

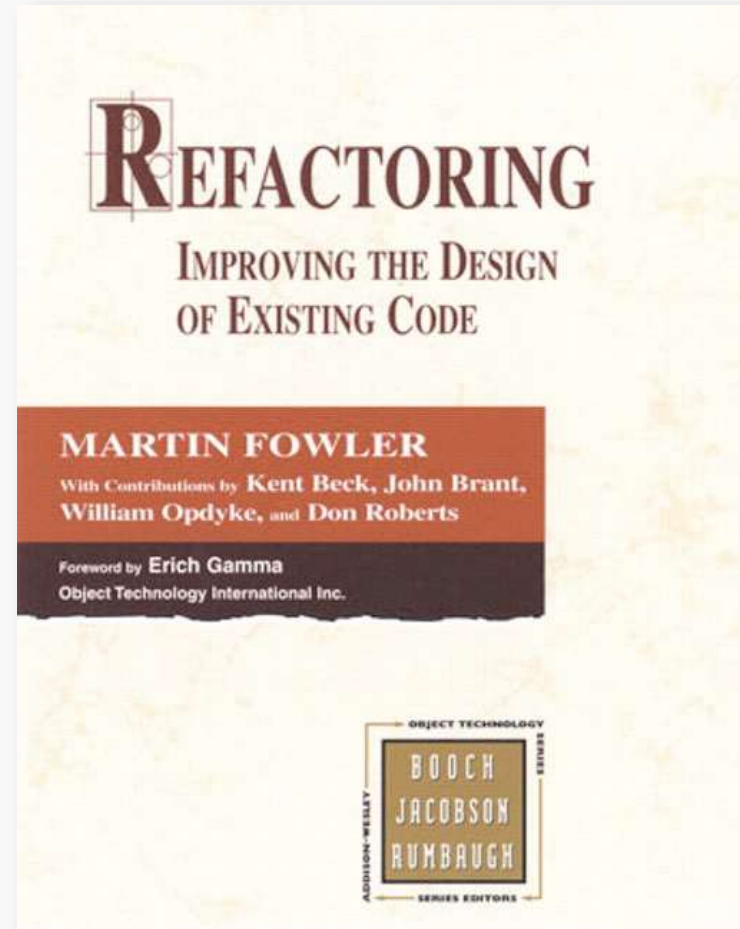
COMP0104 Software Development Practice: Clone Detection

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

Duplicated Code

Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them.



Duplicated Code

- increases code size
- complicates software maintenance
 - X errors may have been duplicated, too
 - X changes have to be applied to all clones
 - X...

“Cloning considered harmful” considered harmful: patterns of cloning in software

Cory J. Kapser · Michael W. Godfrey

We found that as many as 71% of the clones could be considered to have a positive impact on the maintainability of the software system.

What is a clone?

- Software clones are segments of code that are similar according to some definition of similarity.
- There can be different definitions of similarity based on...
 - text
 - syntax
 - semantics
 - patterns
 - ...

What types of clones exist?

- Type 1: An exactly identical source code clone, i.e. no changes at all.
- Type 2: An exactly identical source code clone, but with indentation, comments, or identifier (name) changes.
- Type 3: A functionally identical clone, but with small changes made to the code to tailor it to some new function.
- Type 4: A functionally identical clone.

Type 1

```

186 } else if (f instanceof SVGGroupFigure) {
187     writeGElement(parent, (SVGGroupFigure) f);
188 } else if (f instanceof SVGImageFigure) {
189     writeImageElement(parent, (SVGImageFigure) f);
190 } else if (f instanceof SVGPathFigure) {
191     SVGPathFigure path = (SVGPathFigure) f;
192     if (path.getChildCount() == 1) {
193         BezierFigure bezier = (BezierFigure)
path.getChild(0);
194         boolean isLinear = true;
195         for (int i = 0, n = bezier.getNodeCount(); i < n; i++)
{
196             if (bezier.getNode(i).getMask() != 0) {
197                 isLinear = false;
198                 break;
199             }
200         }
201         if (isLinear) {
202             if (bezier.isClosed()) {
203                 writePolygonElement(parent, path);
204             } else {
205                 if (bezier.getNodeCount() == 2) {
206                     writeLineElement(parent, path);
207                 } else {
208                     writePolylineElement(parent, path);
209                 }
210             }
211         } else {
212             writePathElement(parent, path);
213         }

```

ImageMapOutputFormat.java

```

217 } else if (f instanceof SVGRectFigure) {
218     writeRectElement(parent, (SVGRectFigure) f);
219 } else if (f instanceof SVGTextFigure) {
220     writeTextElement(parent, (SVGTextFigure) f);
221 } else if (f instanceof SVGTextAreaFigure) {
222     writeTextAreaElement(parent, (SVGTextAreaFigure)

```

```

132 } else if (f instanceof SVGGroupFigure) {
133     writeGElement(parent, (SVGGroupFigure) f);
134 } else if (f instanceof SVGImageFigure) {
135     writeImageElement(parent, (SVGImageFigure) f);
136 } else if (f instanceof SVGPathFigure) {
137     SVGPathFigure path = (SVGPathFigure) f;
138     if (path.getChildCount() == 1) {
139         BezierFigure bezier = (BezierFigure)
path.getChild(0);
140         boolean isLinear = true;
141         for (int i = 0, n = bezier.getNodeCount(); i < n; i++)
{
142             if (bezier.getNode(i).getMask() != 0) {
143                 isLinear = false;
144                 break;
145             }
146         }
147         if (isLinear) {
148             if (bezier.isClosed()) {
149                 writePolygonElement(parent, path);
150             } else {
151                 if (bezier.getNodeCount() == 2) {
152                     writeLineElement(parent, path);
153                 } else {
154                     writePolylineElement(parent, path);
155                 }
156             }
157         } else {
158             writePathElement(parent, path);
159         }

