Research based on Clone Detection

Overview

- An empirical study of code clone genealogies [1]
- A case study of cross-system porting in forked projects [2]

An empirical study of code clone genealogies

Based on Miryung Kim's lecture [4]

Problem Statement

- People believe that code clones indicate bad smells of poor design
  - programmers may introduce bugs when maintaining code clones inconsistently
- Is that true?

4

Findings in Previous Study[3]

- Even skilled programmers create and manage code clones with clear intent
  - Programmers cannot refactor clones because of programming language limitations
  - Programmers keep and maintain clones until they realize how to abstract the common part
  - Programmers often apply similar changes to clones

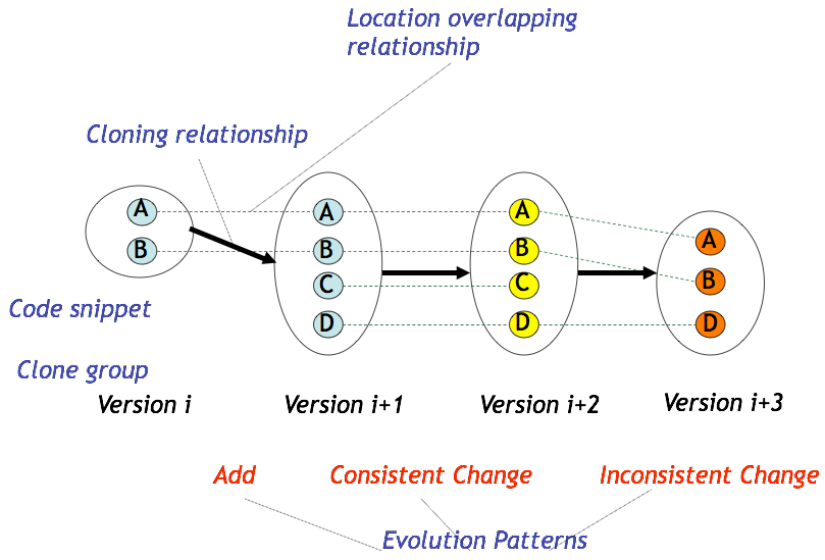
5

Research Questions

- How do clones evolve over time?
  - Consistently changed?
  - Long-lived (or short-lived)?
  - Easily refactorable?

6

Model of Clone Evolution



7

Clone Group Evolution Pattern

- Same: NG = OG
- Add: at least a new clone is added to NG
- Substract: at least an old clone is removed from OG
- Consistent Change: all clones are consistently changed
- Inconsistent Change: clones are changed inconsistently
- Shift: at least one clone in NG partially overlap with a clone OG

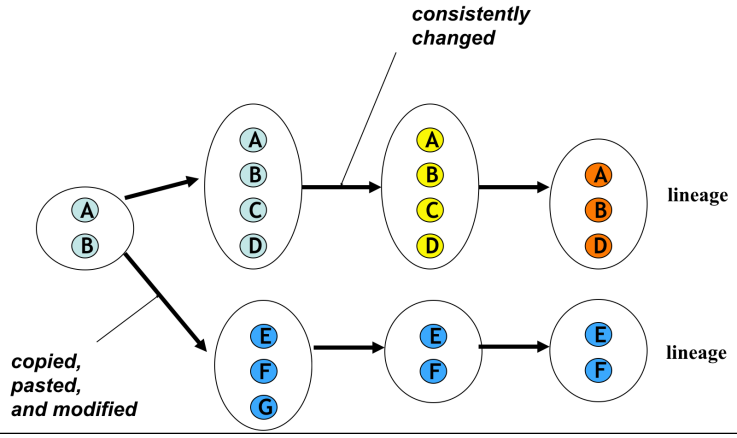
8

- There can also be some other evolution pattern, e.g., copy-paste-modify to generate a whole new clone group

9

Clone genealogy

- A set of clone groups connected by cloning relationship over time



10

Clone Genealogy Extractor (CGE)

