All we like sheep: Cloning as a software engineering tool

Michael W. Godfrey
University of Waterloo

Consider this code…

```
const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
if (err != NULL) {
   return err;
}
ap_threads_per_child = atoi(arg);
if (ap_threads_per_child > thread_limit) {
   ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
            "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
            "value of %d", ap_threads_per_child,
            thread_limit);
   ....
   ap_threads_per_child = thread_limit;
}
else if (ap_threads_per_child < 1) {
   ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
            "WARNING: Require ThreadsPerChild > 0, setting to 1");
   ap_threads_per_child = 1;
}
return NULL;
```


and this code …

```
const char *err = ap_check_cmd_context(cmd, GLOBAL_ONLY);
if (err != NULL) {
   return err;
}
ap_threads_per_child = atoi(arg);
if (ap_threads_per_child > thread_limit) {
   ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
            "WARNING: ThreadsPerChild of %d exceeds ThreadLimit "
            "value of %d threads,", ap_threads_per_child,
            thread_limit);
   ....
   ap_threads_per_child = thread_limit;
}
else if (ap_threads_per_child < 1) {
        ap_log_error(APLOG_MARK, APLOG_STARTUP, 0, NULL,
            "WARNING: Require ThreadsPerChild > 0, setting to 1");
        ap_threads_per_child = 1;
}
return NULL;
```


… or these two functions

```
gnumeric_oct2bin (FunctionEvalInfo *ei, GnmValue const * const *argv)
{
   return val_to_base (ei, argv[0], argv[1],
    8, 2,
     0, GNM_const(7777777777.0),
     V2B_STRINGS_MAXLEN | V2B_STRINGS_BLANK_ZERO);
}


gnumeric_hex2bin (FunctionEvalInfo *ei, GnmValue const * const *argv)
{
   return val_to_base (ei, argv[0], argv[1],
    16, 2,
     0, GNM_const(9999999999.0),
     V2B_STRINGS_MAXLEN | V2B_STRINGS_BLANK_ZERO);
}
```

Or this …

```
static PyObject *
py_new_RangeRef_object (const GnmRangeRef *range_ref){
  py_RangeRef_object *self;
  self = PyObject_NEW py_RangeRef_object,
      &py_RangeRef_object_type);
  if (self == NULL) {
   return NULL;
  }
  self->range_ref = *range_ref;
  return (PyObject *) self;
}
```


… and this

```
static PyObject *
py_new_Range_object (GnmRange const *range) {
  py_Range_object *self;
  self = PyObject_NEW (py_Range_object,
  &py_Range_object_type);
  if (self == NULL) {
   return NULL;
  }
  self->range = *range;
  return (PyObject *) self;
}
```


How to detect software clones

What's in a clone?

- Cloning versus similarity
  - *"Software clones are segments of code that are similar according to some definition of similarity."*
    – Ira Baxter, 2002
  - Hard to compare results!
- Bellon's taxonomy [1]:
  - Type 1: Program text identical; white space / comments may differ
  - Type 2: … also literals + identifiers may be different
  - Type 3: … gaps allowed (can add / delete sections)
  - Type 4: Two code segments have same semantics

Measuring detection effectiveness

- We borrow these terms from IR:
  - Precision: How many of the answers you find are real?
  - Recall: How many of the real answers do you find?

  … but we usually lack "ground truth"

- False positives and filtering:
  - Most detection tools are highly tunable
  - Often set tool for "more hits", then perform customized filtering to remove common false positives

More of the same, only different

- Problems related to software clone detection
  - Plagiarism detection, IP theft
  - DNA sequence analysis
  - Compression
  - SPAM analysis, malware detection

Code clone detection methods

- Strings
- Tokens
- ASTs
- PDGs

Time and complexity / prog lang dependence

- Metrics
- "Lightweight" semantics

See also Roy & Cordy's tech report [4]

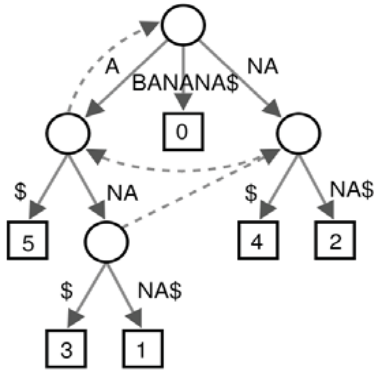
String-based clone detection

- Model:
  - Programs are *sequences* of *character strings*
- Simple to implement
  - Mostly independent of prog lang
- Typical:
  - Pre-process to remove white space + comments
  - Use hashes to speed comparisons of strings
- Variants:
  - Allow gaps in sequences (pattern matching)
  - Use edit distance for near-misses

