SOFTWARE ERROR

OUTLINE

- What is a Software Error?
- Finding, Reproducing and Analyzing a Software Error
- Reporting a Software Error
- A Common Bug Life

What is a Software Error?

- A software error is present when the program does not do what its user reasonably expects it to do.
- It is fair and reasonable to report any deviation from high quality as a software error.

The existence of software errors reflects an impediment on the quality of the product, but does not necessarily imply that the developers are incompetent.

Common Sources of Errors

- You will report all of these types of problems:
  - **Coding Error**: The program doesn't do what the programmer would expect it to do.
  - **Design Issue**: It's doing what the programmer intended, but a reasonable customer would be confused or unhappy with it.
  - **Requirements Issue**: The program is well designed and well implemented, but it won't meet one of the customer's requirements.
  - **Documentation or Specification/ Code Mismatch**: Sometimes the spec is right; sometimes the code is right and the spec should be changed.

13 Common Types of Software Errors

- 1/ User Interface
- 2/ Error Handling
- 3/ Boundary-Related
- 4/ Calculation
- 5/ Initial and Later States
- 6/ Control Flow
- 7/ Handling or Interpreting Data
- 8/ Race Conditions
- 9/ Load Conditions
- 10/ Hardware/Environment Compatibility
- 11/ Source, Version, and ID Control
- 12/Testing
- 13/ Documentation

OUTLINE

- What is a Software Error?
- Finding, Reproducing and Analyzing a Software Error
- Reporting a Software Error
- A Common Bug Life

What to do in a bug finding process?

1. Reproduce the error
2. Analyze the error
3. Report the error

Reproducing a Software Error

- Some bugs are always reproducible, but some are just sometimes or even rarely.
- Bugs don't just miraculously happen and then go away. If a bug happens intermittently, it might be under some certain conditions.
- We hypothesize the cause, then we try to re-create the *conditions* that make the error visible.
- If the bug is non-reproducible, you should always report it, but describe your steps and observations precisely. Programmers will often figure them out.

Why is a Bug Hard to Reproduce?

- Memory dependent
- Memory corruption
- Configuration dependent
  - Software
  - Hardware
- Timing related
- Initialization
- Data flow dependent
- Control flow dependent
- Error condition dependent
- Multi-threading dependent
- Special cases
  - Algorithm
  - Dates
- …

Making an Error Reproducible

- Write down everything you remember about what you did the first time.
- Note which things you are sure of and which are good guesses.
- Note what else you did before starting on the series of steps that led to this bug.
- Review similar problem reports you've come across before.
- Use tools such as capture/replay program, debugger, debug-logger, videotape, or monitoring utilities
- Talk to the programmer and/or read the code.

Analyzing a Software Error

Why Analyze a Reproducible Bug?

- Analyze bugs in order to:
  - Make your communication effective:
    - Make sure you are reporting what you think you are reporting.
    - Make sure that questionable side issues are thoroughly investigated.
    - Create accountability.
  - Support the making of business decisions;
  - Avoid wasting the time of the programming and management staff;
  - Find more bugs.

Analyzing a Reproducible Error

- Start by making sure the error is reproducible.

  1)Describe how to get the program into a known state.

  2)Specify an exact series of steps that expose the problem.

  3)Test your steps to make sure that you can reproduce the problem if you do exactly (and only) what it says in the bug report.

OUTLINE

- What is a Software Error?
- Finding, Reproducing and Analyzing a Software Error
- Reporting a Software Error
- A Common Bug Life

How to Report a Software Error

Bug reports are your primary work product.

A Useful Bug Report

- Written
- Uniquely numbered (ID required)
- Simple (non-compound - one bug per report)
- Understandable
- Reproducible
- Non-judgmental

Report Content

- **Summary**
- **Description**
- **Steps to Reproduce – including expected behavior and observed behavior**
- Reproducible
- Severity
- Frequency
- Priority
- Keyword (Functional Area)
- Resolution

Bug Summary

- This one-line description of the problem is the most important part of the report.
- The ideal summary tells the reader what the bug is, what caused the bug, what part of the program it's from and what its worst consequence is. It runs from 8 to 15 words long.
- We use the following syntax for writing the problem summary:

Symptom + Action + Operating Condition

Bug Summary

Some good examples of Bug Summary:

1. Run-time error when submitting the Contact Us form with first name of more than 256 characters.
2. The main dialog box is resizable
3. Help button does not bring up Help page.
4. Maximize button is still active while the dialog box is maximized.

