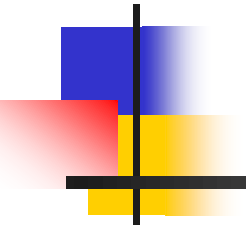
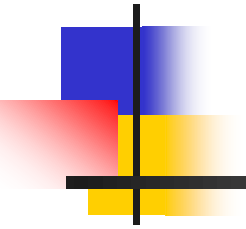


Software Testing & Quality Assurance

Reporting

- Analyzing bugs
- Reproducing bugs and
- Reporting bugs effectively



Credits & Readings

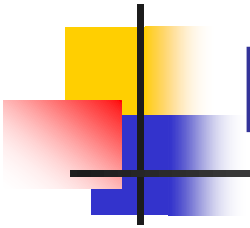
- The material included in these slides are adopted from the following resources:
  - Cem Kaner, Jack Falk, Hung Q. Nguyen, *"Testing Computer Software"* Wiley
  - The Testing Education & Research Center: http://www.testingeducation.org/

Terminology review

- Here's a defective program segment:

```
INPUT A
INPUT B
PRINT A / B
```

- What is the error? What is the fault?
- What is the critical condition (that triggers failure)?
- What will we see as the incident of the failure?



Bug reporting

- Testers report bugs to software developers
- Problem (bug) report forms are commonly used
- If the report is not clear and understandable, the bug will not get fixed
- To write a fully effective report you must:
  - Explain how to reproduce the problem
  - Analyze the error so that it can be described with a minimum number of steps
  - Write a report that is complete, easy to understand, and non-antagonistic

A typical problem report form*

PROBLEM REPORT #____
REPORTER_____ **DATE** ____/____/____

PROGRAM_____ **RELEASE** _____ **VERSION** _____

CONFIGURATION _____
_____
_____

REPORT TYPE ___
1 - Coding error 4 - Documentation
2 - Design Issue 5 - Query
3 - Suggestion

SEVERITY ___
1 - Fatal
2 - Serious
3 - Minor

ATTACHMENTS *(Y/N)*___
*Description:*_____
_____
_____

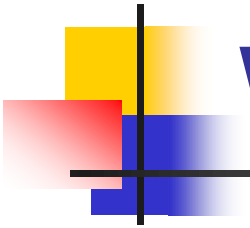
PROBLEM SUMMARY _____

CAN YOU REPRODUCE THE PROBLEM *(Y/N/S/U)* ____

PROBLEM AND HOW TO REPRODUCE IT _____
_____
_____
_____

SUGGESTED FIX (*optional***)**_____
_____
_____
_____

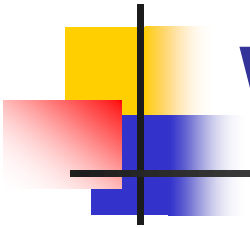
*Cem Kaner, Jack Falk, Hung Q. Nguyen, *"Testing Computer Software"*



What kind of errors to report?

- You may report any of the following:
  - Coding Error: The program doesn't do what the programmer expects it to do
  - Design Issue: It's doing what the programmer intended, but a typical customer would be confused or unhappy with it
  - More on the next slide…

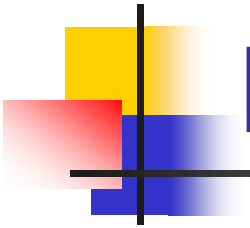
What kind of error to report?



- Requirements Issue: The program is well designed and well implemented, but it won't meet some of the customer's requirements
- Documentation / Code Mismatch: Inconsistency between the code and related documentation
  - Report this to the programmer (via a bug report) and to the writer (usually via a memo or a comment on the manuscript)
- Specification / Code Mismatch: Sometimes the spec is right; sometimes the code is right and the spec should be changed

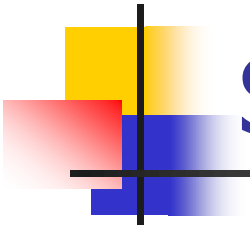
Importance of bug reports

- A *bug report* is a tool that you use to sell the programmer on the idea of spending his/her time and energy to fix a bug
  - Motivate the buyer (i.e. make him/her WANT to fix the bug)
  - Overcome objections (i.e. get past his/her excuses and reasons for not fixing the bug)
- Bug reports are your primary work product as a tester
  - This is what people outside of the testing group will most notice and most remember of your work
- The best tester isn't the one who finds the most bugs or who embarrasses the most programmers
  - An effective tester is the one who gets the most bugs fixed



