# What is Agile Design?

A set of UML diagrams may represent parts of a design, but those diagrams are not the design. The design of a software project is an abstract concept. It has to do with the overall shape and structure of the program, as well as the detailed shape and structure of each module, class, and method. The design can be represented by many different media, but its final embodiment is source code. In the end, the source code is the design.

## The Odors of Rotting Software

* **Rigidity** - is the tendency for software to be difficult to change, even in simple ways. A design id rigid if a single change causes a cascade of subsequent changes in dependent modules. The more modules that must be changed, the more rigid the design.
* **Fragility** - is the tendency of a program to break in many places when a single change is made. Often, the new problems are in areas that have no conceptual relationship with the area that has been changed. Fixing those problems leads to even more problems, and the development team begins to resemble a dog chasing its tail.
* **Immobility**. A design is immobile when it contains parts that could be useful in other operating systems, but the effort and risk involved with separating those parts from the original system are too great. This is unfortunate but very common occurrence.
* **Viscosity** comes in two forms: viscosity of the software and viscosity of the environment. When faced with a change, developers usually find more than one way to make that change. Some of the ways preserve the design; other do not (i.e., they are hacks). When the design-preserving methods are more difficult to use than the hacks, the viscosity of the design is high. It is easy to do the wrong things but difficult to do the right things. We want to design our software such that changes that preserve the design are easy to make. Viscosity of environment comes about when the development environment is slow and inefficient.
* **Needles Complexity**. A design smells of needless complexity when it contains elements that aren't currently useful. This frequently happens when developers anticipate changes to the requirements and put facilities in the software to deal with those potential changes. At first, this may seem like a good thing to do. After all, preparing for future changes should keep our code flexible and prevent nightmarish changes later. Unfortunately, the effect is often just the opposite. By preparing for many contingencies, the design becomes littered with constructs that are never used. Some of those preparations may pay off, but many more do not. Meanwhile, the design carries the weight of these unused design elements. This makes the software complex and difficult to understand.
* **Needless Repetition**. Cut and paste may be useful text-editing operations, but they can be disastrous code-editing operations. All too often, software systems are built on dozens or hundreds of repeated elements. When the same code appears over and over again, in slightly different forms, the developers are missing an abstraction. Finding all the repetition and eliminating it with an appropriate abstraction may not be high on their priority list, but it would go a long way toward making the system easier to understand and maintain. When there is redundant code in the system, the job of changing the system can become arduous. Bugs found in such a repeating unit have to be fixed in every repetition. However, since each repetition is slightly different from every other, the fix is not always the same.
* **Opacity** – is the tendency of a module to be difficult to understand. Code can be written in a clear and expressive manner, or it can be written in an opaque and convoluted manner. Code that evolves over time tends to become more and more opaque with age. A constant effort to keep the code clear and expressive is required in order to keep opacity to a minimum.

## Why Software Rots

In non-agile environments, design degrades because requirements change in ways that the initial design did not anticipate. Often, these changes need to be made quickly and may be made by developers who are not familiar with the original philosophy. So, though the change to the design works, it somehow violates the original design. Bit by bit, as the changes continue, these violations accumulate until malignancy sets in.

We must find a way to make our designs resilient to such changes and use practices that protect them from rotting.

An agile team thrives on change. The team invests little up front and so is not vested in an aging initial design. Rather, the team keeps the design of the system as clean and simple as possible and backs it up with lots of unit tests and acceptance tests. This keeps the design flexible and easy to change. The team takes advantage of that flexibility in order to continuously improve the design; thus, each iteration ends with a system whose design is as appropriate as it can be for the requirements in that iteration.

## Conclusion

Agile design is a process, not an event. It’s the continuous application of principles, patterns, and practices to improve the structure and readability of the software. It is the dedication to keep the design of the system as simple, clean, and expressive as possible at all times.