ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering

Vol. 1, Issue 1, March 2013

Analysis of Software Functional Programming and Imperative Programming

V.Thangadurai[1], Dr.K.P.Yadav[2] , Dr.K.Krishnamoorthy[3]

Research Scholar, Department of CSE, Monad University, Hapur (UP), India

Director, Mangalmay Institute of Technology, Greater Noida (UP), India

Professor, Dept. of CSE, Sudharsan Engineering college, Pudukkottai, Tamilnadu, India

Abstract: Software functional programming is a a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state - i.e. pure mathematical functions. It is a declarative programming paradigm, which means programming is done with expressions. Analysis of functional and imperative programs has the potential to contribute to the control of quality of software. Internal attributes, such as structural properties, measured in the static analysis, are claimed to have a correlation with external at-tributes, such as comprehensibility, maintainability and testability. Tradition-ally, static analysis and related tools focuses mainly on programs written in imperative programming languages. This paper focuses on comparative analysis on various types of models which can be used for both functional and non functional programming.

Keywords: measurement, metrics, models

I. INTRODUCTION

Functional and imperative programming have been an active area of research for many years, but relatively little of this research has been directed towards the software engineering aspects of functional programming. This may partly account for the slow adoption of functional programming in “real world” software development. A software measure is a mapping from the set of objects in software engineering world to a set of objects in the mathematical world. Software measurement is a quantified attribute of a characteristic of a software product or the software process. Objects in the mathematical world may be numbers or vector of numbers. These mapping can be defined on different scales such as nominal, ordinal, interval or ratio. [Zuse, 1991]. A measure can be used to characterize some property of software engineering objects quantitatively.

Measurement permeates everyday life and is an essential part in every scientific and engineering discipline. Measurement allows the acquisition of information that can be used for developing theories and models, and devising, assessing, and using methods and techniques. Practical application of engineering in the industry would not have been possible without measurement, which allows and supports production planning, production monitoring and control, decision making, cost/ benefit analysis, learning from experience.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering

Vol. 1, Issue 1, March 2013

• Software measurement is a technique or method that applies software measures to a (class of) software engineering object(s) to achieve a predefined goal. Such goals of measurement vary along five character-istics (Basili, 1989; Basal and Rombach, (1988): what-software engineering objects are being measured, why they are being measured, who is interested in these measurements, which of their properties are being measured, and in what environment they are being measured.

• Object of measurement. People measure different software engineering objects ranging from products to processes and entire projects. Example products are source code components, software designs, software requirements, and software test cases. Example processes are the architectural design process, the coding and unit test process, and the system test process.
• Purpose of measurement. People measure software engineering objects for different purposes. Examples include characterization, assessment, evaluation, prediction, and improvement.
• Subject of measurement. Different people or organizations are interested in measuring software engineer-ing objects. Examples are the software designer, soft-ware tester, software manager, software quality ensure, and the entire software development organization.
• Measured property. People may be interested in different properties of software engineering objects. Ex-ample properties include cost, adherence to schedule, reliability, maintainability, and correctness. • Context and environment measurement. Software engineering objects are measured in different environments. Example characteristics of an environment may be the kinds of people involved, technology used, applications tackled, and resources available.

II. SOFTWARE MODELS AND MEASURES

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. It is important to have many kinds of models and measures. Models and measures are in separable companions. Measures are intended to characterize some property of software is therefore, impossible to judge the appropriateness of a chosen measure without understanding the underlying property model the measure is supposed to quantify. In the following sections the different types of measures and models needed are discussed and examples of project, product, and process models and measures are given.

A. Types of Measures

There is a distinction between objective and subjective measures, abstract and specific measure, complex and simple measures, product and process measures, and direct and indirect measures. There is a lot of information that cannot be measured objectively; it may be an estimate of the extent or degree in the application of some technique, a classification, or qualification of problem or experience, usually done on a nominal or ordinal male. An example may be the subjective measure team aperients with some technology. Such a measure can be defined in an operational way on a nominal scale as follows:

0. None
1. Have read the manuals
2. Have had a training course

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering

Vol. 1, Issue 1, March 2013

3. Have had experience in a laboratory setting.
4. Have used it on a project before
5. Have used It on several projects before