Bug Summary

Some not-so-good examples of Bug Summary:

1. Software fails
2. Can't install
3. Back button does not work
4. My browser crashed. I think I was on www.foo.com. I play golf with Bill Gates, so you better fix this problem, or I'll report you to him. By the way, your Back icon looks like a squashed rodent. Too ugly. And my grandmother's home page is all messed up in your browser.

Bug Description and Steps to Reproduce

- First, describe the problem. What is the bug? Don't rely on the summary to do this – a short line sometimes cannot state all what you want to say.
- Next, go through the steps that you use to recreate this bug. Start from a known place (e.g. boot the program) and then describe each step until you hit the bug.
- Describe the erroneous behavior and if necessary, explain what should have happened. (Why is this a bug? Be clear.)

Improve the Bug Report

- But you may be able to improve the report in four ways:

1) You might be able to *simplify* the report by:
   - **eliminating unnecessary or irrelevant steps**.
   - **splitting it into two reports**.
2) You might be able to *strengthen* the report by
   - **showing that it is more serious than it first appears**.
   - **showing that it is more general than it first appears**.

1. Eliminate Unnecessary Steps

- ***Look for critical steps -- Sometimes the first symptoms of an error are subtle.***
  - If you've found what looks like a critical step, try to eliminate almost everything else from the bug report.
  - Go directly from that step to the last one (or few) that shows the bug.

2. Split the Report in Two

- When you see two related problems, you *might* report them together on the same report as long as you show that there are two of them.
- If this makes the report become confusing, write two reports instead.
- When you report related problems, it's a courtesy to cross-reference them. For example:

```
Related bug -- see Report # xxx
```

3. Show that it is More Serious

- **Look for follow-up errors**:
  - Keep using the program after you get this problem. Does anything else happen?
- **Look for nastier variants**:
  - Vary the conditions under which you got the bug.
- **Look for nastier configurations**:
  - Sometimes a bug will show itself as more serious if you run the program with less memory, a higher resolution output, more graphics on a page, etc.

4. Show that it is More General

- **Look for alternative paths to the same problem**:
  - Sometimes the bug can happen in some alternative path.
- **Look for configuration dependence**:
  - Sometimes the bug happens because of some hardware configuration.

Reproducible

- You may or may not have this on your form, but you should always provide this information.
  - Never say Yes unless you have recreated the bug
  - If you have tried and tried but you can not recreate the bug, say No. Then explain what steps you tried in your attempt to recreate it.
  - If the bug appears sporadically and you do not yet know why, say “sometimes” and explain.

Severity

- You will have to rate the bug's seriousness. Many companies use a three-level rating:
  - 1 - Critical: This means fatal to the release
  - 2 - Serious: It's a bad bug, but it doesn't cause data loss or a program crash.
  - 3 - Minor: It's a bug, but it's not a big deal.
- Level rating depend on each company.
- Many companies sort their summary reports by severity, so you want to fill in this field thoughtfully.

Frequency

- **Frequency** is usually graded by assessing the following three characteristics:
  - How easy is it for the user to encounter the bug
  - How frequent would the user encounter the bug
  - How often the buggy feature is used

- Many companies use a three-level rating:
  - 1 - Always
  - 2 - Often
  - 3 - Seldom

Priority

- **Priority** rating is either automatically generated by the bug tracking system by assessing the Severity and the Frequency ratings or assigned only by the project manager.

Keyword (Functional Area)

- You may have to categorize the bug according to its functional area.
- The tracking system should include a list of the possible keywords.
- It is important to categorize these bugs consistently. Burying a bug in the wrong category can lead to its never getting fixed.
- If you're creating the list of functional areas, keep it short, perhaps 20 areas.

Resolution

- The project manager has the privilege to assign most of the resolutions in this field.
- Common resolutions include:
  - **New**: The newly submitted bug
  - **To Be Distributed**: The bug is waiting to be distributed
  - **To Be Fixed**: The bug is being fixed.
  - **QA Info Request**: The bug needs more clarification from Tester.
  - **Developer Info Request**: The bug needs more clarification from Developer.
  - **Not Reproducible**: The bug cannot be reproduced.
  - **Fixed**: The bug is fixed.
  - **Not a Problem**: The application works as it is supposed to.
  - **Duplicate**: The bug is just a repeat of another bug.
  - **Deferred**: The bug will be fixed in a later release.
  - **Feature Limitation**: There is some feature limitations that do not allow to fix the bug.