- Given multiple versions of a program
  - Find clone groups in each version using CCFinder
  - Find cloning relationship between clone groups across versions based on text similarity
  - Identify a clone genealogy for each set of connected clone groups
  - Identify clone evolution behaviors in each genealogy

11

Experiment Settings

- Two Java subject programs

Program	carol	dnsjava
LOC	7878 - 23731	5756 - 21188
duration	2 yrs. 2 mos.	5 yrs. 8 mos.
versions	37	224

12

Detected Clone Genealogies

# of genealogies	carol	dnsjava
total	122	140
false positive	13	15
true positive	109	125
locally unfactorable	70 (64%)	61 (49%)
consistently changed	41 (38%)	45 (36%)

13

RQ1: How often do programmers change clones consistently?

- Approach
  - A genealogy has a "consistent change" pattern iff all lineages include at least one consistent change pattern
- Result
  - 38% and 36% of genealogies include a consistent change pattern

14

RQ2: What is the life time of clones?

- Separate live genealogies from dead genealogies
  - Dead genealogies: those which do not contain clones in the final version
- Calculate the life span of each dead genealogies

15

Result

- Among 109 clone genealogies of carol, 53 are dead
- Among 125 clone genealogies of dnsjava, 107 are dead
- Among the dead genealogies:

disappeared within	*carol*	*dnsjava*
2 versions	52%	35%
5 versions	75%	36%
10 versions	79%	48%

16

How do lineages disappear?

reasons	*carol*	*dnsjava*
divergent changes	26%	34%
refactoring or removal	67%	45%
cut off by the threshold	7%	21%

Contrary to conventional wisdom, immediate refactoring may be unnecessary or counterproductive in some cases.

17

RQ3: Are clones easily refactorable?

- A clone group is locally unfactorable if
  - programmers cannot use standard refactoring techniques, or
  - programmer must deal with cascading non-local changes, or
  - programmers cannot remove duplication due to programming language limitations.

18

Example

```
public void exportObject(Remote obj)
throws RemoteException{
 if (TraceCarol.isDebugRmiCarol()) {
  TraceCarol.debugRmiCarol(
   "MultiPRODelegate.exportObject(" ... .
 }
 try {
  if (init) {
   for (Enumeration e
    activePtcls.elemer

   ((ObjDlgt)e.nextElement()).exportObject(o
   bj);
   }
  }
 }catch (Exception e) {
  String msg = "exportObject(Remote obj)
   fail";
  TraceCarol.error(msg,e);
  throw new RemoteException(msg);
 }
```

64% and 49% of genealogies are locally unrefactorable

```
public void unexportObject(Remote obj)
throws NoSuchObjectException {
 if (TraceCarol.isDebugRmiCarol()) {
  TraceCarol.debugRmiCarol(
   "MultiPRODelegate.unexportObject(" ... .
 }
 try {
                                    s(); ...
   ((ObjDlgt)e.nextElement()).unexportObject
   (obj);
   }
  }
 } catch (Exception e) {
  String msg = "unexportObject(Remote obj)
   fail";
  TraceCarol.error(msg,e);
  throw new NoSuchObjectException(msg);
 }
```

19

Summary

- Immediate and aggressive refactoring may be unnecessary for volatile and diverging clones
- Refactoring may not help many long-lived and consistently changing clones
- Q: Do you have other observations?

20

A Case Study of Cross-System Porting in Forked Projects [2]

Based on Baishakhi Ray's slides

Problem Statement

- Software forking is important
  - Developers create a variant product by copying and modifying an existing product
  - E.g., FreeBSD, OpenBSD, and NetBSD evolve from the same code base
- What is the characteristic of code changes ported between peer projects?

22

Research Questions

- How often do developers port edits between projects?
- Are ported changes more defect-prone than others?
- How many developers are involved in patch porting?
- How long does it take for a patch to propagate across projects?
- Where is the porting effort focused on?

