Seven Deadly Sins of Debugging

Roger Orr

OR/2 Limited

www.howzatt.demon.co.uk

ACCU conference 2008

Debugging is the worst case

- The best bug is one that didn't happen.
- Learn and apply techniques to reduce problems up front. These include:
  - clear specifications
  - good design
  - defensive programming
  - static code analysis tools
  - unit testing
  - code reviews
  - pair programming

Debugging is the worst case

- The best bug is one that didn't happen.
- Learn and apply techniques to reduce problems up front. These include:
  - clear specifications
  - good design
  - defensive programming
  - static code analysis tools
  - unit testing
  - code reviews
  - pair programming
- *This* presentation isn't about these.

Debugging is a necessary evil

- "In this world nothing is certain except death and taxes"

Debugging is a necessary evil

- "In this world nothing is certain except death and taxes ... and bugs"

- Bugs “shouldn't” happen, but they do
- There seems to be no silver bullet – despite all our endeavours in preventing bugs some of them always seem to survive

Debugging is a necessary evil

- How well do we handle bugs?

- Better programmers can be 20 times faster
- at finding defects
  - Find higher percentage of defects
  - Spend less time on each defect
  - Introduce fewer fresh problems

- Why is there this differential ?

Debugging is a necessary evil

- There was only one Sherlock Holmes; but Dr. Watson picked up a few ideas about detection as the stories progressed.
- “Here is my lens. You know my methods.”

- What prevents us learning?

The human element

- There are technical challenges in debugging, but before we can solve them we must face our “deadly sins”:
  - Inattention
  - Pride
  - Naivety
  - Anger
  - Sloth
  - Blame
  - Vagueness

'You see, but you do not observe'

- **Inattention** is the first deadly sin of debugging.
  - We don't *see* the real symptoms
  - We *miss* the vital clue
  - We fail to spot *patterns*
  - We make the *same* mistakes again
- **Inattention** is almost entirely negative

'You see, but you do not observe'

- What are the real symptoms?
- Importance of details
  - Specific messages
  - Dates/times
  - Configuration
  - What else is going on
- It's hard to concentrate...
  - automate
  - checklists
  - breaks

'You see, but you do not observe'

- What is the pattern?
- The human brain is good at seeing patterns

'You see, but you do not observe'

- What is the pattern?
- The human brain is good at seeing patterns
- We can see patterns that aren't there ...
- and ignore observations that don't fit
- 18 28 18 28 ?? ??

'You see, but you do not observe'

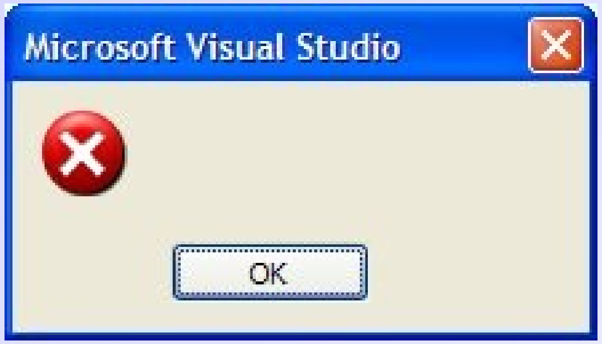
- A second pair of eyes can give a fresh view
- Find different ways to view the problem
  - Example: call method twice during optimising
- Explain the problem to someone else

'You see, but you do not observe'

- **Observation** is the first virtue of debugging.
  - What is our strategy for observing programs?
  - What tools are available, which ones do we use?
  - How can we make programs easy to observe?

“Observation is useless*”

- Sometimes observation is not enough



Microsoft Visual Studio

OK

- Perhaps, if you iron it, words appear?

* with apologies to Douglas Adams

'I used to be proud, but now I'm perfect'

