Code quality is important. Code that meets quality standards is easier to maintain and extend, which minimizes technical debt and maximizes agility.

Speak the language
Naming things is one of the most difficult aspects of software development. That's why agreements and standards on naming are essential to a team's success. When a team uses a common, rigorous language, they find that code is much easier to read and maintain. This language should be based on the domain model used in the software – hence the need for it to be rigorous, since software doesn't cope well with ambiguity.

Document things
One of the core Agile values is "Working Software Over Comprehensive Documentation". This does not mean: no documentation. Apart from documenting standards, it is a good practice to formally document code (e.g. interfaces). This helps team members work with code that they themselves have not written.

Clean code
Clean code does exactly what it promises and can be extended without modification. Good practices such as the Single Responsibility Principle prevent code smells and ensure that developers can respond to change.

Levels and check points of code quality

The levels for *Code quality* are typified as follows:

- **Forming**: *Coding standards are used*
- **Norming**: *Tooling verifies that code lives up to standards*
- **Performing**: *SOLID principles are used to keep code clean*

Please find the checkpoints below.

Forming
1. The whole team uses the same naming convention. 2. There is an agreed standard on other formatting (e.g. bracket on newline or not, indentation). 3. Code is sufficuently formally documented (e.g. Javadoc annotations for the interfaces, but not inline). 4. All naming is related to the problem domain.
Norming

1. Tooling automatically verifies the agreed standards.
2. Tooling automatically checks for language specific code smells.
3. External dependency management is used consciously.

Performing

1. Single responsibility principle is applied.
2. Open/closed principle is applied.
3. Dependency inversion principle is applied.
4. Non-functional requirements are taken into account while writing code.

Terug naar I4Agile