```

SVGOutputFormat.java

```

163 } else if (f instanceof SVGRectFigure) {
164     writeRectElement(parent, (SVGRectFigure) f);
165 } else if (f instanceof SVGTextFigure) {
166     writeTextElement(parent, (SVGTextFigure) f);
167 } else if (f instanceof SVGTextAreaFigure) {
168     writeTextAreaElement(parent, (SVGTextAreaFigure)

```

Type 2

```

186 } else if (f instanceof SVGGroupFigure) {
187     writeGElement(parent, (SVGGroupFigure) f);
188 } else if (f instanceof SVGImageFigure) {
189     writeImageElement(parent, (SVGImageFigure) f);
190 } else if (f instanceof SVGPathFigure) {
191     SVGPathFigure path = (SVGPathFigure) f;
192     if (path.getChildCount() == 1) {
193         BezierFigure bezier = (BezierFigure)
path.getChild(0);
194         boolean isLinear = true;
195         for (int i = 0; n = bezier.getNodeCount(); i < n; i++)
{
196             if (bezier.getNode(i).getMask() != 0) {
197                 isLinear = false;
198                 break;
199             }
200         }
201         if (isLinear) {
202             if (bezier.isClosed()) {
203                 writePolygonElement(parent, path);
204             } else {
205                 if (bezier.getNodeCount() == 2) {
206                     writeLineElement(parent, path);
207                 } else {
208                     writePolylineElement(parent, path);
209                 }
210             }
211         } else {
212             writePathElement(parent, path);
213         }

```

ImageMapOutputFormat.java

```

217 } else if (f instanceof SVGRectFigure) {
218     writeRectElement(parent, (SVGRectFigure) f);
219 } else if (f instanceof SVGTextFigure) {
220     writeTextElement(parent, (SVGTextFigure) f);
221 } else if (f instanceof SVGTextAreaFigure) {
222     writeTextAreaElement(parent, (SVGTextAreaFigure)

```

```

132 } else if (f instanceof SVGGroupFigure) {
133     writeGElement(parent, (SVGGroupFigure) f);
134 } else if (f instanceof SVGImageFigure) {
135     writeImageElement(parent, (SVGImageFigure) f);
136 } else if (f instanceof SVGPathFigure) {
137     SVGPathFigure path = (SVGPathFigure) f;
138     if (path.getChildCount() == 1) {
139         BezierFigure figure = (BezierFigure)
path.getChild(0);
140         boolean isLinear = true;
141         for (int i = 0; n = figure.getNodeCount(); i < n; i++) {
142             if (figure.getNode(i).getMask() != 0) {
143                 isLinear = false;
144                 break;
145             }
146         }
147         if (isLinear) {
148             if (figure.isClosed()) {
149                 writePolygonElement(parent, path);
150             } else {
151                 if (figure.getNodeCount() == 2) {
152                     writeLineElement(parent, path);
153                 } else {
154                     writePolylineElement(parent, path);
155                 }
156             }
157         } else {
158             writePathElement(parent, path);
159         }
160     } else {

```

SVGOutputFormat.java

```

164     writeRectElement(parent, (SVGRectFigure) f);
165 } else if (f instanceof SVGTextFigure) {
166     writeTextElement(parent, (SVGTextFigure) f);
167 } else if (f instanceof SVGTextAreaFigure) {
168     writeTextAreaElement(parent, (SVGTextAreaFigure)
169 }
170 } else {

```


Type 3

```

186 } else if (f instanceof SVGGroupFigure) {
187     writeGElement(parent, (SVGGroupFigure) f);
188 } else if (f instanceof SVGImageFigure) {
189     writeImageElement(parent, (SVGImageFigure) f);
190 } else if (f instanceof SVGPathFigure) {
191     SVGPathFigure path = (SVGPathFigure) f;
192     if (path.getChildCount() == 1) {
193         BezierFigure bezier = (BezierFigure)
path.getChild(0);
194         boolean isLinear = true;
195         for (int i = 0, n = bezier.getNodeCount(); i < n; i++)
{
196             if (bezier.getNode(i).getMask() != 0) {
197                 isLinear = false;
198                 break;
199             }
200         }
201         if (isLinear) {
202             if (bezier.isClosed()) {
203                 writePolygonElement(parent, path);
204             } else {
205                 if (bezier.getNodeCount() == 2) {
206                     writeLineElement(parent, path);
207                 } else {
208                     writePolylineElement(parent, path);
209                 }
210             }
211         } else {
212             writeArcElement(parent, path);
213         }

```

ImageMapOutputFormat.java

```

217 } else if (f instanceof SVGRectFigure) {
218     writeRectElement(parent, (SVGRectFigure) f);
219 } else if (f instanceof SVGTextFigure) {
220     writeTextElement(parent, (SVGTextFigure) f);
221 } else if (f instanceof SVGTextAreaFigure) {
222     writeTextAreaElement(parent, (SVGTextAreaFigure)

