INTERNATIONAL JOURNAL OF ENGINEERING AND MANAGEMENT SCIENCES

www.scienceandnature.org

# PACKAGE ORIENTED REQUIREMENTS ANALYSIS FOR BEST PERFORMANCE AND QUALITY SYSTEMS

**[1]AB. Qayoom Sofi and [2]Ovass Shafi Zargar**
[1]Department of Computer Applications, Govt Degree College, Pattan, Baramulla, J&K 193121, India
[2] Department of Computer Applications, Amar Singh College, Srinagar, J&K, India

**ABSTRACT**
Electronic systems/devices must provide best performance and quality as continuously demanded. It needs developing systems based on the principle of "Oneness" i.e., each component performs just one specialized quality function. The globally unresolved problem is the software crisis problem owing to the low productivity and poor quality of the products. One of the main reasons is the highly complex requirements analysis that forms the foundation and largely affects the quality and productivity. To overcome software crisis, it is necessary to improve processes as well as products. This paper focuses on bringing improvement through following factors.
**Process Effectiveness:** The paper proposes to introduce Package Oriented Software Analysis (Modelling). It evaluates types of requirements and their impact on software quality and productivity and organizes the requirements information into easily manageable reusable quality information components/ packages to effectively evaluate completeness/consistency resulting in high quality Software Requirements Specification document. Significant benefits will be achieved by reduced defects, reduced rework/increased reuse. It improves the productivity by reducing the cost and time of software development/maintenance by increasing the reusability and interoperability of the products/packages.
**Product Quality:** To achieve significant contributions to improving product quality further, this paper reviews quality assessment models to extend/update existing quality model resulting in a better comprehensive/extensive model, whereby total quality (T.Q) of a product is defined as T.Q= P+Q; where P means performance requirements and Q means Qualities requirements.
**Technology Transfer:** The results can be applied using existing Object Oriented technology which already supports Packages and Reusability that further enhances software quality and productivity gains.

**KEYWORDS**: Quality, Productivity, Performance, Requirements Analysis, Requirement, Defect, Packaging, Reuse

## INTRODUCTION

With a growing demand of users for high quality software, the need for the conduct of research to improve the software quality and productivity is increased. The research paper aims to achieve following goals.

1. To improve the Software Quality.
2. To improve the Software Productivity.

In spite of the number of studies conducted, productivity is still an issue for software industry since all factors and their relationships have not been known. Productivity is defined as a ratio of Outputs/Input. It is expressed as ratio of size over effort.

Productivity = Size/Effort.

It is affected by a no of factors which have been broadly classified as

1. Software development environment attributes.
2. Software system product attributes.
3. Project staff attributes.

However, low software productivity is still a globally significant unresolved problem. Software's do fail and the cost of failure is also huge. Software failure occurs due to following three main reasons

1. **Software is late. (Time/Schedule Problem)**
   The main reason for the late delivery of software is either wrong/poor understanding of the problem or the wrong/poor use of the software engineering and project management.
2. **Software is expensive. (Cost Problem)**
   The main reason for the high cost of software is that software development/maintenance is still labor-intensive.
3. **Software is unreliable. (Quality Problem)**
   mainly arises due to the introduction of errors/bugs during the development process.

Software productivity and quality are interrelated. In fact, software productivity depends on the software quality. According to the IEEE Standard Glossary of Software Engineering Terminology [2,3,28], the quality of software products is defined as

1. the degree to which a system, component or process meets specified requirements and
2. the degree to which a system, component or process meets the needs or expectations of a user.

Requirements are capabilities and conditions to which the system/project/software product must conform. Requirements are critical for defining, estimating and managing any project. Quality requirements are essential to the success of any software development project/product. There are two types of requirements

1. **Functional Requirements:** which specify what behaviour a system does.
2. **Non-Functional Requirements:** which specify how well it does.