Defining each value between 0 and S in an operational sense has the advantage of guaranteeing consistent data collection. Without such operational definitions, subjective measures are usually limited, because it is in the eye of the be holder to judge a given team as having an experience /end of either 0 or 5.Most measures reported in the literature are abstract measures in the sense that they are derived from some abstract model. They ward to be tailored to the specific characteristic of an environment before it can be applied, Those tailored measures are called specific. Most measures reported in the literature are abstract measures in the sense that they are derived from some abstract model. They ward to be tailored to the specific characteristics of an environment before they can be applied, Those tailored measures are called specific. For example the abstract Measure of fan-in-fan-out needs to be tailored to specific languages such as Fortran or Ada before it can be applied practically. Simple measures arc those based on a single property; complex measures are those based on a vector of properties.

B.Types of Software Metrics

A software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

Process Metrics: Process metrics are known as management metrics and used to measure the properties of the process which is used to obtain the software. Process metrics include the cost metrics, efforts metrics, advancement metrics and reuse metrics. Process metrics help in predicting the size of final system & determining whether a project on running according to the schedule.

Products Metrics: Product metrics are also known as quality metrics and is used to measure the properties of the software. Product metrics includes product non reliability metrics, functionality metrics, performance metrics, usability metrics, cost metrics, size metrics, complexity metrics and style metrics. Products metrics help in improving the quality of different system component & comparisons between existing systems.

The great promise of software metrics is that they may provide a concrete method to quantify the “quality” of software, and therefore the likelihood of defects appearing in particular sections of program code. However there has been little rigorous work to verify that software metrics can deliver on this promise. Barnes and Hopkins attempted to provide some much needed rigorous analysis of software metrics by analysing the evolution of a large library of numerical routines written in Fortran through a series of releases. They measured path count metric values for each routine in the library, in each release, and counted the number of defects over the lifetime of the library.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering

Vol. 1, Issue 1, March 2013

III.SOFTWARE MEASUREMENT OF FUNCTIONAL AND IMPERATIVE PROGRAMMING

Software measurement has become a key aspect of good software engineering practice. Measurement activities adds value and keeps us actively involved in, and informed of, all phases of the development process. Measurement can help us to make specific characteristics of our processes and products more visible. Measurement encompasses quantitative evaluations that usually use metrics and measures which can be used to directly determine attainment of numerical quality goals. Despite all these advancements and envisaged benefits, software measurement does not seem to have fully penetrated into industrial practices. As far as the use of software measurement for quality evaluation is concerned, [1] have observed that its application has been as yet limited since most software metrics are still being used mainly for cost estimation. Hence, it is clear that everything should be measurable. If it is not measurable, we should make an effort to make it measurable [2]. Software testing is a vital element in the SDLC and can furnish excellent results if done properly and effectively [3] and software measurement can play a key role in increasing the effectiveness of testing process. The role of measurement in software testing has been exemplified by Munson [4]. He maintains that evaluating the test activities will give great insight into the adequacy of the test process and the expected time to produce a software product that can meet certain quality standards

A. *Software Measurement process*

The software measurement process must be an objective, orderly method for quantifying, assessing, adjusting, and ultimately improving the development process. Data are collected based on known, or anticipated, development issues, concerns, and questions. They are then analyzed with respect to the software development process and products. The measurement process is used to assess quality, progress, and performance throughout all life cycle phases [6]. The key components of an effective measurement process are:

- Clearly defined software development issues and the measure (data elements) needed to provide insight into
- those issues;
- Processing of collected data into graphical or tabular reports (indicators) to aid in issue analysis;
- Analysis of indicators to provide insight into development issues; and,
- Use of analysis results to implement process improvements and identify new issues and problems. The Figure 1 shows the software measurement process. The activities required to design a measurement process using this architecture are:
- Developing a measurement process to be made available as part of the organization's standard software process;
- Planning the process on projects and documenting procedures by tailoring and adapting the process asset;
- Implementing the process on projects by executing the plans and procedures; and
- Improving the process by evolving plans and procedures as the projects mature and their measurement needs change.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering

