Course "Debugging"

Debugging Rules 4, 5, 6, 7

Prof. Dr. Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Rule 4:
  - Divide and conquer
- Rule 5:
  - Change one thing at a time
- Rule 6:
  - Keep an audit trail
- Rule 7:
  - Check the plug

The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

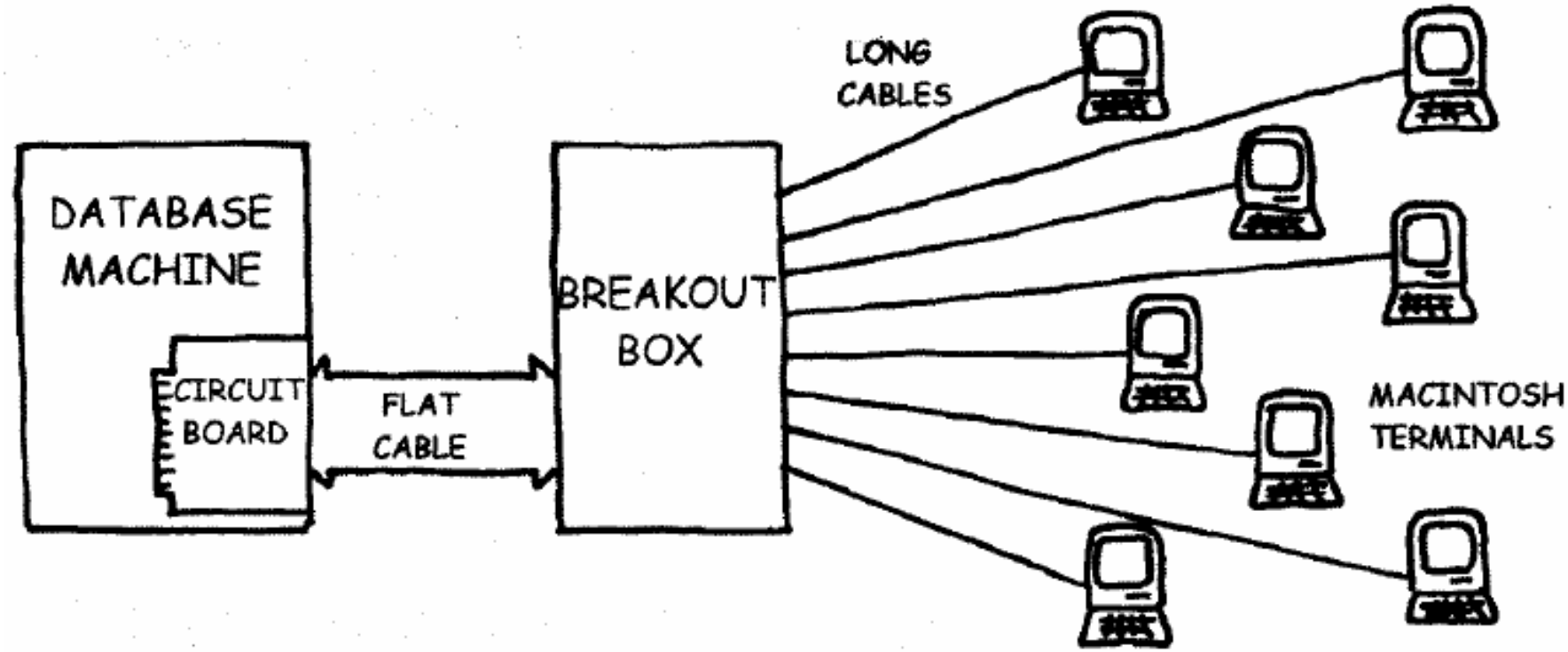
Rule 4: Divide and conquer

- "How often have I said to you that
  when you have eliminated the impossible,
  whatever remains, however improbable,
  must be the truth?"

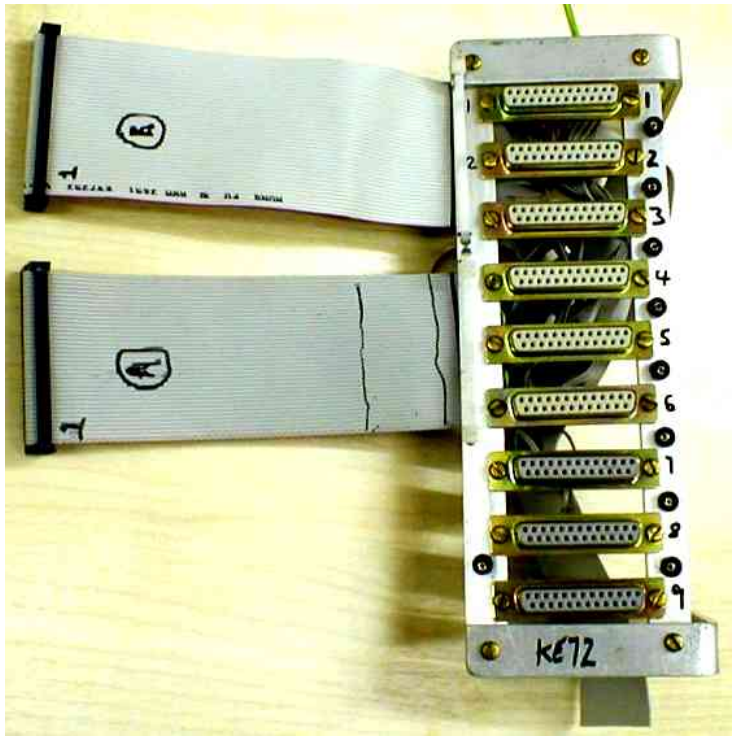
  Sherlock Holmes

Divide and conquer: War story

- Setting: A server (database machine) in a hotel, with 8 Macintosh computers attached as clients
  - Communication over serial cables
- Problem: Database retrieval had become very slow



