CS 6V81-05
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
   - Research motivation
   - Enterprise-level systems-of-systems statistics

2. Reverse engineering holy grail, abstractions from code

3. Prior art academia and industry
   - Systems engineering
   - Industry manual static source code analysis
   - Automatic static analysis for clones, text

   - Source code interviewer
   - Program copies analysis
   - Complexity analysis
   - Clone pairs
   - Data space versus algorithm space
   - Accounting system case study

5. Lessons and futures

6. 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
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

A. 10-year Development, Maintain and Operate Business Model

- Development Activities:
  1. Requirements / Design Artifacts (25%)
  2. Coding (20%)
  3. Testing and Quality Assurance (33%)
  4. User Documents (7%)
  5. Project Management (15%)

B. 10-year Maintain and Operate Business Model

- 226 Staff-Years Development
- 230 Staff-Years Maintenance of 10,000 Level Function Points
- Additional 145 Staff-Years Maintenance, as Function Points Grow from 10,000 to 22,600 over 18 years

- 3.6 Elapsed Years System Development
- 18 Elapsed Years Operational Maintenance
- 21 Time (Years)
Enterprise-level systems-of-systems benchmarks

1. Customer order - Transaction Processing Council (TPC-C)
2. T. C. Jones Project Statistical Database 10,000 function points, 1.3 million Loc C
3. Corporate Industry Productivity Database

<table>
<thead>
<tr>
<th>Lifecycle Phase Artifact Type</th>
<th>Artifact Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.0 Software development phase</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Functional Requirements Text</td>
<td>74 Pages</td>
</tr>
<tr>
<td>1.2 Non Functional Requirements Text</td>
<td>22 Pages</td>
</tr>
<tr>
<td>1.3 NFR Software Interdependency Graph</td>
<td>13 Clouds</td>
</tr>
<tr>
<td>1.4 UML Use Case</td>
<td>110 Bus Fun</td>
</tr>
<tr>
<td>1.5 UML Class Diagram</td>
<td>329 Classes</td>
</tr>
<tr>
<td>1.6 Architecture Confirmation Simulation Model</td>
<td>31 Nodes</td>
</tr>
<tr>
<td>1.7 Context Diagram</td>
<td>22 Entities</td>
</tr>
<tr>
<td>1.8 Initial Software Source Programs</td>
<td>164 Programs</td>
</tr>
<tr>
<td>1.9 UML Deployment Diagram</td>
<td>31 Nodes</td>
</tr>
<tr>
<td>1.10 Data Definition Language Tables</td>
<td>329 Tables</td>
</tr>
<tr>
<td><strong>2.0 Operation implementation phase</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 WAN LAN Network Diagrams</td>
<td>2 Diagrams</td>
</tr>
<tr>
<td>2.2 Services Specifications</td>
<td>65 Pages</td>
</tr>
<tr>
<td>2.3 Server Specifications</td>
<td>31 Servers</td>
</tr>
<tr>
<td>2.4 Storage Specifications</td>
<td>329 Tables</td>
</tr>
<tr>
<td>2.5 Operations Runbook and System Flow</td>
<td>77 Pages</td>
</tr>
<tr>
<td>2.6 Run-time System Performance Constraints SLAs</td>
<td>215 Pages</td>
</tr>
<tr>
<td><strong>3.0 System maintenance/operation phase</strong></td>
<td></td>
</tr>
<tr>
<td>3.1 Run-time Service Level Agreement Monitors</td>
<td>31 Monitors</td>
</tr>
<tr>
<td>3.2 Increase Software Source Programs During 18 yrs</td>
<td>206 Programs</td>
</tr>
</tbody>
</table>
Understanding abstraction artifacts: Requirements, Architecture, Design, Test, Operations, Maintenance
Prior art academia and industry

1. Systems Thinking
2. Systems Dynamics
3. Systems Engineering
4. Static Analysis
   - Program understanding
   - Clone detection
5. Dynamic Analysis
Systems thinking: the analysis, synthesis, and understanding of interconnections, interactions, and interdependencies that are technical, social, temporal, and multi-level.
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

1. 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.

2. Designing, implementing, deploying and operating systems which include hardware, software and people.
Industry manual static source code analysis

1. 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

```
154       CALC.JAVA   11/2001
155
156       public void calculate(double x)
157       {
158           if (lastCommand.equals("+")) result += x;
159           else if (lastCommand.equals("-")) result -= x;
160           else if (lastCommand.equals("*")) result *= x;
161           else if (lastCommand.equals("/")) result /= x;
162           else if (lastCommand.equals("=")) result = x;
163           display.setText("" + result);
164       }

157
260       public void showCalculation(double x)
261       {
262           if (lastCommand.equals("+")) result += x;
263           else if (lastCommand.equals("-")) result -= x;
264           else if (lastCommand.equals("*")) result *= x;
265           else if (lastCommand.equals("/")) result /= x;
266           else if (lastCommand.equals("=")) result = x;
267           display.setText("" + result);
268       }
```
Automatic static analysis for clones, tokens

1. Token-based comparison
Abstract syntax tree comparison
Automatic static analysis for clones, PDG

1. Program dependence graph comparison

```pascal
s1 begin
s2    a:=3;
s3    b:=3;
s4    readln(c);
s5    if c=0 then
s6    begin
s7    d:=functionA(a);
s8    e:=d
s9    end;
s10   else
s11   begin
s12   d:=functionB(b);
s13   e:=d
s14   end;
s15   writeln(e)
s16   end.
```
1. 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)
   - $n = n_1 + n_1$ (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

1. Revolve industry maintenance tool-set
Project time-line

1. 1977 Data affinity analysis
2. 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
1993 Source code interviewer tool (Cobol, PL1, C)
Program copies analysis

1. 1998 Manufacturing company program copies (PL1)
2004 Transportation system complexity analysis (ALC, SabreTalk) - Programs = 4000, source lines of code = 1.24 million, two-path decision logic statements = 68,000 statements

```dos
DO WHILE DATA_EXIST = 'Y';
    ....
    IF DC0LSU > DC0CNT
        THEN DC0PTR = CE1CR6; /* unexecuted path */
        ELSE DATA_EXIST = 'N';
    END;
```
2009 Data structure usage to determine program clone pairs (Multiple languages)
Data space versus algorithm space

1. 2009 Data structure usage to determine program clone pairs (Multiple languages)
Accounting system case study

2010 Accounting system case study metrics

Accounting System Metrics:
- Lines of code: 491,723
- Programs: 333
- Unique Data Structures: 345
- I/O commit statements: READ 77, WRITE 97
Case analysis process

1. Parse program library to find data structures
2. Parse program library to find commit I/O statements
3. Build repository of program library program to data structures relationships
4. Generate descriptive Graph Modeling Language
5. Layout Graph Modeling Language
Case data structure relationships

1. Nodes 20
   a. Programs (P*) 3
   b. Data structures (D*) 17
2. Edges 20

PGM->DS Fruchterman Reingold Edit
2010 High-density and low-density visualization

- **High-Density (Opaque):**
  - Nodes: 678
    - Programs (P*): 333
    - Data structures (D*): 345
  - Edges: 3970

- **Low-Density (Visible):**
  - Nodes: 305
    - Programs (P*): 99
    - Data structures (D*): 206
  - Edges: 1085
Lessons and futures

1. 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

1. 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

References

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