Requirements Analysis is the first phase of software development process that focuses on understanding the problems of the system and the needs of the client and users of the system. The requirements are developed in this stage by a (development) process which consists of a sequence of activities as follows.

1. Requirements Elicitation.
2. Requirements Specification.
3. Requirements Verification & Validation.

The goal of Requirements Analysis is to transform the ideas in the minds of client & users (informal input) into a formal document of clients requirements called Software Requirements Specification (SRS). Evidently, the cost, schedule and quality of the software (project) depends on the quality of the Software Requirements Specification (SRS). So, a high quality Software Requirements Specification (SRS) is a pre-requisite to a high quality software, which requires that the Software Requirements Specification (SRS) be complete, correct, consistent and unambiguous.

However, producing a high quality (reasonable quality) Software Requirements Specification (SRS) requires that the analysis process be such that either errors/defects may not be introduced or fewer defect/errors be introduced at the requirements phase itself. That is, a high quality SRS can be produced by avoiding errors as much as possible or by reducing errors in it, by removing them as far as possible (before the design starts).

However, requirements analysis is still one of the weak areas of software engineering; hence research is required for evaluation and improvement of the requirements analysis phase.

Requirements analysis is a complex process due to involvement of three diverse parties (Client, users and developers) between which a communication gap exists that makes the task of requirements specification difficult and challenging. The problem is complicated by the fact that *often the needs and requirements of the system are not often known and understood by the client and the users of the system.*

Further, the analysis is difficult because it produces a lot of information (about requirements) usually conflicting and the problem is how that information is organized so that it can be effectively evaluated for completeness and consistency. The difficulty occurs when it is not clear "what information is missing". *A good structuring of information (requirements) can help determine the areas of incompleteness.* So, proper structuring of information (requirements) is required because once what needs to be determined is known, determining it is usually not hard. Like any other phase, occurrence of defects during analysis is also inevitable owing to its complexity. It is the presence of these defects that leads to the degradation of quality which in turn may lead to the failure of software.

Further, the greater the delay in detecting errors after it occurs, the more expensive it is to correct it. That is, a requirements error (an error that occurs during requirement phase) if corrected after the system has been developed can cost up to 100 times more than correcting/removing it during the requirements phase itself since the cost of fixing an error increases almost exponentially as the time progresses thereby increasing cost & time of software development which reduces software productivity.

## RESEARCH DESIGN

The research proposal is to improve process effectiveness by improving the requirements analysis phase of the Software Engineering process through the introduction of Package Oriented Software Analysis (Modelling).

It proposes to remove communication problems by modeling real world systems into packages such that the programming language supports their implementation and easy reuse at a higher level (system/sub-system level). Further, it reviews quality assessment models for extension of existing model or introduction of a better comprehensive/extensive quality model, whereby total quality of a product will be defined as T.Q= P+Q; Where P means performance requirements and Q means Qualities requirements.

## PACKAGE ORIENTED SOFTWARE ANALYSIS

Traditional Analysis focuses on Functionality, Object oriented Analysis starts with object modelling, while as most of the real world that we want to model consists of interacting systems and subsystems. There is an increasing need to assemble powerful systems and subsystems. *Packaging just do that i.e it models a real world system into components called Packages.*

*Where a Package is an easily manageable reusable quality information component*. Logically, a package is a group while as physically/technically, a package is a folder at the least level. There are two types of Packages:

1. **Functional:**
   General and Special /Custom Package.
2. **Non-Functional:**
   Performance and Qualities Package.

