A State-of-the-Art Survey on Software Clones

Rainer Koschke
University of Bremen, Germany

First Indian Workshop on Reverse Engineering (IWRE)
Feb 25th , 2010, Mysore, India

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"
- reported redundancy:

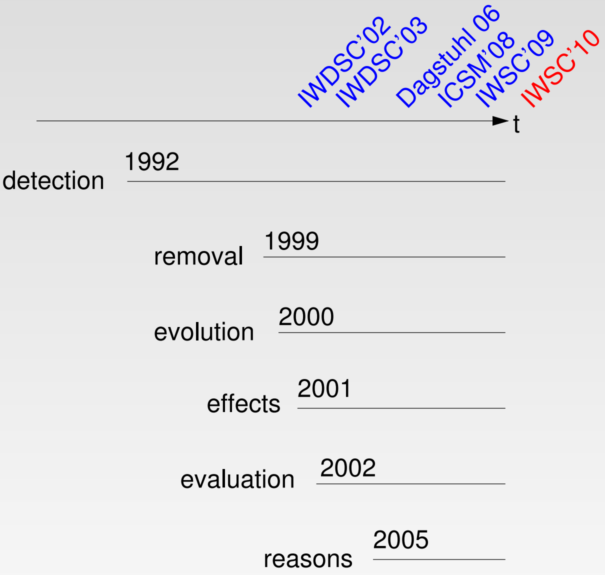
%	system	lines	citation
19	X Windows	≥ 30	Baker (1995)
28	3 subs. of process-control sys.	?	Baxter et al. (1998)
59	payroll system	≥ 10	Ducasse et al. (1999)

- clone sizes larger than 25 are rare (Baxter et al., 1998)

Open Issues

- How much do these numbers depend upon the quality of the clone detector?
- Are these systems representative?
- Do open-source systems have fewer clones?

A Historical Review on Software Clones Research



Type-1 Clone

```
PRIVATE UINT16 typ_length( atn_type *node )
{if ( node->tag == REF )
   node = node->tree.reftype;

 switch ( node->tag )
 {
  case INTEGER : return 4;
  case REAL    : return 8;
  case BOOLEAN : return 1;
  case STRING  : return 4;
  case ARRAY   :
   return typ_length(node->tree.array.type)
      * (node->tree.array.upb
         - node->tree.array.lwb+1);
  case REF     : return 4;
  default :
     log_error(ERR_FATAL, SYSTEM_ERROR,
               E_ILLEGAL_TAG, "type", 0 );
 }
 return 0;
}
```

```
PRIVATE UINT16 typ_length( atn_type *node )
{if ( node->tag == REF )
   node = node->tree.reftype;

 switch ( node->tag )
 {
  case INTEGER : return 4;
  case REAL    : return 8;
  case BOOLEAN : return 1;
  case STRING  : return 4;
  case ARRAY   :
   return typ_length(node->tree.array.type)
      * (node->tree.array.upb
         - node->tree.array.lwb+1);
  case REF     : return 4;
  default :
     log_error(ERR_FATAL, SYSTEM_ERROR,
               E_ILLEGAL_TAG, "type", 0 );
 }
 return 0;
}
```

Type-2 Clone

```
    return TRUE;
}

/* read operand #0 (always present) */

thisOp->op[ 0 ].type
  = va_arg(ap, a3_argument_type);

if (( thisOp->op[ 0 ].type == oCFLOAT ) ||
 ( thisOp->op[ 0 ].type & 16)) // indexed
{
  thisOp->op[ 0 ].val.f[0]=va_arg(ap,INT32);
  thisOp->op[ 0 ].val.f[1]=va_arg(ap,INT32);
}
else
  thisOp->op[ 0 ].val.l = va_arg(ap, INT32);

/* read operand #1 (sometimes present) */

if (( stat_type != A3_GOTO ) &&
```

```
/* read operand #2 (binary op only) */
if ((stat_type == A3_BINARY_OP) ||
    (stat_type == A3_COND))
{
  thisOp->op[ 2 ].type
    = va_arg(ap, a3_argument_type);

  if ((thisOp->op[ 2 ].type == oCFLOAT ) ||
     (thisOp->op[ 2 ].type & 16))
  {
    thisOp->op[ 2 ].val.f[0]=va_arg(ap,INT32);
    thisOp->op[ 2 ].val.f[1]=va_arg(ap,INT32);
  }
  else
    thisOp->op[ 2 ].val.l = va_arg(ap, INT32);
}
else
  thisOp->op[2].type = oNONE;
```

