*Exploring Software Design*

#### Gangadhara Matti

#### Computer Software Engineering

#### Arizona State University, Mesa, USA

#### [gmatti1@asu.edu](mailto:gmatti1@asu.edu)

*Abstract*—Decision making is an important aspect in Software Development Life Cycle. Decisions which are made during architectural or design phase of the project has direct impact on the quality and other aspects of the software. Hence it is of at most importance to consider the design phase properly to avoid major havocs in the development life cycle of the project in the later phase. This paper presents a detailed approach that could be considered for Software Design. Additionally, also discusses about the challenges in each phase and standard methods that are available to solve those problems.

Keywords—Software Design, Software Design Process, Software Design Strategies.

# Introduction

Even though implementation phase in the Software Development Life Cycle seems the most important for actual completion of the project, the project planning and design in the initial phases of the Software Development Life Cycle plays a vital role in delivering products of high quality and minimizing failures. According to Surasak K and Pornpit W [1] 50% of all software projects are total failures and 40% are partial failures or challenged and rest 10% of the projects which results in successful delivery. The main reason being improper complexity estimation which in turn is affected by bad design of the project – as a result of which more money, time and resources are wasted to refactor the project [2]. This paper is structured to give an overview of the breakdown of existing software design topics.

# software design process

In more abstract form a design is nothing but solving a huge problem. This is a tedious task if considered big chuck. It more advisable to follow divide and conquer approach – by dividing the problem into number of independent stand-alone entities. Separating the concerns adds on huge advantage, not only it reduces the problem size. It allows us to specifically focus on a subset of the problem. The chapter 2 of the book [3]– explains the different possible divisions that we could make and the topics that we encounter while planning for a design of a product. The breakdown of Software Design topics is as explained in the following sections.

Section 1 discussed about “Software Design Fundamentals”, section 2 talks about Key Issues that are faced in Software Design Process, section 3 discussed about the structure and architectural styles that we could make use of based on our requirements. Section 4 explains how User Interface Design can be approached. Section 5, 6 and 7 discussed about Software Design Quality and Analysis Evaluation, Software Design Strategies and Methods and Software Design Tools respectively. Each section has a sub-category based to explain the steps in detail.

## Software Design Fundamentals

This section presents the concepts, terminologies, techniques required to understand software design. The article [4] defines all the necessary definitions of the Software Design. We will discuss them as and when needed.

## General Design Concepts

It is more import to understand the design concepts which come across Software Development Life Cycle, it helps to better judge and take proper decisions along the process.

## Context Of Software Design

It is important to know that the Software Design is the most important phase in Software Development Life Cycle. The journal [5] explains clearly the process of different SDLC, where the Software Design fits in. According Nishta Singh [5] the software design is the first phase of SDLC. In waterfall, first requirements gathering, post which is Software Design. In Agile Sprint 0 is where most of the design is taken care. So, it is extremely important to understand where the software design fits in different Life Cycle Models.

## Software Design Process

Software design is where the user requirements are transformed into suitable form, which helps the team or programmer in Software coding and implementation. This is where the problem domain is transformed into solution domain. There are different levels in Software Design which clearly explained in [6], the important ones are listed below.

* Architectural Design: This is the more abstract version of the Solution. This is where, the different components of the system are identified which yields communication between them. This is the step where designers get the idea of proposed Solution domain.
* High-level Design: This is the step which defines how the different components of the system can be implemented as modules and how do they communicate with each other.
* Detailed-Design: This step cares of the implementation of different components identified in the last previous steps. This defines more logical for each module and its interfaces to communicate with other modules.

There are various terminologies which must understood, which helps in implementation of the modules, which will be discussed in the next sections of the paper. The output of this section is a set of modules and artifacts that can be implemented. This considers maintainability of the code and ease of implementation as well.

## Software Design Principles

According to the definition of the Wikipedia principle is defined as “A value that is a guide for behavior or evaluation”. Hence this step provides implantation of key notions of software design approaches and concepts. These include abstraction, coupling and cohesion, modularization, encapsulation, separation of concerns, DRY, KISS. These are explained as below.

