# Visual Detection of Duplicated Code[*]

Matthias Rieger, Stéphane Ducasse
Software Composition Group, University of Berne
ducasse,rieger@iam.unibe.ch
http://www.iam.unibe.ch/∼scg/

**Abstract**

Code duplication is considered as bad practice that complicates the maintenance and evolution of software. Detecting duplicated code is a difficult task because of the large amount of data to be checked and the fact that *a priori* it is unknown which code part has been duplicated. In this paper, we present a tool called DUPLOC that supports code duplication detection in a visual and exploratory or an automatic way.

## 1 Why we need to deal with Duplicated Code

Duplicated code is a phenomenon that occurs frequently in large systems. Although duplication can have its justifications, it is considered bad practice.

The reasons programmers duplicate code are multi-fold [Bak95] :

- making a copy of a code fragment is simpler and faster than writing the code from scratch and, in addition, the fragment may already be tested so the introduction of a bug seems less likely.
- Efficiency considerations may make the cost of a procedure call seem too high a price.

In reverse engineering, we are interested in detecting and subsequently removing unjustified duplicated code for the following reasons:

**Removing bugs.** If one is sure that the code segment where a bug is found occurs only once in the system, one can be confident that the bug has been eradicated from the system.

[*]This work has been funded by the Swiss Government under Project no. NFS-2000-46947.96 and BBW-96.0015 as well as by the European Union under the ESPRIT program Project no. 21975.

**Reducing code bloat.** Refactoring duplicated code into one function reduces the size of the code and subsequently also of the executable.

**Repair design-flaws.** Code duplication may also indicate design problems like missing use of inheritance.

The detection of duplicated code is a difficult task because the amount of data to be checked is large, often thousand of lines of code in separate files. Furthermore, we do not know *a priori* what has been duplicated. In this paper, we present a tool called DUPLOC that supports code duplication detection.

## 2 Principle

Since any part of source code can be copied, we cannot search for specific program clichés but rather have to deduce them from the input itself by comparing every line with every other line of the system under scrutiny. This comparison produces an enormous amount of two-dimensional data, i.e. a matrix where each cell stands for a comparison between two lines of code and contains a value unless the two lines did not match [Hel95]. The next step is then to find zones of interest in the matrix. The matrix can be examined either with an automatic or with an exploratory approach:

**Automatic Examination:** A tool searches for some known configurations of dots in the matrix and delivers a report of the instances that were found. This approach is efficient in that it finds interesting spots automatically which is convenient for large amounts of data. However, the dot configurations to look for must be programmed beforehand. The objects of the automated search can range from simple diagonals with holes like in the example Figure 1 *b)* to dot clusters found using statistical methods.

**Visual Exploration:** Using a tool that displays the comparison matrices graphically, the engineer browses through the matrix, zooms in on interesting spots, and, by clicking on the dots, examines the source code that belongs to a certain match. This exploratory approach and can lead to unexpected findings (see the examples in [Hel95]).

Some interesting configurations formed by the dots in the matrices are the following:

- diagonals of dots indicate copied sequences of source code (see Figure 1 *a)*).
- sequences that have holes in them indicate that a portion of a copied sequences has been changed (see Figure 1 *b)*).
- broken sequences with lower parts shifted indicate that a new portion of code has been inserted (see Figure 1 *c)*).

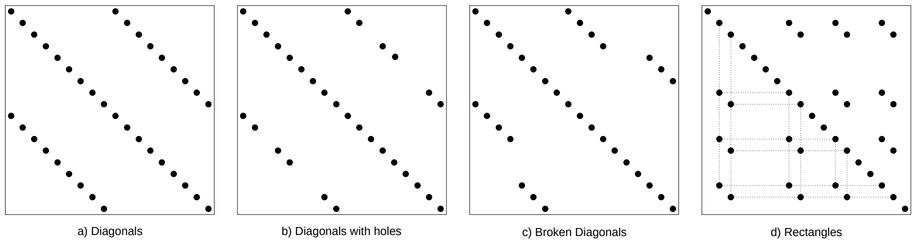


Figure 1: Different Configurations of Dots.

