Symptoms of Poor Software Design?

1. Rigid.
2. Fragile.
3. Immobility.
4. Viscosity.
5. Duplication of code.
6. Needless Complexity.

What is Rigid Software?

- Rigid software is difficult to change, even in simple ways.
- Software is rigid if a single change causes a series of subsequent changes in other areas of the application.
- It makes it difficult for us to give a reasonable estimate of the work required
- If you find yourself saying "It was more complicated than I thought", chances are you are dealing with rigid software.

What is Fragile Software?

- Fragile software breaks in many places when a single change is made.

- A single change introduces unexpected results in other areas of the application.
- Fragile software usually needs to be redesigned, but usually nobody wants to do it or there is no time for redesign.
- If it is not redesigned, they usually get worse the more you fix them.

Immobile Software?

- When there are parts of a design that could be useful in other systems, but there is too much risk in separating those parts from the original system.

Viscosity?

- A Viscous project is one in which the design of software is difficult to preserve.
- Viscosity comes in two forms: viscosity of the software and viscosity of the environment.
- Some software changes preserve the current design, others are hacks.
- When design preserving methods are more difficult to reuse than hacks, the viscosity of the software is high.
- We to design software such that changes that preserve the design are easy to make.
- Viscosity of the environment is the development environment is slow and inefficient.
- Examples are long compile times and inefficient source control systems.

Duplication of Code?

- Cut and paste
- Usually results in the same code happening over and over again in slightly different forms.
- Bugs found in redundant code, have to be fixed in every repetition.
- This can be very frustrating, because each repetition can be slightly different and the fix is not always the same.
- Code duplication can be fixed by introducing an abstraction, which will make the code easier to understand and maintain.

Needless Complexity?

- This is when the code contains elements that are not currently useful.
- Usually happens when we try to put in facilities to anticipate some potential changes.
- Some of the changes may pay off, many may not.
- This results in software that carries the weight of many unused sections, which can make it difficult to understand.

Some Object Oriented Principles that can help prevent symptoms of bad design.

- Single Responsibility Principle.
- Open/Close Principle.
- The Dependency Inversion Principle.
- Interface Segregation Principle.

These principles are not a product of a single mind but represent the integration of a large number of software developers and researchers.

Single Responsibility Principle?

- There should never be more than one reason for a class to change.

Open/Close Principle?

- Software entities should be open for extension but closed for modification.
- Open for extension – We can make the application respond to changes in requirements or new additions.
- Closed for modification – Adding behavior does not result in changes in original code.