Software Clone Detection

State-of-the-Art Survey

Rainer Koschke
University of Bremen, Germany

Universität Bremen

BAUHAUS
SOFTWARE
TECHNOLOGIES

Axivion

Saarbrücken, June 28 2007

No two parts are alike in software...

> *Software entities are more complex for their size than perhaps any other human construct because* **no two parts are alike** *(at least above the statement level).* **If they are, we make the two similar parts into a subroutine** *— open or closed. In this respect, software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound.*

– by Frederick P. Brooks, Jr: *No Silver Bullet: Essence and Accidents of Software Engineering*

Software Redundancy

copy&paste is common habit:

- number 1 on Beck and Fowler's "Stink Parade of Bad Smells"
- typically 5–30 % of code is similar (Baker, 1995; Baxter et al., 1998)
- in extreme cases, even up to 50 % (Ducasse et al., 1999)

Roadmap I

1. What is a clone?
2. Why do they exist?
3. What are the consequences of cloning?
4. What are costs and benefits of clone removal?
5. How do clones evolve?
6. How can we detect clones?
7. How can we compare clone detectors?

Roadmap II

8. How can we present clones to a user?

9. Clone Detection in Forward Engineering

What is a software clone?

> *Software clones are segments of code that are similar according to some definition of similarity.*
>
> – Ira Baxter, 2002

There can be different definitions of similarity based on . . .

- text
- syntax
- semantics
- pattern

What types of clones exist?

Clone detection experiment (Bellon, 2002a):

- type 1: identical code segments except for differences in layout and comments
- type 2: structurally identical segments except for differences in identifiers, literals, layout, and comments
- type 3: similar segments (additions, modifications, removals of statements)
- type 4: semantically equivalent segments

→ degree of similarity

properties:

- type-1, type-2, and type-4 clones form an equivalence relation
- semantic equivalence guaranteed only for type-4 clones

Open Issues

- What are suitable definitions of similarity for which purpose?
- Is there a theory of program redundancy similar to normal forms in databases?
- What other categorizations of clones make sense (e.g., syntax, semantics, origins, risks, etc.)?
- What is the statistical distribution of clone types in real-world programs?
- Which strategies of removal and avoidance, risks of removal, potential damages, root causes, and other factors are associated with these categories?

Why do clones exist?

Ethnographic study by Kim et al. (2005):

- Limitations of programming language designs may result in unavoidable duplicates in a code.
- Programmers often delay code restructuring until they have copied and pasted several times.
- Copy&paste dependencies often reflect important underlying design decisions, such as crosscutting concerns.
- Copied text is often reused as a template and is customized in the pasted context.

Investigation of clones in large systems by Kapser and Godfrey (2006): patterns of cloning:

- forking
- templating
- customization

Open Issues

More empirical research needed. Other potential reasons:

- insufficient information on global change impact
- badly organized reuse process (type-4 clones)
- questionable productivity measures (LOCs per day)
- time pressure
- educational deficiencies, ignorance, or shortsightedness
- intellectual challenges (e.g., generics)
- professionalism/end-user programming (e.g., HTML, Visual Basic, etc.)
- development process (Nickell and Smith (2003): XP yields less clones?)
- organizational issues, e.g., distributed development organizations

→ fight the reasons, not just the symptoms

What are the consequences of cloning?

Only plausible arguments, such as clones increase maintenance effort.

Very few empirical studies on effects of cloning

Monden et al. (2002):

- 2,000 Cobol modules with clones with at least 30 lines (1 MLOC, 20 years old)
- max clone length versus change frequency and number of errors

→ most errors in modules with a 200-line clone

→ many errors for modules with clones of less than 30 lines, too

→ lowest error rate for modules with 50-100–line clones

What are the consequences of cloning?

Chou et al. (2001) investigate hypothesis that if a function, file, or directory has one error, it is more likely that is has others

- additional observation for Linux and OpenBSD:
  - this phenomenon can be observed most often where programmer ignorance of interface or system rules combines with copy-and-paste

  → programmers believe that "working" code is correct code

  → if copied code is incorrect, or it is placed into a context it was not intended for, the assumption of goodness is violated

What are the consequences of cloning?