Vol. 1, Issue 1, March 2013

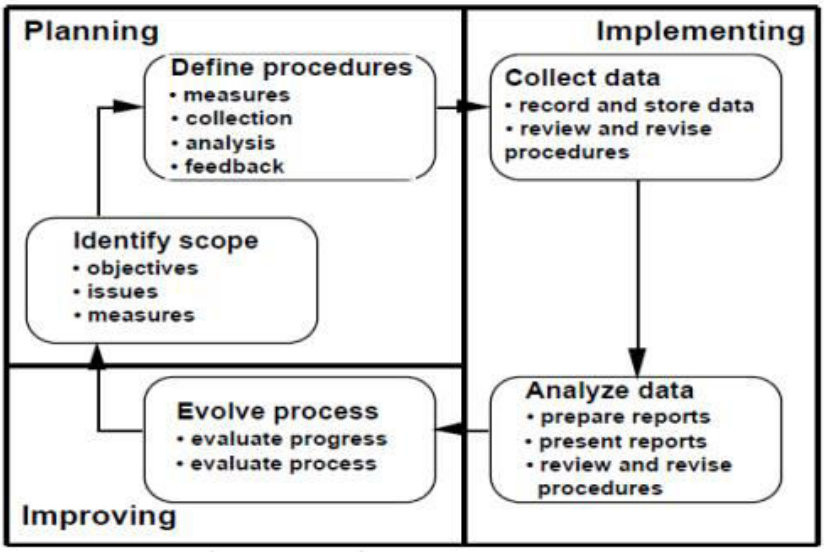

Figure 1 – Software Measurement Process

***B**. CHARACTERISTICS OF A GOOD MEASUREMENT*

The characteristics of good software measurement are:

- Reliability - The outcome of the measurement process is reproducible. (Similar results are gotten over time and across situations.)
- Validity - The measurement process actually measures what it purports to measure.
- Sensitivity - The measurement process shows variability in responses when it exists in the stimulus or situation.

IV.CONCLUSION

This paper has focused on the analysis of software with respect to functional and imperative programming techniques. Also this work has done the investigation on different types of models which can be used for both the above said programming. Therefore this work attempts to address this gap with the following contributions. • A collection of metrics for use with functional programs has been identified from the existing metrics used with other paradigms. The relationship between the metrics and the change history of a small collection of programs has been explored. The above said work has also focused on software measurement towards functional and imperative programming. A collection of metrics for use with functional and imperative programming has been identified from the existing metrics used with other paradigms. The relationship between the metrics and the change history of a small collection of programs has been explored.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering

Vol. 1, Issue 1, March 2013

REFERENCES

[1] Höfer, A. and Tichy, W. F. (2007). ,"Status of empirical research in software engineering.", In Empirical Software Engineering Issues, volume 4336/2007, pages 10–19. Springer.

[2] Aggarwal, K.K and Singh, Yogesh "Software Engineering Programs Documentation Operating Procedures (Second Edition)" New Age International Publishers, 2005.

[3] Quadri, S.M.K and Farooq, SU, "Software Testing – Goals, Principles, and Limitations", International Journal of Computer Applications (0975 – 8887) Volume 6– No.9, September 2010.

[4] Munson, J. C. (2003). "Software Engineering Measurement". CRC Press, Inc., Boca Raton, FL, USA

[5] Zuse, software complexity, walter de gruyter, Berlin, 1991

[6] Sheikh Umar Farooq, S. M. K. Quadri, Nesar Ahmad, Software Measurements And Metrics: Role In Effective Software Testing, International Journal of Engineering Science and Technology, Vol. 3, Issue 1, January 2011, pp.671-680

[7] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. IEEE Computer, 27(8):44–49, 1994.

[8] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. Software Engineering Metrics and Models. Benjamin-Cummings Publishing Co., Inc., Boston, MA, USA, 1986.

[9] M. Dao, M. Huchard, T. Libourel, C. Roume, and H. Leblanc. A new approach to factorization - introducing metrics. In Eighth IEEE Symposium on Software Metrics, pages 227–236, Ottawa, Canada, June 2002. IEEE Computer Society Press.

[10] Fenton, N. E. and Pfleeger, S. L., "Software Metrics: A Rigorous and Practical Approach", 2nd Edition Revised ed. Boston: PWS Publishing, 1997.

[11] Stark, George E; Durst, Robert C; and Pelnik, Tammy M. "An Evaluation of Software Testing metrics for NASA's Mission Control Center" 1992.