Breakout boxes



Divide and conquer: War story (2)

Checks of the technician (in order):

- 1. Guess: There are data transmission problems.
  - Check found error messages in the communication log
- 2. Guess: No SW changes, so problem is probably HW.
  Guess: All terminals work, so it's probably not them
  - Oscilloscope check at server port
    found good signals going out to
    the terminals, and
    weak signals coming back
- 3. Checked halfway between there and the Macs:
  At the terminal sockets of the breakout box
  - Inverse situation: weak signals coming from server, and
    strong signals coming from the clients

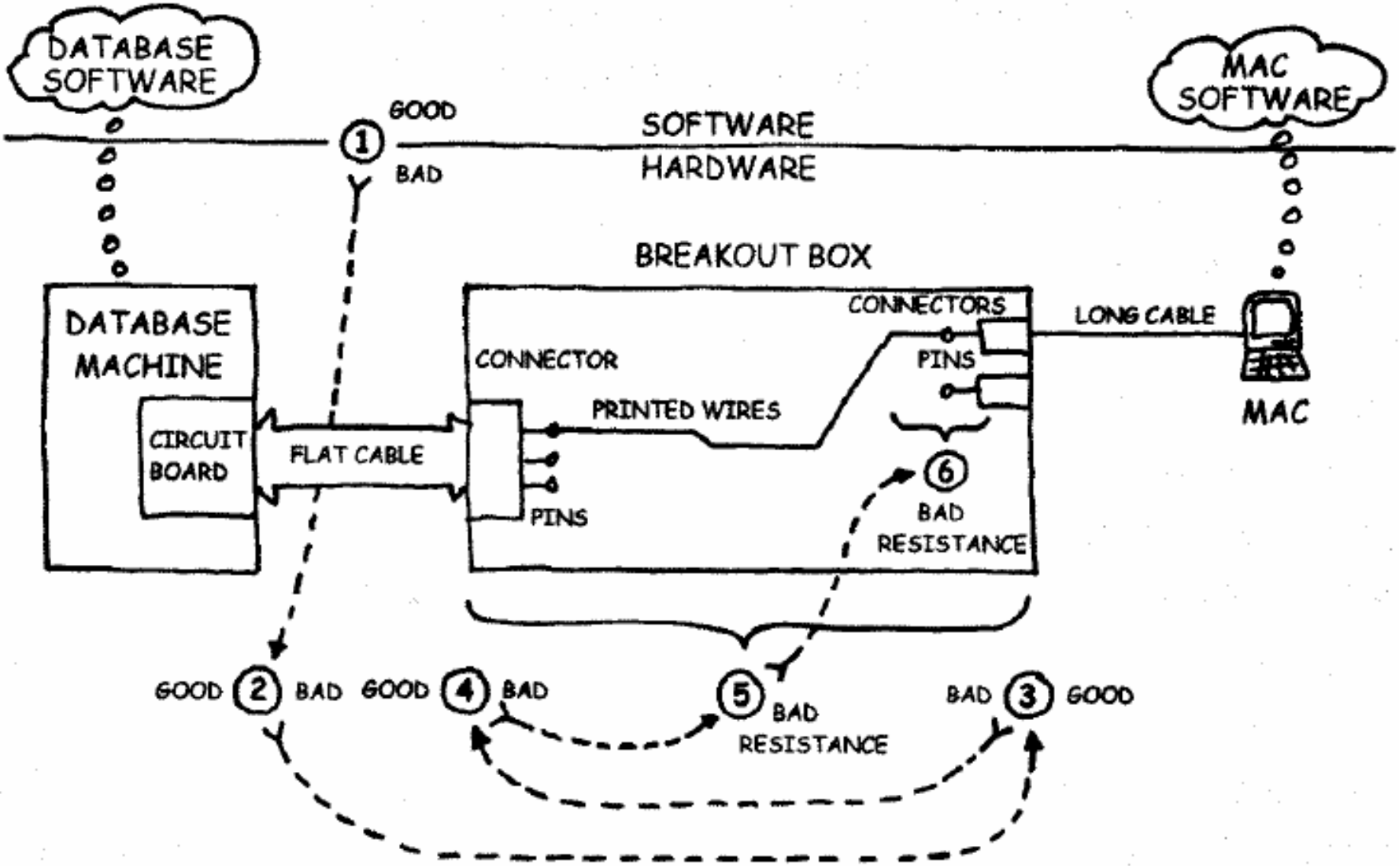
Divide and conquer: War story (3)