```

```

132 } else if (f instanceof SVGGroupFigure) {
133     writeGElement(parent, (SVGGroupFigure) f);
134 } else if (f instanceof SVGImageFigure) {
135     writeImageElement(parent, (SVGImageFigure) f);
136 } else if (f instanceof SVGPathFigure) {
137     SVGPathFigure path = (SVGPathFigure) f;
138     if (path.getChildCount() == 1) {
139         BezierFigure bezier = (BezierFigure)
path.getChild(0);
140         boolean isLinear = true;
141         for (int i = 0, n = bezier.getNodeCount(); i < n; i++)
{
142             if (bezier.getNode(i).getMask() != 0) {
143                 isLinear = false;
144                 break;
145             }
146         }
147         assert isLinear;
148         if (bezier.isClosed()) {
149             writePolygonElement(parent, path);
150         } else {
151             if (bezier.getNodeCount() == 2) {
152                 writeLineElement(parent, path);
153             } else {
154                 writePolylineElement(parent, path);
155             }
156         }
157     } else {

```

SVGOutputFormat.java

```

162 } else if (f instanceof SVGTextFigure) {
163     writeTextElement(parent, (SVGTextFigure) f);
164 } else if (f instanceof SVGTextAreaFigure) {
165     writeTextAreaElement(parent, (SVGTextAreaFigure)
f);
166 } else {
167     System.out.println("Unable to write: " + f);

```

Does duplicated code exist?

- typically 5-30% of code is similar
- in extreme cases, even up to 50%

Why is code cloned?

- Productivity is measured in LOCs per day
- Programmers are not well educated
- Process does not allow to change code from somebody else
- Time pressure

There are good reasons for cloned code

- Limitations of programming languages may result in unavoidable duplicates in a code.
- Programmers often delay code restructuring until they have copied and pasted several times
- Similar code often reflects important design decisions, such as crosscutting concerns.
- Copied text is often reused as a template and is customised in the pasted context.

Patterns of cloning

- Forking
- Templating
- Customization

What are the consequences of cloning?

- Do clones increase the maintenance effort?
- Hypotheses:
 - Cloned code increases code size.
 - A fix to a clone must be applied to all similar fragments.
 - Bugs are duplicated together with their clones
 - Will all bug instance be fixed?
- Cloning external code is a legal risk!

Do Clones lead to Bugs?

Rahman et al., “Clones: What is that Smell”, MSR 2010

- most bugs have very little to do with clones
- cloned code contains less buggy code
- larger clone groups don't have more bugs than smaller clone groups
- making more copies of code doesn't introduce more defects

Code Reuse and legal implications

- German et al. studied cloning between BSD and Linux kernels and license implications.
- Migration from BSDs to Linux is frequent, the opposite was significantly less frequent
- Code fragments migrated from third-parties into Linux and BSDs.

Open Source Code Reuse is a Risk

- GPL is viral: If you use GPL code, your code must be GPL code.
- Outsourcing:
Where is the code **really** coming from?
- Unmanaged use of externally sourced code can compromise intellectual property rights and create unknown royalty obligations



Example Cases

- Cisco acquired Linksys in 2003.
- Linksys used GPL software, but Cisco did not make that software freely available to end users, as required by the GPL.
- In 2008, the Free Software Foundation (FSF) has filed a copyright infringement lawsuit against Cisco.

Example Cases

- gpl-violations.org has helped uncover and negotiate more than 100 GPL violations and has obtained numerous out-of-court settlement agreements.
- Oracle vs. Google:
"at least eight Android source code files are actually decompiled Oracle object code"

Source Code Plagiarism

Why do we care?

- “If the open source code is published under a BSD or Apache type license then the plagiarism is perfectly legal”
- Is it?

Source Code Plagiarism

Why do we care?

- “If the open source code is published under a BSD or Apache type license then the plagiarism is perfectly legal”
- Plagiarism is never legal
- Both licenses require that all copyright (and patent, trademark, and attribution) notices are retained.

How can clones be removed?

Manual

- refactor clones into parameterised functions, class hierarchies, generics, templates etc.
- introduce design patterns
- use aspect oriented programming

Automatic

- automatic refactoring
- replacing with macros

But...

- ...never change a running system(?)
- ...don't fix something that is not broken(?)
- Removal of clones is **preemptive** maintenance
- Removal of clones is hard if variation exist

How can clones be detected?

- Comparison based on text:
lines are compared to other lines
- Comparison based on tokens:
token sequences are compared to sequences
- Comparison based on syntax:
syntax (sub) trees are compared to each other
- Comparison based on structure:
(sub) graphs are compared to each other

Typical steps in the clone detection process

- Pre-processing
- Transformation
- Match detection
- Formatting
- Post-processing
- Aggregation

Pre-processing

- Remove uninteresting parts
 - generated code
 - testing code
- Determine source units (largest fragments)
 - files, classes, methods/functions, blocks...
- Determine comparison units
 - what is actually compared?
 - methods/functions, blocks, lines, tokens...