Type-3 Clone

```
static void r32 (int arg, int ap) {
 /* read operand #0 (always present)*/
 thisOp->op[ 0 ].type = 0;

 if ((thisOp->op[ 0 ].type == oCFLOAT)
  || (thisOp->op[ 0 ].type & 16) )// indexed
 {
   thisOp->op[ 0 ].val.f[0]=arg;
   thisOp->op[ 0 ].val.f[1]=arg;
 }
 else
   thisOp->op[ 0 ].val.l = ap;
 }
}
```

```
static void r64 (int arg, int ap,) {
 /* read operand #2 (binary op only)*/
 thisOp->op[ 2 ].type = 0;

 if ((thisOp->op[ 2 ].type == oCFLOAT)
    || (thisOp->op[ 2 ].type & 32 ))
 {
   thisOp->op[ 2 ].val.f[0]=arg;
   thisOp->op[ 2 ].val.f[1]=arg;
 }
 else
   thisOp->op[ 2 ].val.l = ap;
   thisOp-¿op[2].val.r = ap & 32;
 }
}
```

Is There a Consensus on a Definition of a Clone?

Exploratory study by Walenstein et al. (2003):

- function clones of Bellon Benchmark investigated
- four raters
- rater instructions required clones to be worthwhile for refactoring

$\longrightarrow$ little consensus

Discussion at Dagstuhl seminar on software clones (Kapser et al., 2006)

- segments of code were presented to clone researchers
- clone researchers debated whether the segments are clones

$\longrightarrow$ little consensus

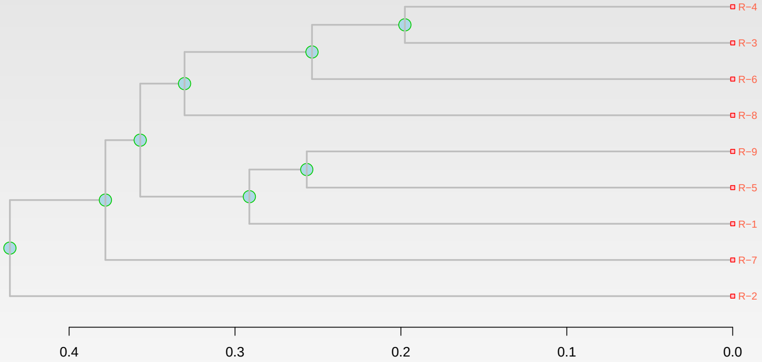
$\longrightarrow$ people tend to use task-oriented definitions

Is There a Consensus on a Definition of a Clone?

Own study (Mende et al., 2009):

- function clones of Linux driver subsystem for wireless LAN
- nine raters
- rater instructions in software product lines context: function variants that are worthwhile for refactoring

→ sufficient agreement



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 categorizations beyond type 1/2/3 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?

How Can We Detect Clones?

Granularity

- functions
- statements

Comparison of . . .

- text
- identifiers
- tokens
- syntax trees
- control/data dependencies

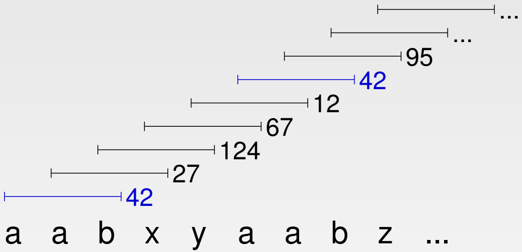
Techniques used

- textual diff
- dotplot
- data mining
- suffix tree
- tree matching
- graph matching
- latent semantic indexing
- metric vector comparison
- hashing

Textual Detection

Comparison of. . .

