Effective Programming Practices for Economists

Debugging

Strategies for debugging

Janoś Gabler and Hans-Martin von Gaudecker

Agans' rules

0. Get it right the first time
1. What is it supposed to do?
2. Is it plugged in?
3. Make it fail
4. Divide and conquer
5. Change one thing at a time, for a reason
6. Write it down
7. Be humble

Get it right the first time

- Avoiding debugging is better than being good at debugging
- Software engineering is all about that
- Your major tools are:
  - Unit testing
  - Error handling
  - Writing readable and modular code

What is it supposed to do?

- First step is knowing what the problem is
- "It doesn’t work" is not good enough
- What exactly is going wrong?
- How do you know?
- You will learn a lot by following execution in a debugger and trying to anticipate what the program is going to do next

Is it plugged in?

- Are you actually exercising the problem that you think you are?
- Are you giving it the right test data?
- Is it configured the way you think it is?
- Is it the version you think it is?
- Did you activate the environment?
- Why are you sure?
- Maybe the reason you cannot isolate the problem is that it is not there

Make it fail

- You can only debug things when they go wrong
- Find a test case that makes the code fail and simplify it as much as possible
- Use the scientific method: Formulate a hypothesis, make a prediction, conduct an experiment
- Each experiment becomes a test case

Divide and conquer

- The smaller the gap between cause and effect, the easier the relationship is to see
- Take the simplest test case that makes your code fail
  - Step through it with a debugger to learn what goes wrong
  - Write tests for untested functions that are called
  - Add error handling where necessary
- Use what you have learned to make your test case even simpler
- Repeat until the bug is located

Change one thing at a time, for a reason

- Most important: Don’t make things worse!
- Make a git commit before you start debugging
- Replacing random chunks of code is unlikely to help
- So always have a hypothesis before making a change
- Every time you make a change, re-run all of your tests immediately
- Undo changes that were not helpful
- Don’t work on any new features or simple other things while debugging

Write it down

- Science works because scientists keep records
  - Did $\gamma_1 = 0$ together with $x_1 = 50$ cause the crash?
  - Or was it $x_2 = -5$?
  - Or was γ_2 not set to the default value after all?"
- Records are particularly useful when getting help
  - People are more likely to listen when you can explain clearly what you did

Be humble

- If you cannot find it in 15 minutes, ask for help
  - Just explaining the problem aloud is often enough
  - "Never debug standing up." (Gerald Weinberg) — rushing makes things worse
- Do not keep telling yourself why it should work
  - If it does not, it does not
  - Never debug while grinding your teeth, either …
- Keep track of your mistakes
  - Just as runners keep track of their time for the 100m sprint
  - "You cannot manage what you cannot measure." (Bill Hewlett)