The Art of Design and Development

Debugging

What’s the big deal on bugs?

- ‘Programmers spend 50.1% of their work time not programming. Half of their time spent programming is spent debugging.’
- Majority of the time is spent on making code work and fixing bugs
- We knew it!
  - ***Developer/someone had seen the bug (maybe during development) but let go!***
- Many companies acknowledge that there’ll be bugs but often have inadequate instrumentation to deal with them esp. on the field
- Fixing the mindset first!

Strategies

- System needs to have a defined way to use debugging interfaces
  - Possibly different for developers, QA and operations folks each
- Experienced programmers often
  - Place self checking code (e.g. assertions)
  - Log - sometimes these logs can be enabled/disabled
  - Use debuggers judiciously
    - Needs good understanding of the debugger itself
    - Good when the scope has been zeroed down on
      - Asking wrong question gets you the right answer but takes you in the wrong direction
    - Some systems may not have a debugger
    - Normally difficult to use in production
    - Better to avoid using it :)

Strategies

- Look at
  - Program output carefully (don’t ignore ‘stray’ error/warning messages including those from compiler)
  - The stack trace, esp values of variables if available in your function calls leading to the call causing the crash
  - Backward reasoning
  - Familiar patterns
  - Code additions/modifications made immediately prior
- Track similar sections in your code elsewhere
  - Same mistake could have been made in other places
- Not ignoring bugs
  - Esp. when spotted first time
    - *Most* if not all bugs have been spotted by some developer/QA/ops engineer
  - Bug negotiations between QA and dev teams can result in customer reporting them eventually

Strategies

- Carefully review the code before modifying, else
  - Powerful urge to get to the keyboard & see if making ‘some’ modification to get the program to work. You may end up introducing another bug this way!
  - Print out critical sections (not the full code) to review
  - Seeing if it’s really the right fix
    - Take a break and then come back to it, and then review your problem & your fix
  - Draw/Write it out on the board
- Avoid relying totally on ‘help’ websites/opinions in forums
  - Expert opinion on some forums could be right but it may not apply to your situation 1-1
  - Do your own research and verify
  - Better to learn fundamentals *first* from a sound source
- Explain your problem code to others
  - It causes explaining the bug to yourself better and you might find the fix just as you converse
  - If no human being is available, explaining to a tree or teddy bear also works!
- Peer reviews

Tough bugs

- Esp. Unreproducible bugs
  - Really rare in reality
  - Most of this category of bugs are eventually found reproducible
  - *Do more than being patient*
- The Many causes
  - Version changes of dependencies
    - Compilers
    - Libraries
    - Inputs : sources of data : local inputs, network
    - Operating System/Environment/other conditions
  - Production environment different than development/testing
  - Regression - unintended consequence of fixing some other bug
  - Some sets of conditions make it happen more often
  - Timing issues, Wrong assumptions, Race conditions...
- Bug reports are incomplete/inaccurate
  - without compiler, OS, hardware, software, library versions as applicable

Tough bugs - some ways to deal

- Track system where it's happening
  - Use desktop recording software if it can help
  - Record command/event history
  - Log all events
    - Tricky timing related issues sometimes tend to go away in different conditions
- Scope input range
- Pattern identification
  - E.g. Are there numeric patterns in bugs?
    - E.g Source input to output 1-1 - occasional byte missing
- Increased logging & examining display output
- Adding assertions, code checkpoints
  - Possibly checking before and after certain operation
- Using pattern matching tools for large outputs/logs
- Flush buffers

Tough bugs - some ways to deal

- Initialization
  - Check for seeding via random numbers for some variables
  - Check for Uninitialized variables: ensure all variables are initialized
  - Results from using third-party libraries, user inputs
- Memory & Pointers
  - Writing more bytes to a memory location than allocated
    - May not always crash
  - Dangling pointers
  - Write your own allocation and freeing functions if applicable
- Analysis tools can help
- Use diff
  - Outputs of clean run and error run (consider keeping project-wide scripts to purge out timestamp etc. which will cause diff to flag time differences etc.)
  - Source code, check what's been checked in across versions of suspect files

Tough bugs - some more..

- Conversion/Typecasting
  - Conversion from one type to another esp when values mean different things
    - E.g. conversions like unsigned integer <> signed integer has resulted in disasters
- Packing
  - Padding (usually done by the compiler) adds extra bytes to the data structure
    - Every allocation needs that many more bytes
    - Size calculations go wrong esp. when these are not considered
    - Performance consideration but easier to fix when at work

```
struct message {
    text1 bool
    messageCode1 int16
    text2 bool
    messageCode2 int16….
    mC1 int32
}
```

```
struct message {
    mC1 int32
    messageCode1 int16
    messageCode2 int16
    text2 bool
    text1 bool
}
```

Tough bugs - some ways to deal

- Solid testing
  - Automated testing
    - Unit
    - Integration
    - System
    - Get good code coverage and handle wide range of inputs/environments using simulation if necessary
  - Use Virtual machines/Containers to test in different environments/combinations
  - Sanity testing before checking in to make sure critical functionality is never broken
  - Don’t ignore garbage messages
- Microservices ?
- Make sure bug reports are complete and well understood
  - Can result in wild goose chase