Token-based clone detection

- Model:
  - Programs are *sequences* of *tokens*
- Low dependence on program lang!
- Typical:
  - Use suffix trees/arrays to detect results

Suffix tree / array



[Diagram from Wikipedia page on suffix trees]

- For each token stream ("string"), build a tree that represents all possible suffixes
  - Compare each new string to the set of existing trees
  - Comparisons are fast, but uses a lot of space
- Suffix array is a *sorted* list of all suffixes:
  1. a
  2. ana
  3. anana
  4. banana
  5. na
  6. nana

Token-based clone detection

- Variants
  - Generalize token streams
    … but add more info back in to analysis later [Baker]
    - CC-Finder treats all identifiers as "the same", it is the keywords and symbols that "count" most
    - 1-to-1 mapping of identifiers is imposed on possible matches
  - External tools "mark up" entity boundaries of token stream
    - We used `ctags` to make sense of CCfinder output
  - Add knowledge of prog lang to improve results
    - eg, `switch` stmts cause many false positives in C/C++/Java

AST-based clone detection

- Model:
  - Generate, then compare abstract syntax trees / graphs
    - Hijack an existing compiler
- Computationally expensive, but also very accurate
  - Usually, be selective about when to "go deep"
- Obviously, it's prog lang dependent

AST-based clone detection

- Variants:
  - Use parse trees instead
  - Use suffix trees to store ASTs [Koschke]
  - Walk trees to generate metrics instead of comparing trees structurally [Kostas]
  - Combined metrics / AST [Deckard, ICSE-07]

PDG-based clone detection

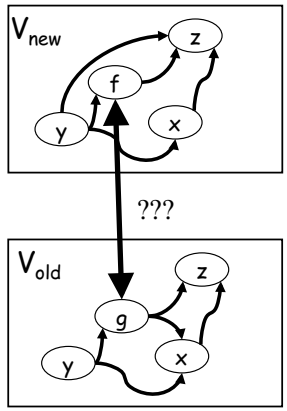
- Model:
  - Program dependence graphs (PDGs) are used in *program slicing* to compute the sub-programs that concern only particular variables / statements
- Can trace through / compare "paths of interest"
  - Easy to ignore gaps / interruptions in code
  - Largely immune to simple re-orderings of lines
- Naïve version is very expensive
  - So need to be clever about when to do it

Metrics-based detection

- Compare measurements instead of structures
- Main advantage: speed
  - Do expensive processing in one pass
  - … then compare numbers / vectors pairwise
- Also, largely immune to simple re-orderings
- Just about any of the previous methods can use metrics judiciously
  - Hash complex structures to simple values
  - Within buckets, selectively perform more expensive comparisons

Lightweight semantics

- "Origin analysis": [TSE-05]
  - Given consecutive versions of a system, which "new" entities really are new, and which are "old" entities that have been moved, renamed, merged, split, refactored
- Perform via
  - Entity analysis (clone detection)
  - Relationship analysis (eg call sets)
  - Known patterns of change



Detection Summary

- Approaches vary in complexity, programming language dependence
- Combining shallow + deep analysis judiciously seems key
  Cheap: Comparing hashes, metrics values, feature vectors
  Expensive: AST/PDG walking, dynamic programming
  – Many SE researchers use a tweaked token-based and/or suffix tree/array implementations
- Unclear how to measure progress, compare results

Is software cloning really a problem?

Just how bad is software cloning?

- Most research has concentrated on mere detection
- Recently focus has been shifting to clone *analysis*:
  – What techniques are effective to study large systems?
  – What kinds of clones are there? What properties do they have?
  – Does cloning really hurt the design in the long run?
- Case studies suggest that cloning is common practice in industrial software
  – 5-15% is common; up to 50% in some systems
  – Unclear how "bad" this is

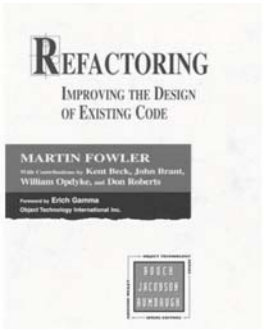
Quotes on source code cloning

"Q: You want to stop developers cutting and pasting code? Why?

A: This is Software Engineering 101, for heaven's sake!!"

