General Pattern Identification of Debugging System

Falahah[1]
Informatics Department, Universitas Widyatama, Bandung
School of Electrical Engineering and Informatics,
Institut Teknologi Bandung
Indonesia
falahah@widyatama.ac.id

Iping.S.Suwardi[2)], Kridanto Surendro[3)]
[2,3)]School of Electrical Engineering and Informatics,
Institut Teknologi Bandung
Indonesia
iping@informatika.org [2)], endro@informatika.org [3)]

Abstract— Debugging is an important work in software development. Along with the increasing complexity of the software debugging process also becomes as simple as the original. This paper will discuss the generic pattern of debugging system. Vary approaches and algorithms that have been proposed to build a good debugger system and easily used by system developers. The bug can be the caused by internal or external sources. All debugger software is ideally fulfil 4 principle Heisenberg principle ie, truthful debugging, program context information system development and debugging trails. But actually not all debuggers can meet this requirement. Debugger also can classify in many ways such as source-level and machine-level, stand-alone and IDE, 4GL and 3GL, OS Kernel and Application Level, and Application-specific or in-circuit emulation. Debugger architecture can typically divide into 5 layers ie the user interface layer, the kernel layer, OS APIs, OS and CPU, before it touch the user program. Researchers also proposed many techniques in debugging methodologies such as Darwin, message oriented, backtracking approach, and concept assignment (CA). Implementation of CA opens new opportunities to a proposed new model of debugging that can leverage into a high level of software as a part of information system. The CA approach can implement According to the V-model of software development approach.

Keywords— debugging, debugger, techniques, general pattern, concept assignment, v-model..

I. INTRODUCTION

Debugging a code is a common activity in application development. Debugging process is done by programmer since early programming language has been used and it became more complex as complexity of programming environment increased. Usually, programmer doing debugging process using specific tools depends on programming language is used or the platform of the software. Despite of varies debugger tools, the study show that the debuggers usually have similar process and architecture.

The software become more complex and all interactions cannot be predicted, specifications usually are not written to the level of programming details, and implementation is an inherently difficult and error prone process. As software gets continuously more complex, debuggers become more and more important in tracking down problems.

Based on the definition, debugging is the process of searching for a bug, where nowadays many opportunities cause a bug in the software production process. The bug can be caused by internal sources such as library supporters unavailable, inaccessible certain services on the operating system, the version of services is not compatible, and so on, or external such as the programmer fault, syntax error, or requirement misinterpretation. All the bug can caused the software crash and according to software testing jargon that it is impossible to make a software 'free of error' but the to minimize the error, so bug usually come in any kind of software, even exist on a commercial software that have been used widely.

On the other side, debugging process can be done into several levels, from source-code level, into lower level on machine-code. Some research show that mostly debugging tools can debug program but some of them still have constraint in fulfil the 'basic' debugging requirement.

The aim of this paper is to review the generic pattern of debugging system and some approach in debugging techniques. Debugging not only can be done in lowest level of programming activity, but also can be mapped into general system requirement using concept assignment approach. It can offer more opportunity to extend the research on debugging into higher conceptual level of software design and development.

II. BASIC CONCEPT OF DEBUGGER

Rosenberg [4] describe the debuggers are software tools that help determine why the program does not behave correctly. The inner working of debugger required a suite of sophisticated algorithms and data structures to accomplish their tasks.

Debugger is used by developer, maintainer, tester and also adapter, and used by rerunning the applications in conjunction with the debugger tool itself. The debugger carefully controls the application using special facilities provided by the underlying operating system to give the very fine control over the program under test.

In scope of source code, bug can occurs in many way in program as explain by Spohrer and Soloway [10]:

1. Boundary problem, include off-by-one bugs (i.e. loops that terminate one iteration too early or too late).
2. Plan dependency problem, describe misplaced code often related to nesting, such as output statements that should be within a specific clause of an if-then statement rather than after the completed if-then construct.
3. Negation and whole part problems, are related to misuse of logical constructs, such as using an OR when an AND is needed.
4. Expectations and interpretation problems, are misinterpretation of how certain quantities are calculated.
5. Duplicate tail digit problems, involve dropping the final digit from a constant with duplicated tail digits.
6. Related knowledge interference problems occur when correct knowledge is similar to incorrect knowledge and the two can be confused.
7. Coincidental ordering problem, occur when arithmetic operations are to be carried out in the order in which they appear (left to right) but parentheses are still necessary to override operator precedence.