Transformation

- Extraction
 - Tokenisation
 - Parsing
 - Control and data flow analysis
- Normalisation
 - Removal of white space and comments
 - Replacing identifiers
 - Pretty printing

Match detection

- The comparison units are actually compared to find matches
- Adjacent units are joined to form larger units
 - find the longest sequence of similar units
 - up to the determined source units
- Depends on the technique used
 - textual
 - token based
 - syntax based

Last steps

- Formatting
 - maps the detected clones back to the source code
- Post-processing / Filtering
 - Ranking of the detected clones
 - Visualisation
- Aggregation
 - Sometimes only pairs are detected and are aggregated into classes

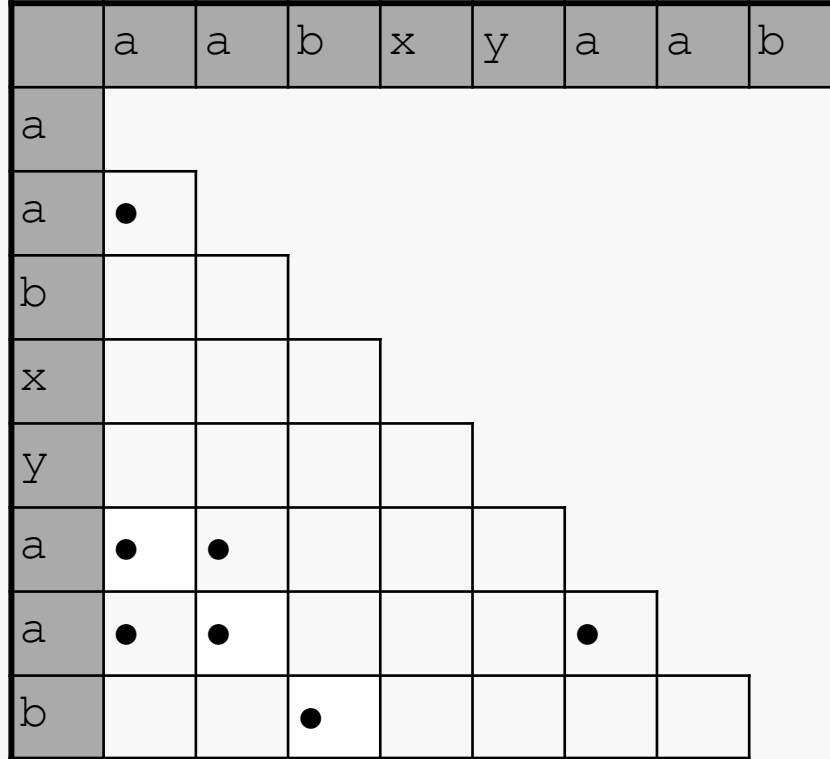
String-based clone detection

- Model: programs are sequences of strings
- Mostly independent of programming language
- Typically includes some preprocessing
 - removal of white space
 - removal of comments
- Can be implemented with different techniques

Visual detection through dot plots

	a	a	b	x	y	a	a	b
a	–	●				●	●	
a	●	–				●	●	
b			–					●
x				–				
y					–			
a	●	●				–	●	
a	●	●				●	–	
b			●					–

Visual detection through dot plots



Naive string-based approach

- All (normalised) source files are joined into one sequence or stream (one string)
- Find all largest substrings that appear more than once
- Report the position and the length of the largest substrings and map them back to the source code
- Consider only substrings larger than a threshold

Naive implementation

String s , length n ($s[1..n]$)

Minimal size t (threshold)

For all $i=1..n$

– For all $j=i+1..n$

- If the length m of the longest common prefix of $s[i..]$ and $s[j..]$ is larger than t ,
- then return $(i:i+m-1, j:j+m-1)$ as a match

Example

- String is `aabxyaab`, threshold is 3
- Will find (1:3,6:8) as a match (with length 3)
- String is the same, threshold is 2
- What happens now?

1	2	3	4	5	6	7	8
a	a	b	x	y	a	a	b

Example

1	2	3	4	5	6	7	8
a	a	b	x	y	a	a	b

- String is `aabxyaab`, threshold is 3
- Will find (1:3,6:8) as a match (with length 3)
- String is the same, threshold is 2
- Will now find (1:2,6:7) with length 3
and (2:3,7:8) with length 2
- Overlapping matches should be removed,
i must be advanced after a find

Optimisations for the naive implementation based on substring search

- The naive implementation is too slow
- The threshold can be used to as a *window*
- For all $i=1..n-(t-1)$
 - Find all substrings $s[i:i+t-1]$ in s at j
 - For all j
 - find the longest common prefix of $s[i:]$ and $s[j:]$, m is the length of the prefix.
 - and return $(i:i+m-1, j:j+m-1)$ as a match and increase i by the length
- substring search can be made efficient (e.g. Rabin-Karp)

Hash-table based implementation

- Hash-tables can be used to find the substrings
- For all $i=1..n-(t-1)$
 - $h = \text{hash}(s[i:i+t-1])$
 - Store i in the hash-table at h
- For all pairs (i,j) with hash collisions
 - If the length m of the longest common prefix of $s[i:]$ and $s[j:]$ is larger than t ,
 - then return $(i:i+m-1, j:j+m-1)$ as a match
- Prefix comparison can be improved by hashing

Alternative approaches

Suffix arrays are a sorted list of all suffixes


aabxyaab has the following suffixes:

- 1 aabxyaab
- 2 abxyaab
- 3 bxyaab
- 4 xyaab
- 5 yaab
- 6 aab
- 7 ab
- 8 b

Alternative approaches

Suffix arrays are a sorted list of all suffixes

aabxyaab has the following sorted suffixes:



6 aab
1 aabxyaab
7 ab
2 abxyaab
8 b
3 bxyaab
4 xyaab
5 yaab

Exercise:

What is the suffix array for “banana”?

