Software Reliability Estimation Based on Static Error Detection

M. Moiseev, M. Glukhikh, A. Karpenko, H. Richter
Importance of Software Reliability Analysis

Modern software contains errors

Software Reliability Analysis

Error detection should be organized

Errors can lead to disasters
Known Approaches

- Heuristics approaches
  - Program metrics
  - Development process
- Dynamic approach
- Architecture-based approach
Known Approaches – Program Metrics

- Based on simple code properties, such as
  - number of statements
  - number of conditions
  - number of loops
  - number of functions
  - ...

Software Reliability Estimation Based on Static Error Detection
Known Approaches – Development Process Metrics

- Based on development process properties, such as
  - duration of development
  - number & qualification of developers
  - number & qualification of testers
  - methodology used
  - automation tools used
Known Approaches – Others

- Runtime
  - Based on failures observed at run-time
- Architecture-based
  - Based on known reliability of program components
Our Approach

- Based on source code static analysis
- Delivers
  - Ranking of errors (based on failure probability)
  - Reliability characteristics
- Limitations
  - Single-threaded C programs
- Error types
  - uninitialized variable use
  - incorrect pointer dereference
  - pointer out of bounds
Features of Our Approach

- Analysis of a program model
- Analysis of all possible execution paths
- Advantages
  - Reliability estimations is based on real errors
  - Results are applicable for any exploitation conditions
  - Makes debugging more effective
- Drawbacks
  - Does not consider quantitative time
  - Does not consider normal program exploitation
  - Execution path probability estimation
  - False positives problem
Reliability characteristics used

- **Computational programs**
  - Probability of whole program successful execution: $P(\infty)$

- **Server programs**
  - Probability of $n$ statements successful execution: $P(n)$
  - Mean executed statement number before failure: $\bar{n}$
Algorithms

Model building

State determination

Error detection

Error ranking

Reliability estimation
Program Model Features

- Control flow graph
- Three-operand assignment form $A = B \text{ op } C$
- If and Phi statements
State Determination Algorithms

- State representation
- Control flow analysis
  - Statement analysis
    - Sequential
    - If statement analysis
    - Phi statement analysis
- Loop analysis
- Interprocedural analysis
Program State Representation

- Based on objects, values, and probabilities
  - set of triples \( Q = \{ (o_j, v_k, p_{jk}) \} \)
  - state probability \( P(Q) \)

- Object values
  - intervals
  - pointers
  - resource descriptors
Probability normalization

- Control flow normalization
  \[ \sum_{Q_{j}^{in} \in \text{Input}(s)} P(Q_{j}^{in}) = \sum_{Q_{j}^{out} \in \text{Output}(s)} P(Q_{j}^{out}) \]

- State normalization
  \[ \forall o_{j} : \exists (o_{j}, v_{k}, p_{jk}) \in Q \Rightarrow \sum_{(o_{j}, v_{k}, p_{jk}) \in Q} p_{jk} = P(Q) \]
Sequential Statement Analysis

\[ Q^{in} = \begin{cases} (b, (1..2), 1), \\ (c, (3..6), 1), \ldots \end{cases} = \begin{cases} \left( b, \frac{1}{2} \right), \\ \left( c, \frac{1}{4} \right), \left( c, \frac{1}{4} \right), \ldots \end{cases} \]

\[ Q^{out} = \{ (a, (4..8), 1), \ldots \} = \begin{cases} \left( a, \frac{1}{8} \right), \\ \left( a, \frac{1}{4} \right), \left( a, \frac{1}{4} \right), \ldots \end{cases} \]
If Statement Analysis

- True and false combination consideration

\[
P(Q^{\text{true}}) = \sum_{c \in C^{\text{true}}} \prod_{(o_j, v_k, p_{jk}) \in c} p_{jk},
\]

\[
P(Q^{\text{false}}) = \sum_{c \in C^{\text{false}}} \prod_{(o_j, v_k, p_{jk}) \in c} p_{jk}.
\]