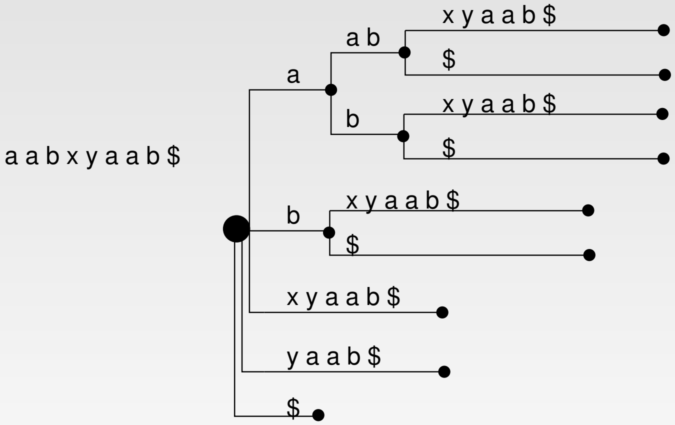
- identifiers and comments (information retrieval)
  - latent semantic indexing (Marcus and Maletic, 2001)
- text
  - string comparison using fingerprints (Johnson, 1993, 1994)
  - line-based comparison via dot plots (Ducasse et al., 1999; Rieger, 2005)



How Can We Detect Clones?

Comparison of . . .

- tokens
  - type-1/-2 clones: suffix trees for parameterized strings per line (Baker, 1995)

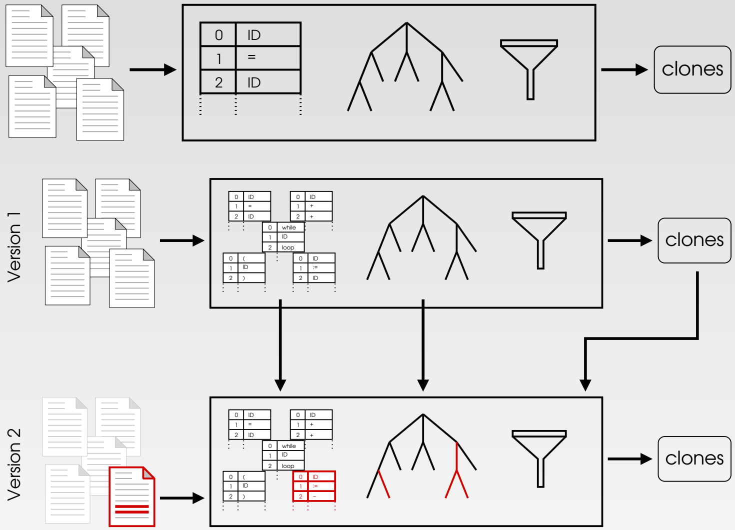


How Can We Detect Clones?

Comparison of . . .

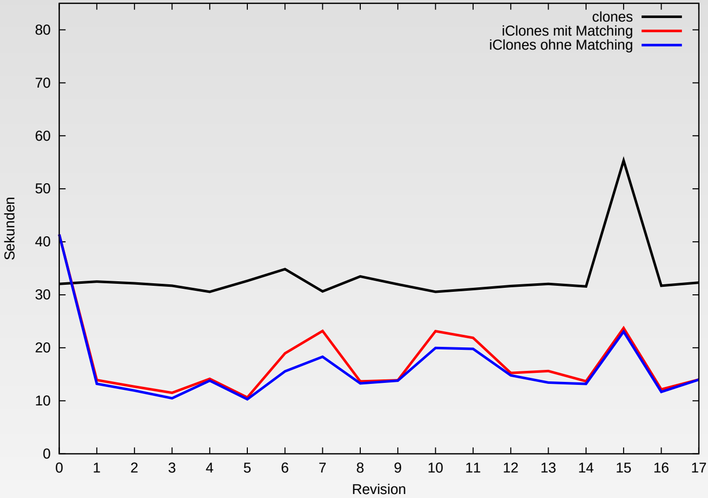
- tokens
  - type-1/-2 clones: suffix trees for parameterized strings per line (Baker, 1995)
  - type-3: concatenation of type-1/2 clones with gaps (Baker, 1995)
  - per token plus normalization of token stream (Kamiya et al., 2002)
  - lexical clones fully contained in syntactic unit:
    - lexical post-processing (Higo et al., 2002)
    - lexical pre-processing (Synytskyy et al., 2003; Cordy et al., 2004)

Incremental Token-Based Detection



– Göde (2008); Göde and Koschke (2009)

Evaluation (Göde and Koschke, 2009)



How Can We Detect Clones?