Li et al. (2006) use clone detection to find bugs when programmers copy code but rename identifiers in the pasted code inconsistently.

Systems analyzed: *Linux kernel*, *FreeBSD*, *Apache*, and *PostgreSQL*.

Findings:

- 13 % of the clones flagged as copy-and-paste bugs turned out to be real errors
- 73 % are false positives
- 14 % of the potential problems are still under analysis by the developers of the analyzed systems.

Open Issues

More empirical research needed on relation of cloning to quality attributes (bugs, costs, performance, etc.).

What are costs and benefits of clone removal?

We know various techniques to remove clones:

- automatic refactoring (Fanta and Rajlich, 1999)
- functional abstraction (Komondoor and Horwitz, 2002)
- macros (e.g., *CloneDr* by *Semantic Designs*)
- design patterns (Balazinska et al., 1999, 2000)

Cordy (2003) argues that companies are afraid of the risks.

What are costs and benefits of clone removal?

clone detection integrated in development process (Lague et al., 1997):

(1) preventive control: addition of a clone is reported for confirmation

(2) problem mining: find other pieces of code to be changed

benefits analyzed post-mortem:

(1) is assessed by the number of functions changed that have clones that were not changed; i.e., how often a modification was missed potentially

(2) is assessed by the number of functions added that were similar to existing functions; i.e., the code that could have been saved

What are costs and benefits of clone removal?

Open Issues

Empirical investigations of costs and benefits of clone removal are needed:

- clone types and their relation to quality attributes
- relevance ranking of clone types
- suitable removal techniques with costs and risks

How do clones evolve?

- Cloning is common and steady practice in Linux kernel (Godfrey and Tu, 2000, 2001; Antoniol et al., 2001, 2002)
- Clone genealogies (Kim et al., 2005):
  - show how clones derive in time over multiple versions of a program from common ancestors
  - many code clones exist in the system for only a short time

  → extensive refactoring of such short-lived clones may not be worthwhile if they likely diverge from one another very soon
  - many long-living clones that have changed consistently with other elements in the same group cannot easily be avoided because of limitations of the programming language.

How do clones evolve?

Open Issues

- How do clones evolve in industrial systems?
- What does their evolution tell about the development organization?
- What affects cloning likelihood over time?
- How we can track and manage clones over versions?
- Can we use history information to improve clone detectors?

How can we detect clones?

Comparison of . . .

- text
    - string comparison (Johnson, 1993, 1994b) based on fingerprints (Karp, 1986; Karp and Rabin, 1987)
    - line comparison based on dot plots (Ducasse et al., 1999; Rieger, 2005)
    - for whole files (Manber, 1994)
- identifiers (information retrieval techniques)
    - latent semantic indexing (Marcus and Maletic, 2001)

How can we detect clones?

Comparison of . . .

- tokens
  - type-1/-2 clones: suffix trees (McCreight, 1976; Kosaraju, 1995) for parameterized strings **per line** (Baker, 1992, 1993, 1995, 1996, 1997, 1999)
  - type-3: dynamic programming (Baker and Giancarlo, 2002).
  - **per token** plus normalization of token stream (Ueda et al., 1999; Inoue et al., 2001; Kamiya et al., 2002, 2001b; Kamiya, 2001; Nakae et al., 2001; Ueda et al., 2001, 2002a,b; Kamiya et al., 2001a)
  - post-processing to find clones fully contained in syntactic unit (Higo et al., 2002)
  - pre-processing to find clones fully contained in syntactic unit (Synytskyy et al., 2003; Cordy et al., 2004) using island parsers (Moonen, 2001)
  - parsing to obtain syntactic scopes (Gitchell and Tran, 1999)

How can we detect clones?

Comparison of . . .

