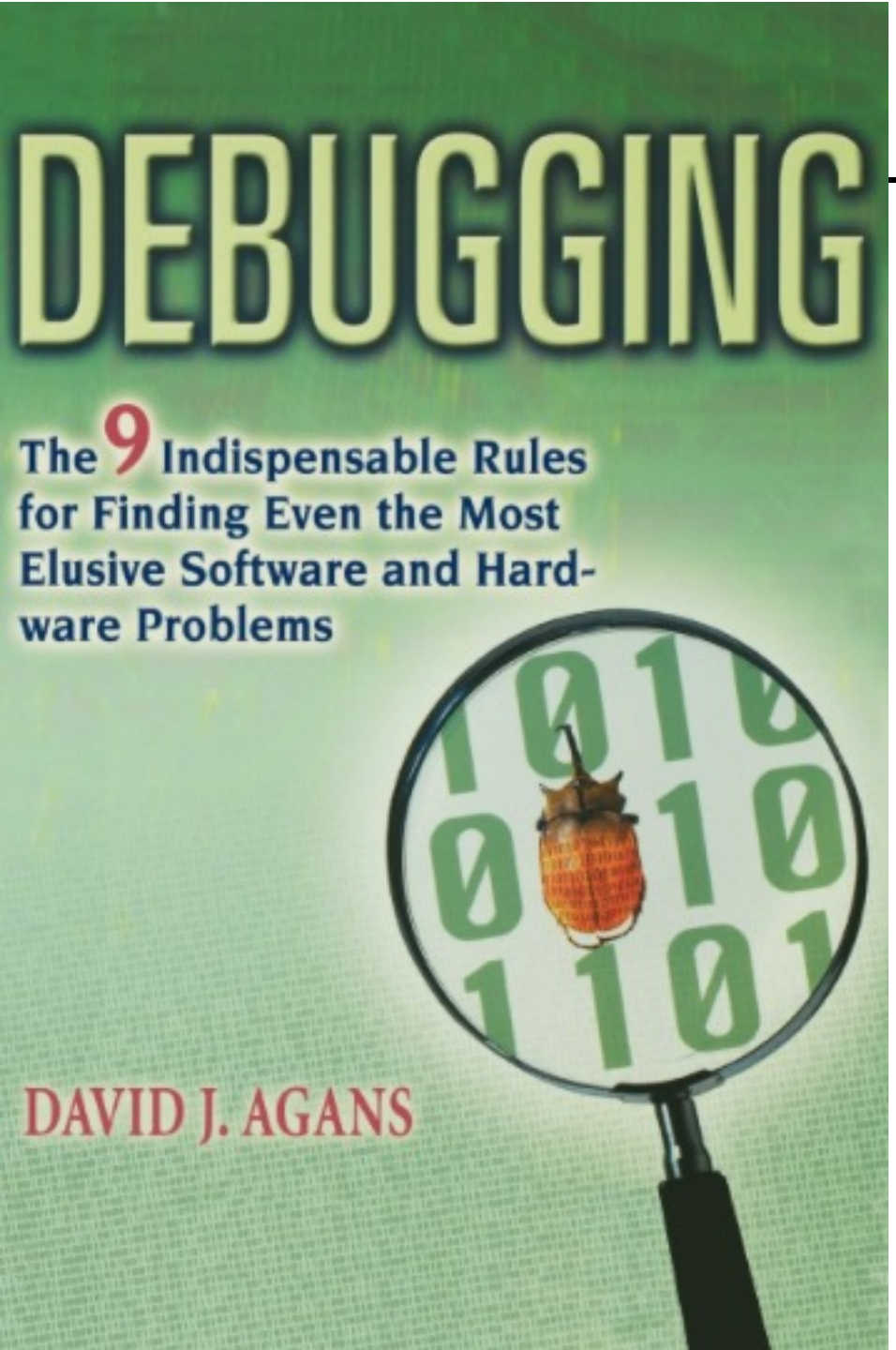

CIS 5710 is in many ways a class about **debugging**

9 Rules for Debugging (c/o David Agans)

- www.debuggingrules.com

1. Understand the system
2. Make it fail
3. Quit thinking and look
4. Divide & conquer
5. Change one thing at a time
6. Keep an audit trail
7. Check the plug
8. Get a fresh view
9. If you didn't fix it, it ain't fixed

goal: think about debugging as a **scientific** process
instead of a random one

1. Understand the system

- Need to know how system is supposed to behave
  - understand the specification
- e.g., LC4 CMP instruction treats inputs as *signed*
  - while CMPU treats them as *unsigned*
- Don't go overboard: don't need to know every detail of the ZedBoard for this class

2. Make it fail

- Can’t debug something that is not repeatable
- Bugs that are reliably triggered are really great!
  - Almost as good as having no bug in the first place
- If steps to reproduce are complicated, consider scripting
  - usually end up needing to reproduce the bug several times

3. Quit thinking & look

- See what the system is actually doing
  - Staring at code is typically not an efficient use of your time
  - sometimes documentation is wrong
- Developing hypotheses about a bug is good
  - but you need to test them!
- Debuggers are great
  - they show you what is actually going on
- printf/$display() is ok
  - faster to start with
    - upgrade to debugger when things get more complicated
  - sometimes printf is all you have
    - sometimes you don’t even have printf!

4. Divide & conquer

- Most systems have different processing steps
  - "garbage in, garbage out"
  - Try to determine where the garbage begins
- E.g., if I have an X output from some module, check the inputs to see if any are also X

5. Change one thing at a time

- If you only follow one rule, this is probably the one!
- If multiple things have changed, how do you know which one is responsible for a behavior change?
  - git is helpful: use branches, stash changes, or revert
  - maybe a change introduced a new bug!

6. Keep an audit trail

- track the **One Thing** that changed
  - what was changed
  - what I expected to happen
  - what actually happened
- sometimes these changes are useful later on!
- audit log format
  - can be very informal, 1-2 sentences often suffices
  - text files, google doc, Github issues

7. Check the plug

- Question your assumptions
  - are you measuring what you think you are?
- Are you running the code you think you are?
  - maybe the file isn't saved, or it's in another directory
  - add a printout, or change the output in a clear way
  - does your change have the expected effect?

8. Get a fresh view

- Talk to someone about your problem
  - go to office hours ☺
  - or talk to a domain expert or colleague
- Take some time off from the problem
  - your brain will keep working in the background

9. If you didn’t fix it…it ain’t fixed

- problems don’t magically go away
  - something changed!
  - if “the same code” works for me but not you, then it’s not the same!
  - often, our scope for the problem is too small
- When you think something is fixed, revert the fix
  - see that it fails again!
  - this provides strong evidence that the fix is responsible

0xA. Take time to reflect

- After the bug is fixed, don't just go onto the next bug!
- Think about:
  - Were there debugging rules that were helpful?
  - Were there debugging rules that I ignored, but shouldn't have?