Software Design Skills

150 points to have better software design skills

By
S K Srivastava

Connect with me:

LinkedIn
YouTube
Twitter

Version 1.1
Purpose : Share software design skills with engineers to have good software design skills
Author : Suresh Kumar Srivastava
Date : 12 July 2022

1.
Software design is a thought process. You just have to start the thought process.

2.
Thought process to design can be alone and good if with team.

3.
Don't worry of initial design, it will be refined again and again.

4.
You just have to think about classes, relationship and how they will work together.

5.
Do your thought process to design with paper & pencil or UML tool.

6.
Understanding of general real life system increases the software design capability.

7.
If you can groom the team with better thought process then its all "DONE".

8.
Think of flexible and extensible design which can accommodate future requirements.

9.
Factory Method Design Pattern video
https://www.youtube.com/watch?v=Q4ktJLM4zaQ

10.
If design is done properly then absolutely "NOT MUCH" left for programming.

11.
Good design of software makes its life longer.

12.
Think of design as solution in form of library or framework.

13.
Cost of design issue is way higher than programming/bug fixing.

14.
Think of reusable design, they are great asset for any organization.

15.
Having a good team of software designers is a game changer.

16.
OO examples are full of Employee, Animal, Student, Car. "NONE" helps when you do real software design.

17.
Abstract Factory Design Pattern video
https://www.youtube.com/watch?v=RDlNwJsX6Qg

18.
Remember these 3 for a good design - Flexibility, Extensibility, Reusability.

19.
Make no mistake, Designing flexible, extensible and reusable software is very very very difficult.

20
The biggest mistake we do is not recording a good design.

21.
It takes a long time to learn good object-oriented design.

22.
Reuse solutions that have worked in the past.

23.
Think about changes in design now or bear the high cost of redesign tomorrow.

24.
Having a thought on how the system can change in future will make a design better.

25.
Redesign affects the product, small part or whole depends on changes.

26.
The question of good design lies heavily on anticipation of changes.

27.
Who, What, Where... is your tool for software design.

28.
Side effect is the most worrisome part of changes in design.

29.
Requirements change, but design should not.

30.
If you feel your design is not right, that means its not right. Look at it again.

31.
Good design requires keeping the big picture in mind.

32.
Have a habit to identify the design solution, record them and use them appropriately in future.

33.
You have to protect your design from changes.

34.
Changes may be unknown, but where to accommodate in design may be known.

35.
Isolate the changes in parts of design.

36.
Think how changes in one part will not affect other part.

37.
Hide the implementation from client.

38.
The hardest part in design is finding the right classes.

39.
Program to an interface, not an implementation.

40.
Favor object composition over class inheritance.

41.
Find what varies and encapsulate it.

42.
Delegation is wonderful, delegate it.

43.
Delegation is a tool in hand to make the design reusable.

44.
Delegation makes composition excellent for reuse.

45.
Builder Design Pattern
https://www.youtube.com/watch?v=Uqdx6ytRcqY

46.
Think about reusability at 3 levels - Application, Library, Framework

47.
Its a design thought process to see and visualize.

Someone is doing something.
Someone can do something.
Someone is changing.
Someone can do something in future.
Someone will do something in future.

Someone is doing something with someone.
.....
.....