- metrics (Kontogiannis et al., 1994, 1995; Kontogiannis, 1997; Mayrand et al., 1996; Kontogiannis et al., 1996b,a; Lague et al., 1997; Balazinska et al., 1999, 2000; Merlo et al., 2002; Patenaude et al., 1999; Merlo et al., 2004)
    - for web sites (Di Lucca et al., 2002; Lanubile and Mallardo, 2003)
- statements using data mining (Wahler et al., 2004; Li et al., 2004)

How can we detect clones?

Comparison of . . .

- syntax trees
  - hashing plus tree matching (Baxter et al., 1998; Leitao, 2003)
  - tree matching plus dynamic programming (for file comparison) (Yang, 1991)
  - suffix trees for serialized syntax trees (Falke, Frenzel, and Koschke, 2006)
- program dependency graphs (Komondoor and Horwitz, 2001a,b; Krinke, 2001)

How can we detect clones?

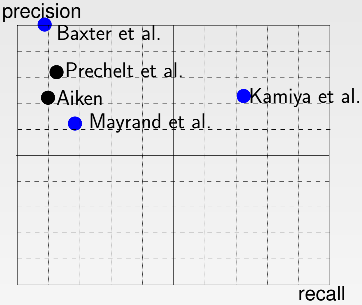
Combined approach by Leitao (2003)

- syntactic transformations in canonical form
- plus semantics in terms of comparison functions for
  - call graphs
  - commutative operators
  - user-defined equivalences that yield degree of evidence

How can we compare clone detectors?

Quantitative comparison of clone detectors by Bailey and Burd (2002)

- human oracle for 16 KLOC program
- comparison of
  - token-based (Kamiya et al., 2002)
  - syntax-based (Baxter et al., 1998)
  - metric-based (Mayrand et al., 1996) (reimplemented)
  - token-based (plagiarism) (Prechelt et al., 2000)
  - text-based (plagiarism) (Schleimer et al., 2003)



How can we compare clone detectors?

Quantitative comparison of clone detectors by Bellon and Koschke (2002b; 2007) for 4 Java and 4 C systems of 850 KLOC in total

	Baker	Baxter	Kamiya	Krinke	Merlo	Rieger
Basis	Token	AST	Token	PDG	Metric	Text
Clone type	1, 2	1, 2	1, 2, 3	3	1, 2, 3	1, 2, 3
Speed	+ +	-	+	- -	+ +	?
RAM	+	-	+	+	+ +	?
Recall	+	-	+	-	-	+
Precision	-	+	-	-	+	-
Hidden	42 %	28 %	46 %	4 %	24 %	31 %

Later re-used and extended by Falke, Frenzel, and Koschke, 2006

How can we compare clone detectors?

Qualitative study by Van Rysselberghe and Demeyer (2004):

- text-based Ducasse et al. (1999); Rieger (2005)
- token-based Baker (1995)
- metric-based Mayrand et al. (1996)

Criteria:

- how easy it is to adapt to a new language
- recall and precision
- effort
- suitability for particular task

Conclusions:

- text-based detection suitable for first overview
- token-based detection suitable for refactoring at statement level
- metric-based detection suitable for refactoring at method level

Benchmarks

Open Issues

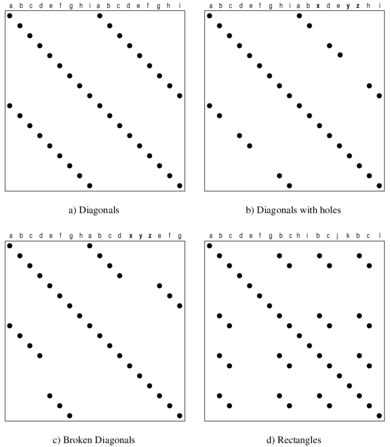
Limitations of current benchmarks

- single oracle (until recently)
  - differences among different human raters for clone candidates (Walenstein et al., 2003) when clones ought to be removed.
- yes/no decision rather than degree of confidence
- clones length measured as lines rather than tokens
- insists on contiguous lines/tokens
- clone pairs rather than clone classes

Benchmarking should become standard procedure of the community.

