CS1632, LECTURE 4: DEFECTS

Bill Laboon

What do we mean by “defect”?

Bug, n.: An unwanted and unintended property of a program or piece of hardware, esp. one that causes it to malfunction. Antonym of feature.

-Eric S. Raymond, The Jargon File

Better definition:

Some condition in a system which does one of the following:

1. Violates a requirement
2. Violates end-user expectations
3. Causes the program to malfunction or end prematurely
4. Produces an incorrect result

Where do defects come from?

- Gaps or mistakes in code
- Gaps or ambiguity in requirements
- Other:
  - *Compiler broken*
  - *Bad hardware*
  - *Broken operating system*
  - *Gamma rays from space*
- Guess which are the two areas on focus on?

A Defect Is Visible to the User!

```
// Program shall always print out “wombat”
// Program shall never print out “cephalopod”
// Is there a defect in this code?
int k = 4;
if (k > 100) {
   System.out.println(“cephalopod”);
} else {
   System.out.println(“wombat”);
}
```

Bad Code != Defect

- This does NOT mean that it’s a good thing to have bad code!
- Don’t tell people Prof. Laboon is saying that it’s OK to have ugly code if it’s not visible to the user
- But from a definitional perspective, a defect is something that impacts the functionality of the program

If the behavior of a system under test does not meet the requirements (implicit or explicit), or the expectation of a user, it can be considered a defect.

- A user may have expected an error message when a string is entered when an integer was expected.
- A user may not have cared whether a word is capitalized or not.
- A user may want negative numbers to be treated as positives.
- You may need to have a discussion with systems engineers or requirement analysts if the behavior of the system as defined by the requirements is not what the user expects.
  - *Remember verification vs validation!*

Defects vs Enhancements

- If the software does not meet the requirements, or is unstable (which is an implied requirement), then there's a DEFECT.
- If the user wants to ADD or MODIFY a requirement/feature/etc, that's an ENHANCEMENT request.

A defect does not have be severe to be a defect

- Images are sized 1 pixel too small
- Delays are 1 ns longer than required
- Upon shutdown, typo in final statement
- Seldom-used feature does not work correctly
- Background color is slightly off
- There should be three periods in an ellipsis, not two..

Non-trivial software will ship with defects. Get used to it.

- Hopefully, you can catch many of them ahead of time, even if they can’t be fixed.
- A KNOWN bug is much better than an UNKNOWN bug.
- Your customer will thank you.

When testing, focus on finding important defects:

- Faulty data
- System crashes
- Extreme resource usage
- Not meeting requirements

Defects can be ambiguous

- Communication
- Communication
- Communication

- What makes a defect?
- What makes a defect serious?
- How should I report defects?
- How do I interpret the requirements?
- Answers to these will vary based on the project, company, and test team.

How to report defects?

Varies based on company/project, but there are some common concepts.

The template I like to use:

- SUMMARY
- DESCRIPTION
- REPRODUCTION STEPS
- EXPECTED BEHAVIOR
- OBSERVED BEHAVIOR
- IMPACT
- SEVERITY
- NOTES

Summary - *A succinct (one-sentence or so) description of the problem.*

- Title does not display after clicking "Next“
- CPU pegs after addition of any two cells
- Total number of widgets in shopping cart not refreshed after removal of more than one
- Page title is "Alll Entries", should be "All Entries“
- If timezone is changed during execution, idle tasks never wake up

DESCRIPTION - *A more detailed explanation of the problem.*

- If more than one widget is removed from the shopping cart, the number of widgets is not changed from the initial value. This value is updated if the widgets are removed one at a time.

DESCRIPTION

Be careful not to overgeneralize (or undergeneralize, but this tends to be less of a problem) here.

REPRODUCTION STEPS - *Specify an EXACT SEQUENCE OF STEPS to reproduce the problem.*

- Make sure you give:
  - *Exact values*
  - *Exact steos*
  - *Exact manner of execution*
- It’s usually better to err on the side of overspecificity

REPRODUCTION STEPS

- BAD: Put some things in the shopping cart. Take a couple things out.
- GOOD:
  - *1. Add three widgets to shopping cart*
  - *2. Note number of widgets listed is 3*
  - *3. Remove two widgets from shopping cart*
  - *Observe number of widgets listed*

EXPECTED AND OBSERVED BEHAVIOR

- EXPECTED BEHAVIOR: This should note, as precisely as possible, what you expected to see according to the requirements.
- OBSERVED BEHAVIOR: This should note what you ACTUALLY saw.

BAD EXAMPLE

- Expected Behavior: Number is correct.
- Observed Behavior: Number is incorrect.

GOOD EXAMPLE

- EXPECTED BEHAVIOR:
  The number of widgets in the shopping cart is 1.
- OBSERVED BEHAVIOR:
  The number of widgets in the shopping cart is 3.

EXPECTED VS OBSERVED BEHAVIOR

- What you saw versus what you expected to see is the CRUX of a bug report.
- Make sure you get it right!
- Be as PRECISE as possible.

IMPACT – How does this defect impact the user of the software?

BAD:The user will hate this because everything is wrong!

GOOD:The user will see an incorrect number of widgets in their shopping cart, meaning they could purchase fewer widgets than they expect.

SEVERITY – how severe is the problem?

Note that this differs from PRIORITY, the ordering of which defects should be worked on first. However, the two are not orthogonal; ceteris paribus, a higher-severity bug will take precedence over a lower-severity one.

SEVERITY

Severity is a combination of several factors:

1. How bad is the problem when it does occur?
2. How often does it occur?
3. Is there a workaround?

LEVELS OF SEVERITY (Bugzilla)

- BLOCKER
- CRITICAL
- MAJOR
- NORMAL
- MINOR
- TRIVIAL

NOTES – Technical and detailed notes that can help understand and fix the problem.

- Stack traces
- Log file excerpts
- Environment
- Anything that may be helpful to a developer fixing this defect

Tracking, Triaging, and Prioritizing Defects

Once you find defects, you need to report them and eventually they need to be fixed.

Tracking Defects

- Defects are usually numbered, not named.
- They should have the following information:
  - *1. Identifier*
  - *2. Source - associated test case, if applicable*
  - *3. Version of software found*
  - *4. Version of software fixed, if applicable*

Lifecycle of a defect

- Discovery
- Recording
- Triage
- Sub-triage (optional)
- Fixed
- Verified

Triage (or "Defect Review")

- This is where relevant stakeholders meet to determine:
- 1. Final severity
- 2. Final priority
- 3. Validity of defect
- 4. Need for more information
- etc.

Fixing

The developer then works on a fix for the bug. This may be an iterative process, with the developer and tester working hand in hand to ensure that the fix is correct and complete, and does not break other parts of the software.

NOTE: This is where automated test suites and unit tests help IMMENSELY!

Verification

Finally, the tester verifies that the bug was actually fixed and is not causing any other issues.

Sub-Triage

For very large projects, there may be a "system triage" and sub-triages, say for each functional group or IPT.

At this point, systems-level triage usually does more filtering and sorting, with less emphasis on prioritization.