The Programmer's Guide to Wisdom

@vedang

ESR: the UNIX Philosophy

Rule of Modularity :: Write simple parts connected by clean interfaces.
Rule of Clarity :: Clarity is better than cleverness.
Rule of Composition :: Design programs to be connected to other programs.
Rule of Separation :: Separate policy from mechanism; separate interfaces from engines.
Rule of Simplicity :: Design for simplicity; add complexity only where you must.
Rule of Parsimony :: Write a big program only when it is clear by demonstration that nothing else will do.
Rule of Transparency :: Design for visibility to make inspection and debugging easier.
Rule of Robustness :: Robustness is the child of transparency and simplicity.
Rule of Representation :: Fold knowledge into data so program logic can be stupid and robust.
Rule of Least Surprise :: In interface design, always do the least surprising thing.
Rule of Silence :: When a program has nothing surprising to say, it should say nothing.
Rule of Repair :: When you must fail, fail noisily and as soon as possible.
Rule of Economy :: Programmer time is expensive; conserve it in preference to machine time.
Rule of Generation :: Avoid hand-hacking; write programs to write programs when you can.
Rule of Optimization :: Prototype before polishing. Get it working before you optimize it.
Rule of Diversity :: Distrust all claims for the "one true way".
Rule of Extensibility :: Design for the future, because it will be here sooner than you think.

Doug McIlroy: the UNIX Philosophy

Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
Expect the output of every program to become the input to another, as yet unknown, program.
Design and build software to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

Rob Pike: Rules of Programming

No Speed Hacks :: You can't tell where a program is going to spend its time. Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.
Measure before tuning :: Don't tune for speed until you've measured, and even then don't unless one part of the code overwhelms the rest.
No Fancy Algorithms :: Fancy algorithms are slow when n is small, and n is usually small. Fancy algorithms have big constants. Until you know that n is frequently going to be big, don't get fancy.
Use Simple Data structures :: Fancy algorithms are buggier and harder to implement than simple ones. Use simple algorithms and simple data structures.
Data Dominates :: If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.

Gene Kim: The Five Ideals of DevOps

The First Ideal :: ***Locality and Simplicity***. Build simple, decoupled systems that can iterate in isolation.
The Second Ideal :: ***Focus, Flow and Joy***. Work in small batches with fast and continuous feedback.
The Third Ideal :: ***Improvement of Daily Work***. Make sure that daily work can be done with minimum impediments. Prioritize productivity over everything else.
The Fourth Ideal :: ***Psychological Safety***. Solving problems requires preventing problems, which requires honesty, which requires the absence of fear.
The Fifth Ideal :: ***Customer Focus***. Build only that which actually matters to our customers.

Gene Kim: The Three Ways of Doing Excellent Work

Flow :: Maximizing the rate of flow of work is the key to success. Limiting the work in progress is the fastest way to achieve Flow.
Fast Feedback :: Setup systems to get fast feedback at every stage of work, from concept through shipping to maintaining in production.
Experimentation and Learning :: Keep dedicated time for experiments, at every level of the company. A culture of innovation is necessary for achieving and maintaining Flow and Feedback.

Gene Kim: The Four Types of Work

Business Projects :: "Feature Work". This is the most visible type of work.
Internal IT Projects :: Release Automation, QA Automation, Developer Tooling and other internal enablers. Mostly un-tracked and invisible, but crucial to long-term success. Focus on this if you want to speed up Business projects.
Updates and Changes :: Generally generated from the above two categories of work. Ignoring this type of work increases the lead time of the prior categories.
Unplanned Work :: Fire-fighting at all levels of the company. Ruins planned work, so root causes need to be aggressively remediated.