How can we present clones to a user?

Dot plots (Church and Helfman, 1993; Ducasse et al., 1999; Ueda et al., 2002b):


a b c d e f g h i a b c d e f g h i
a b c d e f g h i a b x d e y z h i
a b c d e f g h a b c d x y z e f g
a b c d e f g b c h i b c j k b c l

a) Diagonals

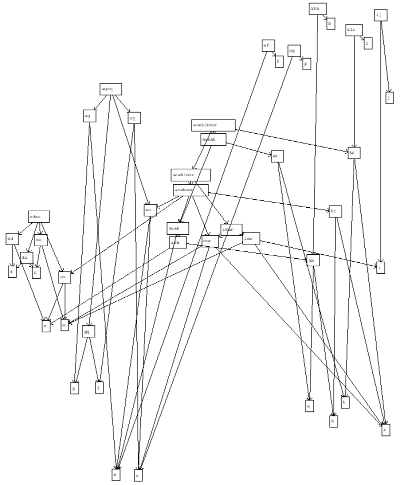
b) Diagonals with holes

c) Broken Diagonals

d) Rectangles

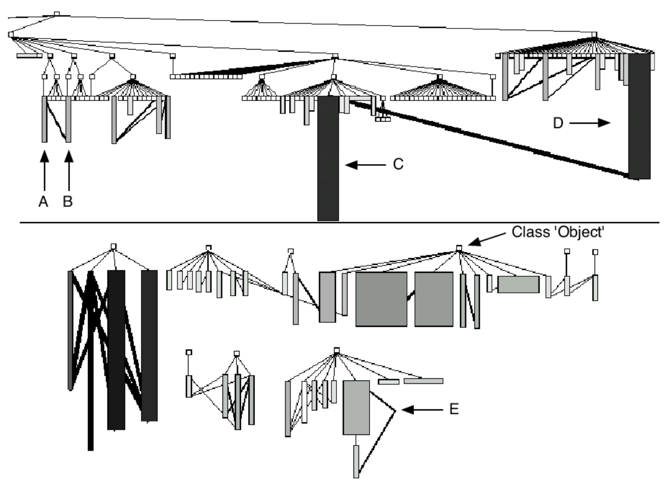
How can we present clones to a user?

Hasse diagram by Johnson (1994a):



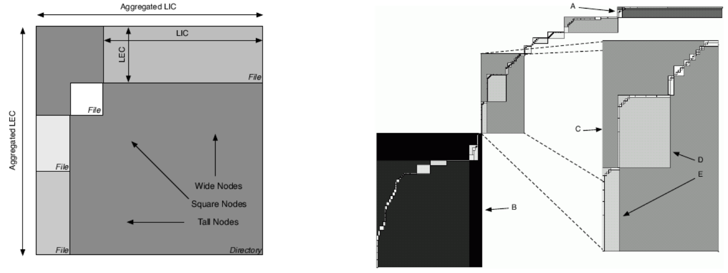
How can we present clones to a user?

Polymetric view by Rieger et al. (2004):



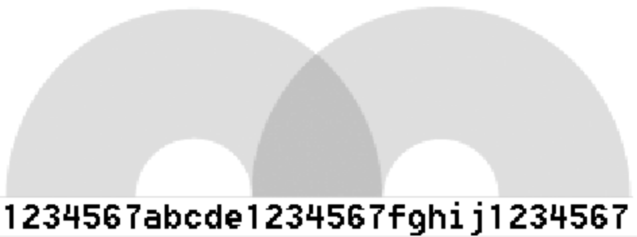
How can we present clones to a user?

Tree map variation by Rieger et al. (2004):



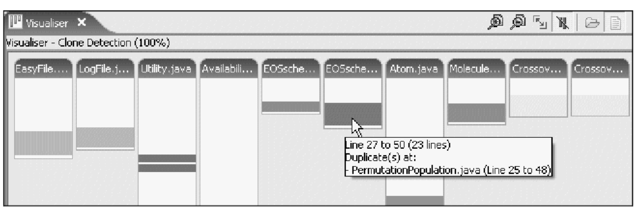
How can we present clones to a user?

