Software Debugging

CSE 403 - Spring 1997

Outline

- Why Debugging is Hard
- Various Debugging Techniques & Tools
- Self Promotion
  - Event Query Based Debugging

Debugging

- Problem definition:
  - Locate a defect that is responsible for an observed fault/failure.
  - Once located, repair the defect in such a way that removes the observed fault/failure without introducing new ones.

Debugging is HARD!

- Expensive.
  - 40 - 70% of total expense (telecomm industry statistics). [Seviora 87]
- Unpredictable impacts on schedule.
  - A defect that looks innocuous may take a long time to repair.
- Non-local bugs are particularly difficult to repair.
  - The distance between a fault/failure and defect is large in a *non-local bug*.
  - To successfully locate the defect, a programmer must bridge this distance.
  - To do so, a programmer needs to bring together related computation taking place at distant locations in program execution into a single entity.

Typical Debugging Activities

- Locate a fault/failute.
  - Usually easy.
  - Well supported.
    - Debuggers, assertions.
- Locate a defect.
  - Hard!
  - Need to trace backward to the defect from the fault.
- Rapir a defect.
  - Once located, usually eay. [Oman, et al. 91]
  - Why?

It is Difficult, Because...

- Partial Understanding
  - Size and complexity.
  - Someone else wrote it...
  - Misleading Manifestation
    - It's not what you think...
- Referential Problem.
  - How to specify all the necessary parameters that uniquely identify instance of interest.
  - E.g., inovocation of `free` with the same pointe value in different contexts.

It is Difficult, Because...

- Grouping Problem.
  - Need to group the collected information per instance.
  - E.g., memory allocation and the use of the allocated memory at a later point.
- Abstractions may hide necessary details.
  - E.g., I/O positions in Unix file I/O abstraction.

Debugging Tools

- Global Text Search Tool.
- Probe Insertion
  - E.g., `printf`
- Assertions
- Break-and-inspect Debuggers
  - Graphical User Interface
  - Data Structure Visualization
- Program Slicing
- Reverse Execution
- Algorithmic Debugging

Global Text Search Tool

- E.g., `grep`, `agrep`, etc.
- Widely used.
- Strengths
  - Simple to use.
  - Immediacy of use.
    - Start the search with texts available at inpsected location.
- Weaknesses
  - Lmited to textual matches.
  - Programmer must determine the relevance.
  - Could return too many hits or miss instances of interests.

Probe Insertion

- Insert output statements at various locations in the source code to gain necessary information.
  - E.g., printf
- Strengths
  - Minimal effort to add.
  - Output may be inspected backward.
- Weaknesses
  - Must know where to insert them.
  - Some type of outputs may not be possible.
    - E.g., scoping rule
  - May generate a lot of outputs.
  - May alter program behavior
  - Turn-around time can be high.

Assertions

- State conditions that must be satisfied at a location in the source code.
  - I.e., state programmer assumptions.
  - Each time the locus of execution passes the location, the condition is evaluated.
    - If it is true, nothing happens.
    - If it is false, suspend a program execution after outputing a message indicating the assertion that failed.

Assertions

- Strengths
  - This is more of a prevention than a debugging tool.
    - Make the program break at an earlier well-known location with a violated condition clearly stated.
      - I.e., reduce the fault-defect distance.
  - The original programmers can probably state them "good" ones at lower cost.
- Weaknesses
  - It is hard to write "good" ones for programmers who do not understand the code.
  - Turning them off may introduce new bugs.

Break-and-inspect Debuggers

- Surprisingly, it does not seem to be the most favorite tool for debugging programmers.
- Why?
- What improvements have been attempted?
  - Graphical user interface
  - Data structure visualization.

Other Debugging Tools

- Program Slicing
  - Given a variable used at a location in the source code, calculate an equivalent program that gives the same behavior per this variable.
  - Aims to help program comprehension.
  - Limitation of analysis.
- Reverse Execution
  - Supports backward reasoning better.
  - Issues with the amount of information to be kept.
  - How far to trace back?
- Algorithmic Debugging