The Cathedral and the Bazaar

Cathedral: A large church, with a highly structured architecture.

Bazaar: A marketplace, with different stalls selling different goods.

In 1997 software developer Eric S. Raymond presented a paper called “The Cathedral and the Bazaar” looking at two different philosophies of writing and releasing open source computer programs:

- The “Cathedral” approach is a highly structured one, where software (and source code) is made available one version at a time, as “releases”, and in between releases it is worked on by an exclusive group of programmers.
- The “Bazaar” approach is a far more open one, where the source code (and all changes made to it) is publicly available online, and any programmer can register and contribute to the code between releases.

Raymond released a book in 1999 also titled “The Cathedral and the Bazaar” which included his original paper, and further essays on software development. The central idea of the paper and book is that the more programmers who can see code in development, the more likely that the majority of the errors (“bugs”) in it will be discovered. He formulates this as "*given enough eyeballs, all bugs are shallow*”, and refers to it as “Linus's law”, named in honour of Linus Torvalds, the initial creator of the Linux operating system, who used a “Bazaar” approach in the development of that system. Although there are some disagreements in the software development community as to whether or not this is a universal law, most of the empirical studies seem to provide support for Linus's law.

He also identified some key lessons that can help ensure the success of an open source software development process. Programmers should be interested in the project, they should base their first draft on some existing code, but should not be afraid to completely redraft their code (and redraft it again). They should release their code early and often, and should treat their users and testers as co-developers. When bugs are found in software, if there is a large group of programmers working on it, at least one of them will know how to fix that bug in a simple way. If the overall project can be divided into distinct and independent sub-projects, all of the developers don’t need to communicate with each other, so there’s less confusion.