Bugs and Defects

JD Kilgallin

CPSC:480

10/26/22

Photo: Edith Windsor, New York Times
Known for: 1950s IBM engineer achieving highest technical role, owner of first PC in New York City, Time person of the year runner-up, lead plaintiff in 2013 supreme court case that overturned federal ban on same-sex marriage

Bugs and Defects

JD Kilgallin

CPSC:480

10/26/22

Pressman Ch 15-16

Learning objectives

- Software defect concepts
- Categorizing bugs
- Software defect lifecycle
- Tracking bugs
- Root Cause Analysis

What is a bug?

- "A flaw, failure, error or fault in computer software or system **programming** that causes it to produce unexpected or incorrect output."
- "An incorrect **implementation** of a valid requirement and design."
- "The location of the programming error causing the incorrect behavior".
- Bugs, glitches, defects, faults, flaws, errors, mistakes, and failures **may** be used interchangeably.
- Pressman describes an *error* as an implementation mistake caught before release, and a *defect* as one caught after release.
- A *crash* is a fatal bug that causes the software to stop running.

What is not a bug?

- Not all software *defects* (i.e. unexpected or undesired results of program operation) are *bugs* (a result of erroneous programming).
- Issues arising at runtime due to improper or ambiguous requirements, inadequate design, or poor architectural decisions.
- Software misconfiguration, communication or hardware issues, etc.
- Compile-time syntax errors are not usually bugs per se.
- Functionality that has not yet been implemented is not a bug, although the line can be blurred.
- An *exception* is a condition that requires a different program flow than the default or standard path; may not be a bug or defect at all.

Additional Terminology

- Severity – The degree to which a bug prevents the software from meeting user requirements.
- Priority – The degree to which a bug impacts the vendor's business.
- Root cause – The most basic factor that leads to the incorrect output.
- Regression – A bug introduced to functionality that previously worked
- Edge case – A scenario where the input is at the extreme end of its range or is otherwise rare, but requires special handling. More likely to trigger bugs.
- Corner case – The intersection of two or more edge cases, or where two or more pieces of input data are at extremes.

Impact of bugs

- Unhappy users (refunds and lawsuits in some cases)
- Distraction from other development tasks
- Loss of reputation and trust in the product or business
- Lost revenue (e.g. income from ads if facebook goes down) or increased operating expenses (e.g. excessive cloud resource usage from a memory leak)
- Estimated 0.6% of GDP lost to impact of bugs!
- Opportunity for malicious actors to access other systems or data
- Real world harm, including death

Famous bugs

- Grace Hopper found an *actual* moth in a Harvard Computer in 1947. Contrary to some stories, this did not originate the term.
- Y2K – Sensationalized issue where some software products encountered issues around the year 2000 due to storing only two digits for the year, when "must work for 50 years" was not a requirement originally considered.
- Mars Climate Orbiter – NASA mission mixed English and metric units resulting in invalid computations that crashed the orbiter into Mars.
- Ariene 5 – Rocket's angle exceeded software design parameters, causing a cascade of failures destroying the mission.
- Gangnam Style got negative 2 billion Youtube views after overflowing a 32-bit signed integer.
- Apache server bugs caused Sony PlayStation Network and Equifax breaches (improper validation of XML before deserialization and processing)

How are bugs found?

- Some potential bugs manifest when the developer is testing a change-in-progress. The developer just finds and fixes the issue before retesting and checking in, and it doesn't become a bug.
- Caught by manual or automated testing before release.
- Caught by technical writers *documenting* the software.
- Caught by other stakeholders in alpha or beta testing.
- Caught by sales or operations staff deploying the software.
- Caught by other developers reading the source code while making other changes to related code.
- Caught by external code review, static analysis, or security research.
- Caught by end users.
- Caught by malicious actors exploiting the bug (Zero Day).

Classifying bugs by nature

- Functional – The software does not perform the intended calculation or operation correctly. For example, an "add" function that returns "5" for add(2, 2).
- Performance – The software performs the intended operation with unreasonable delay or resource usage. For example, the program takes 10 minutes and 1 GB RAM to say add(2,2) == 4.
- Usability – Input/output does not cause the intended operation to occur. Users may still be able to perform the desired action another way. e.g. clicking "add" button doesn't do anything but API works.
- Security – The software performs an operation that it was not properly authorized to perform. For example, add("SELECT * from BankAccountNumbers",0) returns database records.