- 4. Checked end of flatband cable (input to breakout box)
  - Again inverse situation: strong signals coming from server, again and weak signals coming from the clients

- 5. Conclusion: The problem must be in the breakout box
  Check: Measure resistance of the breakout box
  Result: Resistance is much too high
  Consequence: Open the breakout box

- 6. Measure resistance between various points
  - Found hairline cracks where serial connector pins were soldered to circuit board

Divide and conquer: War story (4)



Divide and conquer: War story (5)

- Repair:
  - Disconnected all clients and the server
  - Reheated all spots with a soldering iron
  - Reconnected all clients and the server
  - Checked that the clients worked fast again
  - Only now re-assembled the breakout box:
    - Disconnect all clients and server
    - Reassemble and reinstall breakout box
    - Reconnect all clients and server

- (Subrule: Never reassemble more than absolutely necessary before checking that your fix works as intended)

Divide and conquer: The essence

- "Divide and conquer" is the central rule of debugging
  - All others are just auxiliary
- Think of it somewhat like binary search:
  Of the range of all possibilities, pick one half,
  check it, and then
  - if the defect is in it: cut it in half again
  - if not: <u>check</u> the other half!
    - if the defect is in it: cut it in half again
    - if not: think again what your range is and why
- Let's try: *I chose a number between 1 and 100.* Guess it.
- Debugging is successive approximation of the cause of a phenomenon
  - Unfortunately, the "range" is not often as obvious as in the breakout box example, which even continued:

Divide and conquer: War story (6)

- After the breakout box repair, all Macs were working fast again – except one (Number 8)
  - That also had been the slowest of all before
- 1. After another soldering attempt, the technician started over by looking at the communication log again
  - it now showed errors for outgoing data only
- 2. Opened the serial connector plug at the breakout box
  - the outgoing signal looked OK
- 3. Guess: The problem must be "downstream".
  Opened the connector at the client end of that cable.
  Found that one of the wires was not even connected.
  - Cable had 6 wires; only 4 had to be used
  - Blue had been connected instead of purple
    - 30 m of cable length induced enough electrical coupling

Divide and conquer: The essence (2)

Debugging as binary search for a cause:

- The rules' purpose is
  - to help understand what the range is
  - to help understand what useful halfs may be
  - to pick the more likely half
  - to simplify the checking
  - to make sure your check works
  - to help interpreting the check result
  - to help you find another range entirely in case you went wrong altogether (or have no idea at all)
  - etc.

- We will come back to this view later

Divide and conquer: War story revisited

Freie Universität Berlin

As neat and clean as the divide and conquer looks our technician used other rules as well:

- He understood the system
  - used debug logs, then zoomed in on the hardware part
- He quit thinking and looked
  - rather than starting to replace lots of hardware
  - (which probably would have failed miserably)
- He made the system fail frequently
  - In fact, the regular "I am still here"-traffic between clients and server did it for him
  - But he made sure he really saw the failure (by measuring resistance)

Divide and conquer: Subrules

- Subrule: Hit the right ballpark
  - Be careful when choosing your initial "range"
  - Wrong picks can be costly (cf. number guessing)

- Subrule: Inject easy-to-spot patterns
  - Often checking is difficult or unreliable, because the data is too complicated
  - Then simple, artificial data will often help
    - cf. 00 55 AA FF in the slave processor example
    - cf. waving in the motion estimation example
  - However, make sure you don't end up simulating rather than stimulating your problem: Make it fail!

Divide and conquer: upstream/downstream

- Typically, the "range" over which divide-and-conquer is applied is not geometric in nature, but chronological:
  - A sequence of events that turns bad at some point
  - e.g. the data flow in a program

- If at some point, the problem has manifested itself, that point is "bad" or "wrong"
  - Think of the event sequence as a stream
  - Then everything before the point is "upstream"
  - and everything after that point is "downstream"

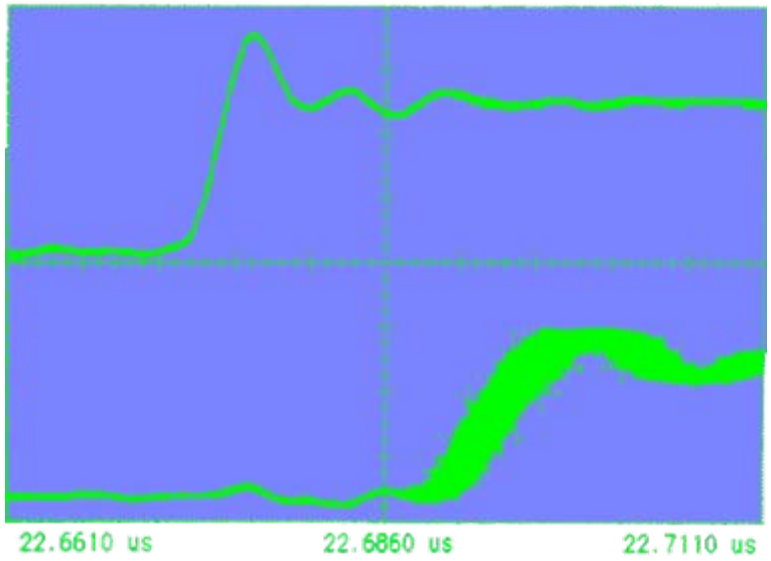
Divide and conquer: Subrules (2)