1 banana

2 anana

3 nana

4 ana

5 na

6 a

6 a

4 ana

2 anana

1 banana

5 na

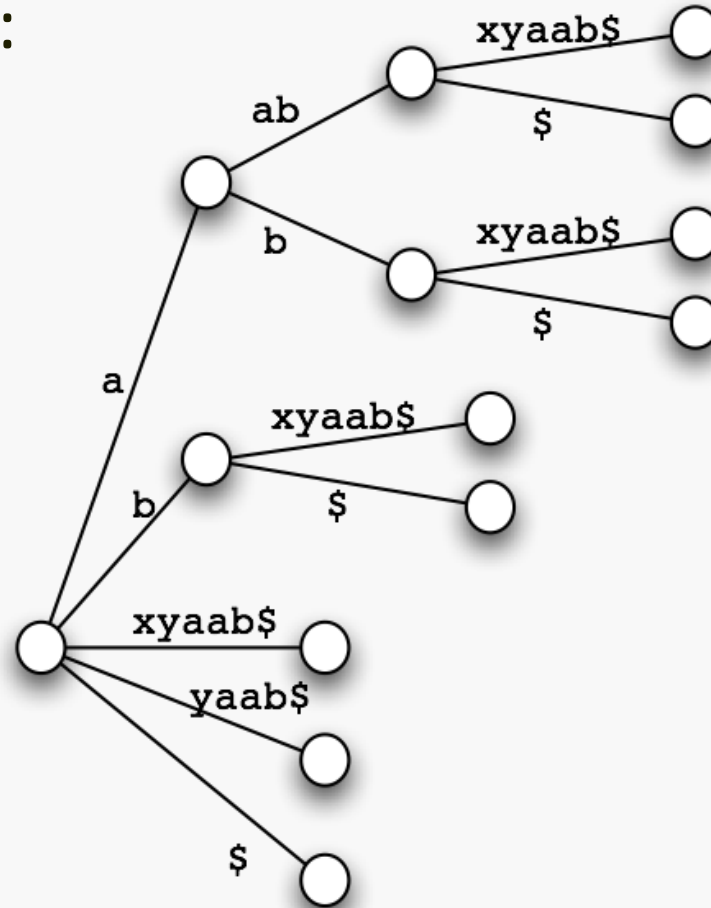
3 nana

Alternative approaches

Suffix arrays are sorted lists of all suffixes

- Only entries following on each other can have common prefixes
- Efficient implementations for suffix arrays are known and available
- The search for clones based on suffix arrays is faster than substring search

Alternative approach: Suffix trees



Suffix trees are often used for clone detection

- Efficient implementations exist
- No search for prefixes necessary, all needed information is available

What are the elements of the strings?

- Character based detection is too expensive
- Source files are transformed into tokens
 - Still expensive
 - Problem: Match token positions to source code
- Fingerprints (or hashes) are generated for every source code line (see diff)
 - Minor problem: match positions to lines in files
 - Is sensitive to source code layout

Type 2 Clones, Variant 1

Example:

`axaxbayaz` - `x`, `y`, `z` are identifiers, threshold 3

- Variant 1: Ignore name of identifiers,
`axaxbayaz` becomes `aiaibai`

- Results:
(1:4,6:9) – “`axax`”, “`ayaz`”

- Subsumed: (1:3,6:8), (2:4,7:9)

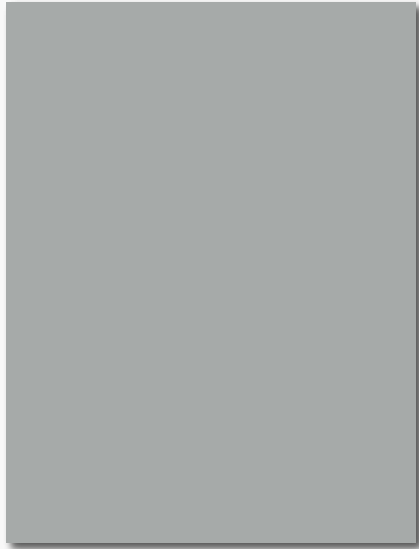
Type 2 Clones, Variant 2

Example:

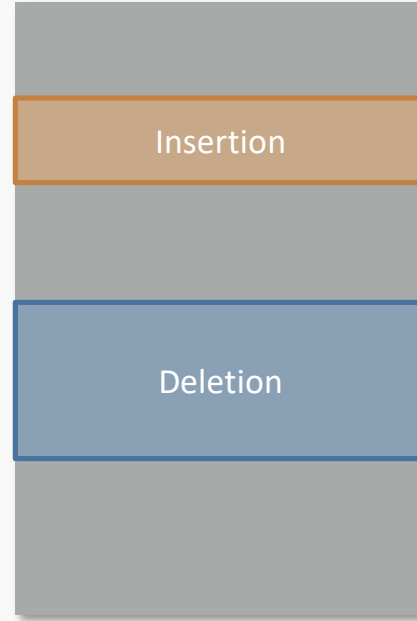
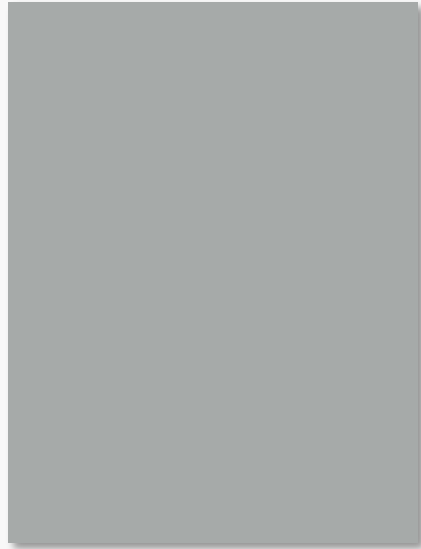
axaxbayaz - x, y, z are identifiers, threshold 3

