Software Reusability Factor Based Reuse Readiness Levels

RanjanaNallamalli*[1], Prof Durg Singh Chauhan[2]

Ranjana Nallamalli[1], Scientist 'G', Simulation and Modelling Centre, DRDO;
Prof D S Chauhan[2], Vice Chancellor, GLA University

Abstract- This paper identifies the critical parameters which govern the reuse of software in systems development. This paper attempts to quantify the reuse parameters in order to assess the reuse maturity.The parameters are assigned weightage as per their effect on the reuse maturity. A quantification method is proposed for assessing reusability factor. In order to practically implement reuse, reuse readiness levels are proposed based on the reusability factor computed from the identified parameters.

Keywords —*Reuse Readiness Level (RRL); Reusability Factor (RF); Reuse Readiness Parameters*

1. INTRODUCTION

In systems design and development 'software' is one of the many aspects of the system. The focus is on the delivery of the system rather than the engineering the software. Airborne systems, defence systems, space applications, medical devices, hardware engineering systems and automotive products are some of the examples where 'systems thinking' leads to software development.

Though the software development in the context is domain and product specific, there is considerable commonality at some levels. It is worthwhile to note that software development in these domains follow rigorous quality standards. This indicates that if the software is reused, the effort optimisation will improve the outcome. However, it is difficult to reuse software in an organisation where software development happens within domain boundaries. Establishing a software reuse framework is important for an organisation which has not initiated the software reuse.

In 2010 NASA's Software Reuse Working Group released a report on Reuse Readiness Levels (RRLs) after examining the measures of technology and software maturity[18]. The definition of RRLs in this paper, are based on the premise that the technology maturity measurement factors TRLs did not address the software maturity. The descriptive approach of topic area definition and RRL estimation is not directly applicable for implementing reuse program in a distributed organisation. The application and experience with implementation of these RRLs are not available in literature.To assess the reuse readiness level, a quantification approach is presented in this paper. The concerns of software development life cycle maturity, product validation and organisational operations are addressed in defining the parameters that shall govern the computation of reusability factor. The reuse readiness levels are thus derived from this reusability factor.

An extensive survey of software reuse in commercial scenario is given in *Section 2*. Software reuse scenarios in the embedded and systems development scenario are highlighted in *Section 3*. *Section 4* delineates a method to objectively assess the reuse parameters. *Sections 5* and *Section 6* bring out the reusability factor computation and reuse readiness levels assessment respectively.

2. LITERATURE SURVEY

The improvement of software productivity has been of paramount importance ever since large scale software based systems have come to exist. By nature, reuse of previous effort is an obvious choice to optimise the output of current assignment. The early ideas on reuse based software development were mooted by [2] [Mcllroy 69]. Designing the software development environments for improving software productivity was discussed by [3] [Williams, Pyster, Stuckle, Penedo, Boehm 84]. Late 80s and 90s saw extensive discussions on software reuse in various forms. A reuse based software development methodology was proposed by [4] [Kang, Cohen et al 1992]. An approach of classifying software for reusability was presented in [5] [Prieto-Diaz and Freeman, 1987]. Evolutionary development approach based on reusing specifications is mentioned in [6] [Bellinzona, Fugini et al 1995]. Further to these, AI based software reuse was discussed in [7] [Ostertag, Hendler, et al 92] and [8] [Aarthi Prasad, Park, 93].

In [1] [Frakes 94] author has mentioned that systematic reuse requires domain focus, repeatable process and reuse of higher level life cycle artefacts such as requirements, designs and subsystems. The author elucidates the success factors for reuse comprehensively in this article. In his research on software reuse [9] [Frakes, Pole 94] further brings out the quantitative indicators for identifying the reusability based on metrics and models.Software reuse frameworks like algorithm strategy, computation design, execution pattern, implementation strategy and structural design are proposed along with software reuse approaches in [11] [Soora, 14].

In the commercial software development fuelled by web technologies, distributed object technologies[10] [Emmerich, Kaveh, 2002] like COM, CORBA, Java

Remote Method Invocation, Java Beans and EJB. Reuse libraries, component based software engineering (CBSE) or component based development (CBD) emphasize the separation of concerns which is a reuse based approach to defining, implementing and composing loosely coupled independent components into systems. The author also highlights some CASE tools developed for domain engineering like Family-Oriented Abstraction, Specification and Translation, Domain Analysis and Reuse Environment, Product Line UML-Based Software Engineering, Feature-Oriented Reuse Method and a few more. [12] [Frakes, Kang 2005] classified the software product line engineering models as reactive, proactive and extractive models based on process, organisational and technical issues. An overview on software product line engineering is given by [13] [Sugumaran, Park, Kang 2006]. Aspect Oriented Programming approach was proposed as a software development paradigm for handling software modifications easily. [14] [Kiczales, Lamping et al, 97] proposed various mechanisms of aspect oriented programming like join points, point cuts and aspect weaving. AOP is both a design and programming technique and is highly useful if software is evolving [15] [Cazolla, Pini, Ancona 2005].

