Προγραμματισμός II
Πρακτικές αποσφαλμάτωσης

Διομήδης Σπινέλλης
Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας
Οικονομικό Πανεπιστήμιο Αθηνών

dds@aueb.gr
http://www.dmst.aueb.gr/dds
@CoolSWEng

2025-01-17

Overview

- High-Level Strategies
- Methods and Practices
- Tools and Techniques
- Debugger Techniques
- Programming Techniques
- Compile-Time Techniques
- Runtime Techniques
- Debugging Multi-threaded Code

High-Level Strategies

- Handle all problems through an issue-tracking system
- Use focused queries to search the web for insights into your problem
- Confirm that pre-conditions and post-conditions are satisfied
- Drill up from the problem to the bug or down from the program’s start to the bug
- Find the difference between a known good system and a failing one
- Use the software’s debugging facilities
- Diversify your build and execution environment
- Focus your work on the most important problems

Methods and Practices

- Enable the efficient reproduction of the problem
- Minimize the turnaround time from your changes to their result
- Automate complex testing scenarios
- Enable a comprehensive overview of your debugging data
- Consider updating your software
- Consult third-party source code
- Use specialized equipment
- Increase the prominence of a failure’s effects

- Enable the debugging of unwieldy systems from your desk
- Automate debugging tasks
- Houseclean before and after debugging
- Fix all instances of a problem class

Tools and Techniques

- Analyze debug data with Unix command-line tools (diff, comm, sort, cut, sed, awk, …)
- Explore debug data with your editor
- Optimize your work environment
- Hunt the causes and history of bugs with the revision control system (git log/blame/bisect)
- Use monitoring tools on systems composed of independent processes (Nagios, Ganglia, …)

Debugger Techniques

- Use code compiled for symbolic debugging
- Step through the code
- Use code and data breakpoints
- Familiarize yourself with reverse debugging
- Navigate along the calls between routines
- Look for errors by examining the values of variables and expressions
- Attach a debugger to a running process
- Work with core dumps
- Tune your debugging tools
- View assembly code and raw memory

Programming Techniques

- Review and manually execute suspect code
- Go over your code and reasoning with a colleague (or talk to a rubber duck)
- Add debugging functionality
- Add logging statements
- Use unit tests and assertions
- Verify your reasoning by perturbing the debugged program
- Minimize the differences between a working example and the failing code
- Simplify the suspect code
- Consider rewriting the suspect code in another (higher-level) language
- Improve the suspect code’s readability and structure
- Fix the bug’s cause, rather than its symptom

Compile-Time Techniques

- Examine generated code
- Use static program analysis (FindBugs, PMD, Coverity, …)
- Configure deterministic builds and executions
- Configure the use of debugging libraries and checks

Runtime Techniques

- Find the fault by constructing a test case
- Fail fast
- Examine application log files
- Profile the operation of systems and processes
- Trace the code’s execution (strace, dtrace, …)
- Use dynamic program analysis tools (Valgrind, ASan, Intel Inspector, Jalangi, …)

Debugging Multi-threaded Code

- Analyze deadlocks with post-mortem debugging
- Capture and replicate
- Uncover deadlocks and race conditions with specialized tools
- Isolate and remove non-determinism
- Investigate scalability issues by looking at contention
- Locate false sharing by using performance counters
- Consider rewriting the code using higher-level abstractions

Άδεια διανομής

Εκτός αν αναφέρεται κάτι διαφορετικό, όλο το πρωτότυπο υλικό της σελίδας αυτής του οποίου δημιουργός είναι ο Διομήδης Σπινέλλης παρέχεται σύμφωνα με τους όρους της άδειας Creative Commons Αναφορά-Παρόμοια διανομή 3.0 Ελλάδα.

SOME RIGHTS RESERVED