- Subrule: Start with the bad
  - If you see a problem, do not start at the source of the stream, working downstream
    - There are too many branching points and too many things that are NOT broken
  - Rather, work upstream from the first obvious symptom
    - As soon as you get past the problem, you will be close to a failure point

- Subrule: Fix any problem that you find
  - If you stumble across a second defect during debugging, you may think it "can never have anything to do with" your failure. You are probably wrong, so fix it anyway
  - Reason: It is extremely difficult to understand the behavior of interacting misbehaviors
  - This is a corollary of "Understand the system"

Divide and conquer: Subrules (3)

- Subrule: Fix the noise first
  - Certain typical kinds of mild misbehavior are very likely to produce other problems. Search for and fix them first.
    - Hardware examples: Signal noise, glitches, ringing, jitter, unstable power
    - Software examples: bad thread synchronization, uninitialized variables, dangling references, overflows etc.
  - However, make sure your corrections are cost-efficient
    - Repairing dozens of mild design defects can be a waste of time and may even introduce new defects



The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

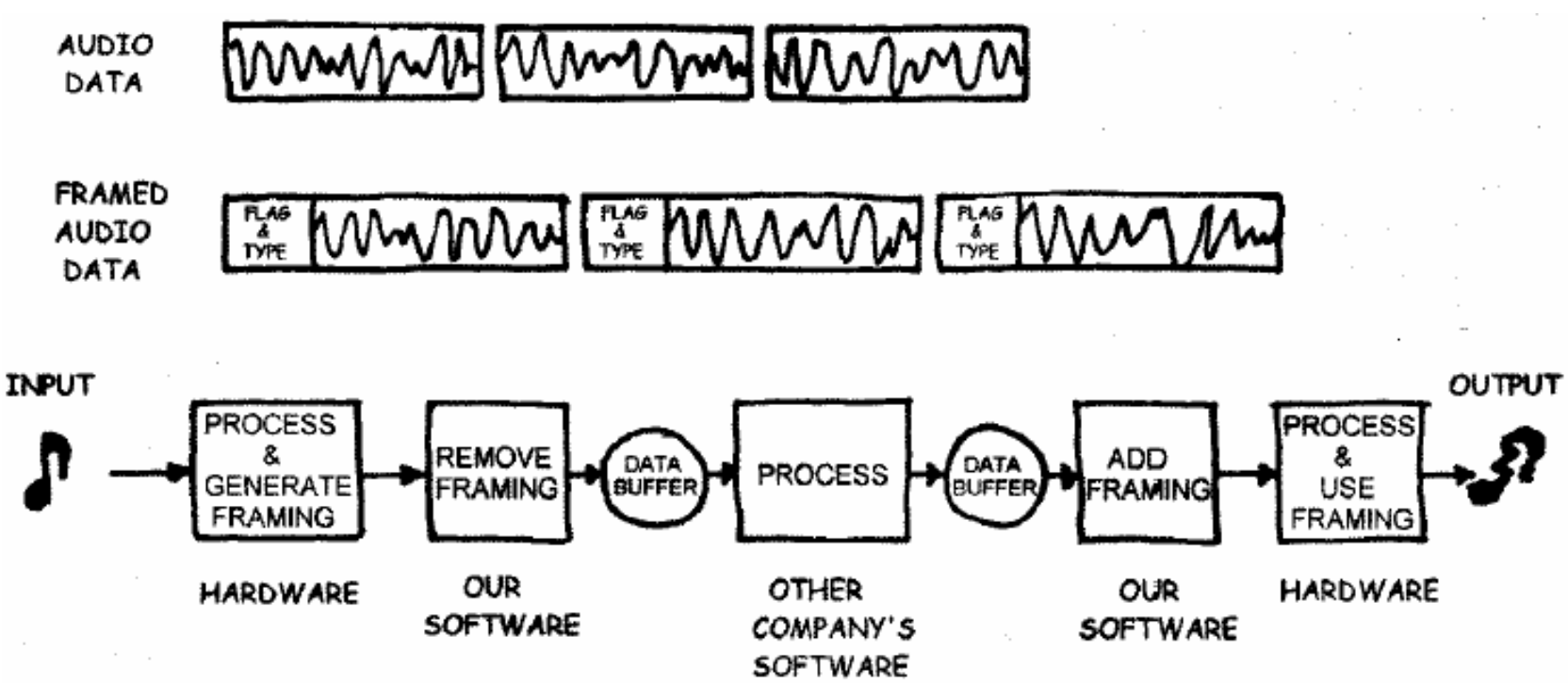
Rule 5: Change one thing at a time

- "They say that genius is an infinite capacity for taking pains.
  It's a very bad definition,
  but it does apply to detective work."

