First, I want to focus on software code quality (relative to larger topics, including problem definition, documentation, UX, design, etc.). High quality software is software with fewer errors (and a shorter tail to fix the remaining problems). There are many other things, such as code readability, maintainability, debuggability and so on, which can be easily classified into the quality umbrella. Let's focus on the core of the product running as expected.

I visualize the software development process as a conduit from ideas to products used by customers. There are other ways to imagine it. However, the most important thing is that it has to go through multiple steps to obtain available / delivered products.

The way to get high quality is quite simple. We need to find problems in the process of performing these steps.

I don't believe that there is a magical way to solve all problems. Therefore, we need to conduct defense in depth. We need to have multiple doors through this pipe, which should gradually filter out all problems. The more doors you have, the more likely you are to end up with problems.

Doors that find problems earlier in the process are better because it is cheaper to fix them earlier. Automatic doors are better than manual doors. Blocking doors that prevent you from entering the next stage are more effective than doors that sit on the side. Doors that capture a higher proportion of problems are also better.

OK, all the above contents are very general / vague descriptions, which are very useful for a theoretical book on quality. However, without specific instructions, it is useless. Give me an insight into the real content that applies to most bread and butter software companies.