CSE306
SOFTWARE QUALITY IN PRACTICE

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

www.cse.buffalo.edu/faculty/alphonce/SP24/CSE306

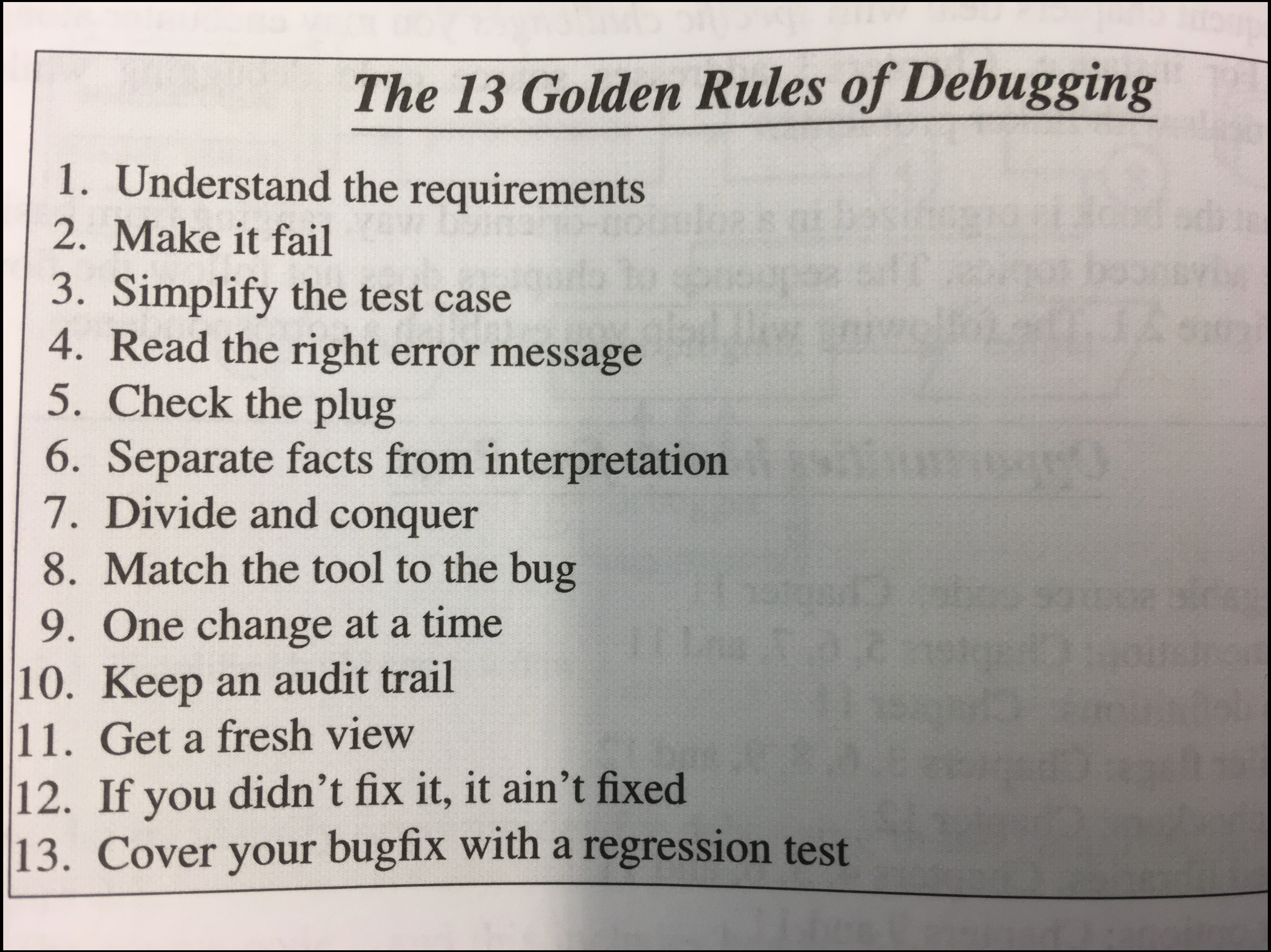
LATE JOINERS

- Today is the last day for Add/Drop
- TopHat and AutoLab rosters will reflect add/drop changes as of Thursday morning
- If you missed your lab session, do the LEX as soon as you can on your own time: post questions and requests for assistance in Piazza.
- We will NOT be strict on the deadlines for LEX01 and LEX02 (to accommodate students registering through end of add/drop)

ANNOUNCEMENTS

- Team formation will be finalized by tomorrow.
  - if you wish to pick your teammates form your team before 5:00 PM today
  - after 5:00 PM today I will assign remaining students to teams
- PRE (the first team project) will be posted on the course website as soon as team assignments are completed.
- Every team will have a Piazza group useful for intra-team communication, necessary for team-staff communication.

TEXT, PG 8



The 13 Golden Rules of Debugging

1. Understand the requirements
2. Make it fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate facts from interpretation
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didn't fix it, it ain't fixed
13. Cover your bugfix with a regression test

10. KEEP AN AUDIT TRAIL

- Keep track of all changes you have made (and what the result of each change was).
- Use a code repository! This lets you back out changes that were not productive.
- Audit trail is useful for you, whoever works on this code in the future, and for documenting your progress.