The step of using debugger can describes in four steps as follow [4]:

1. Debuggers are used at program inception time, when only part of the implementation of design is complete.
2. When an identifiable module or subsystem is completed and ready for use, to test the integration of a module with other components.
3. As testing progresses on a completed program and uncovers new defects.
4. As changes and adaptations are made to existing program that introduce new complexities and therefore destabilize previously working code

There are numerous approaches to debugging, perhaps as many as there are bugs. Some example techniques that common used are printing statements, printing to log files, sprinkling the code with assertions, using post mortem dumps, having program that provide function call stacks on termination, profiling, heap checking, automated data flow analysis, reverse execution, system call tracing tools, and interactive source-level debugging.

Interactive debugging tools also come in assorted flavors such as:

1. Kernel debugger, dealing with problem with an OS kernel on its own, or interaction between heavily OS dependent application and the OS.
2. Basic machine level debugger for debugging the actual running code (machine instruction), as they are processed by the CPU.
3. In-circuit emulator (ICF), which emulates the system services so all interactions between an application and the system can be monitored and traced.
4. Interpretative programming environment provided the debugger that well integrated into the run-time interpreter and has very tight control over the running application
   such as in Basic, Smalltalk, and Java.

Source-level symbolic debugging is the most frequently used technique for debugging end-user application, and on this approach, the high-level language source code is executed directly by the CPU

III. Principle of Debugger Design and The Classification

A. Debugger Design Principles

Four principles of debugger design and development that arise some problem are [4] Heisenberg principle, truthful debugging, program context information and debugging trail system development

Heisenberg Principle

Heisenberg principle state that the debugger must intrude on the debuggee in a minimal way. The act of debugging an application should not change the behavior of the application. In the case of software debugging, which the application and debugger is controlled by the same operating system, this basic principle is sometimes has violated by the debugger. The problem in fulfill Heisenberg principle has considered by OS and debugger designer, because, usually, the better they are able to keep the debugger from being intrusive and from impacting the behavior of the debuggee, the fewer bugs will disappear and become elusive only when run under the auspices of a debugger, and no effective way for developer to proceed.

Truthful Debugging

The principle states that the debugger must never mislead the user. Any misinformation will devastate the user, send the user off in the wrong direction potentially and cause a general lack of trust to develop between the user and the debugging tool.

Program Context Information

Program context may refer into several different types of information such as source code, stack back-trace, variable values, thread information and more. When the application crashed, the developer expects the debugger to show the location of source code and highlight the line. Sometimes, this may not point to the actual cause of the bug. Many bugs occur in one place but their effect (a crash) does not show up until much later. Stack back-trace is the process that allows developer to see the list of functions that the program passed through on the way to the current location.

Debugging Trails System Developments.

This principle states that system development occurs long before any corresponding strong debugging support for the new system development is available. Debugger developers need to push the system vendors to provide the necessary infrastructure to enable support of the latest technologies. Application developer also needs to push the debuggers to have ability to debug applications that being more complicated

B. Debugger Classification

Generally, the debugger can classify as bellow:

1. Source level (symbolic) and machine level.

Source level debugger usually works on compilation process, which transforms the source code into machine instruction. Usually, compilation process also provide extensive debug information about the source code and how it was mapped into machine code. Source-level debugger need to provide the low-level information that usually done by providing a CPU view that includes disassembly information, register values, a memory dump facility, and perhaps other machine-specific information.

2. Stand-alone and Integrated Development Environments

A stand-alone debugger is a program dedicated solely to debugging and is separate from compiling and editing. Integrated Development Environment (IDE) provide inline debugging so programmer can work more productive. The tools allow programmer to set some breakpoints and make changes to the source code via normal editing functions. But, GUI Debugger is more intrusive than a simpler, character mode debugger. But, stand-alone debugger using run-time library and lack of access to the persistent compiler symbol tables and to the compiler itself.

3. 4GL and 3GL.