Arc diagram by Wattenberg (2002):



How can we present clones to a user?

Clones visualiser view in Eclipse (Tairas et al., 2006):



Line 27 to 50 (23 lines)
Duplicate(s) at:
- PermutationPopulation.java (Line 25 to 48)

How can we present clones to a user?

Open Issues

No systematic empirical research on the appropriate type of visualization for a particular task (clone monitoring, detection, removal, etc.).

Tool Support in Forward Engineering

Current software engineering tools have poor support for identifying reusable code templates or maintaining them during software evolution.

– Kim et al. (2005)

Cloning is a good strategy if you have the right tools in place. Let programmers copy and adjust, and then let tools factor out the differences with appropriate mechanisms.

– Ira Baxter, 2002

Further Reading and Resources

- http://www.informatik.uni-bremen.de/st/lehredetails.php?id=&lehre_id=44
  Lecture on software reengineering (slides and video) including techniques for clone detection
- http://www.bauhaus-stuttgart.de/clones/
  Material on experiment to compare clone detectors
- http://drops.dagstuhl.de/portals/index.php?semnr=06301
  Dagstuhl seminar on clone detection, slides and proceedings

A. Aiken. A system for detecting software plagiarism (moss homepage). Web Site, 2002. URL `http://www.cs.berkeley.edu/~aiken/moss.html`.

G. Antoniol, G. Casazza, M. Di Penta, and E. Merlo. Modeling clones evolution through time series. In International Conference on Software Maintenance, pages 273–280. IEEE Computer Society Press, 2001.

G. Antoniol, U. Villano, E. Merlo, and M.D. Penta. Analyzing cloning evolution in the linux kernel. Information and Software Technology, 44 (13), 2002.

John Bailey and Elizabeth Burd. Evaluating clone detection tools for use during preventative maintenance. In Workshop Source Code Analysis and Manipulation, pages 36–43. IEEE Computer Society Press, 2002. URL `http://csdl.computer.org/comp/proceedings/scam/2002/1793/00/17930036abs.htm`.

Brenda Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. SIAM Journal on Computing, 26 (5):1343–1362, October 1997.

Brenda S. Baker. A theory of parameterized pattern matching: Algorithms and applications (extended abstract). In Proc. 25th ACM Symposium on Theory of Computing, pages 71–80, May 1993.

Brenda S. Baker. On finding duplication and near-duplication in large software systems. In L. Wills, P. Newcomb, and E. Chikofsky, editors, Second Working Conference on Reverse Engineering, pages 86–95, Los Alamitos, California, July 1995. IEEE Computer Society Press. URL `http://citeseer.nj.nec.com/baker95finding.html`.

Brenda S. Baker. Parameterized diff. In ACM-SIAM Symp. on Discrete Algorithms, page S854S855. ACM Press, January 1999.

Brenda S. Baker. Parameterized Pattern Matching: Algorithms and Applications. Journal Computer System Science, 52(1):28–42, February 1996. URL `http://citeseer.nj.nec.com/baker94parameterized.html`.

Brenda S. Baker. A program for identifying duplicated code. In Computer Science and Statistics 24: Proceedings of the 24th Symposium on the Interface, pages 49–57, March 1992. URL http://citeseer.nj.nec.com/baker92program.html.

Brenda S. Baker and Raffaele Giancarlo. Sparse dynamic programming for longest common subsequence from fragments. Journal Algorithms, 42 (2):231–254, February 2002.

M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Partial redesign of java software systems based on clone analysis. In Working Conference on Reverse Engineering, pages 326–336. IEEE Computer Society Press, 1999.

Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and Kostas Kontogiannis. Advanced clone-analysis to support object-oriented system refactoring. In Working Conference on Reverse Engineering, pages 98–107. IEEE Computer Society Press, October 2000.

Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone Detection Using Abstract Syntax Trees. In T. M. Koshgoftaar and K. Bennett, editors, International Conference on Software Maintenance, pages 368–378. IEEE Computer Society Press, 1998. ISBN 0-7803-5255-6, 0-8186-8779-7, 0-8186-8795-9.

Stefan Bellon. Detection of software clones – tool comparison experiment, 2002a. URL `http://www.bauhaus-stuttgart.de/clones/`. Official homepage of the experiment.

Stefan Bellon. Vergleich von techniken zur erkennung duplizierten quellcodes. Diploma thesis, no. 1998, University of Stuttgart (Germany), Institute for Software Technology, September 2002b.

Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. IEEE Computer Society Transactions on Software Engineering, 2007. Accepted for publication.

Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson R. Engler. An empirical study of operating system errors. In Symposium on Operating Systems Principles, pages 73–88, 2001. URL `citeseer.ist.psu.edu/chou01empirical.html`.

K. W. Church and J. I. Helfman. Dotplot: A program for exploring self-similarity in millions of lines for text and code. Journal of American Statistical Association, Institute for Mathematical Statistics and Interface Foundations of North America, 2(2):153–174, June 1993.

James R. Cordy, Thomas R. Dean, and Nikita Synytskyy. Practical language-independent detection of near-miss clones. In Conference of the Centre for Advanced Studies on Collaborative research, pages 1–12. IBM Press, 2004.

J.R. Cordy. Comprehending reality: Practical challenges to software maintenance automation. In International Workshop on Program Comprehension, pages 196–206. IEEE Computer Society Press, 2003.

G.A. Di Lucca, M. Di Penta, and A.R. Fasolino. An approach to identify duplicated web pages. In International Computer Software and Applications Conference, pages 481–486, 2002.

Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A Language Independent Approach for Detecting Duplicated Code. In International Conference on Software Maintenance, pages 109–118, 1999.

Raimar Falke, Pierre Frenzel, and Rainer Koschke. Clone detection using abstract syntax suffix trees. In Working Conference on Reverse Engineering. IEEE Computer Society Press, 2006.

Richard Fanta and Václav Rajlich. Removing clones from the code. Journal on Software Maintenance and Evolution, 11(4):223–243, July/Aug. 1999.

David Gitchell and Nicholas Tran. Sim: a utility for detecting similarity in computer programs. In SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education, pages 266–270. ACM Press, 1999. ISBN 1-58113-085-6. doi: http://doi.acm.org/10.1145/299649.299783.

M. Godfrey and Q. Tu. Growth, evolution and structural change in open source software. In Workshop on Principles of Software Evolution, September 2001.

M. Godfrey and Q. Tu. Evolution in open source software: A case study. In International Conference on Software Maintenance. IEEE Computer Society Press, 2000.

Yoshiki Higo, Yasushi Ueda, Toshihro Kamiya, Shinji Kusumoto, and Katsuro Inoue. On software maintenance process improvement based on code clone analysis. In International Conference on Product Focused Software Process Improvement, volume 2559 of Lecture Notes In Computer Science, pages 185–197. Springer, 2002. ISBN ISBN:3-540-00234-0.

Katsuro Inoue, Toshihiro Kamiya, and Shinji Kusumoto. Method for detecting code clones. Computer Software, 18(5):47–65, 2001. in Japanese.

J. Howard Johnson. Identifying redundancy in source code using fingerprints. In Conference of the Centre for Advanced Studies on Collaborative research, pages 171–183. IBM Press, 1993.

J. Howard Johnson. Visualizing textual redundancy in legacy source. In Conference of the Centre for Advanced Studies on Collaborative research, page 32. IBM Press, 1994a.

J. Howard Johnson. Substring matching for clone detection and change tracking. In International Conference on Software Maintenance, pages 120–126. IEEE Computer Society Press, 1994b.

T. Kamiya, F. Ohata, K. Kondou, S. Kusumoto, and K. Inoue. Maintenance support tools for java programs: Ccfinder and jaat. In International Conference on Software Engineering, pages 837–838, 2001a.

