Debugging

1. Identify the Problem
2. Gather Data
3. Hypothesis
4. Test the Hypothesis (Experiment)
   - Yes: Update Hypothesis
   - No: Does the New Data Agree?
Debugging, in the Trenches

Rasala put his hands on his desk and buried his face in them. It was just another routine day down at debugging headquarters.

In the back of Veres’s mind still lies a small suspicion that the problem might after all be noise. And now – much to Guyer’s delight, when he finds out later on – it is Veres himself who disconnects the I-cache. Then he runs the program past the point of failure, and everything works. He puts the I-cache back in and once again Gollum fails. This doesn’t prove the IP is to blame, but it does tend to eliminate noise as a suspect, once and for all. . .

from THE CASE OF THE MISSING NAND GATE (Chapter 10 of Kidder’s The Soul of a New Machine)
Kidder’s book tells the story of the development of a micro-computer in the early 80s. The book’s a classic – won the American Book Award for non-fiction. Anyone who cares how computers are made (or how people work) should read it.

Chapter 10 is a classic story of debugging a hardware problem. The ideas apply just as well to software, and this is the best description of heavy-duty debug I’ve ever seen.
Debugging

- Debugging is really hard – even with a good failing test case in hand

- One of the most time-consuming tasks in software development

- Locating the fault is the most time-consuming part of debugging
Debugging

- Takes as much as 50% of development time on some projects
- Arguably the most scientific part of “computer science” practice
  - Even though it’s *usually* done in a totally *ad hoc*, haphazard way!

“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.” - Brian Kernighan
Debugging and Testing

- What are test cases *for*, anyway?
  - Often: so we can locate and fix a fault
  - Or: so we can understand how serious the failure is, and triage / “flight rule” it away
    - If we have many bugs, and some may not be important enough to merit resources
    - Or if there is a reason we can’t change the code and have to work around the problem

- In either case, we have a “debugging” task at hand – must at least *understand the failure*, even if only to triage
Scientific Debugging

- Test cases (ones that fail and ones that succeed) can be the experiments we perform to verify our hypotheses
  - The failing test case informs us that there is a phenomenon to explain (apple on the head)
  - Generate (or examine) more test cases to find out more about what is going on in the program
Scientific Debugging

Failing Test

Hypothesis

Prediction

Experiment

Observation + Conclusion

Hypothesis is supported:
refine hypothesis

Hypothesis is rejected:
create new hypothesis

Diagnosis

Code

Test

More Tests
Testing for Debugging

Several ways to use test cases in debugging:

- Test case minimization
  - Shrink the test case so we don’t have to look at lots of irrelevant or redundant operations

- Fault localization
  - Give suggestions about where the fault may be (based on test case executions)

- Error explanation
  - Give a “story” of causality
    - (A causes B; B causes C; C causes failure)

attempt to automate part of scientific debugging
What’s a problem?

- A *problem* is a questionable property of a program run
- It becomes a *failure* if it’s incorrect…
- …a *request for enhancement* if missing…
- …and a *feature* if normal behavior.

*It’s not a bug, it’s a feature!*
Problem Life Cycle

- The user *informs* the vendor about some problem.
- The vendor
  1. *reproduces* the problem
  2. *isolates* the circumstances
  3. *locates* and *fixes* the defect
  4. *delivers* the fix to the user.
Vendor Challenges

- How do I organize the life cycle?
- Which problems are currently open?
- Which are the most severe problems?
- Did similar problems occur in the past?
User Challenges

• Solve my problem!
Problem Report

- A problem comes to life with a *problem report*.
- A problem report includes all the information the vendor needs to fix the problem.
- Also known as *change request* or *bug report*.
Problem report #1

From: me@dot.com
To: zeller@gnu.org
Subject: Crash

Your program crashed. (core dumped)
From: me@dot.com
To: zeller@gnu.org
Subject: Re: Crash

Sorry, here's the core - cu

<core, 14MB>
Problem report #3

From: me@dot.com
To: zeller@gnu.org
Subject: Re: Crash

You may need that, too (just in case)