23

Methodology

- Repertoire: Detect ported edits by finding code clones in diff files using CCFinder
- Accuracy measurement
  - Construct a ground truth set of known ported edits, and use it to evaluate precision and recall of Repertoire
  - 94% precision and 84% recall

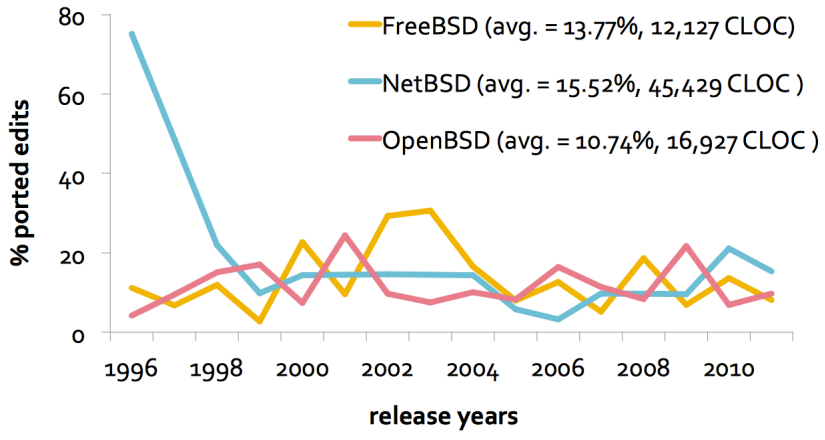
24

RQ1: How often do developers port edits?

$$avg\_porting\_rate = \frac{\sum_{releases} ported\_edits}{\sum_{releases} total\_edits}$$

25

Result



Porting is significant in the BSD family evolution, and it is not necessarily decreasing over time.

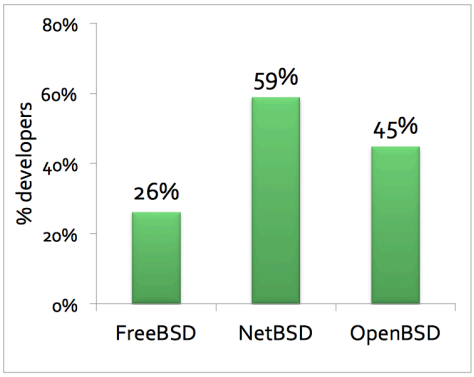
RQ2: Are ported edits more error-prone than others?

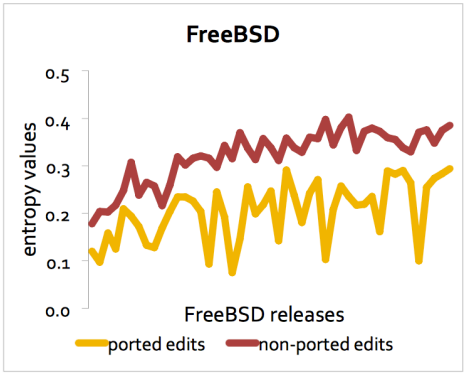
	CLOC	Ported CLOC	Non-Ported CLOC
FreeBSD	4754862	654858	4100004
Correlation with bugs	0.26	0.15	0.25
p-value	< 2.2e-16	< 2.2e-16	< 2.2e-16
NetBSD	4097338	636006	3461332
Correlation with bugs	0.41	0.36	0.42
p-value	< 2.2e-16	< 2.2e-16	< 2.2e-16
OpenBSD	4728360	507810	4220550
Correlation with bugs	0.37	0.32	0.38
p-value	< 2.2e-16	<2.2e-16	< 2.2e-16

- CLOC: Cumulative number of changed lines
- The correlation between bug fixes and ported edits is weaker than that between bug fixes and non-ported edits
- Q: Any improvement for the experiment?