Toshihiro Kamiya. Code clone detection method. In Proceedings of Winter Workshop in Kanazawa, IPSJ SIGSE, pages 21–22, 2001.

Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoui. A token-based code clone detection technique and its evaluation. IEICE SS2000-42, 100(570):41–48, 2001b.

Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code. IEEE Computer Society Transactions on Software Engineering, 28(7):654–670, 2002.

Cory Kapser and Michael W. Godfrey. "clones considered harmful" considered harmful. In Working Conference on Reverse Engineering, 2006.

R. M. Karp. Combinatorics, complexity, and randomness. Communications of the ACM, 29(2):98–109, February 1986.

R. M. Karp and M. 0. Rabin. Efficient randomized pattern-matching algorithms. IBM Journal Research and Development, 31(2):249–260, March 1987.

Miryung Kim, Vibha Sazawal, David Notkin, and Gail C. Murphy. An empirical study of code clone genealogies. In European Software Engineering Conference and Foundations of Software Engineering (ESEC/FSE, 2005.

R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In Proc. Int. Symposium on Static Analysis, pages 40–56, July 2001a.

Raghavan Komondoor and Susan Horwitz. Eliminating duplication in source code via procedure extraction. Technical report 1461, UW-Madison Dept. of Computer Sciences, December 2002.

Raghavan Komondoor and Susan Horwitz. Tool Demonstration: Finding Duplicated Code Using Program Dependences. In European Symposium on Programming, volume 2028 of Lecture Notes in Computer Science, page 383ff, 2001b. URL `http://citeseer.nj.nec.com/487759.html`.

K. Kontogiannis. Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics. In Working Conference on Reverse Engineering, 1997.

K. Kontogiannis, R. DiMori, M. Bernstein, and E. Merlo. Localization of design concepts in legacy systems. In International Conference on Software Maintenance, pages 414–423. IEEE Computer Society Press, 1994.

K. Kontogiannis, R. DeMori, M. Bernstein, M. Galler, and E. Merlo. Pattern matching for design concept localization. In Working Conference on Reverse Engineering, pages 96–103. IEEE Computer Society Press, July 1995.

K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M.Bernstein. Pattern matching techniques for clone detection. Journal of Automated Software Engineering, 3:77–108, 1996a.

K. Kontogiannis, R. De Mori, E. Merlo, M. Galler, and M. Bernstein. Pattern matching for clone and concept detection. Automated Software Engineering, 3(1/2):79–108, June 1996b.

S. Rao Kosaraju. Faster algorithms for the construction of parameterized suffix trees. In In Thirty-sixth Annual Symposium on Foundations of Computer Science (FOCS'95), pages 631–639, 1995.

Jens Krinke. Identifying Similar Code with Program Dependence Graphs. In Working Conference on Reverse Engineering, pages 301–309, 2001.

B. Lague, D. Proulx, J. Mayrand, E.M. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In International Conference on Software Maintenance, pages 314–321, 1997.

F. Lanubile and T. Mallardo. Finding function clones in web applications. In European Conference on Software Maintenance and Reengineering, pages 379–386, 2003.

Antonio Menezes Leitao. Detection of redundant code using r2d2. In Workshop Source Code Analysis and Manipulation, pages 183–192. IEEE Computer Society Press, 2003.

Z. Li, S. Lu, S. Myagmar, and Y. Zhou. Cp-miner: A tool for finding copy-paste and related bugs in operating system code. In Operating System Design and Implementation, pages 289–302, 2004.

Z Li, S Lu, S. Myagmar, and Y. Zhou. Copy-paste and related bugs in large-scale software code. IEEE Computer Society Transactions on Software Engineering, 32(3):176–192, March 2006.

Udi Manber. Finding similar files in a large file system. In Proceedings of the Winter Usenix Technical Conference, pages 1–10, 1994.

A. Marcus and J.I. Maletic. Identification of high-level concept clones in source code. In International Conference on Automated Software Engineering, pages 107–114, 2001.

J. Mayrand, C. Leblanc, and E.M. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In International Conference on Software Maintenance, pages 244–253, 1996.

