Open Source Development Guidelines (ESR's Lessons)

1. Every good work of software starts by scratching a developer's personal itch.
2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
3. Plan to throw one [version] away; you will, anyhow. (Copied from Frederick Brooks' The Mythical Man Month)
4. If you have the right attitude, interesting problems will find you.
5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.
6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.
7. Release early. Release often. And listen to your customers.
8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.
9. Smart data structures and dumb code works a lot better than the other way around.
10. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.
11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.
13. Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away. (Attributed to Antoine de Saint-Exupéry)
14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.
15. When writing gateway software of any kind, take pains to disturb the data stream as little as possible—and never throw away information unless the recipient forces you to!
16. When your language is nowhere near Turing-complete, syntactic sugar can be your friend.
17. A security system is only as secure as its secret. Beware of pseudo-secrets.
18. To solve an interesting problem, start by finding a problem that is interesting to you.
19. Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

Open Source Benefits

1. **Customizable** – Ability to modify base level functionality to meet unique local needs
2. **Participate in project governance**: anyone can contribute to defining the direction of the project: technical aspects, functionality/features priorities, decision-making, community practice, etc.
3. **Organizational audibility**: ability to assess how well the open source community, project and governance is aligned with local (adopting organization's) needs/goals/expectations.
4. **Community audibility**: ability to assess what the level of shared knowledge and experience is within the community; number of participants, contributors and commiters there are in the project; what the level of adoption/deployments is.
5. **Business process audibility**: ability to evaluate how needs are identified and assessed, how work is prioritized, what the workflows and practices are, etc.
6. **Technical audibility**: ability to assess quality of code, architecture, development practices, etc.
7. **Reduced development time**: more contributors and contributions to solve problems, write code, test enhancements, document versions, etc.
8. **Avoid vendor lock-in**: not tied to, and controlled by any third party and their direction, interests, plans, migrations, enhancements, upgrades, sunsetting, integrations, dependencies, etc.
9. **Broader support options**: options range from local internal resources, to communities of practice, and multiple commercial vendors rather than a single commercial provider (the developer)
10. **Greater security**: vulnerabilities can be discovered more quickly (the more folks with access to the code, the more likely issues are to be discovered) with the best solutions applied to address issues (more folks involved can offer a greater variety of approaches, ideas solutions)
11. **Faster implementation**: no procurement process
12. **Higher quality**: meritocracy (best approach) is implemented
13. **Mitigates longterm risk**: no chance of discontinuing development/support due to purchase, or a new version (and thus forced migration to stay up to date with support/service contracts)
14. **Higher reliability**: bugs will be discovered and fixed more quickly (see, greater security)
15. **Business/operations continuity**: local organizational operations and practices are not disrupted through forced timelines for migrations, upgrades, enhancements etc.
16. **Professional (personal and organizational) development opportunities**: individuals can gain experience through participation and organizations can gain prestige through contribution.
17. **Try before you buy (Test drive)**: assess one, or many, options, before investing.
18. **Multiple instances (no per copy fees)**: can extend use on demand without additional costs or contract negotiations.
19. **Reduced acquisition costs**: no licensing fees
20. **Emphasizes concepts, not products**: end users are not tied to branded or copyrighted features, workflows, tools.
21. **Breaks the hardware upgrade cycle**: open source software is often designed to require less resource intensive hardware.
22. **Community access**: ability to meet and network with like minded people and organizations–peers.
23. **Standards based**: provides greater integration and interoperability (no proprietary specifications)
24. **Standards setting**: participants are often directly involved in the development of new standards, or learn of them first.
25. **Lower total cost of ownership (cumulative of other benefits)**: no licensing fees, no procurement process, competitive support contracts (even none with internal), no forced SP upgrades, re-purposed legacy hardware, greater scaling