Introduction to Software Design

Outline:

- 💡1. What is Software Design?
- 💡2. Design concepts
- 💡3. Design considerations
- 💡4. Software Modeling
- 💡5. Software development challenges

1. What is Software Design?

Software design is a part of the Software Engineering process. **It is a process of implementing software solutions to one or more set of problems**.
Someone analyses the requirements finds the gap and then go back to the customer to clarify what the type of product the client is looking for, then the design team handles the technical side of things. This team will come up with a design.
Software design is twofolded: there is a high-level design and then a low-level design.

- At the high-level you are looking at what type of tech stack you would like to be using based on the product.
- Then you can look at the finer details of your software design, from a lower level (the more technical and geeky stuff).

So what are some other ways of defining Software Design?
(1) A process of converting the software requirements analysis (SRA) to list specifications used for software delopment to solve problems.
(2) Specifications could be as simple as **bow chart**, **UML diagram**
(3) The main difference between software analysis and design is that the output of a sotware analysis consist of smaller problems to solve.

Software Design vs. Analysis

- Similar action but diff output.
- Design is focused on **solutions to the problem** as whole that may consist sub-problems.
- Analysis consist of **smaller subproblems** to solve.
- Analysis must be same to the multiple designs to the same problem.
- Design may be **platform-independent** or **platform-specific**, depending on the availability of the technology used for the design.

2. Design Concepts

"If you don't understand Design Concepts then you won't understand Software Development"

Design concepts are a foundation from which more sophisticated methods can be applied.

Abstraction and Refinement

🧠***Abstraction*** - A process or result of generalization by reducing the information content. Only practical and relevant information is retained.

- Hiding lower levels of detail, moves from lower to higher levels.
  🛠***Refinement*** - More specific to a certain process. Complementary to abstraction.
- Movement of high levels of detail to lower levels of detail.

Software Architecture

Software Architecture - Overall structure of the software, a good architecture will yield to good quality product. Effectiveness is measured in terms of performance, quality, schedule and cost.

Modularity - How software is divided into smaller components called ***modules.*** It's the separating of functionality of a program into interchangable sections of code.

- Cleaner and streamlined solutions.
- Allows you to flexibly add or remove code.

Structure

Hierarchy - A program structure that represents the organization of a program component.
Structural Partitioning - The program structure can be divided both horizontally and vertically/

- Horizontal partitions define separate branches of modular hierarchy.
- Vertical partitions suggest that control and work should be distributed **top down**.
  Data Structure - It is a representation of the logical relationship among individual elemtns of data.
  Software Procedure - It focuses on the processing of each modules individually.
  Information Hiding - Hide implementation details.

3. Design Considerations

There are many aspects to consider in the design of a piece of software based on the goals the software. Some of these aspects are:

- ***Compatibility*** - The software works with other products.
- ***Extensibility*** - New capabilities can be added to the software without major changes to the architecture.
- ***Fault-tolerance*** - The software is resistant to and able to recover from component failure.
- ***Maintainability*** - A measure of how easily bug fixes or functional modifications can be accomplished.
    - High maintainability can be the product of modularity and extensibility.
- ***Modularity*** - Smaller modules for each individual task.
    - Easy to test, debug, reuse, and maintain.
- ***Reliability*** - The software is able to perform a required function under stated conditions for a specified period of time.
- ***Reusability*** - It can be reused in other applications and can be extended easily.
- ***Robustness*** - the Software is able to operate under stress or tolerate unpredictabe or invalid input.
- ***Security*** - The software is able to withstand hostile acts and influences.
- ***Usability*** - User friendly and self explanatiory. Default values for the parameters.
- ***Performance*** - The software performs its tasks within a user-acceptable time.
- ***Scalability*** - The software adapts well to increasing data or user base.

4. Software Modeling Language

Software models are used to represent software design.

- It's a graphical view that gives an approachable and informative outline of a design.
  Textual and/or Graphical languages are used to express the software design.
  Each diagram has a certain consistent set of rules.

UML

Unified Modeling Language (UML) is the **most commonly used** software modeling language.
UML is a general modeling language to describe software both structurally and behaviorally.

- A standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

5. Software Development Challenges

Software is:

- Used for a long time.

- Updated and maintained by people who did not write it ~~ a different team will inherit the same code from a past team.
  Software requirements specification:
- Initially may be incomplete.
- Clarified through extensive interaction between user(s) and system analyst(s).
- Needed at the beginning of any software project.
- Designers and users should both approve it.

Requirements change!

- Users needs and expectations change over time.
- Software uses reveals **limitations** and **flaws**
- Desire for increased convenience, functionality.
- Desire for increased performance.

Environments Change!

- Hardware, OS, software packages will change freqeuntly with updates.
- Need to intteract with clients, parent org., etc.
- Law and regulation changes.
- Ways of doing business.
- Trends will affect style and influence your software.

Other Challening Factors:

- Resources can change: time, budget, expenses
- Organizational changes: people come and go, goals change, companies merge, and acquisitions
- Technological constraints: OS, software languages, frameworks, databases, security, everchanging technology, innovations.

Risks In Development

Write code based on what you know.
Minor changes can become a can of worms with a wide impact on your software.

- You probably did not do a good job in the designing of your software if a minor change breaks the functionality of your software.
  Deadlines can easily get affected due to bugs and roadblocks.

Efforts to Minimize Risk

1. Understand that change is inevitable
2. No surprises
3. Feedback is critical.
4. Frequent feedback is necessary. **"Ask questions"**
5. You want to know right away if you broke the code isn't it?
6. ***Test to break it break to test it.***
    - Test every single piece of code you have written. Take a break to test what you have written. NEVER NEVER overook a potential bug in any line of your code.

Motivation to Good Design

1. A good design leads to a good product.
2. In Engineering Construction is expensive, Design is relatively Cheap
3. In Software Development Construction is Cheap and design (which involves modeling and coding) is expensive.
4. Can't we quickly test our design (since construction is cheap)?
5. Testing is the Engineerign Rigot in Software Development.