E. McCreight. A space-economical suffix tree construction algorithm. Journal of the ACM, 32(2):262–272, 1976.

E. Merlo, M. Dagenais, P. Bachand, J.S. Sormani, S. Gradara, and G. Antoniol. Investigating large software system evolution: the linux kernel. In International Computer Software and Applications Conference, pages 421–426, 2002.

E. Merlo, G. Antoniol, M. Di Penta, and V.F. Rollo. Linear complexity object-oriented similarity for clone detection and software evolution analyses. In International Conference on Software Maintenance, pages 412–416, 2004.

A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto. Software quality analysis by code clones in industrial legacy software. In IEEE Symposium on Software Metrics, pages 87–94, 2002.

L. Moonen. Generating robust parsers using island grammars. In Proceedings of the Working Conference on Reverse Engineering, pages 13–22. IEEE Computer Society Press, Oct. 2001.

Daikai Nakae, Toshihiro Kamiya, Akito Monden, Hiroshi Kato, Shin ichi Sato, and Katsuro Inoue. Quantitative analysis of cloned code on legacy software. IEICE, 100(570):57–64, 2001.

Eric Nickell and Ian Smith. Extreme programming and software clones. In Working Conference on Reverse Engineering. IEEE Computer Society Press, 2003.

J.-F. Patenaude, E. Merlo, M. Dagenais, and B. Lague. Extending software quality assessment techniques to java systems. In International Workshop on Program Comprehension, pages 49–56, 1999.

L. Prechelt, G. Malpohl, and M. Philippsen. Jplag: Finding plagiarisms among a set of programs. Technical report, University of Karlsruhe, Department of Informatics, 2000.

M. Rieger, S. Ducasse, and M. Lanza. Insights into system-wide code duplication. In Working Conference on Reverse Engineering, pages 100–109. IEEE Computer Society Press, 2004.

Matthias Rieger. Effective Clone Detection Without Language Barriers. Dissertation, University of Bern, Switzerland, 2005.

S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In Proceedings of the SIGMOD Aiken (2002), pages 76–85. URL `http://www.cs.berkeley.edu/~aiken/moss.html`. Web Site.

Nikita Synytskyy, James R. Cordy, and Thomas Dean. Resolution of static clones in dynamic web pages. In Workshop on Website Evolution, pages 49–56, 2003.

Robert Tairas, Jeff Gray, and Ira Baxter. Visualization of clone detection results. In Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange, pages 50–54, 2006.

Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. On detection of gapped code clones using gap locations. In Proceedings Ninth Asia-Pacific Software Engineering Conference (APSEC'02), pages 327–336. IEEE Computer Society Press, December 1999.

Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Source code analysis system using code clone detection tool. IEICE, 101 (240):17–24, 2001. in Japanese.

Yasushi Ueda, Yoshiki Higo, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Gemini: Code clone analysis tool. In International Symposium on Empirical Software Engineering, volume 2, pages 31–32, 2002a.

Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Gemini: Maintenance support environment based on code clone analysis. In IEEE Symposium on Software Metrics, pages 67–76. IEEE Computer Society Press, 2002b.

F. Van Rysselberghe and S. Demeyer. Evaluating clone detection techniques from a refactoring perspective. In International Conference on Automated Software Engineering, 2004.

V. Wahler, D. Seipel, Jürgen Wolff von Gudenberg, and G. Fischer. Clone detection in source code by frequent itemset techniques. In Workshop Source Code Analysis and Manipulation, pages 128–135, 2004.

Andrew Walenstein, Nitin Jyoti, Junwei Li, Yun Yang, and Arun Lakhotia. Problems creating task-relevant clone detection reference data. In Working Conference on Reverse Engineering. IEEE Computer Society Press, 2003.

Martin Wattenberg. Arc diagrams: Visualizing structure in strings. In Proceedings InfoVis, pages 10–17. IEEE Computer Society Press, 2002.

Wuu Yang. Identifying syntactic differences between two programs. Software–Practice and Experience, 21(7):739–755, July 1991.