– ACNP Software
`http://www.anticutandpaste.com/`

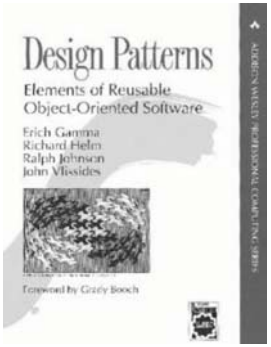
Quotes on source code cloning



"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."

– "Bad Smells"
[Beck/Fowler in *Refactoring*]

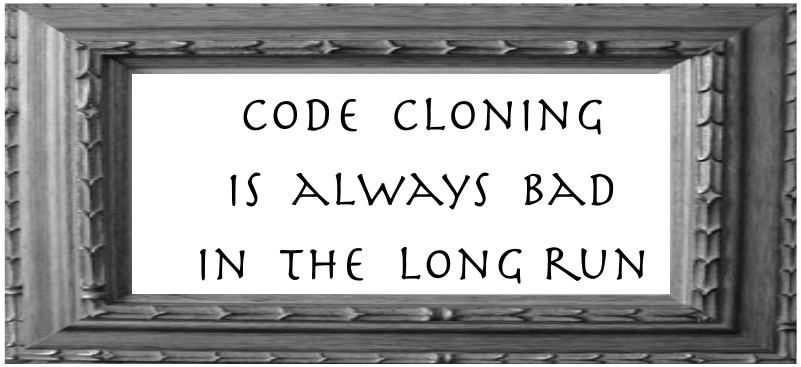
Quotes on source code cloning



"So, copy-and-paste is not necessarily bad in the short run, if you are copying good code. But it is always bad in the long run."

– Ralph Johnson, 2004 blog [3]

Myth



Why cloning is supposed to be bad

- Duplicated code leads to bloat
  - Hard to understand, less "essential"
  - Will all bug instances be fixed?
- Duplication is a sign of inexperienced developers
  - Copy/paste is often "easy", JIT comprehension
  - Cruft will accumulate as developers fear changing working code
- Duplication is a sign of poor design / extensibility
  - Need to keep doing same kinds of things, but there's no easy way to automate it

What you are supposed to do instead

1. Identify commonalities across code base

2. Refactor duplicate functionality to one place in the code
   - Functions with parameters
   - Base class encapsulates commonalities, derived classes specialize peculiarities
   - Generics / templates for classes / functions

Cloning is bad?

- Whoops!
- How did *that* happen?
- Have we been led astray?

Handel's Messiah

All we like sheep
All we like sheep
All we like sheep
All we like sheep
… have gone astraaaaaay

Formula, repetition, duplication

- In the arts and life, we seek to explore the new through careful venturing away from the familiar
  - Watch a toddler exploring his/her world
- The familiar can be
  - a narrative structure (e.g., a fairy tale, a knock-knock joke),
  - a chorus (new to us, but repeated),
  - a theme (e.g., sacrifice for love), …
- Humans also seem to find comfort in ritual
  - We seek refuge in the familiar when the external world seems unpredictable and frightening

Formula, repetition, duplication

- But this is *engineering*!
  - And we have no need of ritual in a utilitarian design!
- Ritual no, but repetition yes!
  - In traditional engineering, we scale up by repeated instantiations of design elements

Formula, repetition, duplication

- But this is *software* engineering!
  - We don't need duplication in a *software* design, right?


Investors Flock to VMware on First Day of Trading - New York Times http://www.nytimes.com/2007/08/15/technology/15soft.html?_r=1...

The New York Times nytimes.com

August 15, 2007

Investors Flock to VMware on First Day of Trading

By STEVE LOHR

Shares of VMware, the fast-growing Silicon Valley software maker, jumped 76 percent in their first day of trading yesterday as investors bet that the company would continue to fend off competitors like Microsoft for years to come.

VMware's initial public offering raised about $1.1 billion, making it the largest I.P.O. by a technology company since Google's in 2004. Defying a weak market, the shares closed the day at $51, or $22 higher than the offering price of $29.

The company hopes that the enthusiastic first-day reception in the stock market will also have strategic and marketing benefits. A valuable stock will give VMware the currency to make acquisitions, Diane Greene, the chief executive, said in an interview.

Formula, repetition, duplication

- Replication of trusted design elements works in software too!
  - We do need familiarity as a learning tool
  - And we can — and should! — employ duplication within a disciplined engineering process

Cloning as software design practice: Bronze an exemplar!