. **Package Oriented Requirements Analysis**

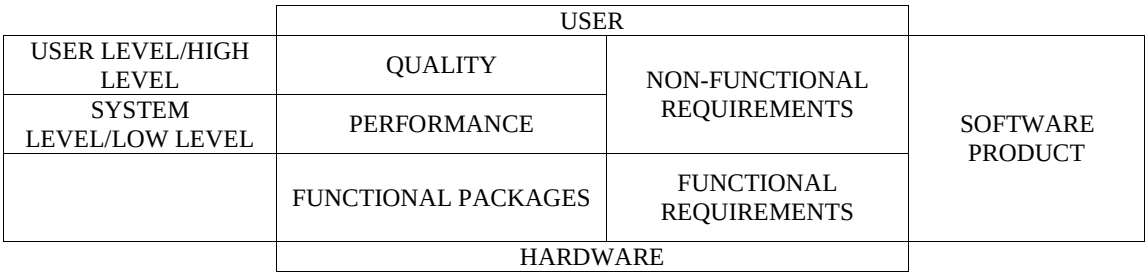
| Macro Analysis (Packaging) | Micro Analysis (OMT) |
|---|---|
| Identify sub-systems, departments, sections | Identify objects, relationships of the sub-systems |

**Advantages of Packaging and Reusing of Packages**

As shown above, analysis has been divided into two phases namely Macro- Analysis and Micro-Analysis. Macro-Analysis is actually a new sub-phase introduced to organize information requirements at the analysis level

itself rather than during the design phase in order to avoid errors as early as possible. In this sub-phase of analysis, packaging (identification and classification of components) will be actually done and detailed analysis later identifies objects and their relationships in these sub-systems identified during the macro-analysis.

The high cost of developing software means that we must make reuse of existing software effective in order to improve productivity and quality. Environments must be developed that make all forms of reuse easy to practice. Further, software is difficult and costly to maintain and yet there is a high demand to modify and/or enhance it because organisations must keep adapting to changing environments. Object-oriented techniques only partly solve the problem, as it only deals with low level reuse. Much greater progress will occur if we reuse whole systems and subsystems rather than just objects. This can be achieved by having systems/sub-systems as packages which can be reused later. It means we also need languages that support not only programming but also analysis (modeling), design (modeling), implementation and reuse at a much higher level (i.e. at the system/subsystem level).




Architecture of the Model

**MAIN COMPONENTS/PACKAGES**

1. **FUNCTIONAL PACKAGES**
   organize all functional requirements/needs of the system.
2. **NON- FUNCTIONAL PACKAGES**
   **Qualities Packages** Organize quality attributes as seen by a user.
   **Performance Packages**:- organize other quality attributes/characteristics of the system including performance from low level system's perspective.

**Features of Package Oriented Software Analysis**

1. Simplification of Requirements Analysis.
2. Introduction and use of "Packages" to organize requirements and to overcome missing areas.
3. Use of existing technologies and languages for modeling and implementation.
4. Effective Elicitation (Modeling) of requirements.
5. Effective specification of requirements.
6. Producing structured/packaged RS document.
7. Prevention of defects.
8. Prioritization of Packages.
9. (Easy) reuse of Packages.
10. Simple, easy to implement, quick integration.

**SOFTWARE QUALITY MODEL**

A number of software quality models have been proposed by the researchers/scholars for assessment of software quality. A quality model is "the set of characteristics, and the relationships between them that provides the basis for specifying quality requirements and evaluation". The models constructed define the fundamental factors (characteristics), and within each of them the sub factors (or sub-characteristics). Each sub-factor has been assigned a metric for the real evaluation of overall quality.

**Important Quality Models**

1. **McCall's Software Quality Model** / Classical Quality Model [McCall et al. in 1977]

2. **Boehm's Software Quality Model** [1978] Boehm et al. proposed using McCall's quality model.

3. **McCall's quality model** was later adapted and revised as the MQ Model by Watts in 1987.

4. **ISO/IEC 9126 Quality Model** [1991, ISO/IEC.] **First Standard Model**.

5. **FURPS Quality Model [1992]. FURPS+**

6. **Dromey's Software Quality Model** [1995] R.G. Dromey.

7. **SQuaRE Model**: The ISO 9126 Model was updated in 2007 by the ISO 25010 that redefines the fundamental characteristics.