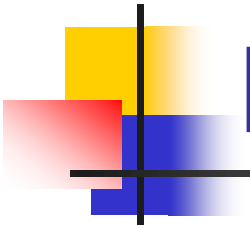
Motivating the bug fixer

- Some reasons that will often make programmers want to fix the bug:
  - It looks really bad
  - It looks like an interesting puzzle and piques the programmer's curiosity
  - It will affect lots of people
  - Getting to it is trivially easy
  - It has embarrassed the company, or a bug like it embarrassed a competitor
  - Management (that is, someone with influence) has said that they really want it fixed



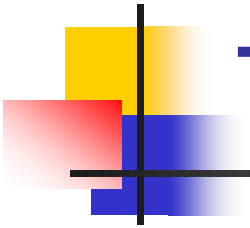
Show the magnitude of faults

- Identifying the symptoms versus curing the disease
  - When you run a test and find a failure, you're looking at a symptom, not at the underlying fault
  - You may or may not have found the best example of a failure that can be caused by the underlying fault
- Therefore you should do some follow-up work to try to prove that :
  - *A defect can be more serious than it first appears*
  - *A defect is more general than it first appears*



Follow-up testing

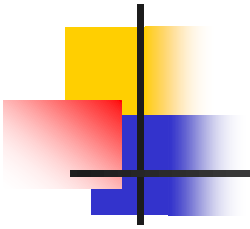
- Vulnerable program state
  - When you find a coding error, you have the program in a state that the programmer did not intend and probably did not expect
- Leverage vulnerability
  - Keep testing and you might find that the real impact of the underlying fault is a much worse failure, such as a system crash or corrupted data



Types of follow-up testing

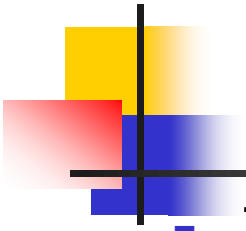
1. Vary the behavior (change the conditions by changing what the test case does)
2. Vary the options and settings of the program (change the conditions by changing something about the program under test)
3. Vary the software and hardware environment

Let's describe each one in more detail…



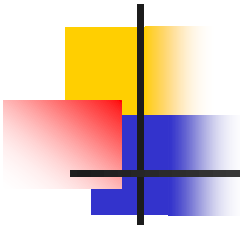
1. Vary the behavior

- Keep using the program after you see the problem (stay with it)
- Bring it to the failure state again and again (multiple times)
  - If the program fails when you do X, then do X many times
  - Is there a cumulative impact?
- Try things that are related to the task that failed
  - For example, assume that the program unexpectedly but slightly scrolls the display when you add two numbers
  - Then try doing X, see the scroll. Do Y then do X, see the scroll. Do Z, then do X, see the scroll, etc. (If the scrolling gets worse or better in one of these tests, follow that up, you're getting useful information for debugging)



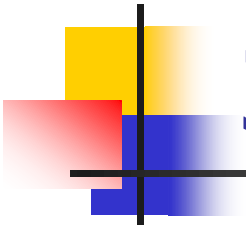
1. Vary your behavior--continued

- Try things that are related to the failure
  - If the failure is unexpected scrolling after adding then
    - Try scrolling first, then adding
    - Try repainting the screen, then adding
    - Try resizing the display of the numbers, then adding
- Vary the speed
  - Try entering the numbers more quickly or
  - Change the speed of your activity in some other way
- Also try other exploratory testing techniques
  - For example, you might try some interference tests
    - Stop the program or pause it just as the program is failing
    - Try it while the program is doing a background save
    - Does that cause data loss corruption along with this failure?