  Sherlock Holmes

Change one...: Audio war story

- A system handling audio data, involving special hardware, our own SW, third party SW, and a speaker.
  - Audio is processed in chunks
  - Sometimes these chunks are 'framed', sometimes raw
- The resulting audio sounded bad



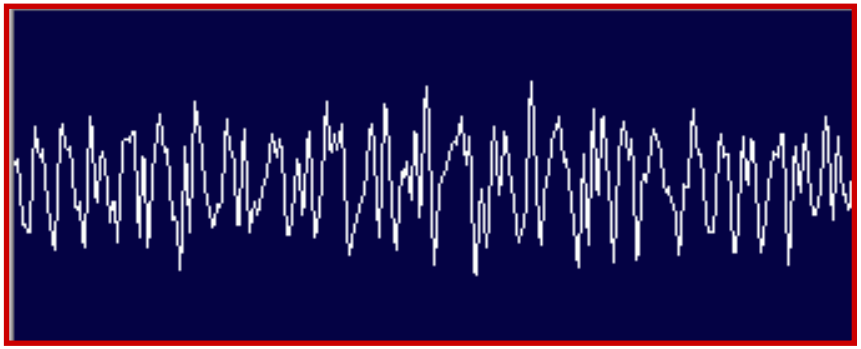
Change one…: Audio war story (2)

- 1. Guess: The engineer suspected that at one point in the process framing was missing
- 2. He added framing in the SW at that point
  - The audio still sounded bad
- 3. He could not think of other reasons and called in a debugging wizard (DW)
  - DW insisted they tried with simple, known data and instrument the SW
- 4. After some work, they could see the data getting clobbered and traced the cause to a pointer defect
  - They fixed the defect and found that the test data got through the defect spot unharmed
- 5. They tried real audio and it still sounded bad

Change one...: Audio war story (3)

- 6. They took an hour to reconfirm that their fix would really fix the problem

- 7. They went to reconfirm that their test system was really running the corrected version of the SW
  - At this point, the engineer recognized that he had not taken out his previous, useless 'fix' that inserted framing
  - After taking it back out, the system ran perfectly



Change one thing at a time: The essence

- "In debugging, always use a rifle, never a shotgun"
  - If you change multiple things at once, you learn little about the effect of each one

It may be still better to change nothing at all, until you really understand what is going on as good as possible:

- (See the following subrule)

Change one thing at a time: The essence (2)

- Subrule: Grab the brass bar with both hands!
  - According to legend, nuclear submarines have a horizontal brass bar in front of the power station control panel
  - The engineers are trained to grab that bar with both hands when any status alarms go off
  - They have to hold on until they have analyzed and understood all information presented on the panel
    - Overcome the urge to do something

Grab the brass bar: Fireplace war story

- At a party, the stereo was connected such that the cable to one speaker ran through the (always unused) fireplace
  - Sure enough, somebody ignited the fireplace
  - The isolation melted, the cable shorted, the fuse blew, the left speaker went dead
  - The organizers did not bother to analyze, but decided to check, whether the problem was the speaker or the amplifier:
    - They switched the left and right speaker cables at the amplifier
- Result: The rest of the party had no music at all

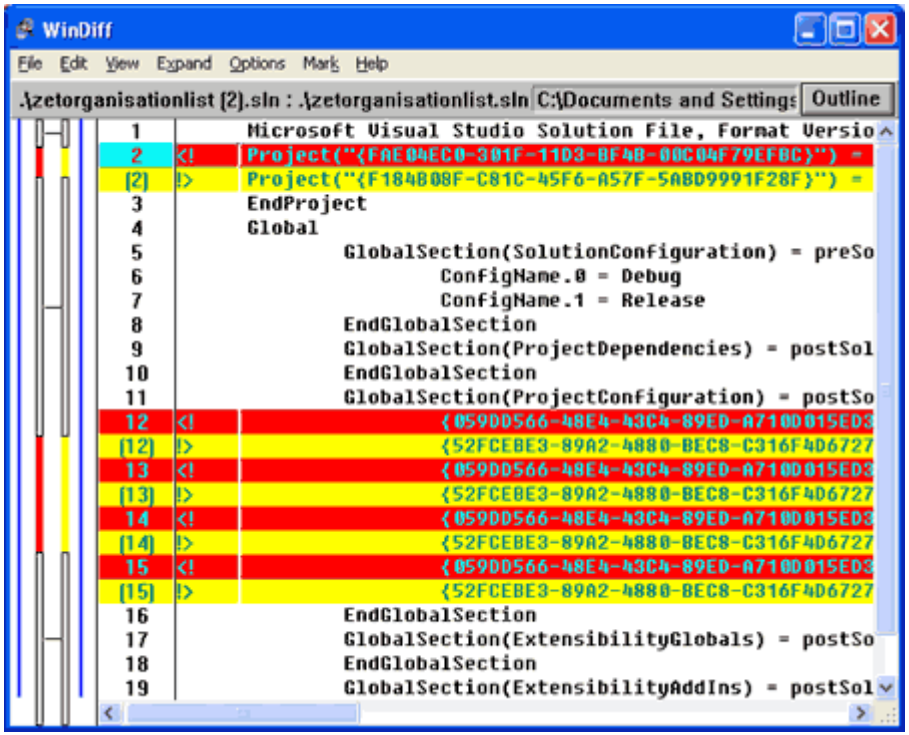
Change one thing at a time: Compare

Freie Universität Berlin

- Subrule: Compare with a good one
  - A very effective debugging method is often direct comparison of a "good" run and a "bad" run
    - Compare trace output, logs etc.
    - Use a differencing tool (after removing the timestamps)
  - The less you change between those two, the easier the comparison
    - The more instrumentation you have, the better the chances of the comparison
  - If your change is minimal, the differences may point directly to the cause
    - Ideally, you can use a 'diff' program to do the comparison
    - But even that will leave a lot of irrelevant data to look at