11. GET A FRESH VIEW

- Ask for someone else to have a look — but not before having done steps 1 - 10!
- Even just explaining the situation can help you better understand what is happening.

12. IF YOU DIDN'T FIX IT, IT AIN'T FIXED

- Intermittent bugs will recur.
- If you make a change to the code and the symptom goes away, did you really fix it? You must convince yourself that the fix you applied really did solve the problem!

13. COVER YOUR BUG FIX WITH A REGRESSION TEST

- Make sure the bug doesn’t come back! Just because it worked yesterday doesn't mean it still works today. This is especially important in team environments where you are not the only person touching the code.

ESSENTIAL TOOLS

- compiler (e.g gcc)
- debugger (e.g. gbd)
- memory checker (e.g. memcheck)
- runtime profiler (e.g. gprof)
- automated testing framework (e.g. cunit)
- build tool (e.g. make)
- code repository (e.g. git)
- organization/collaboration tool (e.g. ZenHub)
- pad of paper / whiteboard

CLASSIFICATION OF BUGS

- Common bugs (source code, predictable)
- Sporadic bugs (intermittent)
- Heisenbugs (averse to observation)
    - race conditions
    - memory access violations
    - (programmer) optimizations
- Multiple bugs - several must be fixed before program behavior changes - consider violating rule #9 "one change at a time"

WHY HEISENBUGS?
THE UNCERTAINTY PRINCIPLE...

…the uncertainty principle, also known as Heisenberg's uncertainty principle, is any of a variety of mathematical inequalities[1] asserting a fundamental limit to the precision with which certain pairs of physical properties of a particle, known as complementary variables, such as position x and momentum p, can be known.

https://en.wikipedia.org/wiki/Uncertainty_principle

OBSERVER EFFECT

…the term observer effect refers to changes that the act of observation will make on a phenomenon being observed. This is often the result of instruments that, by necessity, alter the state of what they measure in some manner.

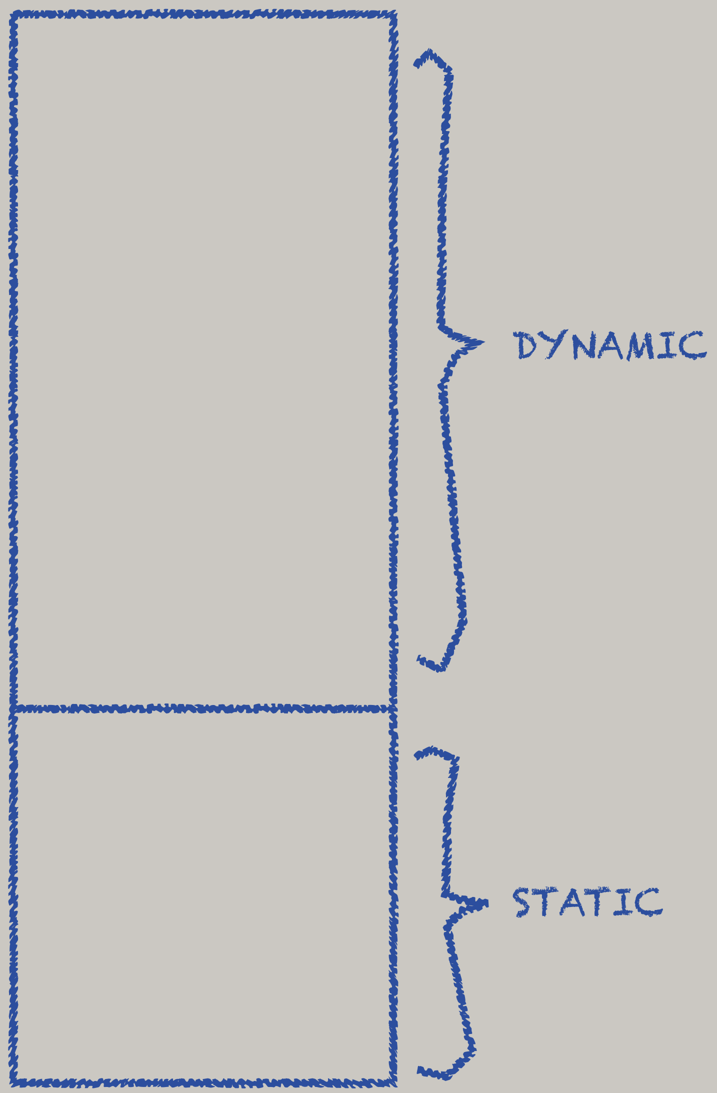
https://en.wikipedia.org/wiki/Observer_effect_(physics)

DEBUGGING TOOLS

- instrument code during compilation
- instrumented code may behave differently than uninstrumented code
- in other words: the act of using a debugger may mask a bug, causing its symptoms to disappear, only to reappear when run without instrumentation

MEMORY ORGANIZATION

MEMORY ORGANIZATION



Each process (a running program) has a chunk of memory at its disposal.

This memory is divided into "static" memory (allocated/structured before execution begins) and "dynamic" memory (allocated while the program executes.