CS1632, Lecture 5: Defects

Wonsun Ahn

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 of defect:

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

1. Violates an explicit requirement (both functional / non-functional)
2. Violates an implicit requirement (no malfunction or crash)

- Something that end-user wants but not in the requirements is an *enhancement*

Defects vs Enhancements

- If software does not meet requirements, then it's a DEFECT.
- If user wants to ADD a requirement, that's an ENHANCEMENT.
- Sometimes the boundary may get blurry
  - Is the problem a malfunction or just a candidate for improvement?
  - When things get intense, it may even end up in a lawsuit:
    If defect, developer must pay for any resulting damages;
    If enhancement, user must pay for the improvement

Defect or Enhancement?

- Example: Program loses internal data on system power outage
  - Suppose requirements didn't specify behavior on power outage
  - If program is a database, probably a defect
    Why? Implicit assumption is no data loss should happen in any circumstance.
  - If program is a solitaire game, may be just a candidate for enhancement
- Example: Program may become unresponsive for 1 second
  - If program is a real-time game, probably a defect
    Why? Implicit assumption is a real-time game must be responsive at all times
  - If program is a batch file copy tool, may be just a candidate for enhancement
- Note that confusion arose because requirements were not explicit
  - More reason why requirements should be complete and unambiguous

Make requirements explicit!

- Whether something is a defect or an enhancement is beside the point
  - The result is the same: low quality software that does not fulfill user needs

- Never leave gaps or ambiguity in the requirements

- Communication!
- Communication!
- Communication!

Defects can come from external sources

- Gaps or mistakes in code
- Gaps or ambiguity in requirements
- Other:
  - Compiler broken
  - Faulty hardware
  - Broken operating system
  - Cosmic rays from space (not joking)
- Guess which are the two areas we are going to focus on?

Defects can come from external sources

- Faulty external software
  - Faulty compiler
  - Faulty operating system
- Faulty hardware
  - Faulty CPU, DRAM, I/O device
  - Cosmic rays from space (not joking)

- … but let’s only focus on defects in our software in this course

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!
- It’s not OK to have ugly code even if it’s not visible to the user
- But it is still not a defect if it doesn’t impact behavior

A defect does not necessarily have to be severe

- 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.

- It will contain KNOWN bugs as well as UNKNOWN bugs

- Why ship when there are known bugs?
  - Bug may not be severe enough to impact everyday usage
  - Bug may have a workaround (ways to avoid the bug)

- Knowns bugs should be well-documented and advertised
  - Your customer will thank you

When testing, focus on important defects:

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

Again, Context is Important

- What makes a defect?
- What makes a defect serious?
- How should I report defects?
- How do I interpret the requirements?
- Answers will vary based on 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 - *succinct description of problem*

- Usually a one sentence description
- Examples:
  - Title does not display after clicking "Next“
  - CPU pegs at 100% after addition of any two cells
  - Total number of widgets in 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 - *details of problem*

- A detailed description of everything the tester discovered about the problem that may help the developer
- Example:
  - 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.
- Be careful not to overgeneralize (or undergeneralize, but this tends to be less of a problem)
  - Describing the contours of the issue accurately helps developer

REPRODUCTION STEPS
- *Exact sequence of steps to reproduce problem.*

- Make sure you give:
  - Exact values
  - Exact steps
  - 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
    4. 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.

- This is the CRUX of the defect report.
  - Make sure you get it right!
  - Be as PRECISE as possible.

EXPECTED AND OBSERVED BEHAVIOR

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

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

IMPACT – 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?

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)

- CRITICAL
- MAJOR
- NORMAL
- MINOR
- TRIVIAL

SEVERITY is different from PRIORITY

- *Priority*: the ordering of which defects should be work on first

- Usually a higher severity bug will be given higher priority
  - But not always; other considerations may take precedence

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.

Sub-Triage

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

- With sub-triage, systems-level triage usually does filtering and sorting, sub-triage doing the prioritization

Fixing

- Developer works on a fix for the bug

- Usually an iterative process, with the developer and tester working hand in hand
  - To ensure that the fix is correct and complete
  - To ensure it does not break other parts of software (In other words, tester does regression testing)
  - Automated test suites and unit tests help ensure above
  - After bug is fixed, test case for bug is added to test suite

Verification

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

Now Please Read Textbook Chapter 9

- Be sure read Chapter 9.3 carefully since you will be using the defect template for the first in-class exercise.