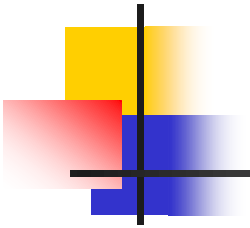
2. Vary options and settings

- Try to reproduce the bug when the program is in a different state:
  - Change the values of environment variables
  - Change how the program uses memory
  - Change anything that looks like it might be relevant that allows you to change as an option
- For example, suppose the program scrolls unexpectedly when you add two numbers
  - Maybe you can change the size of the program window, or the precision (or displayed number of digits) of the numbers



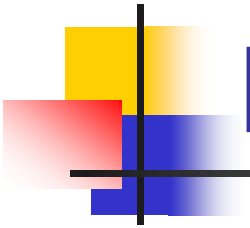
3. Vary the configuration

- A bug might show a more serious failure if you run the program with less memory, a higher resolution printer, more device interrupts coming in etc.
  - If there is anything involving timing, use a really slow (or very fast) computer, link, modem or printer, etc.
  - If there is a video problem, try other resolutions on the video card
  - Try displaying MUCH more (less) complex images



Old bugs

- In many projects, an *old bug* (i.e. found in a previous release of the program) might not be taken very seriously, if there weren't lots of customer complaints
  - If you know it's an old bug, check its history
  - The bug will be taken more seriously if it is new
  - You can argue that it should be treated as new if you can find a new variation or a new symptom that didn't exist in the previous release
  - What you are showing is that the new version's code interacts with this error in new ways
  - *So, that's a new problem…*

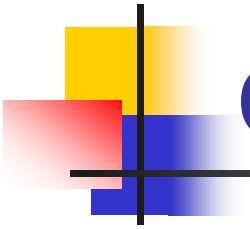


Bugs credibility

- Bugs that don't fail on the programmer's machine are much less *credible* (to that programmer)
- Look for configuration dependence
- If they are configuration dependent, the report will be much more credible if it identifies the configuration dependence directly

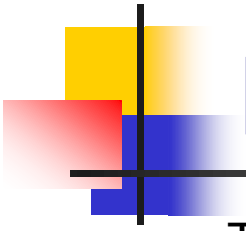
Configuration dependence

- It is recommended to perform testing on two separate machines (standard in many companies)
  - Do your main testing on Machine 1
    - Maybe this is your powerhouse: latest processor, newest updates to the operating system, fancy printer, video card, USB devices, huge hard disk, lots of RAM, cable modem, etc.
  - When you find a defect, use Machine 1 as your bug reporting machine and replicate on Machine 2
    - Machine 2 is totally different. Different processor, different keyboard and keyboard driver, different video, barely enough RAM, slow, small hard drive, dial-up connection with a link that makes turtles look fast
- Some people do their main testing on the turtle and use the power machine for replication



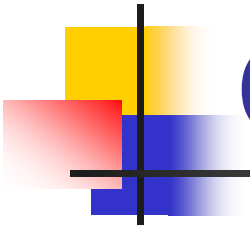
Configuration dependence-continued

- Write the steps, one by one, on the bug form at Machine 1
- As you write them, try them on Machine 2
  - If you get the same failure, you've checked your bug report while you wrote it (a good thing to do)
  - If you don't get the same failure, you have a configuration dependent bug. Time to do troubleshooting



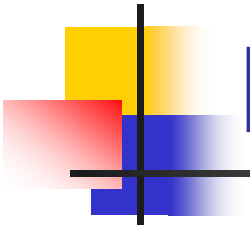
Extreme vs. mainstream values

- Test at extreme values (most likely to show a defect)
  - If they yield failure, then do some troubleshooting around the extremes
  - Is the bug tied to a single setting or is there a small range of cases?
  - In your report, identify the narrow range that yields failures
    - The range might be so narrow that the bug gets deferred
    - That might be the right decision. Your reports help the company choose the right bugs to fix before a release, and size the risks associated with the remaining ones
- Try mainstream values
  - These are easy settings that should pose no problem to the program
  - Do you replicate the bug? If yes, write it up, referring primarily to these mainstream settings
  - This will be a very credible bug report



Overcoming resistance

- Things that will make programmers resist spending their time on fixing the bug you have reported:
  - The programmer can't replicate the defect
  - Strange and complex set of steps required to induce the failure
  - Not enough information to know what steps are required, and it will take a lot of work to figure them out
  - The programmer doesn't understand the report
  - Unrealistic (e.g. "corner case")
  - It's a feature



Non-reproducible errors

- Always report non-reproducible errors
  - When you realize that you can't reproduce the bug, write down everything you can remember (do it now, before you forget even more)
  - As you write, ask yourself whether you're sure that you did this step (or saw this thing) exactly as you are describing it. If not, say so. Draw these distinctions right away. The longer you wait, the more you'll forget
  - Maybe the failure was a delayed reaction to something you did before starting this test or series of tests. Before you forget, note the tasks you did before running this test
  - Check the bug tracking system. Are there similar failures? Maybe you can find a pattern
  - If you report them well, programmers can often figure out the underlying problem
- You must describe the failure as precisely as possible
  - If you can identify a display or a message well enough, the programmer can often identify a specific point in the code that the failure had to pass through