- The *Prototype* design pattern [GoF]
  - Used to create copies of families of complex objects
- The *Self* programming language [Ungar at al.]
  - Supports evolutionary designs better than trad OO langs
  - Helps with the fragile base class problem

Cloning as software design practice

- The Rule of Three (eXtreme Programming)
  - Premature abstraction is the root of much evil!
  - Design the simplest thing that could possibly work.
- Boiler-plating is key to industrial-strength COBOL development
  - Reliable designs and working systems are golden!

"Cloning considered harmful" considered harmful

'Cloning considered harmful' … considered harmful*

1. Forking
   - Hardware variation
   - Platform variation
   - Experimental variation
2. Templating
   - Boilerplating
   - API / library protocols
   - Generalized programming idioms
   - Parameterized code
3. Customizing
   - Bug workarounds
   - Replicate + specialize

**Best paper at 2006 Working Conference on Reverse Engineering [4]*

1. Forking

- Often used to "springboard" new or experimental development
  - Clones will need to evolve independently
  - Big chunks are copied!
- Works well when the commonalities and difference of the end solutions are unclear.

1. Forking: Platform variation

- Motivation:
  - Different platforms ⇒ very different low level details
  - Interleaving the platform-specific code in one place may be very complex
- Advantages of cloning:
  - Each (cloned) variant is simpler to maintain
  - No risk to stability of older variants
  - Platforms are likely to evolve independently, so maintenance is likely to be "mostly independent"

1. Forking: Platform variation

- Disadvantages:
  - Evolution in two dimensions: user requirements + platform support
  - Change to the interface level means changes to many files
- Management and long-term issues:
  - Factor out platform independent functionality as much as possible
  - Document the variation points and platform peculiarities
  - As number of platforms grows, the interface to the system effectively hardens

1. Forking: Platform variation

- Structural manifestations:
  - Cloning usually happens at the file level.
    - Clones are often stored as files (or dirs) in the same source directory
- Well known examples:
  - Linux kernel "arch" subsystem
  - Apache Portable Runtime (APR)
    - Portable impl of functionality that is typically platform dependent, such as file and network access
    - `fileio -> {netware, os2, unix, win32}`
    - Typical changes: insertions of extra error checking or API calls.
    - Cloning is clearly obvious and is documented

2. Templating

- Code embodying the desired behavior already exists
  - … but the impl. language does not provide strong support for the desired abstraction
- Linked editing or source auto-generation can be used
- Examples
  - COBOL boilerplate code
  - C routines that treat floats and ints analogously
  - (old) Java code that could have used generics
  - API usages for common tasks (eg GUI creation)
  - Language / platform idioms, such as safe pointer handling

3. Customization

- Existing code solves a similar problem but you can't or won't change it
  - May not own the code [Microsoft: "Clone and own"]
  - May not want to risk change there
  - Changing may be too complex
- Examples:
  - Replicate and specialize
  - Bug workarounds

Two case studies

Group	Pattern	Good	Harmful	Good	Harmful
Forking	Hardware variation	0	0	0	0
Forking	Platform variation	10	0	0	0
Forking	Experimental variation	4	0	0	0
Templating	Pre-forking	5	0	5	7
Templating	API	0	0	0	0
Templating	Macro	0	12	1	1
Templating	Parameterized code	5	12	10	34
Customizing	Replace + specialize	12	4	16	16
Customizing	Bug workarounds	0	0	0	0
Total		**36**	**28**	**32**	**67**

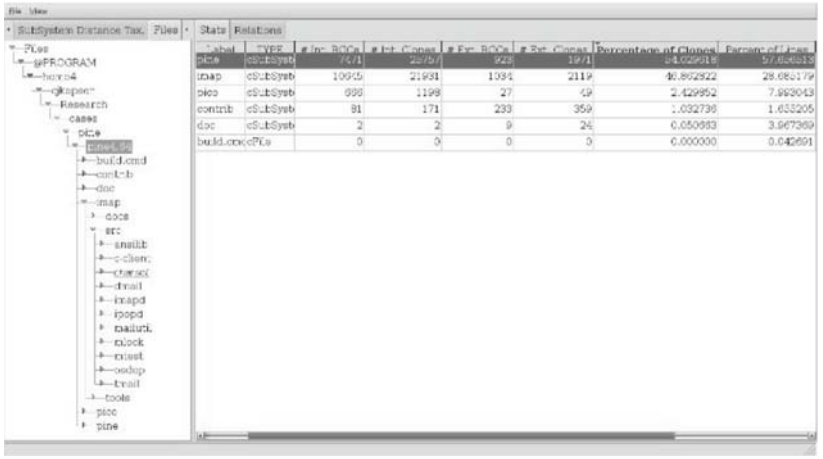
Apache httpd 2.2.4 - 60 Tokens

CLICS – A tool for clone analysis [Kapser]