* Abstraction: It is a process in which exposing only the relevant information to the client and hiding all the implementation details. There are different kinds of abstraction – Data Abstraction and control abstraction.
* Coupling and Cohesion: Coupling is the measure of interdependence between the modules. Whereas Cohesion is the measure of strength of association of the elements within a module. Technically an application should have low coupling and high Cohesion.
* Modularization: It’s a technique of dividing the software system into multiple discrete, independent modules, which are expected to carry out tasks independently. This results in smaller components which will be easier to maintain, yields in high abstraction, concurrent execution can be made possible and this helps for components with high cohesion to be reused again.
* Encapsulation: It’s a process of binding required data and operations into single independent entity. This allows for separation of concerns and results in high modularity.
* Separation of Concerns: Separate your modules which serves one specific task and focuses on one problem or concern. Concern can be an action or process, a data structure, a property of something. This makes code cleaner and easy to scale.
* DRY (Don’t Repeat Yourself): It’s a principle which emphasized to not to repeat and code. This highly emphasizes code reusability and avoid duplicated code. This helps in easier edit and scale the code.
* KISS (Keep It Simple and Stupid): Keeping your code simple and avoid unnecessary complexity. This helps in less chance of bugs, smaller file size and easier to debug, modify, write and read.

## Key Issues in Software Design

The increasing complexity of today’s systems has created several challenges or issues that makes it hard for Software Engineers to meet the customer demand and also maintain a higher quality of the software. These challenges have made Software Engineers to give the most attention Software Design process and consider widely known design principles and strategies to avoid these problems. [7] discussed major challenges some of these include requirements volatility, quality issues (reliability, availability, scalability and maintainability), distributed Software Development, allocation of resources, limited budgets, fast-changing technology and the transformation of requirements to actual product.

* 1. *Concurrency:* When we want to model or control or monitor real-world systems, one must deal with concurrency. Decomposing Software into modules, process and threads which will help to run applications which share same memory or resources.
  2. *Control and Handling of events:* This issue deals with how to organize the control flow of the application. How to share data among entities. This also deals with issues related to organizing the data.
  3. *Data Persistence:* This is one of the most puzzling issue. How to handle the persistence of the long-lived data. [8] discussed various available databases for persistence – RDBMS and NoSQL are the popular once. There are some tradeoffs (like efficiency, speed, accuracy) that can be made while choosing the type of database for the application.
  4. *Distribution of Components:* [9] talks about distribution of components in software to achieve the best possible availability, scalability, reliability and maintainability. It considers the interaction of the full application, how the components including hardware, software, middleware should be handled in Software Design.
  5. *Error and Exception Handling and fault tolerance:* Errors are common and handling errors are the most tedious. Everything is error prone, the whole system might not work at times, the network failure issue. A good system should be able to handle most possible errors. Fault tolerance is the property of a system which continue to operate even in the event of failure of some of its components.
  6. *Interaction and Presentation:* This deals with separation of concerns as well. How to handle the interaction with application users and presentation of information to the users. There are various design techniques which is discussed in section 3.1 of this paper.
  7. *Security:* Software security is an idea which is used to prevent software from malicious attack and other hackers. [10] explains the importance of software security and issues related to it.

## Software structure and architecture

The fundamental structures of the software systems and the disciplines and rules that must be followed while developing that structure is called Software Architecture [11]. The paper [12] discusses about systematic methodology for identifying and communicating architecture design decisions. They are classified into 2 broad categories – Architectural Styles and Design Patterns.