Change one thing at a time: Go back in time

- Subrule: What did you change since the last time it worked?
  - If a misbehavior occurs after a design change, it is useful to understand precisely what was changed:
    - Either one of the changes was wrong
    - or one exposed a defect that had lurked for some time
  - In SW, this is greatly aided by a revision control system
    - So check in frequently!
    - And use visual differencing tools
    - See also the next rule: Keep an audit trail



The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Rule 6: Keep an audit trail

- "There is no branch of detective science which is so important and so much neglected as the art of tracing footsteps."

  Sherlock Holmes

Keep an audit trail: Plaid shirt war story

Freie Universität Berlin

- Debugging a video compression chip used for video conferencing
  - Setup used live signals from a camera
  - Target data rate: 30 frames per second; usually possible
- Sometimes, the chip would slow down to 2 fps
  - and stay there until it was restarted
- The failure cause had nothing to do with uptime
  - sometimes it failed quickly, sometimes not for hours
- Once the chip did not fail a whole day
  - The tester considered different room temperature as the cause
  - Tried heating and cooling the chip – no effect

Keep an audit trail: Plaid shirt war story (2)

- Then suddenly the tester noted the chip failed just when the tester got up from his chair
- In fact this failure was repeatable
  - He sat back down, restarted the chip, got up: it failed
  - It also worked the other way round: restart while standing, then sitting down: it failed
- But then why had it not failed all day yesterday?
  - He had gotten up multiple times then, too
- What might be the actual technical cause?

Keep an audit trail: Plaid shirt war story (3)

Solution:

- The tester usually wore plaid flannel shirts
- Yesterday was an exception: plain blue formal shirt
- The chip gave up when it tried compressing a very complex signal (the moving plaid pattern)

Lesson:

- The seemingly insignificant **does** matter!
  - At least sometimes
  - But you never know when or what

Keep an audit trail: Everyday application

- When you have a food allergy, the doctor will make you protocol
  - all that you eat and drink (when, what, how much) and
  - the symptoms you get (when, what, how much)
- The food list alone or symptoms list alone are not very useful
- Even both lists together, but without the times, will be not very useful

- The audit trail must
  - be complete and detailed about all relevant events
  - and must correlate events

Keep an audit trail: Write it down

- Subrule: Write down what you did, in what order, and what happened
  - Or else your short-term memory will be overloaded
  - In your head, you cannot analyze for more than your current hypothesis or focus – much work will be lost

- Subrule: The shortest pencil is longer than the longest memory
  - Written audit trails can be copied,
  - attached to logs,
  - forwarded and shown to other people, and
  - reproduced weeks later when investigating something else

Keep an audit trail: Be specific

Subrule: Be specific!

- e.g. when a program crashes, do not just write down "crashed"
  - Did it produce a proper UI-level error message? Which?
  - If yes, did it terminate afterwards?
  - Did it stop with an exception message, stack trace, memory dump? Contents?
  - Did it freeze? In which observable state?
- if more than one machine is involved, always indicate which one you are talking about
- if a symptom has describable nature, describe it
  - e.g. size, intensity, color, duration, shape etc.

Keep an audit trail: Correlate

Subrule: Correlate events

- Often you have some information in a log and other information is directly observed
- Make sure you know where the observations fit into the log
  - Time stamps are a good way of doing this
  - Synchronize clocks as precisely as you can
  - In particular if multiple machines keep logs
- If you can't, instrument accordingly

Keep an audit trail: Belly war story

- A certain workstation would sometimes produce spurious input characters
  - Other machines running the same software worked OK
  - The hardware of the given machine had passed all tests and seemed to be in order
- Then somebody noted that the events correlated with work shifts
  - They occured only when Fred was on duty
- Solution:
  Fred had such a big stomach that it sometimes worked the keyboard when he reached for his coffee pot

Keep an audit trail: Audit trails of design

- Subrule: Create design audit trails
  - When a new problem occurs in a once-working product, understanding what was changed in between is crucial
  - It narrows your range for divide-and-conquer dramatically
  - Version control systems allow for such design trails
  - Better still: Configuration management systems
    - that also keep track of precisely which tools were used for building the system etc.

Keep an audit trail: WRITE IT DOWN!

- *"The horror of that moment"*, the king went on,
  "I shall never forget."
  "You will, though", the queen said,
  "if you don't make a memorandum of it".