- Variant 2: parameterise identifiers,
parameterised substring search maps x, y, z
- Results:
(1:3,6:8) – “axa”, “aya”
but not (1:4,6:9) “axax”, “ayaz”

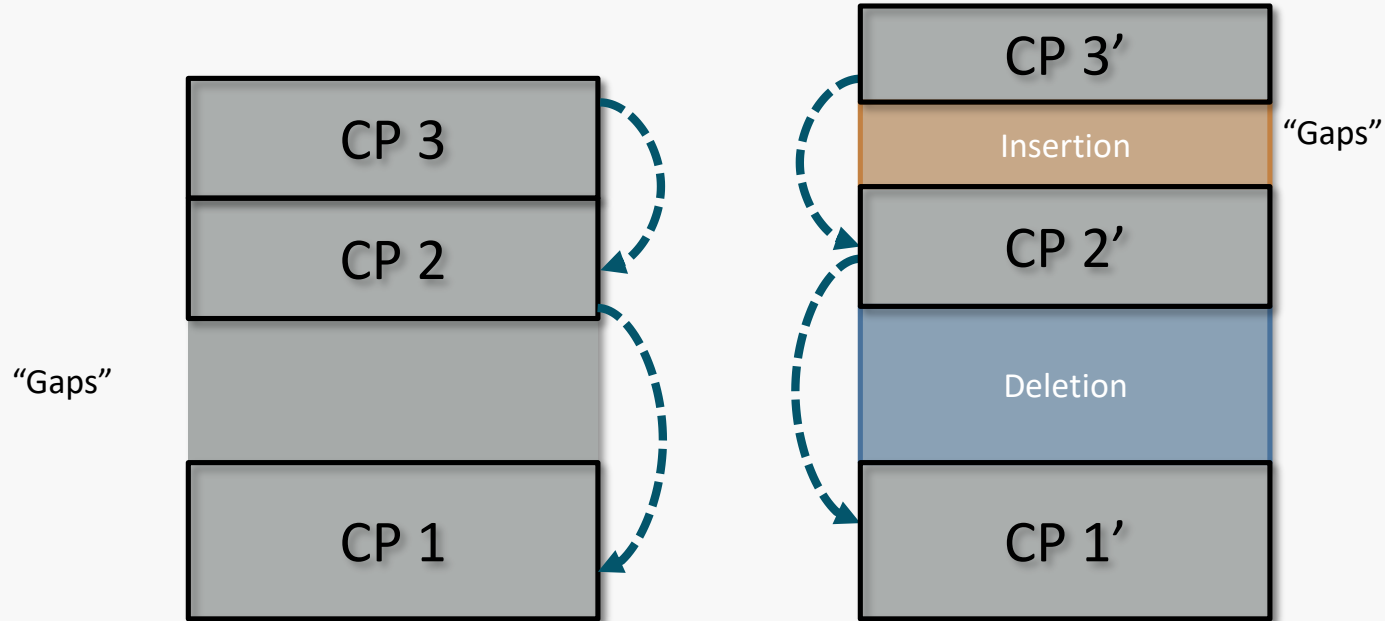
Type 3 Clones



Type 3 Clones



Type 3 Clones



Clone Detectors



JCCD

CCFinder

KClone

ACD

Dup

SDD

SimScan

CloneDr

Scorpio

ConQAT

CPC

PMD

JCD

Bauhaus

NICAD

Duplix

Duplo

Simian

Clone Tracker

CLICS

CPD

Shinobi

Clone Board

Clone Digger

Clone detectors

- There exist a lot of clone detectors
 - as standalone tools
 - as eclipse plugins
 - in visual studio
- Typical tools
 - CCFinder
 - Simian, CPD (PMD)
 - NICAD, SourcererCC
 - Siamese

Clones Elsewhere

- Clones in Source Code are well studied
- What about other areas?
 - Requirement Documents
 - Visual Dataflow Languages

Clones in Requirements

- Juergens et al., “Can Clone Detection Support Quality Assessments of Requirements Specifications?”, ICSE 2010
- Case Study on 28 specifications, 8667 pages
- Significant amount of cloning in all kinds of information
- Significant blow up of specifications
- Inconsistencies may lead to errors

Clones in Graphs

- Programs can be represented as graphs
- Visual Languages are purely graph-based
- Detection of isomorphic subgraphs is a NP-complete problem
- Typical approach:
Find a seed pair and expand up to a limit
- Multiple approaches exist.

Related Areas

- Differencing
- Plagiarism Detection
- Compression
- DNA Sequence Analysis
- Malware Detection

Citation from an typical clone related paper

“the existence of clones is known to worsen productivity in software maintenance”

Is it really?