4GL are used primarily in high-productivity business-oriented application-generation tools with an emphasis on database-based application. Most of 4GL tools is based on an interpreted language such as Basic or Smalltalk. Debugging is made dramatically simpler because the interpreter provide a safe, protected environment in which both the target application and the debugger can run. Some features that usually available on 4GL debugger such as breakpoints, watch expressions, single step, procedure step, out-of-line interpreter, and procedure call stack.

4. OS Kernel and Application-level.

Kernel debugging is a necessary part of developing device drivers. Modern OS have set of APIs and tools to allow the modification of OS behavior via the addition of specialized device drivers. Kernel debugging is normally set ip in such a way that two machine are involved, the host machine and the target machine being debugged. This way can give the debugger the complete picture during debugging. As the machine crashes, debugger on the crashing machine could not generate the report, but the remote machine can capture this critical information.

5. Application-specific and In-circuit emulation

Application-specific debuggers are general-purpose, high level debuggers that control one of just a few application at one time. The debugger notify the OS of their intention and get notification from the OS when important events occur within one specific application.

In-circuit emulators sit between operating system and the bare hardware and can watch and monitor all processes and all interaction between applications and the operating system. These kind of debugger usually used for development of add-on hardware or for very special type of heavily system-interacting applications.

IV. General Debugger Architecture, Process and Services

Rosenberg [4] describes the typical architecture of debugger as shown on figure 1. The figure shows the basic element of debugger system. Outer-most ring represent user interface and inner-most circle represent the core of the debugger interfacing to the underlying operating system.

The common steps that followed when debugging has observed by Gould and Drongowski (1974) and Gould (1975) consists of four steps [5]:

1. Choose a debugging tactic (such as read the line of code one by one until something suspicious was detected).
2. Find a clue (something suspicious in one of the information sources) to a bug.
3. Reporting the line containing the clue of error.
4. If nothing suspicious was detected, chose another debugging tactic.

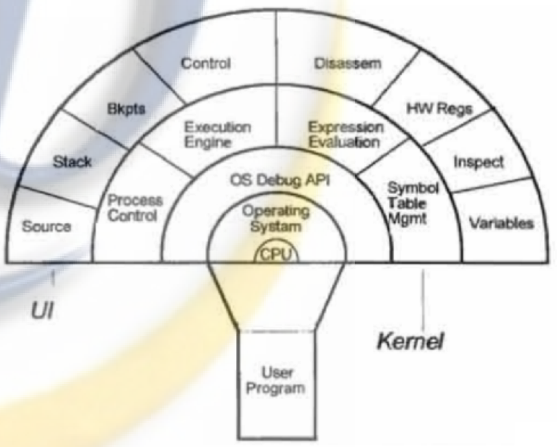

Figure 1. Typical debugger architecture [4]

Another approach of debugging tactic is goal subjects as explain by Vessey[13] :

1. Determine the problem with the program (compare correct and incorrect output)
2. Gain familiarity with the function and structure of the program
3. Explore program execution and / or control
4. Evaluate the program, leading to a statement of hypothesis of the error;
5. Repair the error

Vessey also draw a debugging strategy paths to describe the error possibilities, as show in figure 2.

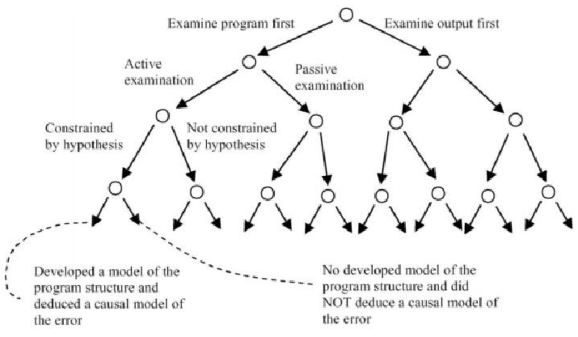
Examine program first
Examine output first
Active examination
Passive examination
Constrained by hypothesis
Not constrained by hypothesis
Developed a model of the program structure and deduced a causal model of the error
No developed model of the program structure and did NOT deduce a causal model of the error

Figure 2. Vessey's possible debugging strategy paths [13]

Tubaishat [12] modeling the conceptual architecture of BugDoctor tool to aid programmers in finding errors. The model is based on cognitive science studies describe the conceptual model for software fault localization as shallow reasoning, that used test cases, output, a mental model, and diagnostic rules of thumb, and deep reasoning which is characterized by intensive use of program recognition. The last one is supported by a knowledge base of programming concepts library which include a hierarchy of problem domain, algorithmic, semantic, and syntactic programming language knowledge/rules.