  Lewis Carroll ("Through the looking glass")

The nine rules

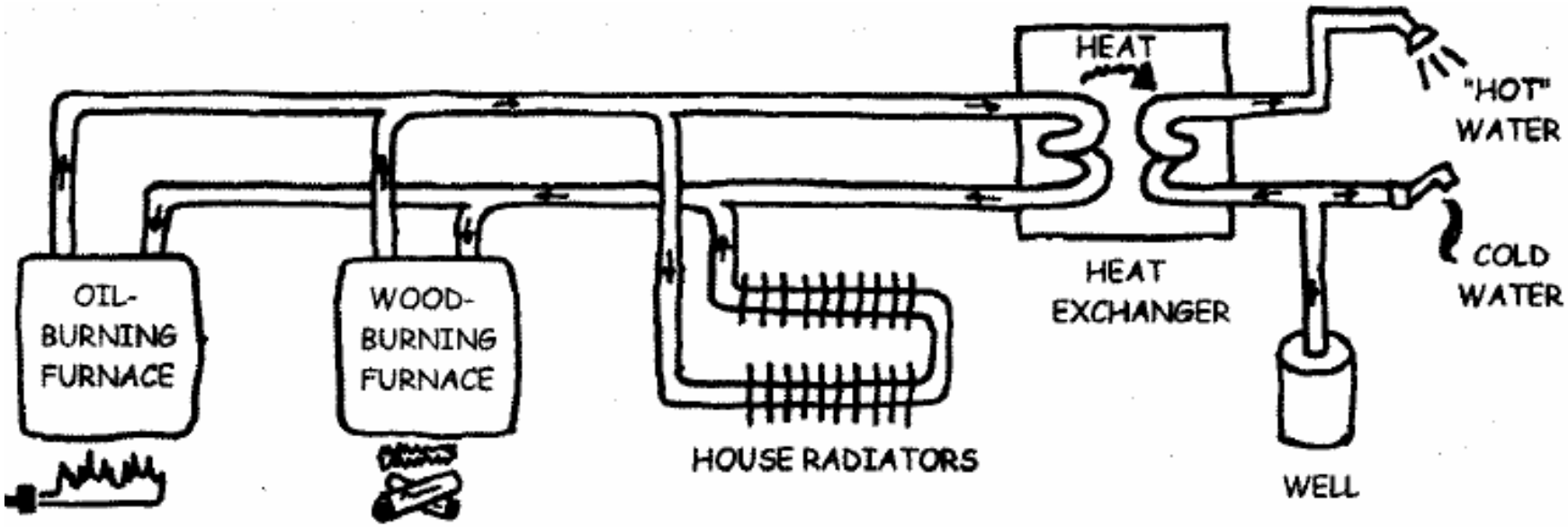
1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Rule 7: Check the plug

- "There is nothing more deceptive than an obvious fact."

  Sherlock Holmes

Check the plug: Old house war story



- 90 year-old house; most things were present twice
- Heating: Previous owner had added an oil furnace as a backup to the primary furnace heated with wood
- The new owner took the wood furnace out of service

Check the plug: Old house war story (2)

- Problem: When showering, the water would quickly turn cold

- Idea 1: Hot-water pressure drops
  - Possible solution: A pressure-balanced valve
  - But such a thing was already in place!
- Idea 2: Not enough hot water available
  - But the system was instantaneous: It cannot run out of hot water, as it heats it just-in-time
- Idea 3: Hot water production temperature not set hot enough (at heat exchanger)
  - But it was set to 60˚C, enough even for the dishwasher

- Solution: Oil furnace was set to only 74˚C rather than the required 88˚C
  - Reason: It had been meant as a backup only!

Check the plug: Old house war story (3)

Analysis:

- A wrong assumption was at work:
  That the furnace would produce enough heat
- Consequently, the divide-and-conquer approach started with a range that was too narrow
- The assumption was discovered only when the radiators also did not work well in autumn
  - thus pointing to a different kind of problem than previously considered

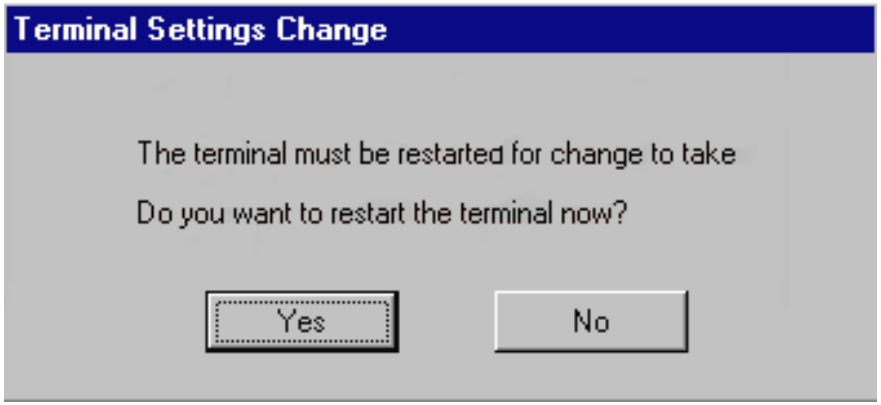
- This kind of "foundation factor" is particularly likely to be overlooked
  - We are often too deep into details to consider the basics

Check the plug: Typical things to check in SW

Freie Universität Berlin

If some service does not work:

- Is it installed?
- All parts of it?
  - driver, libraries, application
- Is it configured to be used?
- Is it turned on?
  - manual start/initialization?
- Do you need to reboot?
- etc.