**Limitation of Existing Models:-**

1. **All the models have ignored the aspect of communication** as one of the quality factors although there is a need for quality components for communications at all levels especially because of Internet.

2. **Reusability** has not been considered as main characteristic except in McCall's Model, although easy and effective reuse is the need of the modern technology.

## THE PERFORMANCE QUALITY MODEL

With rapid change in technology and to cover past, present as well as future technology, Quality needs to be redefined/reviewed/extended. So, with ISO 25010 Quality Model as a base/reference, integrating some new characteristics/factors and sub-characteristics/sub-factors is proposed to have a comprehensive quality model called Package-Oriented Software Quality (POSQ) Model/PQ Model, with two main quality groups as Performance and Qualities as shown below.

| Non-Functional Requirements | |
|---|---|
| **LOW LEVEL/SYSTEM LEVEL/ INTERNAL** | **HIGH LEVEL/USER LEVEL/ EXTERNAL** |
| Portability | Quality |
| Efficiency | Usability |
| Reliability | Acceptance /Affordability |
| Functionality | Legality/Licensing/Language |
| Operability | Interoperability |
| Reusability | Testability |
| Maintainability | Integrity |
| Adaptability | Expandability |
| Networkability | Security |
| Compatibility | Supportability |
| Evolvability | |

## INTERPRETATIONS AND APPLICATION OF PROPOSED MODEL

### Measurement for Improvement

To improve products and processes, it is necessary to make meaningful measurements. Once we have the measurements, we can use them to improve the quality of the software. To achieve useful impact, the measurements must be linked to a software product quality model in a systematic way.

The most desirable situation for applying measurement is to define an ideal for some property and then make measurements of a product to see if the ideal is realized. If the ideal is not realized then their is a basis and guidance for improvement. The present proposal will try to measure things with respect to an ideal which is constructive to quality improvement. Accordingly, the (software) products can then be classified as having

1. Best quality.
2. Average quality.
3. Poor quality.

## CONCLUSION

The major thrust of this paper is to propose package oriented requirements analysis model for effective specification and organization of complete information about requirements into functional packages which can be reused.

Further, the focus is to ensure the quality of these packages through the proposal of Performance Qualities model whereby the quality attributes have been refined into Performance and Qualities groups implemented as non-functional packages. The models are supported by the various existing software engineering tools like Holmes, Egidio, UML, Interstage Apworks.

## REFERENCES

Bill Davey, RMIT University, Melbourne, Australia and Kevin R. Parker, Idaho State University, Pocatello, Idaho, USA Requirements Elicitation Problems: A Literature Analysis

R.G. Dromey, Software Quality Institute, Griffith University, Nathan, QLD., 4111, AUSTRALIA: Software Quality and Productivity Improvement

Donald Firesmith, Software Engineering Institute, U.S.A. Using Quality Models to Engineer Quality Requirements,

David Mauricio, Glen Rodríguez and José P. Miguel, A REVIEW OF SOFTWARE QUALITY MODELS FOR THE EVALUATION OF SOFTWARE PRODUCTS,

Pankaj Kumar, Noida Institute of Engineering & Technology, Greater Noida: ASPECT-ORIENTED SOFTWARE QUALITY MODEL: THE AOSQ MODEL

Kazuhiro Esaki, Faculty of Science and Engineering, Hosei University, Tokyo, Japan:Verification of Quality Requirement Method Based on the SQuaRE System Quality Model

Walt Scacchi, Understanding and Improving Software Productivity.

Melanie Ruhe and Stefan Wagner, A Systematic Review of Productivity Factors in Software Development.

Anupriya and Saya. Survey on Various Productivity Measures of Software Development Teams, June 2014,

Abimbola Soriyan[1,] Ishaya Gambo[1] and Philip Achimugu[2] Software Architecture Performance Quality Model: Qualitative Approach