Classifying by root cause

- Logic error – The programmer wrote code that the computer interprets differently than the programmer intended.
- Resource error – An attempt to access a resource (typically memory) without meeting the proper preconditions.
- Concurrency error – The program depends on an invalid assumption about order of execution in a parallel or distributed program.
- Input validation error – The program does not handle input outside of the expected range.
- Interface error – Communication between two (or more) modules does not occur according to one of the module's expectations.
- Improper merge – Code is checked in that does not reflect both developers' intentions.
- Documentation mismatch – Programmer relies on incorrect or outdated guidance from program documentation or code comments.

Logic errors

- The programmer wrote code that the computer interprets differently than the programmer intended.
- Most common cause of software bugs is human error of this form.
- May occur due to poor code quality, subtleties of a programming language or framework, and many other reasons.
- The best way to catch these bugs is code reviews and testing.
- Some examples include
  - Arithmetic: float average(float a, float b) { return a + b / 2; }
  - Off-by-one: for(int i = 0; i <= argc; i++) { cout << argv[I];}
  - Infinite loop: while(x->next != null) { k = x; n = x->next; process(k,n); }

Concurrency errors

- The program depends on an invalid assumption made by the developer about order of execution in a multi-threaded or distributed program.
- Each component in isolation may do exactly what the developer intended, but may still produce incorrect results as a whole.
- Hard to catch in review because concurrent reasoning is difficult, and hard to catch with tests since failures may not be consistently reproduceable.
- Deadlock – Two or more processes are both waiting for each other to finish before they resume execution.
- Race condition – A process produces correct output only when events happen in a certain order. Even if there's a 0.0000001% chance for the "bad" order to happen, a 3GHz=3*billion* ops/sec, giving a 95% chance of the event occurring.
- Thread starvation – One process may never get to finish its operation, or not in a reasonable amount of time.

Severity and Priority

- A high-severity bug that impacts a small number of non-paying users may be classified as low priority (e.g. doesn't work on old browsers).
- A low-severity bug that impacts a large number of important customers may be classified as high priority (e.g. a feature that simplifies management of a large number of records).
- A highly-visible low-severity bug may be classified as high priority (e.g. loading screen counts backward from 100% to 0%).
- A low-severity bug which could lead to significant reputational harm may be classified as high priority (e.g. unauthorized users are able to access a small amount of non-sensitive data).

Concepts Recap

- Bugs are software defects caused by errors in programming.
- Bugs can be serious issues causing significant harm.
- Bugs can be found at any point in the product lifecycle after the code has been written, but the sooner they're caught, the better.
- Bugs may impact product functionality, performance, usability, or security.
- There are many possible causes of bugs.
- Severity measures user impact, priority measures vendor impact.

Bug Lifecycle

- Bug fixes are software development and follow a lifecycle similar to development for new requirements.
- Bugs are reported, triaged, assigned, investigated, reproduced, & fixed
- Fixes must be verified and released, and test cases should be added to prevent regression.
- Bugs may not make it all the way through the lifecycle and end in another terminal state (possibly to be reopened in the future). Common reasons to close a bug report include:
  - "not a bug" or "working as intended"
  - "not reproduceable"
  - "can't fix" or "won't fix"
  - "need more information"
  - "duplicate"

Tracking bugs

- Product manager/owner is responsible for ensuring bugs follow the appropriate process, but other stakeholders need some insight into the status of a reported bug and potential fix.
- Bugs are usually captured in dedicated bug-tracking software, which may be coupled to version control/project management software.
- The internal representation of the bug is usually different than any publicly-visible issue tracking system (e.g. GitHub issues), so that it can be treated similarly to a backlog item that's in initial development.
- Bug reports can have extended comment threads with discussion among developers and stakeholders regarding nature/severity/priority.

Reporting bugs

- The first step in fixing a bug is bringing it to the awareness of the development team that can fix it.
- Reporting process will vary by product and who is reporting it:
  - QA engineers and other team members may file directly in internal bug tracking system used during the bug's lifecycle.
  - Staff from other departments like sales and support may report issues across departments with tickets in the support helpdesk or customer relations management software (Salesforce).
  - End users may find a public issue-tracking system to use, contact their representative, or use the published product support process.

Triage

- Product management and engineering management teams meet periodically (weekly) to review new bug reports from various systems.
- A new bug report is discussed so that everyone understands the report
- Intended resolution ("approved" or closed for another reason) is decided. May require a deeper system search for duplicate reports.
- Intended behavior, precise definition of the bug, severity and priority are established, and target date or product version for a fix.
- Intended resolution may be communicated to support/stakeholders.
- Time required for a fix to be designed, implemented, and tested is estimated.
- Bug is added to the backlog for assignment and fix in a later sprint.

