] C. K. Roy and J. Cordy, "A Survey on Software Clone Detection Research," Queen's University at Kingston, Tech. Rep. 2007-541, 2007.
[6] S. Eick, J. Steffen, and J. Sumner, E.E., "Seesoft – A Tool for Visualizing Line Oriented Software Statistics ," *IEEE TSE*, vol. 18, no. 11, pp. 957–968, 1992.
[7] B. Johnson, "TreeViz: Treemap Visualization of Hierarchically Structured Information," in *CHI*, 1992, pp. 369–370.

[1]http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=12071