The problem report form[1]

- A typical form includes many of the following fields
  - **Problem report number**: must be unique
  - **Reported by**: original reporter's name. Some forms add an editor's name
  - **Date reported**: date of initial report
  - **Program (or component) name**: the visible item under test
  - **Release number**: like Release 2.0
  - **Version (build) identifier**: like version C or version 20000802a

The problem report form[2]

- **Configuration(s)**: h/w and s/w configurations under which the bug was found and replicated
- **Report type**: e.g. coding error, design issue, documentation mismatch, suggestion, query
- **Can reproduce**: yes / no / sometimes / unknown (Unknown can arise, for example, when the configuration is at a customer site and not available to the lab)
- **Severity**: assigned by tester. Some variation on small / medium / large

The problem report form[3]

- **Priority**: assigned by programmer/project manager
- **Problem summary**: 1-line summary of the problem
- **Key words**: use these for searching later, anyone can add to key words at any time
- **Problem description and how to reproduce it**: concise step by step reproduction description
- **Suggested fix**: leave it blank unless you have something useful to say
- **Status**: Tester fills this in: Open / closed / resolved

The problem report form[4]

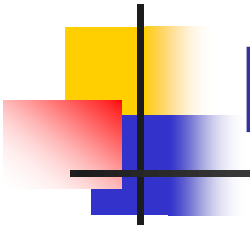
- **Resolution**: The project manager owns this field. Common resolutions include:
  - **Pending**: the bug is still being worked on
  - **Fixed**: the programmer says it's fixed. Now you should check it
  - **Cannot reproduce**: The programmer can't make the failure happen. You must add details, reset the resolution to Pending, and notify the programmer
  - **Deferred**: It's a bug, but we'll fix it later
  - **As Designed**: The program works as it's supposed to
  - **Need Info**: The programmer needs more info from you. He/she has probably asked a question in the comments
  - **Duplicate**: This is just a repeat of another bug report (XREF it on this report). Duplicates should not close until the duplicated bug closes
  - **Withdrawn**: The tester withdrew the report

The problem report form[5]

- **Resolution version**: build identifier
- **Resolved by**: programmer, project manager, tester (if withdrawn by tester), etc.
- **Resolution tested by**: originating tester, or a tester if originator was a non-tester
- **Change history**: date-stamped list of all changes to the record, including name and fields changed

The problem report form[6]

- **Comments**: free-form, arbitrarily long field, typically accepts comments from anyone on the project. Testers, programmers, tech support (in some companies) and others have an ongoing discussion of reproduction conditions, etc., until the bug is resolved. Closing comments (why a deferral is OK, or how it was fixed for example) go here
  - This field is especially valuable for recording progress and issues with difficult or politically charged bugs
  - Write carefully. Just like e-mail and web postings, it's easy to read a joke or a remark as a flame. Never flame.

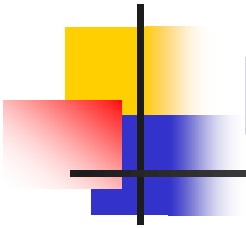


Problem summary

- This one-line description of the problem is the most important part of the report
  - The project manager will use it in when reviewing the list of bugs that haven't been fixed
  - Executives will read it when reviewing the list of bugs that won't be fixed. They might only spend additional time on bugs with "interesting" summaries
- The ideal summary gives the reader enough information to help him/her decide whether to ask for more information. It should include:
  - A brief description that is specific enough that the reader can visualize the failure
  - A brief indication of the limits or dependencies of the bug (how narrow or broad are the circumstances involved in this bug)?
  - Some other indication of the severity (not a rating but helping the reader envision the consequences of the bug)

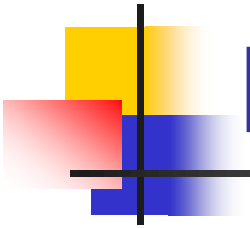
Can you reproduce the bug?