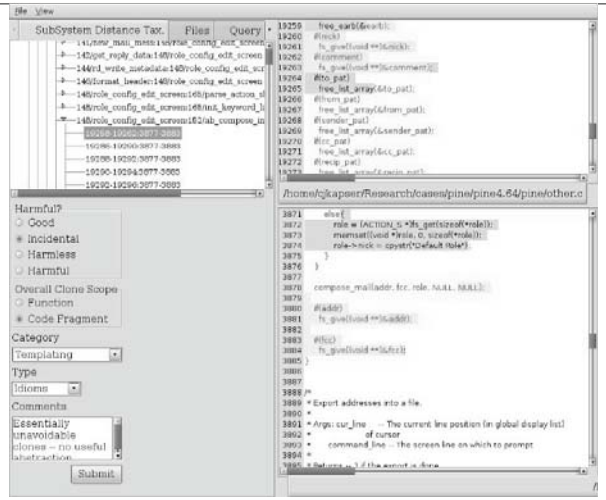
- Detection has two phases:
  1. Vanilla token-based matching
     - Identifiers all mapped to `<id>`
     - Observe func def boundaries
     - Uses suffix arrays (much like CC-Finder)
  2. Aggressive filtering reduces false positive rate
     - Require stronger matches for `switch / if`
     - Overlay overlapping clones and keep only ones with highest similarity

       e.g., Pine had 1.5M clones, but 1.2M were removed by this step

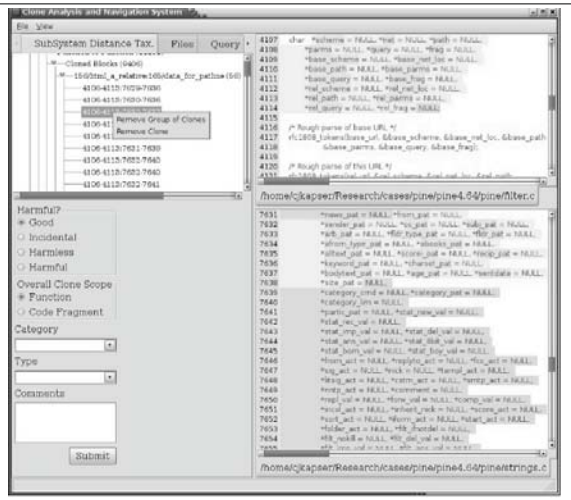
CLICS – Clone Analysis



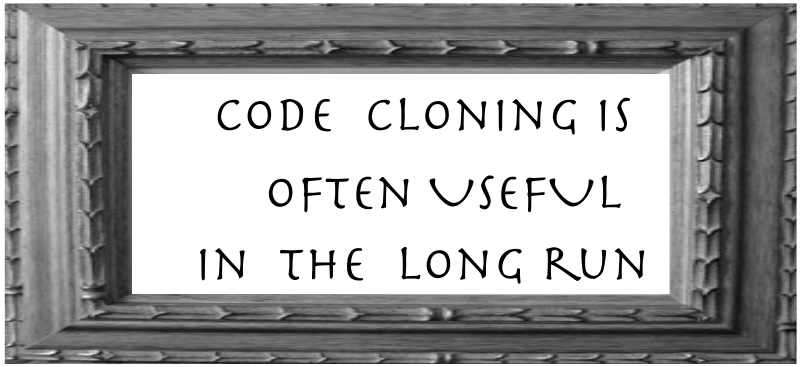
CLICS – Clone Analysis



CLICS – Clone Analysis



~~Myth~~ Motto



Special thanks

- To Cory Kapser, my PhD student
  - who will be graduating Real Soon Now!

References

1. Rainer's Koschke's completely awesome slides on source code cloning:
   - http://www.st.cs.uni-sb.de/edu/softmine2007/Slides/Koschke.pdf

2. Chanchal Roy + Jim Cordy's totally rad tech report on cloning
   - http://www.cs.queensu.ca/TechReports/Reports/2007-541.pdf

3. Ralph Johnson's blog:
   - http://www.cincomsmalltalk.com/userblogs/ralph/blogView?showComments=true&printTitle=Why_duplication_is_bad&entry=3271050352

4. "`Cloning considered harmful' considered harmful",
   - Cory J. Kapser and Michael W. Godfrey,
     2006 Working Conference on Reverse Engineering.

All we like sheep: Cloning as a software engineering tool

Michael W. Godfrey
University of Waterloo