* 1. *Architectural Styles:* As per [13] software architecture is an implementation of system in terms of its architectural constructs (components, aspects, concerns). This has shown effective for realizing and managing large scale applications. There are various architectural styles some of which are discussed below.
* Structural – This deal with actual structure of the applications and how they interact. Some of these are component-based, Monolithic application, Layered, Pipes and filters.
* Shared Memory – Here applications share the same resources such as memory. These include Database-centric, Rule-based.
* Messaging – If applications wants to communicate via sharing messages through pipelines, this architecture holds good. Publisher-Subscriber, Asynchronous Messaging are the different types of Messaging architectures.
* Distributed Systems – Here the applications are distributed. The application is divided into number of standalone applications which communicate via each other to act as a single application. Client-Server (2-tier, 3-tier, n-tier), shared nothing architecture, peer-to-peer, Representational State Transfer, Service Oriented and Cloud Computing patterns.
  1. *Design Patterns:* [14] describes nicely the design patterns that could be used in software development. This paper is summarized as below. Design Pattern describes a repeated problem during software designing. Applying design patterns enables developers to solve this recurring problem and apply the widely accepted solution to the problem. This also helps other developers in understating the structure or code easily rather then re-inventing the wheel [15].
* Creational Patterns: They deal with object creational problem. This solves the problem of trying to create objects in a way. There are 5 such design patterns – Singleton, Builder, Factory, Abstract Factory, Prototype.
* Behavioral Patterns: This identify communication problems between objects and realize this as a pattern. There are 10 such patterns – Chain of Responsibility, Command, Iterator Design Pattern, Mediator, Memento, Observer, State, Strategy, Template Method and Visitor.
* Structural Patterns: This shows how to glue different pieces of the software together in a flexible and extensible fashion. There are 7 of them – Proxy Design Pattern, Flyweight Design Pattern, Façade Design Pattern, Composite Design Pattern, Bridge Design Pattern, Adapter Design Pattern, Decorator Design Pattern.

That said none of these patterns force the user to do anything, these are just guidelines to solve that problem in that context, in a way.

## User Interface Design

This section talks about UI considerations as per [16] and [17] that one must take care while designing interface for communicating with the user. As the number of interfaces and diversity grow, there are some considerations that one must take care for effective interface design.

* 1. *User Interface Design Principles:* These are worldwide accepted guidelines one must follow. Below are few of them.
* Learnability – The UI should be easy for user to learn and get acquainted with it without much efforts.
* User familiarity – The interface should be designed keeping in mind the users who will use this software.
* Consistency – There shouldn’t be any inconsistent actions which makes user struggle to know its function.
* No Surprise – The behavior of the software should be predictable with less surprises.
* User guidance – There should be meaningful errors for user to understand. Remember user is a non-technical person.
* User Diversity – The interface should consider diversity and people with different backgrounds. Should be accountable for people with disabilities.
  1. *User Interface Design Issues*: There are number of issues encountered while designing user interfaces. It is subjective, one should choose ways to minimize the trade-offs.
  2. *User Interface Design Process:* This is an iterative process. This has to do with the feedback of the customers and the users who will use this software. The process of interface design has 3 steps as explained below.
* User Analysis – Analyze the end user, who will be using this software. Understand how they interact with other users, what is the manual process they follow. These help us in understanding and coming up with the best design possible.
* Software Prototyping – Prototypes helps users to comment and give constructive feedbacks which can be incorporated.
* Interface Evaluation – It is best to get the interface evaluated with the actual end users, after completion.

## Software Design Quality and Analysis Evaluation

[18] and [19] discusses about the importance of analyzing the quality of software design. It also says about the impacts of not having a good quality design – which leads to increase in budget for refactoring, delay in project completion. There are many principles and characteristics that could be considered in developing a good quality software. Design patterns, antipatterns, code smells, technical debt form an internal bridge between internal attributes of the system and external quality characteristics [20]. Below are the tools that help in evaluating software design quality.

* Static Analysis: It is a method of computer code debugging that is examined without running the code. This helps in understanding the unused code and complexity of the software.
* Simulation and Prototyping: This is where one runs the code to simulate the actual working environment and testing. This could also be done first prototyping the model and the go ahead with actual implementation.
* Software Design Reviews: Getting reviewed the working software from the expected end users is one of the best evaluation metrics. This reveals the bugs which could be fixed before deploying the software to production.

## Software Design Strategies and Methods

There are number of strategies and methods to help the design process. Some of which are mentioned below.

* Function Oriented Design: In this method one identifies the design as a function. The problem can be broken down into top-down or bottom-up.
* Object Oriented Design: Here the solution to the problem is given in terms of number of classes and interaction between them. [21] clearly describes a framework that could be used to achieve this object-oriented design.
* Data Structure-Centered Design: Here the solution to the problem is seen in terms of data structures that could be used.
* Component-Based Design: The problem is split into independent functional components and interactions between them realizes the solution to the problem. This mainly focuses on code-reuse.
* Aspect-Oriented Design: Here the main focus on the aspects rather than classes as in Object-Oriented Design.