During deep reasoning, the developer usually uses two level approach in code recognition and fault localization: coarse-grained analysis and fine-grained analysis.

Traditionally, debugger has several tasks such as [5]:

1. Setting breakpoints that can be done by a suer through an interface. When the process that executed reach the breakpoint, an interactive session begins.
2. Step over, is an operation consists of moving to the following node in the code after having interpreted the current node.
3. Step-into, consists of moving to the next node in the code according to the application control flow.
4. Continue, the execution of an application may be resumed by continuing it
5. Terminate, the execution might be prematurely ended with the terminate operation.

Figure 3 explain the debugger services in class model.

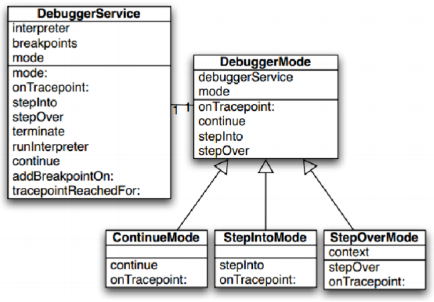

Figure 3. General services on debugger system [5]

V. Debugging Approach

Roychoudhury and Vaswani [6] proposed 'Darwin' methodology for debugging evolving program. Using this approach, debugger collect and suitably compose the path conditions of the failed test case in two program version to generate an alternate test input, and then in the next phase, debugger compares the trace of generated test input with the trace of failed test input to produce a bug report. Trace comparison proceeds by employing string alignment method (widely used in computational biology for aligning DNA sequences) on the trace, and the branches which cannot be aligned appear in the bug report.

Stanley, et.al [10], proposed the message-oriented as approach on distributed debugger. The debugger is post-mortem and consists of effective strategy to find the source of unintended side-effects, start with the chain of expressed intentions. The method is tested into distributed program.

Backtracking approach is a common approach on debugging techniques and is implemented in many debugger tools. Agrawal et.al, argue that using backtracking facility in interactive source debugger, allows programmers to mirror their though processes while debugging by working backwards from the location where an error is manifested and determining the conditions under which the error occurred, and add the 'what if' facility to allow the programmer change program characteristics and re-execute from arbitrary points within the program under examination.

Silva [9] compares some algorithm in debugging strategies implemented on ET (execution tree) such as single stepping (Shapiro, 1982), top down search (Av-Ron, 1984), Top-down Zooming (Maeji and Kanamori, 1987), Heaviest First (Bink, 1995), and promote new algorithm : less YES First (Silva, 2006). The research focuses on algorithm for visiting node on a piece of code (execution tree).

Concept Assignment (CA) [2] is a process for high-level program comprehension and relates human-oriented concepts to implementation-oriented artefacts. Human-oriented concepts often expressed using UML diagrams or other high-level specification schemes. On the other side, implementation-oriented artefacts are expressed directly in term of source-code features, such as variables and method calls.

CA attempts to work backward from source code to recover the 'concepts' that the original programmer where thinking about as they wrote each part of the program [7]. Generally, each individual source code entity implements a single concepts and the granularity of CA can breakdown into the smallest such as per token, per-line, or larger such as per block or per-method. CA sometimes visualized by coloring each source-code entity with the color associated with that particular concept. Adopting the set concepts, CA can be expressed mathematically.

Given U as set of source code unit, u_0, u_1, …u_n, and a set of concepts C, c_0,c_1,c_2,…,c_m. The CA can construct a mapping from U to C, or mapping itself. It is known as concept assignment [7].

Implementing concept assignment include two phases:

1. Concept selection: determine the set C of human oriented concept that are implemented in the source code.
2. Concept mapping: map each source code entity (at some specified granularity) to a particular concept from C.