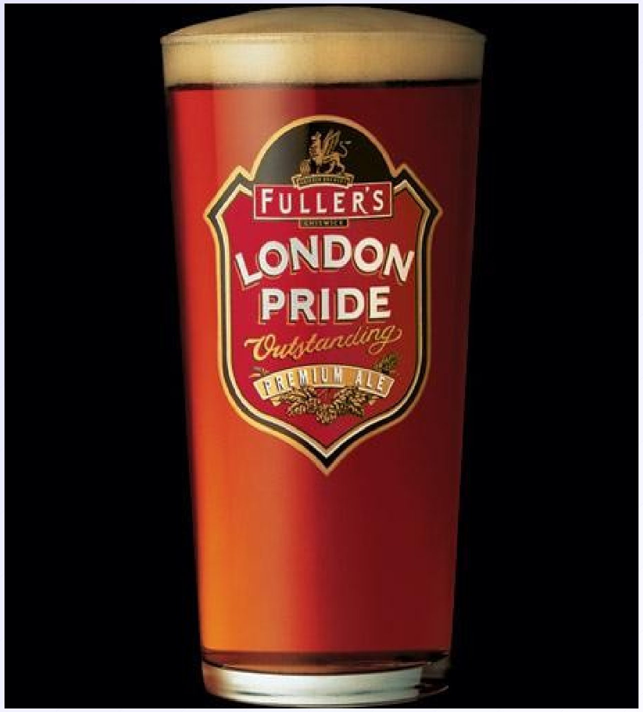
- **Pride** keeps us debugging longer than we should
  - Not *my* problem, so spend time looking at other, often less accessible, parts of the system
  - We don't ask for help
  - We don't look for, or don't use, the right tools to assist with debugging

'I used to be proud, but now I'm perfect'

- **Pride** *can* have a good side
- One of the 'Virtues of a Perl programmer'
  - “the quality that makes you write (and maintain) programs that other people won't want to say bad things about”
- Can help keep you motivated

'I used to be proud, but now I'm perfect'

- **Pride** *can* have a good side
- One of the 'Virtues of a Perl programmer'
  - “the quality that makes you write (and maintain) programs that other people won't want to say bad things about”
- Can help keep you motivated
- Or celebrate afterwards...



'I used to be proud, but now I'm perfect'

- **Pride** keeps us debugging when we ought to stop
  - We can keep on with a hunch rather than re-examining the evidence
  - We are sure we are going to fix the problem and don't look for alternative options

'I used to be proud, but now I'm perfect'

- Our pride may also have got us here in the first place!
  - Inappropriately clever code
  - Writing it yourself rather than using standard parts
  - Not prepared for your code to be wrong

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan, "The Elements of Programming Style", 2nd ed., chapter 2

'I used to be proud, but now I'm perfect'

- The corresponding virtue is **humility**
  - **I could be wrong**
  - Most real experts are humble: “I think the proper definition ought to be...” - sample posting from the C++ moderated newsgroup by Bjarne Stroustrup
  - What don't I know?
  - What options have I ignored?
  - Who can I ask – and how?

Naivety

- We don't seem to learn from our mistakes
- “Plan” for bugs
- Examine the problem domain, and the program
  - What is most likely to go wrong?
  - What are the potential bugs?
  - What information will I need to trouble shoot?
  - What is the cost of a fault?

Naivety

- Naivety *can* have a good side
- The simplest explanation for a bug is often the right one
- “YAGNI” - the code “you ain't gonna need” doesn't need debugging

Naivety

- Can be very dangerous when debugging
- “It is a capital offence to theorize in advance of the facts" - Adventure of the Second Stain.
- Need to collect *enough* information to ensure our hypotheses are verifiable
- Be prepared to try again

Naivety

- Bugs are not isolated
- Where else is this bug?
  - Do we just wait for another occurrence
  - Can we automate the search?
- How will I avoid this in the future?
  - Is the API badly designed?
  - Does the documentation need improving?
  - Can we automatically detect the bug?

Naivety

- The corresponding virtue is **wisdom**
  - **Knowledge** is knowing a tomato is a fruit
  - **Wisdom** is not putting it in the fruit salad

- A big part of wisdom is standing back from the immediate problem and reflecting

  For example:
  - Where have I seen these symptoms before?
  - Pro-actively, what is going to go wrong?

The red heat of debugging

- **Anger** (and other emotions) can cloud our judgement
- If we stop being calm we stop thinking clearly
- Denial of the truth – black and white rather than shades of grey
- Optimising code – you can always make it faster but when do we stop?
  - Emotion can keep us optimising well past the point of return
  - and even when the new code is actually *slower* than the old

The red heat of debugging

- **Emotion** can have a good side
  - It motivates and energises us
  - It can help involve others
  - Fixing the bug gives a 'buzz'

The red heat of debugging

- Getting personal: “I'm going to fix this bug if it's the last thing I do”
- If we *can't* fix this bug what is our plan B?
  - Often reluctant to consider alternatives
- Danger of believing *any* action is better than nothing
  - It may keep our manager (or customer) off our back