- You may not see this on your form, but you should always provide this information
  - Never say it's reproducible unless you have recreated the bug (Always try to recreate the bug before writing the report)
  - If you've tried and tried but you can't recreate the bug, say "No". Then explain what steps you tried in your attempt to recreate it
  - If the bug appears sporadically and you don't yet know why, say "Sometimes" and explain
  - You may not be able to try to replicate some bugs. Example: customer-reported bugs where the setup is too hard to recreate



Explain how to reproduce the bug

- First, describe the problem
- Don't rely on the summary to do this - some reports will print this field without the summary
- Go through the steps that you use to recreate this bug
  - Start from a known place (e.g. boot the program)
  - Then describe each step until you hit the bug
  - Number the steps. Take it one step at a time
  - If anything interesting happens on the way, describe it
  - Use landmarks/beacons
    - You are giving people directions to locate a bug
    - Especially in long reports, people need landmarks

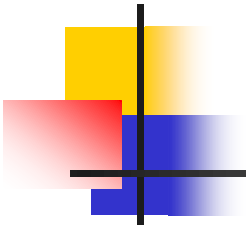


How to reproduce the bug

- Describe the erroneous behavior and, if necessary, explain what should have happened (i.e. why is this a bug? Be clear)
- List the environmental variables (e.g. configuration) that are not covered elsewhere in the bug tracking form
- If you expect the reader to have any trouble reproducing the bug (special circumstances are required), be clear about them
- It is essential to keep the description focused
- The first part of the description should be the shortest step-by-step statement of how to get to the problem
- Add "Notes" after the description such as:
  - Comment that the bug won't show up if you do step X between step Y and step Z
  - Comment explaining your reasoning for running this test
  - Comment explaining why you think this is an interesting bug
  - Comment describing other variants of the bug

Keeping the report simple

- If you see two failures, write two reports
- Combining failures creates problems:
  - The summary description is typically vague. You say words like "fails" or "doesn't work" instead of describing the failure more vividly. This weakens the impact of the summary
  - The detailed report is typically lengthened and contains complex logic like: "Do this unless that happens in which case don't do this unless the first thing, and then the test case of the second part and sometimes you see this but if not then that…"
- Even if the detailed report is rationally organized, it is longer (there are two failures and two sets of conditions, even if they are related) and therefore more intimidating
- You'll often see one bug get fixed but not the other
- When you report related problems on separate reports, it is a courtesy to cross-reference them



Eliminate unnecessary steps[1]

- Sometimes it's not immediately obvious what steps can be dropped from a long sequence of steps in a bug
  - *Look for critical steps -- Sometimes the first symptoms of a failure are subtle*
- You have a list of the steps you took to show the error. You're now trying to shorten the list. Look carefully for any hint of a failure as you take each step
- A few things to look for:
  - Error messages (you got a message 10 minutes ago. The program didn't fully recover from the error, and the problem you see now is caused by that poor recovery)
  - Delays or unexpectedly fast responses
  - More on the next slide…

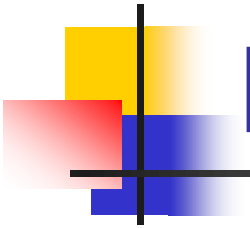
Eliminate unnecessary steps[2]

- Display oddities, such as a flash, a repainted screen, a cursor that jumps back and forth, multiple cursors, misaligned text, slightly distorted graphics, etc.
- Sometimes the first indicator that the system is working differently is that it sounds a little different than normal
- An in-use light or other indicator that a device is in use when nothing is being sent to it (or a light that is off when it shouldn't be)
- Debug messages—turn on the debug monitor on your system (if you have one) and see if/when a message is sent to it

- 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
- If this doesn't work, try taking out individual steps or small groups of steps

Unrealistic cases

- Some reports are inevitably dismissed as unrealistic (having no importance in real use)
  - If you're dealing with an extreme value, do follow-up testing with less extreme values
  - Check with people who might know the customer impact of the bug:
    - -- Technical marketing
    - -- Human factors
    - -- Network administrators
    - -- In-house power users
    - -- Technical support
    - -- Documentation
    - -- Training
    - -- Maybe sales



Editing bug reports

- Some groups have a second tester (usually a senior tester) review reported defects before they go to the programmer
- The second tester:
  - checks that critical information is present and intelligible
  - checks whether she/he can reproduce the bug
  - asks whether the report might be simplified, generalized or strengthened
- If there are problems, she/he takes the bug back to the original reporter