Mining Resources in the Internet Space

Katsuro Inoue
Osaka University
Mining OSS Data

Huge Repository

Analyst  Powerful WS

OSS Data
Maintaining Huge Repository

• Keeping the repository updated is important
  – Time consuming effort
  – Automatic crawling is not easy
Mining OSS Data

Huge Repository

Analyst
Powerful Analyzer
OSS Data
Mining OSS Data

Analyst → Powerful Analyzer → OSS Data
Mining OSS Data

Analyst → Powerful Analyzer → Internet Space

Internet Search Engines → OSS Data
Mining OSS Data

Analyst → Small Interface and Filter → Internet Search Engines → OSS Data

Internet Space
Code History Tracking Model and Ichi Tracker
Developer’s Concerns

Existing Project

New Project

Reusable?

Code Fragment

Developer

• Origin
  - Who?
  - When?
  - License?
  - Copyright?

• Evolution
  - Maintenance?
  - Popularity?
  - Newer version?
  ...

To ease concerns, a support system is needed
Code History Tracking System

OSS Repositories

License: X
Copyright: Y

License: X'
Copyright Y'

Proj. A

Proj. B

Proj. C

Proj. D

Proj. E

Proj. F

Proj. G

Proj. H

Ancestor

Copy Source

Modify

Descendants

Time

Software Engineering Laboratory, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
Design Policy of Ichi Tracker

OSS repository
• Target many OSS projects from old to new ones
• No crawling, no maintenance
  → Do not have local repository, but use external code search engines

Output quality
• Find not only exactly same code fragments, but also similar ones
• Lower false positive results
• No real-time response
  → Use code clone filtering to improve the output quality
Code History Tracking Model

**Input Query Q**
- Code Fragment $q_c$
- Code Attributes (Optional)

**Output Results $R$**
- Code Clones
- Code Attributes

**Integrated Code History Tracker** *Ichi Tracker*

**Code Search Engines**
- SPARS/R
- Google Code Search
- Koders

**Internet**

**Open Source Repositories**

**Search Query SQ**

**Search Results SR**
Process of Ichi Tracker

1) Word Extraction
2) Keyword Selection
3) Query Generation
4) Results OK?
5) Download/Search Result Analysis
6) Code Clone Filtering
7) Result Forming

Input Query $Q$
Output Results $R$

Search Query $SQ$
Search Results $SR$

Keywords
Header List/Files
Control and Data Flow
Evolution Pattern of Texture.java

1. Jmonkeyengine r3448 (K)
2. Simplexe (G)
3. Simplexe (K)
4. Jmonkeyengine r3800 (G)
5. The-project08 (G)
6. Tank Combat Game (K)
7. wrathofthetaboos (G)
8. wrathofthetaboos (K)
9. Lasthaven (G)
10. Jme-cotk (G)
11. Ardor3D (G)
12. Jmonkeyengine r4099 (G)
13. Fairytale-souffire (G)
14. Jmonkeyengine r4490 (G)
15. Jmonkeyengine r4490 (S)
16. Xenogeddon (G)
17. Deathsquadrendezvous (G)
18. Partiendolapana (G)
19. Tholos (G)
20. Ardor3D (G)
21. Cosmic-engine (G)
22. Fregatclient3d (G)
23. Footballmanagerdesia (G)
24. Multiplicity (G)
25. Jmerefactoring (G)
26. Java3dfh (G)

Evolution of jMonkeyEngine Project
Cluster of Same or Similar Files
File in New BSD License
File in zib-libpng License
Evolution Pattern of kern_malloc

Results by G(Google Code Search) and K(Koders):
- File in New BSD License
- File in Original BSD License

Cover Ratio

Last modified time

1993/01/31 1995/10/28 1998/07/24 2001/04/19 2004/01/14 2006/10/10 2009/07/06
We have found many cases of different projects, different licenses and different copyrights
-> We need to check very carefully when we reuse SSHTools code
Usefulness

With simple check of the output of Ichi Tracker, we can get useful information for the history and evolution of code:

- **Origin**
  - Who?
  - When?
  - License?
  - Copyright?

- **Evolution**
  - Maintenance?
  - Popularity?
  - Newer version?
  - ...
Approach and Process

• Choosing good code search engines is a key to get high quality results
  
  GCS >> Koders > SPARS/R
  (GCS, Koders, SPARS/R) >> (Google, Bing)

  Need good code search engine available!

• Keyword selection strategy:
  Incremental strategy: try 1, 2, … keywords until the header list becomes less than 50
  – Decrement strategy, random, less frequently-used keywords, comment keywords, short keywords, …

  Less effective
Other Issues

• Performance
  – Case Studies A and B: 1 to 4 min.
  – Heavily depend on the code search engines and network performance
    Acceptable as non-interactive support system

• Quality of search result
  – Non-removed rate at the code clone filtering is an indicator of effectiveness of keyword search
    e.g., 0.46 (Case Study A(1) default setting)
  – The final output contains no false positives results
Pros and Cons

+ No huge repository
+ No maintenance cost and inconsistency
+ Easy to get the initial results

– Hard to control search results (engines) in details
– Fragile results depending on search engines
Can we use this analogy to other cases than the code search?
Thank you!