Using Reverse Engineering Practices to Improve Systems-of-Systems Understanding

Tom Hill
Department of Computer Science
The University of Texas at Dallas
November 11th, 2011

Outline
1. Systems-of-systems understanding research motivation
2. Research motivation
3. Enterprise-level systems-of-systems statistics
4. Reverse engineering essential design abstractions from source code
5. Prior art academia and industry
6. Systems engineering
7. Industry manual static source code analysis
8. Automatic static analysis for clones, text
10. Source code interview
11. Program copies analysis
12. Complexity analysis
13. Clone pairs
14. Data space versus algorithm space
15. Accounting system case study
16. Lessons and futures
17. References

Research motivation and context
1. Source code is the only truth that remains.
2. Worldwide Cumulative Lines of Code Chart estimate of 900 billion lines of code produced by professional programmers by the year 2011.
3. Tribal memory of architecture fades and documented blueprints are no longer trusted.
4. A legacy of over 200 billion lines of operational Cobol program code.

Problem statement
After five decades of promises, reverse engineering essential design abstractions from source code has fallen short. Software engineers still struggle to understand legacy systems-of-systems to:
- Maintain systems functionality
- Educate replacements for aging workforce
- Port to new platforms
- Implement changing requirements

Enterprise-level systems-of-systems statistics
Twenty two year life cycle with two IT Services business models

Enterprise-level systems-of-systems benchmarks
- Customer order- Transaction Processing Council (TPC-C)
- T. C. Jones Project Statistical Database 10,000 function points, 1.3 million Loc C
- Corporate Industry Productivity Database
Reverse engineering holy grail, abstractions from code

- Understanding abstraction artifacts: Requirements, Architecture, Design, Test, Operations, Maintenance

Prior art academia and industry

- Systems Thinking
- Systems Dynamics
- Systems Engineering
- Static Analysis
  - Program understanding
  - Clone detection
- Dynamic Analysis

Systems thinking

- Systems thinking: the analysis, synthesis, and understanding of interconnections, interactions, and interdependencies that are technical, social, temporal, and multi-level

Systems dynamics

- Systems dynamics is an approach to understanding the behavior of complex systems over time. It deals with internal feedback loops and time delays that affect the behavior of the entire system. What makes using system dynamics different from other approaches to studying complex systems is the use of feedback loops and stocks and flows

Systems engineering

- Systems Engineering is an interdisciplinary field of engineering that focuses on how complex engineering projects should be designed and managed over the life cycle of the project
- Designing, implementing, deploying and operating systems which include hardware, software and people

Industry manual static source code analysis

- Manual review to:
  - Document key system algorithms and trace the flow of variables
  - Construct a data model of data structures and trace their transformations from creation to destruction
Automatic static analysis for clones, text

Source line text comparison

Automatic static analysis for clones, tokens

Token-based comparison

Automatic static analysis for clones, AST

Abstract syntax tree comparison

Automatic static analysis for clones, PDG

Program dependence graph comparison

Automatic static analysis for clones, Halstead metrics

Halstead software science metrics:
- $n_1$ (The number of unique or distinct operators)
- $n_2$ (The number of unique or distinct operands)
- $N_1$ (The total usage of all the operators)
- $N_2$ (The total usage of all the operands)
- $n = n_1 + n_2$ (Known as the vocabulary $n$)
- $N = N_1 + N_2$ (Known as the implementation length $N$)
- Volume ($V$) can be calculated using: $V = N \log_2 n$

Automatic static analysis for clones, McCabe metrics

McCabe complexity metrics: Cyclomatic Complexity, Actual, Module Design, Essential, Pathological, Design, Integration, Object Integration, plus eight data complexity metrics
Automatic static analysis, industry tool-set

- Revolve industry maintenance tool-set

Project time-line

- 1977 Data affinity analysis
- Data element affinity analysis Cobol A 2.3 million lines of code Materials Management Application. Application data element affinity analysis was used to optimize the assignment of program source code to specific programmers for incentive based maintenance activities

Source code interviewer

- 1993 Source code interviewer tool (Cobol, PL1, C)

Program copies analysis

- 1998 Manufacturing company program copies (PL1)

Complexity analysis

- 2004 Transportation system complexity analysis (ALC, SabreTalk) - Programs ~4000, source lines of code = 1.24 million, two-path decision logic statements = 68,000 statements

Clone pairs

- 2009 Data structure usage to determine program clone pairs (Multiple languages)
Data space versus algorithm space

2009 Data structure usage to determine program clone pairs (Multiple languages)

Accounting system case study

2010 Accounting system case study metrics

Case analysis process

2010 Case five-step process

Case data structure relationships

2010 Data structure and program relationships

Case visualization

2010 High-density and low-density visualization

Lessons and futures

Lessons
- Limitations of static analysis
- Eliminate dead code bias
- Indirect references and pointers
- Special input output programs
- Dynamic program call binding
- Poor (non-descriptive) language variable names
- No display is large enough or has sufficient resolution (High-quality E size plots may still have value)
Lessons and futures

Future research
- OS dynamic profile [Oracle/Sun DTrace] analysis integration
- Infer cardinality of relationships
- Static analysis of external files/databases
- Indirect reference chain analysis
- Build quick sub-set parsers for additional languages
- Primary and foreign key discovery
- Analysis of executable for lost source code
- Implement committed data structure discovery and extraction

References

1. J. Koskinen, Software maintenance costs, Department of Computer Science, University of Jyvaskyla, 2000.