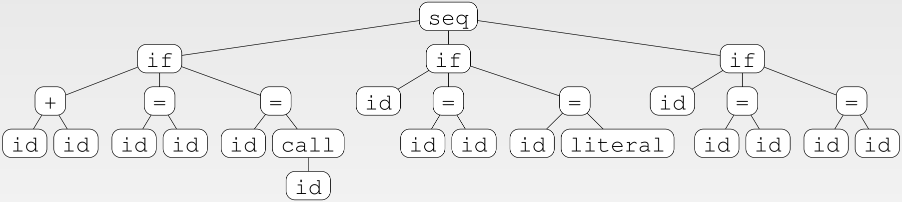
Comparison of . . .

- metrics (Mayrand et al., 1996; Kontogiannis, 1997)
- statements via 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)
  - tree matching plus dynamic programming (for file comparison) (Yang, 1991)
  - suffix trees for serialized syntax trees (Koschke et al., 2006)

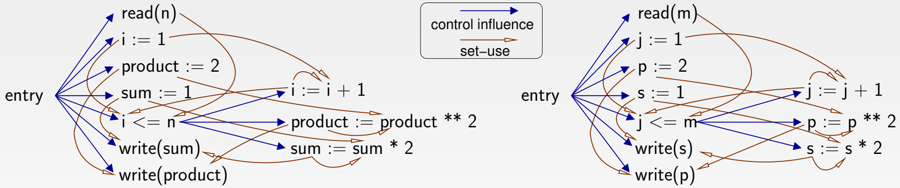


How Can We Detect Clones?

Comparison of Program Dependency Graphs (PDG) (Komondoor and Horwitz, 2001; Krinke, 2001)

```
read (n);
i = 1;
product = 1;
sum = 0;
while (i <= n) {
   product = product * i;
   sum = sum + i;
   i = i + 1;
}
write (sum);
write (product);
```

```
read (m);
s = 0;
p = 1;
for (j = 1; j <= m; j = j + 1) {
   s = s + i;
   p = p * i;
}
write (s);
write (p);
```



How do Clone Detectors Compare?

Quantitative comparison of clone detectors by Bellon and Koschke (2002; 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 Koschke et al. (2006)

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.
$\longrightarrow$ Bellon Benchmark is open source:
`http://www.bauhaus-stuttgart.de/clones/`
TBD: `http://cloneval.sourceforge.net`

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
- 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)
- development process, e.g., XP yields less clones? (Nickell and Smith, 2003)
- organizational issues, e.g., distributed development organizations

$\longrightarrow$ fight the reasons, not just the symptoms

Beliefs About Effects of Cloning

Duplication is undesirable because of its well-known association with bugs.

— Baker (1993)

In the long run the software grows in size and complexity and requires more resources to maintain and enhance.

— Mayrand et al. (1996)

Detection and removal of such clones promises decreased software maintenance costs of possibly the same magnitude.

— Baxter et al. (1998)

Beliefs About Effects of Cloning

During our case studies of large software systems, we found that code cloning can often be used in a positive way.

— "Cloning considered harmful" considered harmful; Kapser and Godfrey (2006)

In particular, refactoring may not always improve software with respect to clones for two reasons. First, many code clones exist in the system for only a short time; [...] Second, many clones, especially long-lived clones that have changed consistently with other elements in the same group, are not easily refactorable due to programming language limitations.

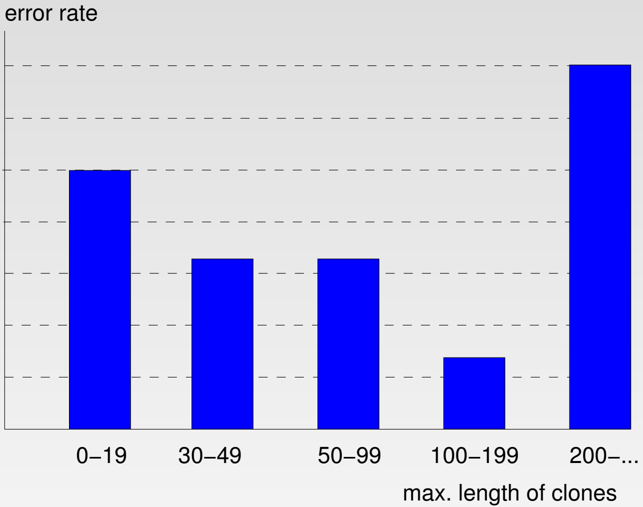
— Kim et al. (2005)

Beliefs in Beliefs About Effects of Cloning

Several recent studies contradict the common wisdom that cloning constitutes a risky practice as found by Kim et al. (2005). As shown in a paper by Kapser and Godfrey (2006), source code clones are not necessarily to be considered harmful [...]