Bug Report Simple Template

Bug ID: 001
Summary
Main Form: The application does not close when clicking on End button

Description: The application can be closed when clicking on X button but cannot be closed when clicking on End button

Steps:

1. Open MiniBank
2. Click on End button
3. Observe the result

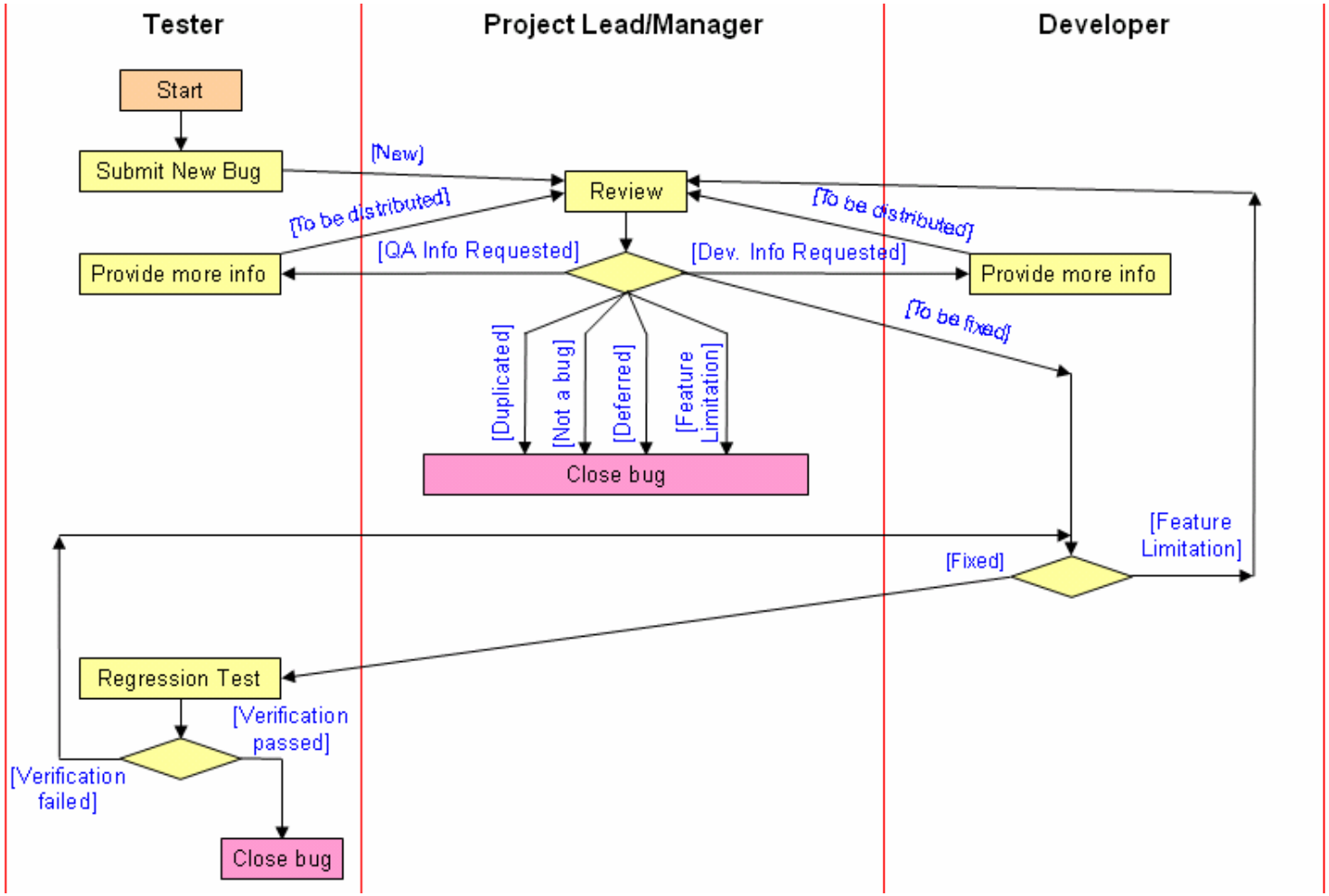
Expected Result: The application is closed

Observed Result: The application is still on the screen

OUTLINE

- What is a Software Error?
- Finding, Reproducing and Analyzing a Software Error
- Reporting a Software Error
- A Common Bug Life

A Common Bug Life



Q&A