If a change has no effect:

- Have you compiled the new version?
- Have you installed it?
- Are you running it?
  - Have you ever terminated the old version?
- Is there some old version still on the system?
  - CLASSPATH
- Do you need to reboot?

Check the plug: File I/O war story

- Benchmarking a new processor for reading and writing video files
  - needs to always achieve a certain speed
- A highly-paid consultant is hired to write the benchmark programs
- The programs are too slow – and slower than expected
  - Reading was even slower than writing
- The consultant carefully optimizes all parts of both programs for several weeks
  - But could never make reading as fast as writing
- He finally gave up
  - The others took over and found the following:

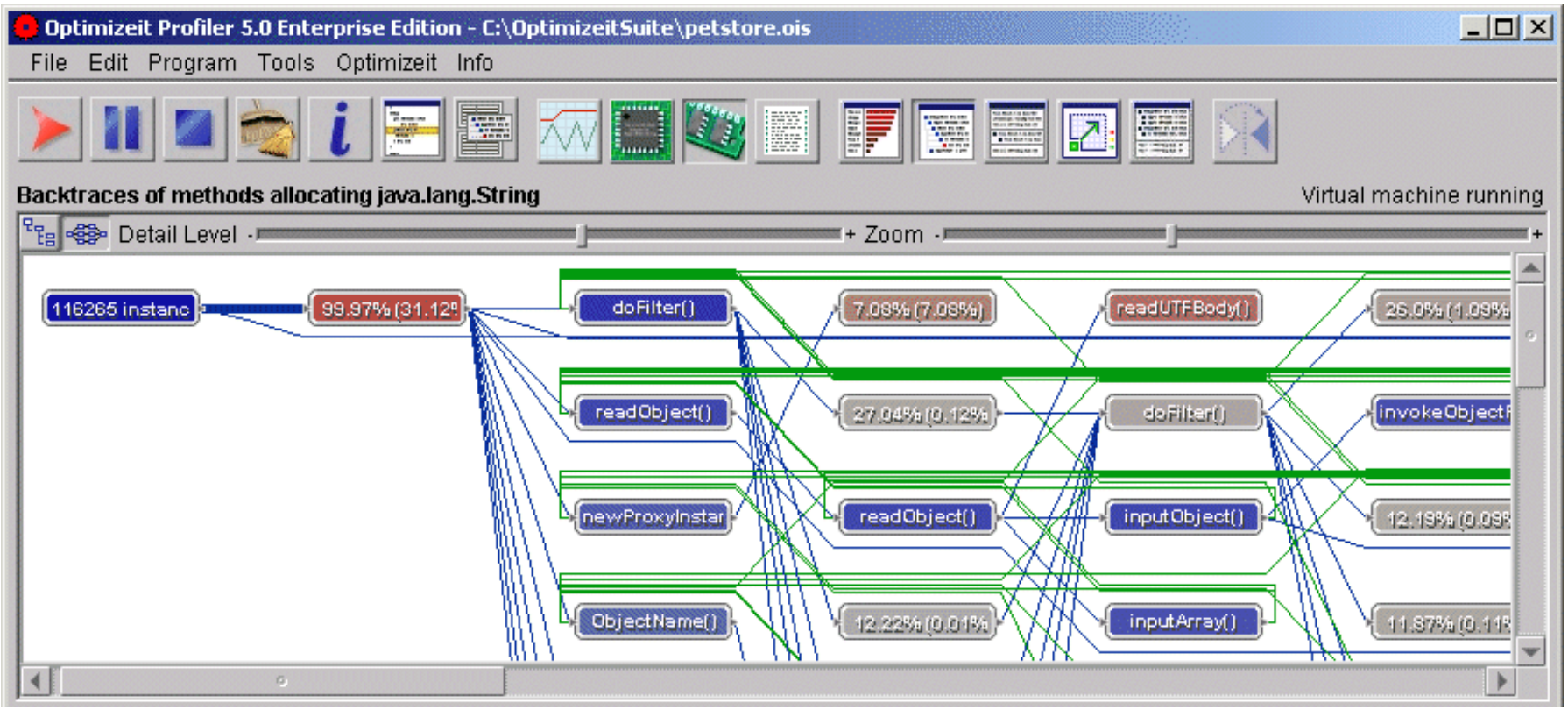
Check the plug: File I/O war story (2)

- The programs opened files without specifying file type
  - text or binary
- The consultant had assumed the default was *binary*, but indeed it was *text*
  - In text mode, writing would convert LF into CR LF and reading would convert CR LF into LF

- Such wrong assumptions about library behavior are also common
- as are wrong assumptions about what tools do
- Note:
  Libraries and tools may be faulty. But most often they are not.

Check the plug: File I/O war story (2)

- Note:
  In such cases of runtime performance 'defects', use a profiler tool to find out what is going on



The nine rules

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide and conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you don't fix it, it ain't fixed

Thank you!