48.
It works. Is it right? Think again.

49.
Think about encapsulating behaviors.

50.
Software design capability improves with ability to raise your level of thinking.

51.
Reuse is the organization culture. You are the change.

52.
DO NOT, I repeat DO NOT make me complex, I will die with my own complexities.

53.
Avoid large inheritance hierarchies.

54.
Always keep in mind the business objectives of product/organization.

55.
I don't want my parent properties, I want "is substitutable for".

56.
DO NOT live only in world of technology, see the big picture.

57.
Avoid subclassing to extend functionality.

58.
Look for common behaviors.

59.
Look for specific implementation of common behaviors.

60.
DO NOT touch, its working. How long?

61.
Make me flexible but 360 degree is a BIG NO.

62.
The power lies in base class to derived class and derived class to base class.

63.
If I require a cup of tea, DO NOT give me a kettle.

64.
Singleton Design Pattern video.
https://www.youtube.com/watch?v=Z53KOPQ2nGc

65.
Avoid having things complex. Someone else has to maintain it.

66.
Think about long term with current objectives.

67.
Avoid having too much dynamic, it will be difficult to understand.

68.
Dependency is the big cause of redesign.

69.
Touch me not, I am omnipresent. Isolate me.

70.
Noone understands me, I will slowly die.

71.
I am too much generalized to be understood.

72.
I can't be changed, Oh I will be outdated.

73.
Technologies are out of question for me.

74.
DO NOT generalize me too much, I am application, not the framework.

75.
Good design is the assurance of better maintenance.

76.
Think how it will be used, NOT how it will be implemented.

77.
DON'T think about all possibilities, think about how changes will be accommodated.

78.
STOP saying inheritance is for reuse, its more about "is substitutable for".

79.
Changes will be from start of product development, only good design will be your friend.

80.
I am interface, and I don't want you to know the implementation.

81.
You know only me (interface), enjoy adding new implementation.

82.
Simplicity is superb, DO NOT underestimate it.

83.
Good design is the future of product, invest wisely.

84.
Design should influence to have better understandable code.

85.
Identify the dynamic behavior in design.

86.
While designing, I am only a designer.

87.
Good understanding of DSA increases design capability.

88.
Design SHOULD NOT have any influence of programming.

89.
Flexible software design is about how easily you can accommodate the changes.

90.
Dynamic class loading helps you to have flexible software design.

91.
Think of configuration to have flexible software design.

92.
Prototype Design Pattern video
https://www.youtube.com/watch?v=jJRDJu9wUyY

93.
Think of hooks to have flexible software design.

94.
Can I replace you? Yes my friend I am flexible.

95.
Capability to replace/change is heart of flexible software design.

96.
Can I replace this algorithm? When did I say, I am not flexible.

97.
Can I replace this class? Yes sir, I said I am flexible.

98.
Can I change this representation? Why not, You only made me flexible.

99.
Can I replace this library? Yes, I am designed to be flexible.

100.
Google "Software Peter principle" and thank me.

101.
What's the "Software Peter Principle"?

The Software Peter Principle is in operation when unwise developers "improve" and "generalize" the software until they themselves can no longer understand it, then the project slowly dies.

—From the book C++ FAQ by Marshall Cline, Greg Lomow, Mike Girou

102.
Extensible software design is about how easily you can add new functionality.

103.
Inheritance provides a way to have extensible software design.

104.
is-a relationship ("is substitutable for") is looked for extensible software design.

105.
Generic code powers seamless extensible software design.

106.
Dynamic binding does the magic for extensible software design.

107.
Can I add this? Yes my friend I am extensible.

108.
Can I add this code? Why not, I am extensible.

109.
Can I add this class? Oh Yes, You only made me extensible.

110.
Can I add this configuration? Don't ask me, I am extensible.

111.
Can I add this module? Do it, I am extensible.

112.
Can I add these parameters? Yes sir, I am extensible.

113.
Think of hooks to have extensible software design.

114.
Think of plugins to have extensible software design.

115.
Think of customization to have extensible software design.

116.
Think of modules to have extensible software design.

117.
Think of components to have extensible software design.

118.
Decouple abstraction and implementation to have extensible software design.

119.
Empty methods helps you to facilitate extensible software design.

120.
Think of default methods to have good design.

121.
Can we think of changing object responsibilities.

122.
Abstraction encapsulates variations.

123.
Design Patterns are your friend and say what worked well again and again.

124.
Antipatterns are your good friend and say what not to do, if done then how to make it right.

125.
Learning what NOT to DO is very very very important, explore the Antipatterns.

126.
Inheritance for reuse is bad inheritance.

127.
Big user community is a good parameter to select the technology.

128.
Right team is key to success of any project.

129.
Software design is key to programming and testing.

130.
I (software design) am victim of my own success.

131.
Demand of more has to be satisfied by good design.

132.
Software design die if it is not able to manage the success of product.

133.
Changing business behavior requires flexibility.

134.
Design Patterns In Java
http://coursegalaxy.com/design-patterns/java.html

135.
Software design has huge impact on development & maintenance. Invest your time on design wisely.

136.
Good Regression Test Suite is very important to know the impact of changes.

137.
Making things complex will not make your job permanent.

138.
Debugging is all about localizing the problem to fix.

139.
Analysis and Debugging is most important for Product maintenance. Fix/Programming is just outcome of it.

140.
Good Regression Test Suite is heart of Product maintenance/release. Never ever compromise on it.

141.
Debugging skill should be in starter kit of new engineers for any product organization.

142.
Finding appropriate solution is most important for Product development/maintenance. Programming/Fix is just outcome of it.

143.
Finding root cause solves multiple problems/problems to come in future for product. "Never ever compromise on it".

144.
Debugging is addiction. Addiction to find the root cause of problem.

145.
Debugging provides X-ray of product to have better analysis.

146.
Debugging is torch-bearer to show the path for better analysis and finding root cause.

147.
Analysis and Debugging skills is most important to look for hiring engineers for product organization.

148.
The curiosity to explore "how it works" increases the analysis capability.

149.
Design should address how old code will call new code.

150.
Design should address how existing code can be reused.