The red heat of debugging

- "Develop your plan to fit the facts, not the other way around. And, do not allow your judgement to be clouded by emotion!" - Sherlock Holmes (again)
- As with observation, we need to see the *actual facts* of the situation and not just
  - What we wish was true
  - What the observer claimed
  - What our prejudices expect (“It's that OS again”)

The red heat of debugging

- We overcome this fault with **patience**
- Take time to reflect, if necessary
  - It may be a slow process
  - Can formalise this: “I will spend one hour doing a direct attack in the debugger and then ...”
- Keep the bug in focus
  - Don't get sidetracked into changing things JIC
  - Don't make the bug consume too much effort

Oh well, it's only a bug

- **Sloth** means we try to avoid necessary work
- Debugging can be hard work - so we put it off
- Usually easier to find bugs up front – the hardest place is after the code is released
- Temptation to just go into the debugger and poke around rather than *think* about the bug

Oh well, it's only a bug

- Laziness: the first virtue of a Perl programmer
  - “The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labour-saving programs that other people will find useful, and document what you wrote so you don't have to answer so many questions about it.”
- Write useful tools and scripts
- Use available programs
- If I document how I solved the problem I avoid support calls
  - Google as a debugging tool

Oh well, it's only a bug

- Failure to complete the analysis, so we fix **a** bug not **the** bug
- Short term-ism – we fix the symptoms not the cause

Oh well, it's only a bug

- Sloth results in ignorance
  - Failure to read around the subject or fully understand the technology used
  - Little knowledge of, or interest in, what's behind the system
  - Lack of knowledge of how other people fix this sort of problem, and what tools exist to help us
- Ignorance makes it hard to debug

Oh well, it's only a bug

- The virtue is **diligence** (not overwork)
- Learn *enough* about the system
- Always approach the debugger with a question or a hypothesis
- Reflect on the work put in – what would have reduced my effort?

A bad workman blames...

- Easy to **blame** – and lots of candidates
  - Other team members
  - Management
  - Users
  - Testers
  - Any third party component you include
  - Compilers
  - Operating systems
  - Hardware
  - Cosmic rays

A bad workman blames...

- Sometimes blame is good: it isn't your fault and nothing you can do can change things

A bad workman blames...

- Blame doesn't fix the problem, but may lose you some allies
  - Poor motivator: the other people just want to deflect the blame rather than fix the problem
- Other systems probably *do* have bugs but am I likely to be the only person to find them?
- Even if you can blame another system, you *still* have a bug to fix

A bad workman blames...

- How do the other people in the project and the company handle blame?
  - Project managers
  - QA/test teams
  - End users
- Do what you can to remove a blame culture from the team, or people try to cover up bugs rather than fixing them.
- Eg: If everyone is shouting “my bit's working” then no-one is looking for what is actually broken!

A bad workman blames...

- **Responsibility** is the antidote to blame.
- Be prepared to admit you got something wrong (or *may* have got something wrong)
- Look at *why* you got it wrong
- Think about the way you use the tools and technologies you have: are you increasing or reducing the chance of discovering bugs in other people's code?

Something's broken again

- **Vagueness** is fatal to effective fault finding
- What *exactly* is the bug
- Related to observation – need for capturing precisely the right pieces of data
- Fixing 'a' bug not 'the' bug
- Experiment – report an unspecified bug in a piece of code and see how long it takes people to find it

Something's broken again

- Focus on what you're doing
- Avoid fixing other things while we're there
- Don't optimise broken code
- When generating error messages ask whether they contain enough (any) useful information
  - Especially under the actual target configuration

Something's broken again

- Precision greatly improves the bug hunting experience
- Contrast “The new release is broken”
- With “When trying to open file *name*, I get this error message (*text*)”
  - Don't forget sin #1 (inattention)

First the bad news

- Debugging is hard, and doesn't seem to be going away.
  - More distributed systems
  - Increasing mix of technologies and languages
  - Increased security (eg user access controls, firewalls, audit requirements, DRM protection)
  - Larger number of dependencies
  - Decreased time to market

Then the good news

- *Genius* debuggers may be born, but great debuggers are **made**
- It's not a black art, and how we approach it can both reduce the pain of doing it and greatly improve the degree of success

Deadly Sins or Cardinal Virtues?

• Inattention	Observation
• Pride	Humility
• Naivety	Wisdom
• Anger	Patience
• Sloth	Diligence
• Blame	Responsibility
• Vagueness	Precision