Mastering Programming

En Español. Chinese.

From years of watching master programmers, I have observed certain common patterns in their workflows. From years of coaching skilled journeyman programmers, I have observed the absence of those patterns. I have seen what a difference introducing the patterns can make.

Here are ways effective programmers get the most out of their precious 3e9 seconds on the planet.

The theme here is scaling your brain. The journeyman learns to solve bigger problems by solving more problems at once. The master learns to solve even bigger problems than that by solving fewer problems at once. Part of the wisdom is subdividing so that integrating the separate solutions will be a smaller problem than just solving them together.

Time

- **Slicing**. Take a big project, cut it into thin slices, and rearrange the slices to suit your context. I can always slice projects finer and I can always find new permutations of the slices that meet different needs.
- **One thing at a time**. We're so focused on efficiency that we reduce the number of feedback cycles in an attempt to reduce overhead. This leads to difficult debugging situations whose expected cost is greater than the cycle overhead we avoided.
- **Make it run, make it right, make it fast**. (Example of One Thing at a Time, Slicing, and Easy Changes)
- **Easy changes**. When faced with a hard change, first make it easy (warning, this may be hard), then make the easy change. (e.g. slicing, one thing at a time, concentration, isolation). Example of slicing.
- **Concentration**. If you need to change several elements, first rearrange the code so the change only needs to happen in one element.
- **Isolation**. If you only need to change a part of an element, extract that part so the whole subelement changes.

- **Baseline Measurement**. Start projects by measuring the current state of the world. This goes against our engineering instincts to start fixing things, but when you measure the baseline you will actually know whether you are fixing things.

Learning

- **Call your shot**. Before you run code, predict out loud exactly what will happen.
- **Concrete hypotheses**. When the program is misbehaving, articulate exactly what you think is wrong before making a change. If you have two or more hypotheses, find a differential diagnosis.
- **Remove extraneous detail**. When reporting a bug, find the shortest repro steps. When isolating a bug, find the shortest test case. When using a new API, start from the most basic example. “All that stuff can’t possibly matter,” is an expensive assumption when it’s wrong.
  - E.g. see a bug on mobile, reproduce it with curl
- **Multiple scales**. Move between scales freely. Maybe this is a design problem, not a testing problem. Maybe it is a people problem, not a technology problem [cheating, this is always true].

Transcend Logic

- **Symmetry**. Things that are almost the same can be divided into parts that are identical and parts that are clearly different.
- **Aesthetics**. Beauty is a powerful gradient to climb. It is also a liberating gradient to flout (e.g. inlining a bunch of functions into one giant mess).
- **Rhythm**. Waiting until the right moment preserves energy and avoids clutter. Act with intensity when the time comes to act.
- **Tradeoffs**. All decisions are subject to tradeoffs. It’s more important to know what the decision depends on than it is to know which answer to pick today (or which answer you picked yesterday).

Risk

- **Fun list**. When tangential ideas come, note them and get back to work quickly. Revisit this list when you’ve reached a stopping spot.
- **Feed Ideas**. Ideas are like frightened little birds. If you scare them away they will stop coming around. When you have an idea, feed it a little. Invalidate it as quickly as you can, but from data not from a lack of self-esteem.

can, but from data not from a lack of self-esteem.

- 80/15/5. Spend 80% of your time on low-risk/reasonable-payoff work. Spend 15% of your time on related high-risk/high-payoff work. Spend 5% of your time on things that tickle you, regardless of payoff. Teach the next generation to do your 80% job. By the time someone is ready to take over, one of your 15% experiments (or, less frequently, one of your 5% experiments) will have paid off and will become your new 80%. Repeat.

Conclusion

The flow in this outline seems to be from reducing risks by managing time and increasing learning to mindfully taking risks by using your whole brain and quickly triaging ideas.