Root Cause Analysis

- The first step a developer must take is to determine what sequence of actions or conditions are prerequisites to encounter the bug.
- Ideally, the bug report will describe exactly what leads to the bug and the developer is able to easily observe the behavior described.
- The developer must then identify where the defect is occurring; i.e. what code leads to the undesired behavior.
  - This is where a *debugger* is used; being able to examine internals of program state & data being processed is the single best tool for finding causes of bugs
- Potential causes are considered, the root cause is found, the desired fix is identified, and the erroneous code corrected. Related changes may be made to add data validation or exception handling so that similar errors do not cause the same behavior.

Fixing Bugs

- The fix is checked in and goes through the code review and QA process like other checkins.
- It is especially important to add automated tests that would have caught the bug, in order to prevent regressions.
- The fix may need to be applied to other versions of the software that have been released (a hotfix or patch), and the same changes must be made in that codebase.
- Fixing bugs is time-consuming and frustrating, and can be reduced with careful product architecture and design, code reviews, and extensive testing.

Vulnerability Disclosure

- Exception to the usual reporting and release process to keep a security issue hidden from potential exploiters until a fix is available.
- Major companies will have a confidential reporting tool or even a bug bounty for finding security issues.
- Frequently, the disclosure of the bug and impending fix needs to be communicated to select other parties, followed by the general public, in a coordinated fashion.
- "Common Vulnerabilities and Exposures" (CVEs) are records of known security issues maintained by MITRE and NIST (US Dept of Commerce)
- US-CERT provides a vulnerability database maintained by Carnegie Mellon and the US Cybersecurity and Infrastructure Security Agency.

"not a bug"/"working as intended"

- The bug's reporter misunderstands the intended functionality and the bug description actually matches the correct results intended for the product.
- The description is accurate but should be filed as another type of task
  - May really be a new requirement: "it doesn't do x" becomes "As a user, I want to do x".
  - May really be a design or architecture change request.
  - May be a documentation update task to change the description of what the software is supposed to do.

"not reproduceable"

- Testing of the software does not result in the behavior described.
- It is not usually possible to know or recreate the *exact* same program state that caused the bug, and it may not be clear what factors (e.g. "happened on a Tuesday") need to be accounted for in attempting to reproduce it.
- May require tools or techniques not available to the development team (especially integrating with other commercial software).
- May have been a transient issue caused by operating environment
- Reporter may be mistaken or lying about what they saw happen.
- May have been fixed by another change.

"can't fix"/"won't fix"

- Some bugs are outside of the development team's control (e.g. a bug in an external system).
- A bug may also require an inordinate amount of work to address, and it's easier to just live with the bug.
- The bug fix might cause more issues than it solves (e.g. breaks backwards compatibility); the cure is worse than the disease.
- A planned redesign of the functionality may render the report obsolete.

"need more information"

- Incomplete report or can't proceed as described.
- It may not be clear from the report:
  - What is being described
  - If it's a duplicate, or a different issue than a similar report
  - What the expected behavior is

"duplicate"

- The bug is already captured in another bug report.
- Reporter may not have access to prior bug reports, or may not check if it's already been reported.
- Two people may describe the same bug very differently, and the reporter is not able to find an existing report using different search terms.
- The reporter believes or hopes that filing a new report will lead to additional discussion.
- Bugs can be subtly related, and it may not be clear until deep investigation of the code that two issues have the same root cause.

Lifecycle recap

- Bug lifecycle includes report, triage, investigation, fix, and verification.
- Bugs are usually reported through software, but multiple systems may be needed to take reports from all types of stakeholder.
- Triage meetings decide on handling of a bug report.
- Fixes must be verified and released, and test cases should be added to prevent regression.
- Root cause analysis involves identifying the underlying code issue causing the bug and identifying the desired solution.
- Good design, code reviews, and automated tests help prevent bugs.

References

- Remembering Edie Windsor: Tech Pioneer, Equality Advocate. AnitaB.org. 2021.
- The 5 Most Infamous Software Bugs in History. Abdallah Aberouch. Nov 2015. OpenMind.
- How to Test Software for Different Types of Bugs. Victor Sachuk. June 2020. ScienceSoft.
- Bug Lifecycle in Software Development. Satyabrata Jena. Nov 2020. Geeks for Geeks.
- Root Cause Analysis: Important Steps. March 2017. Six Sigma.
- What is a CVE? Red Hat. Nov 2021.
- Vulnerability Note VU#971035. Ted Shorter et al. June 2012. Carnegie Mellon Software Engineering Institute.
- CVE-2022-34831. Sept 2022. National Institute for Standards and Technologies.

- *Reading for next lecture: Pressman Ch 17*