- rectangular configurations indicate periodic occurrences of the same code. An example is the break at the end of the individual cases in a C/C++ switch statement or recurring preprocessor commands like #ifdef SOME_CONDITION (see Figure 1 *d)*).

## 3 Duploc

We have implemented a tool, called DUPLOC, which compares source code line–by–line and displays a two-dimensional comparison matrix. The tool is implemented using VISUALWORKS 2.5 and available at http://www.iam.unibe.ch/∼rieger/duploc/

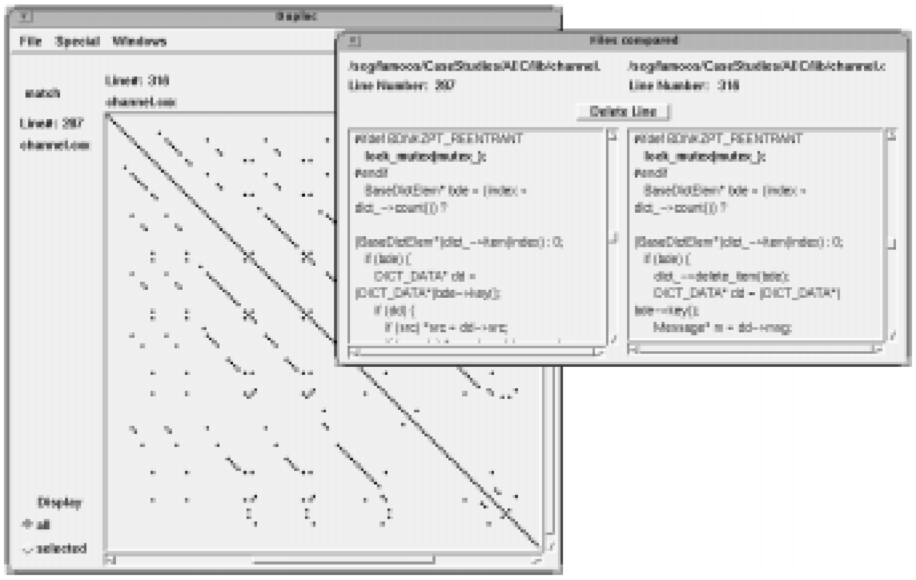
### 3.1 Features

DUPLOC reads source code lines and, by removing comments and superfluous white space, generates a 'normal form'–representation of a line. These lines are then compared using a simple string matching algorithm. DUPLOC offers a clickable matrix which allows the user to look at the source code that produced the match as shown in Figure 2. DUPLOC has the possibility to remove noise from the matrix by 'deleting' lines that do not seem interesting, e.g. the public: or private: specifiers in C++ class declarations. Moreover, DUPLOC offers a batchmode which allows to search for duplicated code in the whole of a system offline. A report file, called a *map*, is then generated which lists all the occurrences of duplicated code in the system.

### 3.2 Evaluation

The tool evaluates favorably according to the following requirements which were formulated for re-engineering tools in [HEH$^+$96, LN95]:

**Exploratory.** The visual presentation of the duplicated code allows the user to detect structures or configurations of dots that are unknown or new. The tool does not



/scg/famoos/CaseStudies/AEC/lib/channel.  
Line Number: 287

```
#ifdef SDNKZPT_REENTRANT
  lock_mutex(mutex_);
#endif
  BaseDictElem* bde = (index <
dict_->count()) ?

(BaseDictElem*)dict_->item(index) : 0;
  if (bde) {
    DICT_DATA* dd =
(DICT_DATA*)bde->key();
    if (dd) {
      if (src) *src = dd->src;
```

/scg/famoos/CaseStudies/AEC/lib/channel.c  
Line Number: 316

```
#ifdef SDNKZPT_REENTRANT
  lock_mutex(mutex_);
#endif
  BaseDictElem* bde = (index <
dict_->count()) ?

(BaseDictElem*)dict_->item(index) : 0;
  if (bde) {
    dict_->delete_item(bde);
    DICT_DATA* dd = (DICT_DATA*)
bde->key();
    Message* m = dd->msg;
```

Figure 2: The DUPLOC main window and a source code viewer

impose a certain view on the data, making unexpected findings possible.

