Getting Clean via Emergent Design
Kent Beck's four rules of Simple Design give great advice for making good software. First, test everything to make sure it works right. Second, get rid of repeating stuff to make things clearer and less likely to mess up. Third, make sure your code shows what you're trying to do, so others can understand it easily. Lastly, keep things simple by not having too many classes or methods. Following these rules helps make software better and makes it easier to apply other important principles like SRP and DIP.

Simple Design Rule 1: Runs All the Tests
This passage talks about how important it is for systems to be easily tested. It says that a good system should work well and be able to be checked through testing. To make testing easier, it suggests making smaller classes that each do one thing well. It also suggests using certain principles and tools to make sure different parts of the system are not too closely connected. By testing systems regularly, it helps make sure they work properly and are well-designed, following important programming goals.

Simple Design Rules 2–4: Refactoring
This passage talks about how important it is to have tests when writing code. Tests help keep the code clean and organized. By regularly updating and reviewing the code's structure, developers can make it better over time without worrying about breaking it, because the tests catch any mistakes. When updating the code, they can follow some key rules like keeping things together that belong together, separating different tasks, and avoiding repeating the same code. Following these rules helps make the code easier to maintain and use.

No Duplication
This passage talks about how it's really important to get rid of repetition when designing systems. This repetition can happen in different ways, like having the same lines of code or similar ways of doing things. When we find and fix this repetition, it makes systems neater and works better. The passage gives examples of how to fix this repetition, both in small and big ways, like by using methods or patterns. Getting rid of repetition helps systems be easier to work with, easier to update, and able to handle more stuff, which means the code is better and it's faster to develop.

Expressive
The passage says it's super important to write code that's easy to understand. That way, it's cheaper to maintain and has fewer mistakes. It talks about how using clear names for things, keeping functions and classes short, and following common coding rules helps other developers understand your code better. It also says that writing tests for your code is like writing instructions for how it should work, and it encourages developers to feel good about making their code easy to read and work with. Basically, the main idea is that writing clear and

well-designed code is really important for working together well and keeping software projects going strong in the long run.

Minimal Classes and Methods

The passage talks about how important it is to find balance when designing software. It says that while we should keep things like class and method sizes small, we shouldn't overdo it because that can make things more complicated. It warns against blindly following strict rules, like making an interface for every class, and suggests being practical instead. The main focus should be on making the system simple, while still keeping classes and functions manageable. It also mentions that while it's good to have fewer classes and functions, other things like testing and making sure the code is clear are more important.