- Normalization of state probabilities
- Normalization of non-affected triples probabilities
If Statement Analysis Example

- 172 combinations where $a < b$
- 28 combinations where $a \geq b$

Normalization: 0.86 for true, 0.14 for false
Phi Statement Analysis

- Identical triples are added together

\[ \forall o_j, v_k : (o_j, v_k, p_{jk}) \in Q_1^{in}, (o_j, v_k, r_{jk}) \in Q_2^{in} \Rightarrow \]
\[ (o_j, v_k, p_{jk} + r_{jk}) \in Q^{out} \]

- Control flow normalization

\[ P(Q^{out}) = P(Q_1^{in}) + P(Q_2^{in}) \]
Error Detection

- Based on incorrect values in state
  - uninitialized variable use \((o_j, v_{noninit}, p_k)\)
  - pointer dereference \((o_j, v_{noninit}, p_k)\)
  \((o_j, v_{null}, p_k)\) \((o_j, v_{invalid}, p_k)\)
  - out of bounds \((o_i, (o_j, offset_j), p_k)\)
  - correct if \(0 \leq offset_j < sizeof(o_j)\)
  - otherwise error is detected
Error Inhibition

(obj, valid, p1)
(obj, invalid, p2)
P(Q)=p1+p2

(obj, valid, p1)
P(Q)=p1
Error Ranking

- Errors are sorted according to probability of occurrence
  - Most dangerous errors can be corrected first

- Probabilities are summarized for same errors in the same statement
Overall reliability estimation

- probability of successful execution
  \[ P(n) = \sum_{\text{end statements}} P(Q) \]

- probability of \( n \) statements successful execution
  \[ P(n) = \sum_{n \text{ statements executed}} P(Q) \]

- mean executed statements number before failure
  \[ \bar{n} = \sum_{n=0}^{n_{\text{max}}} (P(n) - P(n+1)) \cdot n \]
Implementation

- AEGIS static analyzer
  - analysis of C/C++ source code
  - interval, points to, resource analysis
  - loop & interprocedural analysis
  - spread range of program errors detected
- Results
  - error ranking table
  - $P(n)$ table
  - $P(\infty)$
  - mean executed statements number before failure
Experiments made

- **Purpose**
  - Testing of our approach
  - Debugging example
Sample of reliability analysis

while (!feof(f)) // 0.5
{
    i = t = 0;
    // Failure in one of three cases
    prov(&t, strlen(st), st);  
}

- Probability of successful execution is
  \[ 0.75 = 0.5 + 0.5 \times 0.33 + 0.5 \times 0.33^2 + \ldots \]
Amount of errors in real-world projects

- More than 500 errors, 2/3 of considered types
- Density about 0.8/1KLOC
Debugging results

![Graph showing debugging results with original and corrected lines.](image-url)
Directions for Future Work

- Reliability estimation
  - Annotations for path probability estimations
  - Run-time analysis for path probability estimation
  - Execution time estimation
- Static analysis itself
  - Soundness & precision
  - Parallel program analysis
  - Annotations for functional error detection
Conclusion

- Approach for software reliability estimation
  - based on error detection using static analysis
- Implementation in AEGIS tool (prototype)
  - ranking of errors by the probability of occurrence
  - probability of successful execution
  - probability of N statement successful execution
  - mean number of executed statements before failure
Contacts

Saint Petersburg State Polytechnical University
Digitek Labs http://digiteklabs.ru
Mikhail Glukhikh, Mikhail Moiseev, Anatoly Karpenko
E-mail: glukhikh@kspt.ftk.spbstu.ru
E-mail: mikhail.moiseev@gmail.com
E-mail: karpenko@kspt.ftk.spbstu.ru

Clausthal University of Technology
Harald Richter
E-mail: hri@tu-clausthal.de

Software Reliability Estimation Based on Static Error Detection