**Browsing views.** The tool allows one to look at the whole system or at specific parts (files) of it. Scrolling and zooming support is provided.

**Language independent.** Due to the simplicity of the comparison algorithm (string matching), we do not depend on parsers for a specific language.

**Forgiving.** The comparison is not based on source text that has been parsed, so there can be any number of syntax errors in it which do not break the algorithm.

**Complete.** The *map* gives us an overview of the structure of code duplication that occurs in a system as a whole.

**Parameterized.** The comparison algorithm is easily exchanged. The matrix display can be enhanced giving different grey– or color–values to the individual dots to show the percentage of a fuzzy match. It is then also possible to visualize another data dimension. This could be used, for example, to identify different kinds of statements.

**Platform independent.** The tool is implemented in SMALLTALK and runs on UNIX, Mac and Windows platforms.

The drawbacks of the approach can be summarized as follows:

**Detection Only.** The tool can help in detecting occurrences of duplicated code. It does not help to actually decide what should be done with the code, e.g. if it should be left untouched or be re-factored.

**Syntactic level.** The tool compares only syntactic elements of the program. This implies that it does not discover duplicated *functionality*.

**Comparison Mechanism.** Using a simple comparison mechanism like string matching, code parts that were slightly changed will not be recognized. More sophisticated algorithms can be employed [Bak97].

## 4 Related and Future Work

Other work that involves computing comparison matrices is the following:

- DUP [Bak92] is a tool that detects parameterized matches and generates reports on the found matches. It can also generate scatter-plots of found matches. This tools does not support exploration and navigation through the duplicated code.
- DOTPLOT [Hel95] is a tool for displaying large scatter-plots. It has been used to compare source code, but also filenames in a filesystem and literary texts. However, DOTPLOT does not support source code browsing and the *map* facilities.

In the future, we plan to experiment with different algorithms for fuzzy matching. We will also investigate if textual comparison is a useful approach for exploring the structure of data other than source code, like for example program execution traces, where it could be interesting to find recurring patterns of execution sequences.

We also plan to extend the visualization capability of DUPLOC by implementing the *Information Mural* algorithm which allows to visualize large amounts of data on a single screen [JS96].

As a third valuable direction we see the automatisation of the detection process, i.e. the identification of meaningful dot configurations, other than the diagonal sequences we can detect already.

## References

[Bak92] Brenda S. Baker. A Program for Identifying Duplicated Code. *Computing Science and Statistics*, 24:49–57, 1992.

[Bak95] Brenda S. Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In *Proc. Second IEEE Working Conference on Reverse Engineering*, pages 86–95, July 1995.

[Bak97] Brenda S. Baker. Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance. *SIAM J. Computing*, October 1997.

[HEH$^+$96] J.-L. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, and D. Roland. Database reverse engineering: From requirements to CARE tools. In *Automated Software Engineering, Vol. 3 Nos 1/2, June 1996*. 1996. Gives a very good overview on issues and considerations for (database) reverse engineering.

[Hel95] Jonathan Helfman. Dotplot Patterns: a Literal Look at Pattern Languages. *TAPOS*, 2(1):31–41, 1995.

[JS96] Dean F. Jerding and John T. Stasko. The Information Mural: Increasing Information Bandwidth in Visualizations. Technical Report GIT-GVU-96-25, Georgia Institute of Technology, october 1996.

[LN95] D.B. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. In *Proceedings of OOPSLA'95*, pages 342–357, 1995. mixin dynamic and static information for model capture.