<drive_c.zip, 148GB>
What to report

- The *product release*
- The *operating environment*
- The *problem history*
- *Expected and experienced behavior*
- A one-line *summary*
Product Release

- Typically, some *version number* or otherwise unique identifier

- Required to *reproduce the exact version*:
  - Perfect Publishing Program 1.1 (Build 7E47)

- Generalize: Does the problem occur only in this release?
Operating Environment

- Typically, *version information* about the operating system
- Can be simple (“Windows 98 SE”) or complex (“Debian Linux ‘Sarge’ with the following packages…”)
- Generalize: In which environments does the problem occur?
Problem History

- Steps needed to *reproduce* the problem:
  1. Create “bug.ppp”
  2. Print on the default printer…

- If the problem cannot be reproduced, it is unlikely to be fixed

- Simplify: Which steps are relevant?
Expected Behavior

- What should have happened according to the user:
  - The program should have printed the document.

- Reality check: What’s the understanding of the user?
Observed Behavior

- The *symptoms* of the problem — in contrast to the *expected* behavior
  - The program crashed with the following information

  *** STACK DUMP OF CRASH (LemonyOS) ***

  - Back chain  ISA  Caller
    - 00000000  SPC  0BA8E574
    - 03EADF80  SPC  0B742428
    - 03EADF30  SPC  0B50FDDC  PrintThePage+072FC
    - SnicketPC unmapped memory exception at
      - 0B512BD0 PrintThePage+05F50
A one-line summary

- Captures the essential of the problem
  - PPP 1.1 crashes when printing
Things to avoid

- **Humor**
  - PPP (oops, gotta go to the restroom :-) …

- **Sarcasm**
  - Here’s yet another “never-to-be-fixed” bug

- **Attacks**
  - If you weren’t too incompetent to grasp…
Talk back

The Netscape Quality Feedback Agent has captured information that Netscape needs to help improve Communicator’s quality.

Enter your email address (optional), describe how you were using Communicator (optional), then click Send.
Your Email Address (optional):

Andreas Zeller <zeller@cs.uni-so.de>

[ ] Send me information about updates to Netscape products

If failure occurred within the browser please provide URL/location address of the page you were browsing

about:config

Describe what you were doing when Communicator failed (optional):

Printing "about:config" crashes.

Details

Application Information
- Agent Configuration Version
- Agent Library Version
- Application Launch Time
- Build Identifier
- Deployment Identifier
- Interface Version
- Monitor Configuration Version
- Platform Identifier
- Product Identifier

[ ] Include Info rejected above

NetscapeGecko1.4LinuxIntel2003061711
Talk Back + Privacy

- Be sure what to collect and include in an automated report:
  - Pages visited
  - Text entered
  - Images viewed…

- *Privacy* is an important issue here!
Managing Problems

- Alternative #1: A Problem File
  - Only one person at a time can work on it
  - History of earlier (fixed) problems is lost
  - Does not scale

- Alternative #2: A Problem Database
Bugzilla

This is Bugzilla: the Mozilla bug system. For more information about what Bugzilla is and what it can do, see mozilla.org’s bug pages.

Search for bugs

Summary: contains all of the words/stings

Product:
- Browser
- Bugzilla
- Calendar
- CCK
- Chimera

Component:
- Accessibility
- Accessibility APIs
- Account Manager
- Address Book
- Addressbook/LDAP (non-UI)

Version:
- 1.01
- 1.1
- 1.2
- 1.3
- 1.4

Target:
- Future
- 3.0
- Jan
- M1

A comment: contains all of the words/stings

The URL: contains all of the words/stings

Whiteboard: contains all of the words/stings

Keywords: contains all of the keywords

Status:
- UNCONFIRMED
- NEW
- ASSIGNED
- REOPENED
- RESOLVED
- VERIFIED
- CLOSED

Resolution:
- FIXED
- INVALID
- WONTFIX
- LATER
- REMIND
- DUPLICATE
- WORKFORME

Severity:
- blocker
- critical
- major
- normal
- minor
- trivial
- enhancement

Priority:
- --
- P1
- P2
- P3
- P4
- P5

Hardware:
- All
- DEC
- HP
- Macintosh
- PC
- SGI
- Sun

OS:
- All
- Windows 3.1
- Windows 95
- Windows 98
- Windows ME
- Windows 2000
- Windows NT