Both phases can take place manually or automatically. In automated CA, a software tool analyses the source code and automatically selects the most appropriate concept for each source code entity (for some definition of appropriate). Automated system usually implement some artificial intelligence to determine the 'approriate' concepts.

Application of concept assignment mostly in program understanding, such as Biggerstaff et al [2] who is simply present a tool for software comprehension and visualization, Gold and Bennet state that the primary motivation for CA is providing the maintainer with an additional knowledge source from which to work. Kontogiannis et al, use CA information to assist in the detection of cloned source code, but no automated client that uses this information.

Singer [7] argue that a machine-level program understanding should facilitate further automated analysis or transformation of subject program, and by using CA, the information can support automated debugging. The idea is implemented on code generator debugger techniques and the result shows the similarities on structure of source-code with conceptual, and the tools can detect anomalies as potential bugs.

Research on CA opens the new opportunities to elaborate the approach in debugging for higher level abstraction of software, such as considering software as a part of information system. The software can include some component such as library component from operating system, database, data itself, control, and many more. The granularity of CA can elaborate to highest level such as business requirement is mapping into acceptance criteria or it can breakdown into lowest level such as system requirement specification, high level design, and so on, as describe in V model of software development.

VI. Conclusion

Debugging is an important function in software development and to debug the software it needs the debugger. The research on debugging area has conduct in many research areas such on algorithms, technique, approach, and data structure. Many researchers had determined the generic pattern of debugger architecture, functions, processes and services. But the research opportunities still open widely as the programming and software development increase their complexity rapidly.

The adoption in debugging techniques that already proposed includes Darwin method, backtracking method, message-oriented, and concept assignment. The concept assignment approach offer the possibilities to map the human-oriented concept into program-oriented and also can use as an approach to identify disoriented transformation between the concept. It opens new opportunities in research development of build generic debugging system for higher level presentation of software system.

References

[1] Agrawal, Hiralal, et al (1991), An Execution-Backtracking Approach to Debugging, Journal IEEE Software Volume 8 Issue 3, May 1991, Page 21-26

[2] Biggerstaff, Ted J, and Mitbander, et al (1993), The Concept Assignment Problem in Program Understanding. ICSE '93 Proceedings of the 15th international conference on Software Engineering, Pages 482-498.

[3] Ko, A., & Myers, B. (2005). A framework and methodology for studying the causes of sofware errors in programming systems.Journal of Visual Languages and Computing, 16, 41–84.

[4] Rosenberg, Jonathan B, (1996), "How Debugger Work: Algorithms, Data Structures, and Architecture", John Wiley & Sons, Inc.

[5] R McCauley et al, (2008), Debugging: a review of the literature from an educational perspective, Computer Science Education 18 (2), 67-92

[6] Rochchoudhury, Abhik and Vaswani, Kapil,(2008) "DARWIN: An Approach for Debugging Evolving Programs", Technical Report MSR-TR-2008-91, Proceeding of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Pages 33-42.

[7] Singer, Jeremy,(2005), Concept Assignment as a Debugging Technique for Code Generators, Fifth IEEE International Workshop on Source Code Analysis and Manipulation.

[8] Silva, Josep (2005), A Comparative Study of Algorithmic Debugging Strategies, LOPSTR'06 Proceedings of the 16th international conference on Logic-based program synthesis and transformation, Pages 143-159

[9] Spohrer, J., & Soloway, E. (1986a). Analyzing the high frequency bugs in novice programs. In E. Soloway & S. Iyengar (Eds.),Empirical studies of programmers(pp. 230–251). Norwood, NJ:Ablex.

[10] Stanley, Tyler and Miller, (2009) Causeway: A message-oriented distributed debugger, Technical Report of HP, HPL-2009-78,

[11] Stamey, John, and Saunders, Bryan (2005), Unit Testing and Debugging with Aspects, Journal of Computing Sciences in Colleges, Volume 20 Issue 5, May 2005, Pages 47-55

[12] Tubaishat, A. (2001). A knowledge base for program debugging. In Proceedings of the international conference on computer systems and applications(pp. 321–327). Beirut: IEEE Press.

[13] Vessey, I. (1985). Expertise in debugging computer programs: A process analysis.International Journal of Man–Machine Studies, 23, 459–494.

This page is left blank on purpose