ABSTRACT

Software systems are interconnecting software components which form a part of a computer system. Every system is modeled by considering two important aspects, structure and behaviour. A large and intricate software system can be described structurally with the help of its architecture and also the structural aspects of its individual components. Structural characteristics play a significant role in evaluating the reusability of software systems. This research addresses issues related to the structure of software systems such as organization of the system in terms of the components and their relationships, which are static models.

Louis Pasteur has stated that science is as mature as its measurement tools. Producing high quality software is a prerequisite for satisfying the requirements of any software. To accomplish this, more research is needed in defining and measuring software quality as well as a way to derive metrics for quantifying software. Metrics are quantitative measures of the software attributes related to the quality factors. The factors affecting software quality are categorized in terms of three software product activities namely, product operation, product revision, and product transition (Mc Call *et al.* 1977b). The software quality attributes considered in this research are maintainability (in the product revision phase) and reusability (in the product transition phase). Maintainability is the effort needed to find and troubleshoot an error in an operational program while reusability is the effort required to use software in a different application. Design Pattern is a technique to communicate object-oriented design knowledge. Design Patterns play many roles in the object-oriented software development process such as, contribute a universal design vocabulary, help in curtailing system complexity by using abstractions,

represent a foundation of experience for constructing reusable software, and operate as building blocks which form the basis for complex designs.

A new design pattern is proposed with the name Shuffler, which aids to select generic shuffling alternatives that make a system reusable. Shuffling is a method which deals with jumbling or interchanging the objects' position in the collection. It may be applied to games such as Jigsaw, Sudoku, Word games, etc. It has many known uses in the software industry. The design pattern Shuffler is documented in the Gang of Four (GoF) design pattern template so that the design reusers can interpret the significance of the pattern.

The impact of Shuffler design pattern on the quality of software systems is analyzed. The software systems considered for empirical evaluation are gaming applications namely, Jigsaw, Poker, and Scramble. The reusability quality attribute is linked to the low-level quality features such as cohesion, coupling, inheritance, size, and complexity. The Eclipse plug-in Metrics is used to compute the object-oriented metrics linked to the low-level quality features. The gaming applications before refactoring are evaluated with the Metrics tool. Further, the pattern Shuffler is used to refactor the gaming applications and these redesigned applications are again assessed using the tool. A comparative study of the gaming applications before and after refactoring proves that the Shuffler pattern has a positive impact on object-oriented metrics. Additionally, these applications are assessed with reusability metrics for software components. Further, the Shuffler pattern merged with two (Gang of Four) GoF creational design patterns namely, Singleton and Prototype, is used to redesign the applications. Singleton is a design pattern which assures that a class has one instance, and provides a method to access that unique instance. Prototype is a design pattern which defines the type of objects to construct using the typical instance built

initially, and further generate new objects by cloning this prototypical instance. These refactored systems are analyzed further to compute the object-oriented metrics.

A novel design pattern named Bout serves to the effective storage of the client data in software systems with distributed or cloud architectures. This pattern is fitted into the well-known GoF pattern template thus, entertaining the adoption of design patterns. The Bout pattern provides integrity of client information despite multiple requests and responses from clients across the network. When a client makes a request to the server, a session is created and stored in a cache for a specific time period till the session expires. A HashMap is used to store data pertaining to clients in the server. Singleton pattern is used to return the same session object for the same client, though requested from different nodes in the network. When a new session is created, Prototype pattern is used to clone the session object and relevant updates are done. For example, a job portal system is considered and designed with Bout pattern integrated with Factory Method, Observer, and Decorator design patterns. The Factory Method designates a method/function for creating objects though subclasses determine the class to instantiate. The Observer pattern specifies a one-to-many relationship between the objects such that if one of the object changes, all its dependents are kept informed and automatic updates are done. The Decorator design pattern hooks up added responsibilities to an object spontaneously. The impact of these patterns on low-level quality attributes namely, object-oriented metrics, component reusability metrics, and modularity metrics is evaluated. The metrics show noteworthy improvement when the system is refactored with patterns.

The adoption of existing software artifacts or software experiences to construct new systems is termed as software reuse. Reuse may be in different forms namely, code, library, tools, design, packages, and components. This

research narrows down to the importance and measurement of package reuse. The search engines do not provide much support for the rating of pertinent package candidates for reuse. Since Python is the fastest growing programming language, Python's largest repository Python Package Index (PyPI) is considered for extracting Python packages. Two core software reusability measures namely coupling and cohesion are evaluated to compute the package reusability score. This score is compared with the existing reusability metrics namely, Cyclomatic Complexity and Maintainability Index (MI). It is proved that packages with high reusability scores bring about lesser values for Halstead's reuse effort. Further, the package reusability score is validated using code detuners, which inject erratic reusability deficiencies in the source code of Python packages.

A software ecosystem is a synergy of a set of actors which interact with a common technical platform comprising number of software solutions and services. The enormous growth of software ecosystems has necessitated the analysis of quality of individual software, ecosystem dependency health, and developer community activities. The proposed research assesses the health of Python software ecosystem PyPI, based on the health of individual Python packages and their dependencies. The health of a package is evaluated in terms of Maintainability Index (MI), number of active maintainers, download count, and unsolved issues. The health of a package based on its dependency is assessed based on the Number of Modules (NOM) and Lines Of Code (LOC) imported. To evaluate the health of the package dependencies, complex network properties are applied to the package dependency network modeled as weighted directed multigraphs.

This research work, thus evaluates open-source software systems and software packaging ecosystems based on the non-functional aspects which helps developers in code restructuring using design patterns. The empirical

study carried out also measures the reusability of open-source packages to assist reusers in providing non-functional suitability. The health of huge software packaging ecosystems is assessed finally.