NEED FOR DESIGN PATTERNS AND FRAMEWORKS FOR QUALITY SOFTWARE DEVELOPMENT

S.Saira Thabasum MCA.,M.Phil
Research Scholar, Mother Teresa Women’s University
Mail id: saira.thabasum@calydontech.com

ABSTRACT

Designing Object Oriented Software is hard and designing reusable Object Oriented software is even harder. Experienced Object Oriented designers can make good designs, whereas novice designers cannot and tend to fall back on non object-oriented techniques. Designing object oriented software is not as same as procedure oriented software. In software development, 60% of effort goes to Analysis and Design and only 20% for implementation. In this paper, we explore the need of using design patterns and frameworks in building large complex reusable object oriented systems. Using patterns in software development can greatly reduce the development time as known solutions are used. By using well-known patterns reusable components can be built into frameworks which provide reusability and separation of concerns which is a key to software development.

Keywords:
Object-oriented, Software design, Design Patterns, Framework, Reusability

INTRODUCTION

Designing and building high quality object oriented software is difficult. The complexity of software is quick to increase with respect to changing requirements and the size of the software system. Development of software system is primarily a human activity. Indeed, it can be claimed that development of high quality object oriented software systems are amongst the most complex construction undertaken by humans. Today, Computer Engineers have developed various tools and methods to make the
construction of software systems simpler. Among these, the method that is considered to be most important is design patterns and frameworks for object oriented software development.

PROBLEMS WITH OBJECT ORIENTED DESIGN

In Software Engineering, Object Oriented Design is concerned with developing an object oriented model of a software system to implement the identified requirements. It is a part of object oriented development where an object oriented strategy is used throughout the development process. In OOD, designers think in terms of objects instead of operations or functions. These objects are related to the solution to the problem that is being solved. Design Process involves designing the object classes and the relationship between these classes. But the difficult part of object oriented design is decomposing a system into objects. The task is difficult because many functions come into play: Encapsulation, Granularity, Dependency, Flexibility, Performance, Evolution and Reusability. All these factors influence the decomposition in various conflicting ways. Moreover objects can vary tremendously in size and number. A novice designer may not be able to decide what an object should be? Another problem with object oriented design is overuse of inheritance. Designers new to object oriented design tends to overuse inheritance. Though it is a powerful mechanism for reuse of functionality, it binds the classes tightly and makes designs much harder to understand and change. At the heart of great design are a set of principles and patterns. Design Principles form the foundation of good object oriented design and Design Patterns provide general repeatable solutions for common software problems addressed.

HOW DESIGN PATTERN SOLVES PROBLEM?

A Design Pattern is a general repeatable solution to commonly occurring problems in software design. According to GOF “Design Pattern names, abstracts and identifies the key aspects of a common design structure that make it useful for creating a reusable object oriented design”. They help the designers in choosing design alternatives that makes a system reusable and to avoid alternatives that compromise reusability. In addition, they allow developers to communicate using common vocabulary for design of software systems and gives flexibility to change or add an object implementing an interface rather than anticipate all functionality of an object. Each design pattern focuses on a particular design issue and describes the applicability scenarios and also the consequences and trade-off of its use. This greatly helps the novice designer to design an object oriented system with less complexity and errors reduced. With design patterns, even a novice designer can gain expertise equivalent to experienced designers. Design patterns can solve many of the day to day problems faced by object oriented designers in different ways to a maximum extent.
PATTERNS USED IN JAVA API

Importance of Design Patterns can be very well understood by conducting a study on Java API’s. Let’s analyze Java API to understand the usage of patterns for the development of quality software system. GOF patterns are broadly classified into Creational, Structural and Behavioral based on their scope and purpose. There are numerous that are extensively used in Java API’s. One such pattern is Factory Method that belongs to the category of Creational Patterns. This pattern is used in implementing the getInstance() method of java.util.Calendar class. Similarly, Prototype pattern is used in implementing clone() method of Object class. Also, the Proxy pattern is used extensively in the implementation of Java’s Remote Method Invocation and Interface Definition Languages. The structure of Component and Container classes in java.awt provide a good example for Composite pattern. The java.io package provides among other things, a set of classes for reading input streams known as readers. Readers provide specialized functionality. For example, one reader reads from a file, another tracks line number and yet another pushes characters back on the input stream. In all, eight different readers exist to read input streams. It’s often necessary to combine capabilities offered by java.io.readers. This could be done by means of inheritance to provide a wide array of such reader combinations. But this would result in a veritable explosion of classes. Java.io package designers created a design using Decorator pattern that allows combining reader functionality in a way we desire, with only 10 reader classes. Even the interface java.util.Observable and the class java.util.Observer provide support for Observer pattern. The java swing classes implement Command pattern by providing an Action interface and an Abstract Action class. There are also other patterns which are used in the implementation of java API but not discussed in this paper. Use of patterns makes the design of an application transparent. The highlighting features of Java API are easy to use, easy to learn and fast enough. Study of patterns used in java API conforms that pattern provides the benefits of performance, modularization, ease of implementation and improved software quality.

PATTERNS TO FRAMEWORK

Patterns support reuse of software architecture and design whereas Frameworks support reuse of detailed design and code that may increase code productivity. Frameworks are semi-complete applications that provide domain-specific functionality and exhibit inversion of control at run-time.

The key idea behind framework is to group all common cooperating reusable classes that can be used for specific applications. The end result of using framework is a cooperative, synergistic platform, suitable for development teams, independent developers, and everyone in between. A framework embodies dozens of patterns: ex. Struts framework implementing MVC pattern. MVC is a design pattern that separates an
application into three components: Model, View and Controller. The key concern of MVC pattern is to separate the presentation layer of an application from Business Logic. By creating frameworks, each concern is solved with reusability in mind. Struts 2 framework promotes the reuse of application logic and separate presentation, logic and control. Struts actions are configurable at runtime. From this it is very clear that frameworks can improve developer productivity and improve the quality, reliability and robustness of new software.

With correct use of design patterns and building of reusable frameworks, software efficiency can be improved and greater productivity can be achieved.

IMPACT OF PATTERNS AND FRAMEWORKS ON SOFTWARE QUALITY

One of the most important tasks of the software engineering is to assure the quality of the software. Throughout the software life cycle, as the product emerges, quality must be taken into account. Throughout the software development process some quality characteristics of software become essential.

Impact of design patterns on software quality can be studied in the context of software maintenance and evolution. The quality attributes as defined by ISO/IEC 9126-1 quality model are expandability, simplicity, generality, modularity, learnability, understandability, reusability and scalability. Various research work conducted to assess the impact of patterns on development proves that design patterns can considerably reduce the maintenance cost while improving software quality. The research work done by B. Aydinoz observed the relation between error-proneness and OO metrics, and then he applied GOF design patterns to different software samples and examined the change in these metrics and concluded about the effect of these design patterns to software error-proneness. Error-proneness is related to stability of the software and stability is a factor on maintainability. Another aspect of software that may be affected by the use of design patterns is performance, especially important in the context of real-time embedded systems.

CONCLUSION

Reliable object oriented software can be built only with proven practices and methodologies. Among the better software design practices is the application of design patterns. Design patterns allow us to exploit the experience of our predecessors using proven arrangements of objects. Applying design patterns will improve our software design, development and most important, reusability. Many frameworks in use today are incorporated with design patterns and concepts. The objects utilized in a framework follow a design pattern. Hence, it can be concluded that design patterns are more than just formulas for software design solutions.
REFERENCES