— Aversano et al. (2007)

What Are the Effects of Cloning?



– Monden et al. (2002)

What Are the Effects of Cloning?

Hypothesis by Chou et al. (2001):

> *If a function, file, or directory has one error, it is more likely that is has others.*

Additional observation in their study of Linux and OpenBSD:

- can be observed most often when programmer ignorance of interface or system rules combines with copy-and-paste

$\longrightarrow$ programmers believe that “working” code is correct code

$\longrightarrow$ copied code may be incorrect or placed into a context it was not intended for

What Are the Effects of Cloning?

Faults due to type-2 clones with inconsistent renaming

— Li et al. (2006) for *Linux kernel*, *FreeBSD*, *Apache*, *PostgreSQL*.

Faults due to inconsistently changed type-3 clones (Juergens et al., 2009)

What Are the Effects of Cloning?

Effects on Changeability:

- impact $=$ percentage of co-changed methods
- likelihood $= \frac{\text{frequency of method changes with clones}}{\text{frequency of method changes}}$
- work $=$ impact $\times$ likelihood

— Lozano and Wermelinger (2008)

$\rightarrow$ no difference between cloned code and not cloned clone

Open Issues

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

How to Get Rid of Clones?

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)
- generative programming (Jarzabek and Shubiao, 2003; Jarzabek and Li, 2006)

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

How to Get Rid of Clones?

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

Tool support for removing type-3 clones?

How do Clones Evolve?

- Cloning is common and steady practice in Linux kernel (Godfrey and Tu, 2000, 2001; Antoniol et al., 2001, 2002)
- Many code clones exist for only a short time (Kim et al., 2005)
- Most type-1 clones live for a long time (Göde, 2009)
- 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 (Kim et al., 2005)
- Clones are changed consistently (Aversano et al., 2007)
- Only half of the clones are changed consistently (Krinke, 2007)
- Clones are more stable than non-cloned code (Krinke, 2008)

How do Clones Evolve?

Open Issues

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

Summary of Own Research Results

- Research surveys (Koschke, 2007, 2008b,a; Roy et al., 2009; Harder and Koschke, 2008)
- Clone detection for syntax trees in linear time (Koschke et al., 2006; Falke et al., 2008)
- Incremental clone detection (Göde and Koschke, 2009)
- Tracing of clones across versions (Göde, 2009)
- Learning algorithms for clone characteristics (Tiarks et al., 2009a,b)
- Empirical classifications of type-3 clones (Tiarks et al., 2009a,b)
- Empirical investigation of clone removal (Göde, 2010)

Final Quotes

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`
  Bauhaus research project offering various clone detectors
- `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

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 CS 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):755–765, 2002.

L. Aversano, L. Cerulo, and M. D. Penta. How clones are maintained: An empirical study. In European Conference on Software Maintenance and Reengineering. IEEE CS Press, 2007.

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 CS Press. URL `http://citeseer.nj.nec.com/baker95finding.html`.

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 CS 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 CS 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. Vergleich von Techniken zur Erkennung duplizierten Quellcodes. Diploma thesis, no. 1998, University of Stuttgart (Germany), Institute for Software Technology, September 2002.

Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. IEEE Transactions on Software Engineering, pages 577–591, September 2007.

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`.

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 CS Press, 2003.

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, Rainer Koschke, and Pierre Frenzel. Empirical evaluation of clone detection using syntax suffix trees. Empirical Software Engineering, 2008. accepted for publication.

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

Nils Göde. Evolution of type-1 clones. In Workshop Source Code Analysis and Manipulation, pages 77–86. IEEE Computer Society, 2009.

M. Godfrey and Q. Tu. Evolution in open source software: A case study. In International Conference on Software Maintenance, pages 131–142. IEEE CS Press, 2000.

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

Nils Göde. Incremental clone detection. Diplomarbeit, University of Bremen, FB3, AG Softwaretechnik, September 2008. URL `http://www.informatik.uni-bremen.de/st/diplomdetails.php?da_id=11`.