## Software Design Tools

There are various tools available in the market which aids in creation of software design artifacts during the process of software design. They can be used for below functions.

* To transform requirements to design.
* Convert Class Level Design to code.
* To check static and dynamic quality of the code. References

# Conclusion

Software design process should be considered with at most importance. Following a predefined framework or set of rules for design process aids in easy transition of requirements to design.

# ACKNOWLEDGEMENT

I would like to thank Professor Michael Findler for assigning me this interesting paperwork. I got to learn a lot by reviewing other papers. He did help us in gaining a lot of insights in Software Design. Additionally, I would like to thank out TA Jahnvi Rai for helping us all through the start to finish of this work.

# References

|  |  |
| --- | --- |
| [1] | S. K. a. P. W. ongthongtham, "A state of the art review on software project," in *4th IEEE International Conference on Digital Ecosystems and Technologies* , Perth, Australia, 2010. |
| [2] | S. M. A. S. a. G. Kadoda, "Factors that Influence Software Project Cost and Schedule Estimation," in *Sudan Conference on Computer Science and Information Technology (SCCSIT)*, Khartoum, Sudan, 2017. |
| [3] | P. B. a. R. E. (. Fairley, Guide to the Software Engineering Body of Knowledge Version 3.0, Piscataway, NJ: IEEE Computer Society, 2014. |
| [4] | B. Sherlund, "IEEE Guide to Software Design," *IEEE Std 1016.1-1993,* p. 25, 1993. |
| [5] | N. Singh, "Agile Software Development : An Overview," *IJRCEMAS,* vol. 3, no. 4, p. 6, 2016. |
| [6] | S. S. Y. a. J. J.-P. TSA, "A Survey of Software Design Techniques," in *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 1986. |
| [7] | C. E. Otero, "Software Design Challenges," IT Performance Improvement, [Online]. Available: http://www.ittoday.info/ITPerformanceImprovement/Articles/2012-06Otero.html. [Accessed 04 April 2020]. |
| [8] | H. B. S. P. a. K. S. Chong Tang, "Towards Designing Effective Data Persistenceis," in *IEEE International Conference on Software Engineering Companion*, Virginia Charlottesville, 2017. |
| [9] | D. D. Lovre Hribar, "Weibull Distribution in Modeling Component Faults," in *52nd International Symposium ELMAR-2010*, Zadar, Croatia, 2010. |
| [10] | G. McGraw, "Software Security," *GARY,* p. 83, 2004. |
| [11] | T. R. M. B. A. N. a. A. A. Tehreem Aslam, "Quality Based Software Architectural Decision Making," in *International Conference on Communication Technologies (ComTech 2019)*, Islamabad, Pakistan, 2019. |
| [12] | J. Z. K. Eric Harper, "Exploring Software Architecture Context," in *Working IEEE/IFIP Conference on Software Architecture*, Raleigh, NC, 2015. |
| [13] | S. Malek, "Effective Realization of Software Architectural Styles with Aspects," in *Seventh Working IEEE/IFIP Conference on Software Architecture*, VA, USA, 2008. |
| [14] | M. H. a. J. Shuai, "Design Patterns in Software Development," Beijing, 2011. |
| [15] | D. N. a. V. BOZHIKOVA, "One Approach to Improve the Software Quality by Applying Software Design Patterns," in *XVI-th International Conference on Electrical Machines*, Varna, Bulgaria , 2019. |
| [16] | U. Nerurkar, "Web User Interface Design:," *Interface Design,* p. 4, 2001. |
| [17] | "User Interface Design Basics," usability.gov, 27 May 2011. [Online]. Available: https://www.usability.gov/what-and-why/user-interface-design.html. [Accessed 07 April 2020]. |
| [18] | F. Khomh, "SQUAD: Software Quality Understanding," in *Working Conference on Reverse Engineering*, Montreal, Canada, 2009. |
| [19] | J. W. a. H. Z. Qian Zhang, "Tool Support to Model-based Quality Analysis of Software Architecture," in *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, China, 2006. |
| [20] | A. Imran, "Design Smell Detection and Analysis for Open Source Java Software," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Buffalo, New York, 2019. |
| [21] | A. R. A. S. a. A. T. David Parsons, "A "Framework" for Ob ject Oriented Frameworks Design," Netherlands. |