Everything about bugs

Wei Le

January 18, 2021

Outline

- Why should we care about bugs?
- What do we want to know about bugs?
- When and how shall we find bugs?

Why should we care about bugs?

- software bugs cost 1.7 $trillion in financial loss in 2017 [1]
- impact billions of people
- consequential bugs
- bugs are unavoidable: e.g., 391 commits of bugs, 287 commits of other stuffs in one of our studies.

[1] https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/

Top software failure recently [2]

- February 2020: Heathrow disruption: cannot check in e-tickets
- Bitcoin unlimited nodes fall from 800 to 300: memory leak
- Nest thermostat leaves users in the cold: battery drain
- Starbucks's software bugs: 60% stores in US and Canada have to be closed, give coffee away for free as they cannot process payment transactions
- Co-op charges customers twice
- March 2015: F-35 detects targets incorrectly
- 2014, 2015 Toyota recall its cars: bugs on engine control unit, overheat transistors, can cause sudden stops while driving
- Emergery calls go offline for 6 hours

[2] https://www.computerworld.com/article/3412197/top-software-failures-in-recent-history.html

Top software fialure in history

- Therac-25 (radiation therapy machine) had $\geq$6 incidents between 1985-1987 and gave patients radiation doses that were hundreds of times greater than normal, resulting in death (in 3 cases) or serious injury
- written in assembly language
- had both design problems and coding problems including race conditions, arithmetic overflow
- more on wiki page "Therac-25"
- loss of rockets and satellites: NASA Mariner 1 destruction (1962), Airane 5 flight 501 destroyed (1996), Mars climate orbitor

Fun bugs [3]

Impossible Bugs

Many bugs are confusing. Others are are annoying. Some are just impossible. This is a list of those bugs:

1. MRI disabled every iOS device in facility
2. We can't send mail more than 500 miles
3. OpenOffice.org can't print on Tuesday (see comment 28)
4. I can't log in when I stand up. (and another similar story)
5. A story about "magic"
6. Print this file, your printer will jam
7. gcj crashes in April and December, but only if you speak German in Austria
8. Processor 5 doesn't work if you're standing too close
9. A car that is allergic to vanilla ice cream
10. Some employees change the monitor's resolution without touching it.
11. The computer is filled with bees
12. My chair turns off my monitor (via tweet)

[3] https://blog.eitanadler.com/2018/10/impossible-bugs.html

What do we want to know about bugs?

find.66c536bb bug [4]

find -mtime allows to find file according to their age. However, this version interprets -mtime -n and +n in the same way. But it should interpret find -mtime -n as finding files that are *strictly less* than n days old and find -mtime +n as finding files that are *strictly more* than n days old.

```
$ mkdir tmp
$ touch -t 202101251600 tmp/a //today
$ touch -t 202101241600 tmp/b //yesterday
$ touch -t 202101231600 tmp/c //2 days ago
$ touch -t 202101221600 tmp/c //3 days ago
```

If we run the following, we would expect this output

```
$ ./find tmp -mtime -2
tmp
tmp/a
tmp/b
```

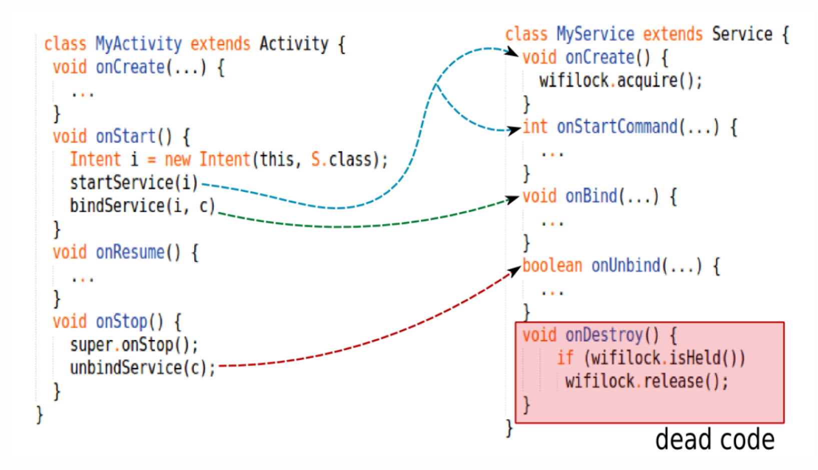
However, I actually get this output:

```
$ ./find tmp -mtime -2
tmp/c
```

Results are the same if I replace -n with +n, or just n.

