Type 2 Clone Detection On ASCET Models

Francesco Gerardi
Reutlingen University
Fakultät Informatik
Reutlingen, Germany
fgerardi@web.de

Jochen Quante
Robert Bosch GmbH
Corporate Research
Stuttgart, Germany
Jochen.Quante@de.bosch.com

Abstract

Clones are a well-known bad smell pattern in software. So far, research has concentrated on detection of clones in textual languages, while model-based development becomes increasingly important. This is in particular true for the automotive domain, where modelling languages like ASCET are used. This paper presents the adaptation and extension of an existing approach for detection of clones in models. The main novelty is a graph analysis that can detect clones of type 2 (i. e., identical structure, but renamed elements) and distinguish between consistently and inconsistently renamed model elements.

1 Introduction

Cloning means the duplication of artifacts within a system. This behavior is considered harmful for software development: Clones are the number one in the stink parade of bad smell patterns [2]. Such redundancies increase the effort for maintenance and consequently the costs for development of software systems.

Clone detection on the level of source code is a well-known and long-term topic in research. Over the years, a lot of approaches have been established for clone detection in textual programming languages [4]. The same problem is also present in higher-level programming, the modelling of software. In embedded software development, an executable application for control units is generated from data-flow oriented models, e. g., ASCET models[1]. Real world models often contain thousands of connected elements. This fact complicates the manual search for clones in such models or even makes the search impossible. However, there have only been very few activities in model clone detection research.

The seminal work in this area is from Deissenboeck et al. [1]. Their approach is publicly available as part of ConQAT[2]. It provides the base functionality for detecting clones in Matlab/Simulink models. We call it CMCD (ConQAT's Model Clone Detection) in this paper. The contribution of this paper is an adaption of this algorithm to ASCET models and its extension for detection of additional clone types.

[1] `http://www.etas.com/en/products/ascet_software_products.php`

[2] ConQAT is an open-source toolkit for various software analysis and quality measurements. `http://www.conqat.org/`

2 Adaption and Extensions

The core of CMCD works on a general graph structure with additional labels for each node and edge. Therefore, the existing ASCET model has to be transformed into a CMCD graph, and the labels have to be determined from the model. The result of CMCD are pairs of subgraphs that are clones of each other. This means that the subgraphs' structures are identical, as well as the labels. We extended the basic CMCD framework by further parameterizable preparation steps (Preprocessing) and detailed graph analyses (Postprocessing) for retrieving better interpretable results.

Preprocessing

Preprocessing includes the correct transformation from the ASCET-specific model elements (nodes) and connectors (directed edges) to the model structure as needed by CMCD as input. Furthermore, a canonical label is built for each model element. It depends on the user-selected clone type that should be detected. Table 1 shows how the use of different attributes as the canonical label allows detection of different clone types. This makes it possible to detect clones with nodes that have different visible labels in the original model, but have other commonalities. For example, a variable with a different name is still a variable.

CMCD's Clone Detection

CMCD starts its clone detection by first identifying clone candidates, which are pairs of nodes with matching labels. Next, each of these candidates is successively enlarged according to the graph structure and

	Attribute	Example
Type 1	Label	`"&&"`
Type 2	Class	`"Operator"`
Type 2x	Class+Category	`"OperatorLogical"`

Table 1: Variants of constructing the node labels.

labels. Only clones of a certain minimum size are reported. Optionally, clones are summarized into clone cluster, a representative form for many identical clone pairs. For details, see Deissenboeck's paper [1].

Postprocessing

The postprocessing step performs further graph analyses on CMCD's clone results. The following analyses and visualizations are provided:

- An overview graph shows which parts of the graph contain clones.
- Optionally, the number of occurrences of each node in all clones can be visualized to identify regions of intensive cloning.
- Each clone pair is checked for inconsistently or consistently renamed labels (for type 2(x))
- One detailed graph per clone pair shows this additional information.
- Uncloned nodes that are directly connected to a cloned node can optionally be included to get more context information.
- Statistics about the size of the found clone pairs, along with information about the number of consistently and inconsistently renamed nodes.

To distinguish between consistently and inconsistently renamed labels, the algorithm in Figure 1 is used. The algorithm assumes that a graph is defined by $G = (V, E, L)$ where V describes the set of nodes, E the set of directed edges and L the labelling function $L : V \cup E \to N$. While the subgraph $G_1 = (V_1, E_1, L_1)$ points to the original artifact, $G_2 = (V_2, E_2, L_2)$ means its corresponding duplicate. The function f maps a node from V_1 to its cloned counterpart.

The algorithm basically checks for each group of identically labelled nodes in G_1 (M) whether the corresponding set of nodes in G_2 also has a unique label (N). If this is the case, then the labels are renamed consistently, otherwise they are not. This check has to be performed in both directions.

Require: $G_1 = (V_1, E_1, L_1), G_2 = (V_2, E_2, L_2)$

1: $Q := \{v \mid v \in V_1 \wedge L_1(v) \neq L_2(f(v))\}$
2: **while** $|Q| > 0$ **do**
3: take one $q \in Q$ arbitrarily
4: $M := \{v \in Q \mid L_1(v) = L_1(q)\}$
5: $N := \{L_2(f(v)) \mid v \in M\}$
6: **if** $|N| > 1$ **then**
7: mark all $v \in M$ as **inconsistent** labels
8: **else**
9: mark all $v \in M$ as consistent labels
10: **end if**
11: $Q := Q \setminus M$
12: **end while**

Figure 1: Algorithm for detecting inconsistencies

3 Evaluation

The approach was implemented and applied to a number of real-world models with up to 1,500 nodes. The calculation was performed in reasonable time (< 35s). It delivered relevant clone pairs of remarkable size (up to 250 nodes). However, there were also a number of false positives among them, in particular for type 2(x) clones.

An indicator for detecting these false positives is the relationship between clone size and the number of inconsistent labels. Small clones with a big share of inconsistencies can mostly be classified as occasional matches. In contrast to that, large clones with a small proportion of inconsistent labels are more interesting and should be considered more intensively. The statistics produced by the postprocessing provides this information. By appropriate sorting of the data sets it is possible to identify relevant clone pairs.

4 Conclusion

So far, there was no possibility to perform tool supported clone detection on ASCET models. The adaptation of an existing approach for clone detection in models makes it now possible for ASCET. The additional postprocessing provides convenient visualizations and interpretation support. The algorithm for detecting inconsistencies is the first designed and implemented for model clones at all. It can as well be applied to clone detection on other kinds of models. The evaluation has shown potential for increasing the accuracy of results by adjusting parameters. Overall, this work is another building block for further improving the quality and efficiency of model-based embedded software development.

References

[1] F. Deissenboeck, B. Hummel, E. Jürgens, B. Schätz, S. Wagner, J.-F. Girard, and S. Teuchert. Clone detection in automotive model-based development. In *Proc. of 13th ICSE*, pages 603–612. ACM Press, 2008.

[2] M. Fowler and K. Beck. *Refactoring: Improving the design of existing code.* The Addison-Wesley object technology series. Addison-Wesley, Reading MA, 1999.

[3] F. Gerardi. Erkennung und Visualisierung von Klonen in modellbasierter eingebetteter Software. Bachelor's thesis, Reutlingen University, Germany, 2012.

[4] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, 2009.