Impact of Clones

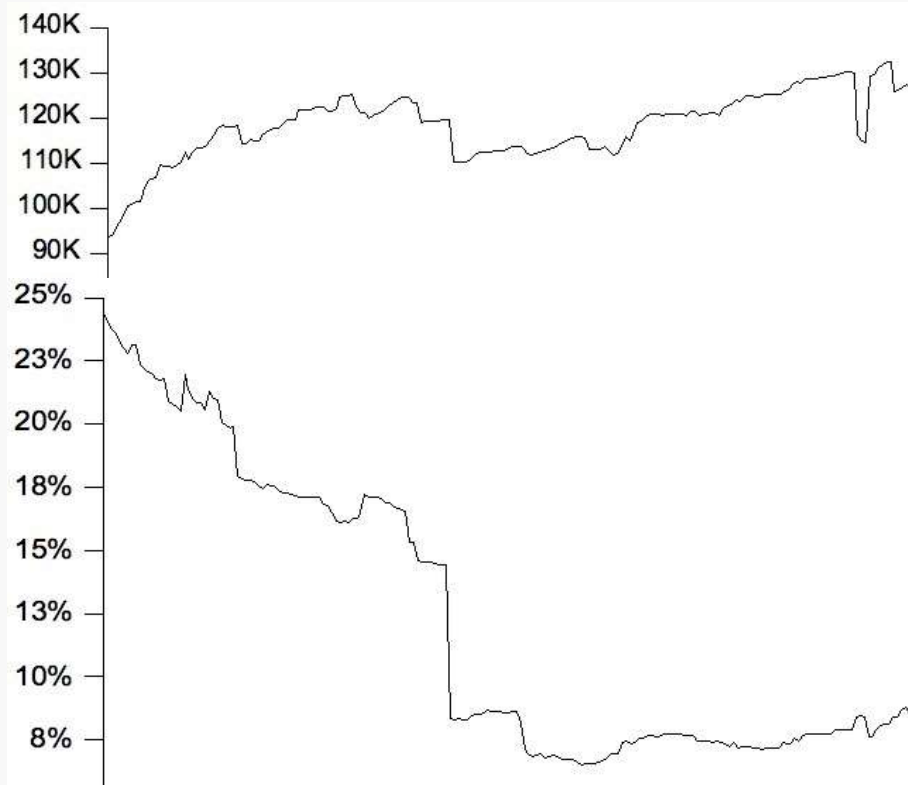
- In the past years, a few studies have been done to identify the impact of clones.
- Some of the studies agree, some disagree.
- How can such studies be done?
- What are the current results?

RQ #1:

Is Clone Code more stable?

- If cloned code is changed often, it requires more attention and is more expensive
- If cloned code is more stable, its maintenance costs will be lower

How to measure stability?



Cloned Code is (not) changed more often

**Cloned code may be more stable
than Non-Cloned Code**

- Lozano et al 2007 and 2010...
- Krinke 2008: Is cloned code more stable than non-cloned code?
- Hotta et al 2010: Is duplicate code more frequently modified than non-duplicate code in software evolution?
- Göde and Harder 2011: Clone Stability
- Harder and Göde 2012: Cloned Code: Stable Code

Typical Approach

- Checkout all revisions from a repository
- Compute the amount and frequencies of changes
- Map changes to cloned and non-cloned code
- Compute Metrics
- Compare Metrics for cloned and non-cloned code

Empirical Study

- 5 open source systems
- 200 weeks of evolution:
200 snapshots
- Clones:
200 sets (using simian)
- Changes (addition, deletions, changes):
200 diffs to the next week
- Changes are mapped to clones

ArgoUML	118.316	12%
jdt.core	192.624	15%
Emacs	227.919	10%
FileZilla	90.302	16%
SQuirreL	69.981	8%

RQ #1:

Is Cloned Code more stable?

- The average percentage of additions, deletions, or other changes to cloned code is lower than the average percentage for non-cloned code
- More often a higher percentage of non-cloned code is added, deleted, or changed in comparison to cloned code
- **Cloned code is more stable than Non-Cloned Code**

A different approach

- Problem:
 - Changes and Clones overlap,
 - Additions and Deletions are special
- Checking all versions is expensive
- Idea: How old is this line of code?

Version Controls Systems can 'blame'

ModeChangeHeight.java:95,105

```
1: 15154 int startOffset = layer.getNodeIndex(startY);
2: 8186 int endOffset;
3: 8533 if (startY > endY) {
4: 8533     endOffset = startOffset;
5: 15154     startOffset = layer.getNodeIndex(endY);
6: 8533 } else {
7: 15154     endOffset = layer.getNodeIndex(endY);
8: 8533 }
9: 8533 int diff = endOffset - startOffset;
10: 8533 if (diff > 0) {
11: 15154     layer.contractDiagram(startOffset, diff);
```

Results

- Cloned code is usually older than non-cloned code.
- On average, cloned code is 85 days older for the observed systems.
- Cloned code in a file is usually older than the non-cloned code in the same file.
- For the observed systems, this holds for 81% of the files.

Pitfalls

The different studies agree in general,
so can we make a general observation?

- If so, they must agree for each system!
- Experiment:
Compare two approaches on 12 systems

Experiment

	Agreement
DNSJava	✗
Ant-Contrib	✗
Carol	✗
Plandora	✗
Ctags	✗
TidyForNet	✗
QMail Admin	✓
Hash Kill	✓
GreenShot	✗
ImgSeqScan	✗
CapitalResource	✓
MonoOSC	✓

- Major Disagreement
- AA only considers the time after the last change
- MF considers the complete history
- AA and MF do not correlate!
- **Be aware of what you measure!**

RQ #2:

Are Clones changed consistently?