[4] https://dbgbench.github.io/find.66c536bb.report.txt

Android bug: causing battery drain



Android bug: causing battery drain

```
class HostListActivity extends Activity {
  public void onStart() {
    this.startService(new Intent(this,
      TrackingRecordingService.class));
    this.bindService(new Intent(this,
      TrackingRecordingService.class),...);
  }
  public void onStop() {
    super.onStop();
    this.unbindService(connection);
  }
}
class TrackingRecordingService extends Service {
  public void onCreate() {
    wifilock.acquire();
  }
  public boolean onUnbind(Intent intent) {
+   if (bridges.size() == 0) this.stopSelf();//patch
+   return true;
 }
  public void onDestroy() {
    if (wifilock != null && wifilock.isHeld())
       wifilock.release();
  }
}
```

What do we want to know about the bugs?

- *bug*: mistakes in software (code, configuration, makefile)
- *fault*: violation of *program property* – conditions hold for all program paths, e.g., *assertion*
- *failure*: dynamic symptoms - crash, incorrect results ... the crash stacks and memory states at the crashes can be captured and reported for postmortem analysis
- *vulneraiblity*: is it exploitable? what is the threat model?
- *root cause*: what is the mistake and how it is propagated and lead faults and failures
- *failure-inducing input*: inputs that can trigger the bug
- *reproduce steps and environment*: how to reproduce the bugs: in addition to inputs, we sometimes also need to know the libraries and system setups
- *patch, program fix*: the modification of code that ensures correct executions

Types of bugs

Coding errors in C/C++/Java /Javascript

1. buffer overflow, integer overflow, null-pointer dereference, double free, dangling pointers – memory bugs
2. deadlock, race conditions – concurrent bugs
3. memory leak, lock/unlock mismatch, file open/close mismatch – resource leaks, typestate violations, source-sink problems
4. program specific, functionality issues

Bug in special types of software: *active research areas*

1. finding bugs in compilers, virtual machine software
2. finding bugs in machine learning software
3. finding bugs in UAVs

When and how should we find bugs?

Bugs and software development lifecycle

The early we find bugs, the cheaper to diagnose and fix bugs. According to the search at IBM [5]:

[5] https://www.celerity.com/the-true-cost-of-a-software-bug

Current approaches for finding bugs

There are both static (don't need to run the program) and dynamic approaches (need to run the program) to find bugs

- *Static analysis* aims to predict such conditions by analyzing the source code.
- *Testing* aims to find such conditions by exercising representative inputs.
- *Dynamic analysis* collects the run time information to determine if a bug has been triggered. Dynamic tools can combine with static analysis tools and testing.
- *Model checking* builds models of software and check it against given properties.

Current approaches for finding bugs

- *Code review*, *code inspection* finds bugs manually to confirm static warnings, to diagnose a failure. The big company like Microsoft prepare a very user-friendly GUI for code inspection.
- A typical process: static analysis/code inspection on the desktop $\rightarrow$ git commit and push to the clouds $\rightarrow$ unit testing / whole program static analysis /team code inspection/ integration testing $\rightarrow$ fuzzing
- *Continous integration*: nightly analyzing and testing changes, providing timely feedback to the developers

Automatic bug finding tools

Software companies such as Google, Microsoft, Facebook has deployed automatic tools; there are also bug finding tool companies:

- Static analysis tools: GammarTech (CodeSonar), Coverity, Fortify, PolySpace, ESP, ESPx, Infer (open source), Findbugs
- Testing tools: AFL (fuzzing tools), SAGE (dynamic symbolic execution)
- Dynamic analysis tools: Purify, Valgrind, AppVerifier
- Model checking tool: SLAM (finding bugs in drivers that can cause blue screens)
- Implementing an algorithm using a week, but building a tool that can handle real-world software will need to handle many engineering challenges

Automatic bug finding tools

Name	Language	Type of Tool	Note
Findbugs	Java	Static analysis	open source, UMD/Google
American Fuzzy Lop	C/C++	Fuzzer	open source
Prefix, Prefast	C/C++	Static analysis	Microsoft
ESP	C/C++	Static analysis	Microsoft
KLEE	C	Static + Dynamic	open source
Infer	Java, C/C++, Objective C	Static analysis	open source, facebook
CodeSonar	C/C++	Static analysis	UW/GrammarTech
Coverity	C, ..	Static analysis	Standford/Synopsys
Valgrind	C/C++	Dynamic analysis	open source
Atlas	C/C++/Java	Static analysis	Iowa State/EnSoft

Behind the scene: the key ideas

- finding bugs: a software engineering problem
- does the program contain an "erroneous/undesired" state?: a program analysis problem
- problem reduction: we need to define what is the "erroneous/undesired", i.e., we need to know the "specification of a bug" in order to detect bugs
  - Statically: a program state is the values of a set of variables at a program point; the set of incorrect values form an erroneous/undesired state. Static analysis tools use Common Weakness Enumeration (CWE)
  - Dynamically: crash, hang, incorrect output

Behind the scene: the key ideas

- Challenge: software consists of a very large state space, we cannot enumerate all the states
- Static analysis: Abstraction
- Testing and dynamic analysis: Sampling
- Prove the correctness of the software with regard to a given property: precondition, postcondition and logic inferences