27

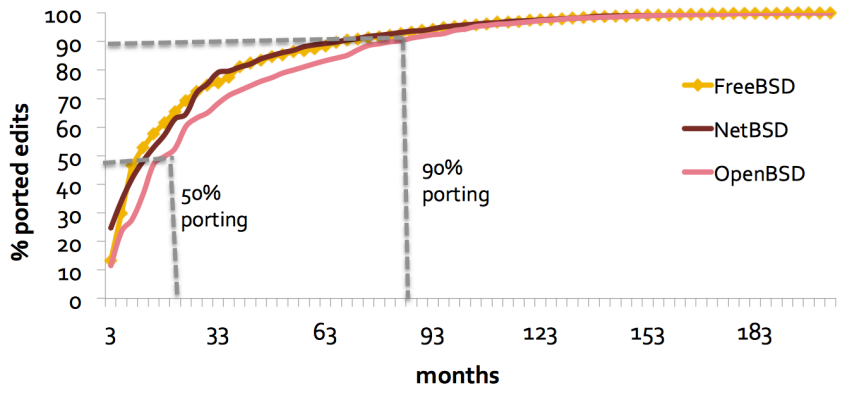
RQ3: How many developers are involved in porting patches from other projects?





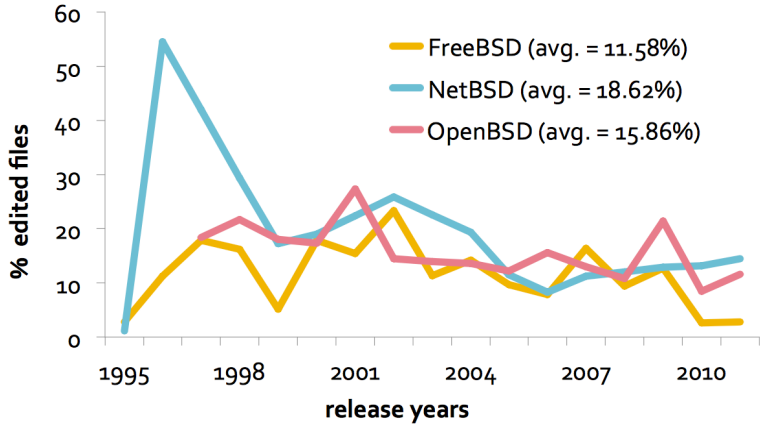
A significant portion of active committers port changes, but some do more porting work than others.

RQ4: How long does it take for a patch to propagate to different projects?



While most ported changes migrate to peer projects in a relatively short amount of time, some changes take a very long time to propagate to other projects.

RQ5: Where is the porting effort focused on?



Ported changes affect about 12% to 19% of modified files and porting effort is concentrated on specific parts of the BSD codebase.

Top 4 directories with the largest amount of ported changes

Rank	FreeBSD		NetBSD		OpenBSD	
1	src/crypto/openssl	21.54%	src/sys/arch	20.34 %	src/sys/dev	24.57 %
2	src/crypto/openssh	13.98%	src/sys/dev	19.96 %	src/lib/libssl	16.36 %
3	src/crypto/heimdal	13.31%	src/crypto/dist	10.61 %	src/sys/arch	11.16%
4	src/sys/dev	8.95%	src/gnu/dist	4.54%	src/usr.sbin/ppp	6.27%

31

Implications

- Call for automated approaches for cross-system porting (implied from RQ1)
- Call for tools to notify developers of potential collateral evolution and cross-system change impact analysis (implied from RQ5)
- Q: Any research questions you want to ask and make implications based on that?

32

Reference

[1] Miryung Kim, Vibha Sazawal, David Notkin, and Gail C. Murphy, "An empirical study of Code Clone Genealogies", ESEC-FSE '05

[2] Baishakhi Ray and Miryung Kim, "A Case Study of Cross-System Porting in Forked Projects", FSE '12

[3] Miryung Kim, Lawrence Bergman, Tessa Lau, David Notkin, "An Ethnographic Study of Copy and Paste Programming Practices in OOPL"

[4] Miryung Kim, "Empirical Studies of Clone Evolution Clone Genealogies", lecture in Fall 2010

33