- If cloned code is changed consistently, it evolves together
- If cloned code is changed consistently, inconsistent changes may be bugs

Hypothesis #1

During the evolution of a system,
code clones of a clone group are changed consistently

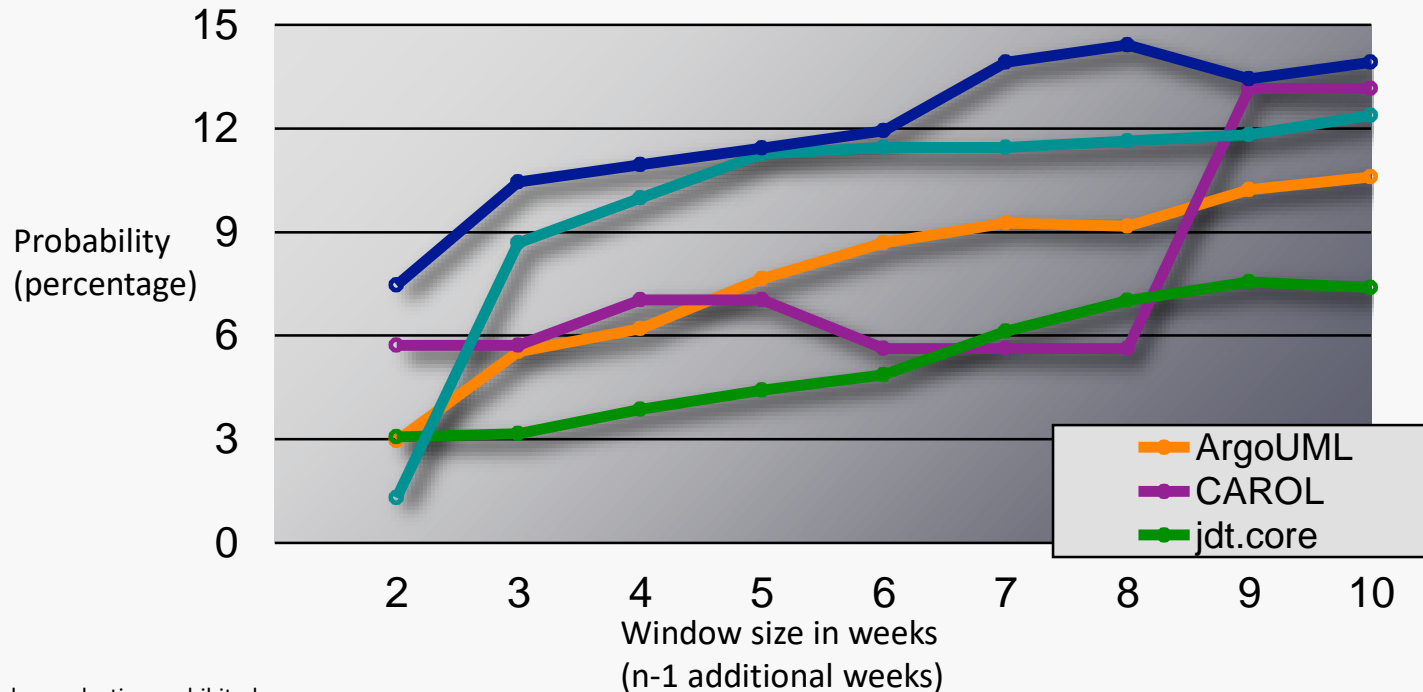
X is only valid half of the time

Hypothesis #2

During the evolution of a system,
if code clones of a clone group
are not changed consistently,
missing changes will appear in a later version
(Late Propagation)

Hypothesis #2

Probability that an inconsistent change will become a consistent change



RQ #2:

Are Clones changed consistently?

- half of changed clone groups are inconsistently changed
- if a clone group is inconsistently changed, there is an increasing probability that it is consistently changed later

Do Clones Lower the Quality?

There is no significant difference between the quality of cloned and non cloned methods for most of 27 metrics.

- 4,421 Java projects, 1,4mio methods)
- Code Complexity, Modularity, Documentation
- Exceptions
 - HEFF (Halstead effort)
 - NEXP (number of expressions)
 - XMET (number of external methods called)

Do inconsistent changes cause problems?

- Bakota et al. 2007: Detected six defects based on their analysis of 60 change anomalies in the evolution of clones.
- Selim et al. 2010: cloned code is not always more risky than non-cloned code; the risk seems to be system dependent.
- Göde, Koschke 2011: The number of unintentional inconsistent changes is small.

Bakota, Tibor, Rudolf Ferenc, and Tibor Gyimothy. 2007. Clone Smells in Software Evolution. International Conference on Software Maintenance.

Selim, Gehan M.K., Liliane Barbour, Weiyi Shang, Bram Adams, Ahmed E Hassan, and Ying Zou. 2010. Studying the Impact of Clones on Software Defects. Working Conference on Reverse Engineering.

Göde, Nils, and Rainer Koschke. 2011. Frequency and Risks of Changes to Clones. International Conference on Software Engineering.

Do Clones cause Bugs?

- most bugs have very little to do with clones
- cloned code contains less buggy code
- larger clone groups don't have more bugs than smaller clone groups
- making more copies of code doesn't introduce more defects

Concepts (1/2)

- Duplication of code is a common practice and often there are good reasons for cloning code, but cloning can introduce maintenance issues.
- Clones are typically detected in steps, including pre-processing, transformation, match detection, formatting, post-processing, aggregation.
- Clone detection can be text based, token based, syntax based, etc.
- A lot of clone detectors exist.

Concepts (2/2)

- Reuse of code can cause legal issues.
- The stability of Cloned code varies.
- Cloned code has only a small impact on bugs.