# EECS4313 week 2

Jerry Wu

2024-01-15

# Contents

# Lecture 1

# Reporting and analyzing bugs

## Bug reporting

### Writing an effective bug report

- Explain the problem and how to reproduce the problem
- Analyze the problem so that it can be described with a **minimum** number of steps
- Write a report that is complete, easy to understand, and non-antagonistic

  *"Be nice" - H.V. Pham 2024*

### What kind of errors to report?

- **Coding error** - The program does not do what the programmer would expect it to do; general bugs (pull request)
- **Design issue** - The program does 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/code mismatch** - Report this to the programmer (via bug report/github issues) and to the writer (via memo or a comment on the manuscript)
- **Specification/code mismatch** - Sometimes, the specs are right, sometimes the code is right and the specs should be changed

# Making effective bug reports

*"you are not the enemy of the programmer" - H.V. Pham 2024*

- Bug reports store all information needed to document, report and fix problems occuring in a software system
- Bug reports are like a pitch to the devs. Good bug reports will "sell" the developer the idea of spending their time and energy to fix said bug (incentive)
- Bug reports are the **primary work product** of a tester. This is what people outside of the testing group will notice and remember most about your work. In other words, include as much detail as possible
- The best tester is not the one who finds the most bugs or who embarrasses the most programmers, but the one who **gets the most bugs fixed**
- The primary goal at the end of the day is to work together with the developer to get the system running properly

## Selling a bug in a bug report

- Time is in short supply, so if you want to convince the dev to spend their time fixing your bug, you have to sell it to them
- Selling revolves around two fundamental objectives:
  - Motivate the buyer (make them **want** to fix the bug)
  - Overcome objections (get past their **excuses** and reasons for not fixing the bug)

## Motivating the bug fix

- When you run a test and find a failure, you are looking at a sympton and not the underlying fault. You may or may not have foun 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 fault:
  - is more serious than it first appears
  - is more general than it first appears
  - affects more versions of the software

- These things will often motivate a developer to fix a bug:
  - The bug looks really bad
  - It looks like an interesting puzzle and piques the programmer's curiosity
  - It will affect lots of people
  - The problem is trivial
  - 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
- As soon as you run into a problem in the software, fill out a **problem report form**. In a well written report, you:
  - **Explain the problem** and how to reproduce the problem
  - **Analyze the error** so you can describe it in a minimum number of steps
  - Include all the steps
  - Make the report easy to understand
  - Keep your tone neutral and non antagonistic
  - **Keep it simple**, report one bug per report
  - If a sample test file is essential to reproducing a problem, reference it and attach the test file.

## Problem report form outline

A typical report form includes some of the following fields

- **Problem report number** - unique number assigned to the report
- **Reported by** - author of the report
- **Date reported** - self explanatory
- **Program/component name** - the visible item/class under test
- **Release number** - self explanatory
- **Version/build identifier** - like version C or 20000802a
- **Configuration(s)** - h/w and s/w configurations under which the bug was found and replicated

- **Report type** - coding error, design issue, documentation mismatch, etc.
- **Can reproduce** - yes/no
- **Severity** - assigned by the tester. Some variation of small/medium/large
- **Priority** - assigned by the programmer/project manager
- **Problem summary** - self explanatory
- **Keywords** - used for searching for open reports in the project, anyone can add keywords at any time
- **Problem desc and reproduction steps**
- **Suggested fix** - don't do this unless you know for sure
- **Status** - Tester fills this in (open/closed/resolved/re-opened)
- **Resolution** - The project manager owns this field, common resolutions include:
  - **Pending** - the bug is currently being worked on
  - **Fixed** - the dev says it is fixed, so the tester should check it again
  - **Cannot reproduce** - the dev was unable to reproduce the issue. Add more details and notify the dev
  - **Deferred** - we'll fix this later (not)
  - **As designed** - the program works as intended
  - **Need info** - not enough information provided
  - **Duplicate** - the same issue has been brought up already
  - **Withdrawn** - tester has withdrew the report
- **Resolution version** - build identifier
- **Resolved by** - name of programmer, project manager, tester, etc
- **Resolution tested by** - original tester, or a tester if the originator was not a tester
- **Change history** - date stamped list of all changes to the record including name and fields changed

# Lecture 2

# Open research questions on bug management