Nils Göde. Clone removal: Fact or fiction? In International Workshop on Software Clones. ACM Press, 2010. submitted for publication.

Nils Göde and Rainer Koschke. Incremental clone detection. In European Conference on Software Maintenance and Reengineering. IEEE CS Press, 2009.

Jan Harder and Rainer Koschke. Empirische grundlagen für das klonmanagement. In Rainer Gimnich, Uwe Kaiser, Jochen Quante, and Andreas Winter, editors, Workshop Software Reengineering, pages 127–133. GI Lecture Notes for Informatics, 2008. ISBN 978-3-88579-220-8.

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.

S. Jarzabek and S. Li. Unifying clones with a generative programming technique: a case study. Journal on Software Maintenance and Evolution, 18(4):267–292, 2006.

S. Jarzabek and L. Shubiao. Eliminating redundancies with a "composition with adaptation" meta programming technique. In European Software Engineering Conference, pages 237–246. IEEE CS Press, 2003.

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. Substring matching for clone detection and change tracking. In International Conference on Software Maintenance, pages 120–126. IEEE CS Press, 1994.

E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In International Conference on Software Engineering. ACM Press, 2009.

Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code. IEEE 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, pages 19–28, 2006.

Cory Kapser, Paul Anderson, Michael Godfrey, Rainer Koschke, Matthias Rieger, Filip van Rysselberghe, and Peter Weißgerber. Subjectivity in clone judgment: Can we ever agree? In R. Koschke, E. Merlo, and A. Walenstein, editors, Dagstuhl Seminar "Duplication, Redundancy, and Similarity in Software" Proceedings 06301. Schloß Dagstuhl, 2006. URL `http://drops.dagstuhl.de/opus/volltexte/2007/970/pdf/06301.SWM.Paper.970.pdf`.

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, pages 187–196, 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 2001.

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

K. Kontogiannis. Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics. In Working Conference on Reverse Engineering, pages 44–53, 1997.

Rainer Koschke. Identifying and Removing Software Clones, pages 15–39. Springer Verlag, 2008a. Editors: Serge Demeyer und Tom Mens.

Rainer Koschke. Survey of research on software clones. In Rainer Koschke, Ettore Merlo, and Andrew Walenstein, editors, Duplication, Redundancy, and Similarity in Software, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Dagstuhl. URL http://drops.dagstuhl.de/opus/volltexte/2007/970[dateofcitation:2007-01-01].

Rainer Koschke. Frontiers on software clone management. In International Conference on Software Maintenance. IEEE CS Press, 2008b.

Rainer Koschke, Raimar Falke, and Pierre Frenzel. Clone detection using abstract syntax suffix trees. In Working Conference on Reverse Engineering, pages 253–262. IEEE CS Press, 2006.

Jens Krinke. Is cloned code more stable than non-cloned code? In Workshop Source Code Analysis and Manipulation, pages 57–66. IEEE CS Press, 2008.

Jens Krinke. A study of consistent and inconsistent changes to code clones. In Working Conference on Reverse Engineering. IEEE CS Press, 2007.

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

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 Transactions on Software Engineering, 32(3):176–192, March 2006.

Angela Lozano and Michel Wermelinger. Assessing the effect of clones on changeability. In International Conference on Software Maintenance. IEEE Press, 2008.

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.

Thilo Mende, Rainer Koschke, and Felix Beckwermert. An evaluation of code similarity identification for the grow-and-prune model. Journal on Software Maintenance and Evolution, 2009. Accepted for publication.

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.

Eric Nickell and Ian Smith. Extreme programming and software clones. In Working Conference on Reverse Engineering. IEEE CS Press, 2003. URL `http://www.cacs.louisiana.edu/labs/SRL/iwdsc2003/papers-pre/smith.pdf`.

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

Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming, 2009. doi: doi:10.1016/j.scico.2009.02.007. accepted for publication.

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.

Rebecca Tiarks, Rainer Koschke, and Raimar Falke. An assessment of type-3 clones as detected by state-of-the-art tools. In Workshop Source Code Analysis and Manipulation. IEEE CS Press, 2009a.

Rebecca Tiarks, Rainer Koschke, and Raimar Falke. An extended assessment of type-3 clones as detected by state-of-the-art tools. Software Quality Journal, 2009b. submitted for publication.

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, pages 285–294. IEEE CS Press, 2003.

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