Email and Numbering

Any of:
- bug owner

Bug Changes

Only bugs changed in the last
- days
Classifying Problems

- Severity
- Priority
- Identifier
- Comments
- Notification
Severity

- **Enhancement.** A desired feature.
- **Trivial.** Cosmetic problem.
- **Minor.** Problem with easy workaround.
- **Normal.** “Standard” problem.
- **Major.** Major loss of function.
- **Critical.** Crashes, loss of data or memory
- **Showstopper.** Blocks development.
Priority

- Every new problem gets a *priority*
- The higher the priority, the sooner the problem will be addressed
- Priority is independent from severity
- Prioritizing problems is the main tool to control development and problem solving
Identity

- Every new problem gets an *identifier*
- (also known as *PR number* or *bug number*)

- The identifier is used in all documents during the debugging process:
  - Subject: PR #3427 is fixed?
Comments

• Every developer can attach *comments* to a problem:
  - I have a patch for this. It's just an unitialized variable but I still need a review.

• Comments may also include files, documents, etc.
Notification

- Developers can attach an e-mail address to a problem report; they will be notified every time the report changes.
- Users can do so, too.
The Problem Lifecycle

- UNCONFIRMED
- NEW
- ASSIGNED
- RESOLVED
- VERIFIED
- CLOSED

- INVALID
- DUPLICATE
- FIXED
- WONTFIX
- WORKSFORME

- REOPENED

- NEW, Status
- FIXED, Resulting Resolution

If the resolution is FIXED, the process can return to the RESOLVED state.
Unconfirmed Problem

- The problem report has just been entered into the database
New Problem

- The report is *valid* and not a *duplicate*.
- *(If not, it becomes *resolved*.)*
- The problem is assigned to a developer
Resolution

**FIXED:** The problem is fixed.

**INVALID:** The problem is not a problem.

**DUPLICATE:** The problem already exists.

**WONTFIX:** Will never be fixed (for instance, because the problem is a feature)

**WORKSFORME:** Could not be reproduced.
Resolved Problem

- The problem report has been processed.
The problem is fixed; the fix has been successful.
A new version with the fix has been released.
Reopened Problem

- Oops – there we go again :-(

Management

- Who *enters* problem reports?
- Who *classifies* problem reports?
- Who sets *priorities*?
- Who takes care of the problem?
- Who *closes* issues?
The SCCB

- At many organizations, a *software change control board* is in charge of these questions:
  - Assess the *impact* of a problem
  - Assign tasks to developers
  - Close issues…
Problem-driven Development

- The whole development can be organized around the problem database:
  - Start with one single problem:
  - “The product isn’t there”
  - Decompose into sub-problems
  - Ship when all problems are fixed
Managing Clutter

- Large problem databases contain garbage
- Get rid of duplicates by
  - simplifying bug reports
  - asking submitters to search first
- Get rid of obsolete problems by searching for old ones that rarely occurred
Problems and Fixes

Use tag in problem reports
Problems and Tests

- Some test fails. Should we enter the problem into the database?
- *No*, because test cases make problem reports obsolete.
- Once we can repeat a problem at will, there is no need for a database entry.
Concepts

- Reports about problems encountered in the field are stored in a *problem database*.

- A problem report must contain everything relevant to reproduce the problem.

- It is helpful to set up a standard set of items that users must provide (product release, operating environment...).
Concepts (2)

🌟 An effective problem report...

- is *well-structured*
- is *reproducible*
- has a descriptive *one-line summary*
- is as *simple* and *general* as possible
- is *neutral* and stays with the facts.
A typical problem life cycle starts with an unconfirmed status.

It ends with a closed status and a specific resolution (such as fixed or worksforme).

Typically, a software change control board organizes priorities and assignments.
Use version control to separate fixes and features during development.

Establish conventions to relate changes to problem reports and vice versa.

Make a problem report obsolete as soon as a test case exists.