The Cathedral and the Bazaar

There is two separate approaches models to build the software those are called as the Cathedral and Bazaar.   
 1) Cathedral Model   
 2) Bazaar Model

Cathedral: Cathedral software developed by group of software developers using a central plan strategy. After developing, identifying bugs, and implementing the maximum suited adjustments, they ultimately release a project after about a year. Comparable to how each detail is carefully created and placed before such a cathedral's gates open.

Cathedral model represent Software is created and use the Cathedral model, which would be restricted (the source code is not available), infrequently published, and professionally managed by users of the system. They have such a vision and a plan to build a spectacular cathedral. It has a more instructive tone.

Main features of cathedral Model:

1. Centralized control and command
2. An organization with different duties and responsibilities
3. Predefined processes and techniques
4. Re - usable parts and construction method
5. A focus on design and documentation
6. Stakeholders and groups operate together well.

Bazaar Model and Represent: Bazar often called as open source software and it is developed more individually. Individual developers can build on a basic kernel, add new features or correcting faults as they see suitable. It is basically program pooling. Famous depictions contain UNIX and Apache.

Main features of Bazaar:

1) Open collaborative and decentralized management

2) Flexible and adaptable techniques

3) Ongoing testing and development

4) Importance on engagement with and accomplishments to the community

5) User-driven development and fast repetitions

6) Resources are shared and code within an open-source license.

Raymond outlines characteristics that are suggestive of quality standards for developing open source software in his list of 19 "learned" from multiple software development initiatives.

Top five lessons I would like to highlight after reading the book

1. **A security system is only as secure as its secret. Beware of pseudo-secrets**

I choose this lesson because of Security will prevent from the maliciousness (unlike controls which prevent from risk). Do not even overprotect yourself; rather than, consider what a malicious person would also do.

1. **Smart data structures and dumb code works a lot better than the other way around**.

I Choose this lesson because with help of the data structures to build the complex algorithms to sort out the problems.

1. **Release early. Release often and listen to your customers.**

I choose this lesson because of if we release the software rapidly into market and we can get the feedback from users and if any bug is there in the code developers fix it ASAP.

1. **Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.**

I choose this lesson because of collaboration is very important between the developers and testers because they will find bugs and provide the fixes rapidly.

1. **The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better**

I choose this lesson because of good ideas always improve the software development skills and concepts understanding.