UCAR Software Engineering Assembly
University Corporation for Atmospheric Research

What is software design?

Software design is the process of defining software methods, functions, objects, and the overall structure and interaction of your code so that the resulting functionality will satisfy your users requirements. See our Requirements page to learn how to write requirements. There are many different ways of designing software, almost all of which involve coming up with an initial design and refining it as necessary. Different developers prefer different amounts of design up front or during implementation phase. Generally the overall design should be well thought out and reviewed before coding starts. Refer to our section on Design reviews to learn how to review your design. It is easier to try out different designs up front and discover problems early in the development cycle than to make a major design change after much of the code has been written.
Your software design should include a description of the overall architecture. This should include the hardware, databases, and third party frameworks your software will use or interact with. This is the big picture of what is running where and how all the parts will interact.
Your software design should include all Application Programming Interfaces (API) that are used by your code or by external code that calls your code. The correct definition of these APIs is very important because once they are agreed to and used it is extremely difficult to change them without breaking lots of other peoples code. This link describes how to design a good API.

- http://www.infoq.com/news/2006/11/joshua-bloch-API-design
  One extreme approach to software design is to come up with a simplistic design and implementation, and extending/refactoring it gradually to include more of the requirements. This solution involves a lot of refactoring and can sometimes make it difficult to keep track of the bigger picture. This tends to provide an initial solution that works for a limited set of requirements.
  An approach on the other extreme is to attempt to design as much as possible before implementation. This approach requires a fairly complete understanding of the requirements, but tends towards creating a solution that fits the needs of the entire system over the needs of each sub-component.
  There are many good choices between these two extremes. Try to think of the overall big issues up front and create a design that addresses those problems. As you start to implement your design you may discover new problems that alter or add to your design and you may have to refactor your code to account for these changes, however these changes should be small. Remember the purpose of designing the software before coding starts is to make sure everyone is working together and building the same thing.

WHAT IS SOFTWARE MODELING?

By software modeling we do not mean expressing a scientific theory or algorithm in software. This is what scientists traditionally call a software model. What we mean here by software modeling is larger than an algorithm or a single method. Software modeling should address the entire software design including interfaces, interactions with other software, and all the software methods.
Software models are ways of expressing a software design. Usually some sort of abstract language or pictures are used to express the software design. For object-oriented software, an object modeling language such as UML is used to develop and express the software design. There are several tools that you can use to develop your UML design.
In almost all cases a modeling language is used to develop the design not just to capture the design after it is complete. This allows the designer to try different designs and decide which will be best for the final solution. Think of designing your software as you would a house. You start by drawing a rough sketch of the floor plan and layout of the rooms and floors. The drawing is your modeling language and the resulting blueprint will be a model of your final design. You will continue to modify your drawings until you arrive at a design that meets all your requirements. Only then should you start cutting boards or writing code.
Again the benefit of designing your software using a modeling language is that you discover problems early and and fix them without refactoring your code.
How do the different methodologies handle software design and modeling?
Please read the following:
[The Rational Unified Process (RUP)]
[Agile]

REFERENCES

- *UML Distilled: A Brief Guide to the Standard Object Modeling Language*
  by Martin Fowler
- *UML 2.0 in a Nutshell* by Dan Pilone
- *Design Patterns: Elements of Reusable Object-Oriented Software* By Richard Helm, Ralph Johnson, and John Vlissides