The concepts of model based design and development have become widely accepted. The model based development is an active area of application and also in research. The large and complex projects like Orion Crew Exploration Vehicle project of NASA has adopted the approach of model based design and development for flight software development [19]. This project promises to pave the way for reuse of models in further projects.

3. SOFTWARE REUSE IN SYSTEM DEVELOPMENT

Embedded software for large systems is tightly coupled with hardware and it is difficult to generalise the software components for the purpose of reuse. As such in an organisation which is not having any formal mechanism of reuse, software reuse happensinformally in the following forms:

1. The software for the next generation product is based on the existing architecture. Thus, utilising basic functionalities and code flow.
2. The functions in the form of code are copied and used in the next version but the architecture (new scheduler design) is changed. The code is utilised in a different execution flow like round robin to priority based to interrupt based.
3. Hardware interfaces are changed preserving the timing and functionality.
4. Time is optimised preserving the functionality and hardware interfaces.
5. The software is ported to a different execution platform preserving functionality.
6. An algorithm is optimised for operating on a different hardware platform.
7. Front ends are changed preserving the back-end logics.

This kind of reuse happens when same team attempts reuse of software in a similar context. The paper in [22] clearly demarcates the code use Vs reuse. The author brings out that the ad hoc reuse of code by recovery and planned reuse in the form of porting, tailoring and assembling of code are not the same. In a systems organisation, where code is reused by recovery and not as a planned reuse, a planned method is required to improve the reuse. The above mentioned method of reuse is ad hoc reuse by code recovery. In this regard it is worthwhile to mention that the code readability is a factor, often ignored. Though there are various metrics to measure the code readability as mention in [25], it is assumed that standards-compliant software (safety criticality classified software) possess high readability index.

Table 1 shows the reuselines of codeand the percentage of software reuse in a developed product version M4 and M5 configuration.

System	LOC in M4	Lines Changed in M5	No of Functions in M4	No of Functions Modified in M5	No of Functions Deleted in M5	No of Functions Added in M5
ES1	9553	864	223	9	-	24
ES2	9624	1770	208	12	19	49
GS1	95144	3364	532	5	8	165
GS2	78517	598	630	11	14	59
GS3	3190	416	82	1	-	28
ES3	4614	105	55	-	-	7
ES4	Used as it is					
ES6	Used as it is					
ES7	Used as it is					

ES – Embedded/Flight System; GS – Ground System

Table 1 – Software Reuse Metrics in Deriving Variants

4. IDENTIFICATION OF REUSE READINESS PARAMETERS

As per [17] [Antovski and Imeri 13], software reuse is not a simple addition to existing software development processes. The factors affecting the reusability are organisational issues of application domains, management commitment, education about software reuse, legal issues, psychological issues, identification of reusable components, repository availability, modification support and measurement of reuse.

Reuse Readiness Levels signify reuse maturity, which lies in moving from white box approach to black box approach i.e from using code pieces to validated and certified components with clear advantage to the organisation. Confidence level of reusable components increases with unit testing, component testing (using various techniques like functional testing, MC/DC coverage and other identified techniques), static testing and code walkthrough done for proving the correctness of the component. Compliance to the specified development process improves the overall quality of software. A practical implementation of certification of reusable components in Ericsson is given in [Mohagheghi, Conradi, 2014]. This study in the form of experiences brings out the certification needs of the architecture and quality scheme for developing new components like inspections, prototyping, unit testing and system testing before it can be made reusable.

There are discussions on systematic methods to verify designs within a product line based on formal verification [16] [Kishi, Noda, 2006]. In order to improve the reuse, formal verification will play a crucial role in improving the confidence on the reusable components. Similarly other factors like software portability, beneficial for reuse, and optimised for reuse play a vital role is defining the reuse maturity.

The ESPRIT-2 project called REBOOT (Reuse Based on Object Oriented Techniques) developed some reusability attributes. Some of these attributes are metrics based and some are subjective checklists based [23]. In NASA paper on Reuse Readiness Levels [18], topic areas are defined and described before qualitatively describing the Reuse Readiness Levels. These topic areas are modified to evolve a set of parameters and these parameters are assigned a value based on given criteria.

4.1 Support And Contact Information

It is possible that there is legacy software in operation and the developer is not available in the organisation. The algorithms and modules are required to be reused in the new scenarios. This is the worst case. In the other cases support is available virtually or by complete handholding. The three levels of thisparameter are given in Table 1.

Table 1 – Definition of Support Levels

S No	Level	Explanation	Level Assigned
a.	Minimum Support	Provision of artefacts	1
b.	Virtual Support	Support through e-mail and telephone	2
c.	Total Support	Handholding through physical presence and explanation	3

4.2 Documentation

The documentation for the identified reusable module will aid in its better adoption across bigger footprints without putting extra effort on the developer agency. The levels defined for this parameter are given in Table 3.

Table 2 – Definition of Documentation Levels

S No	Level	Explanation	Level Assigned
a.	Minimum Documentation	Scanty and un-reviewed documentation	1
b.	Useful Documentation	Reviewed and controlled documentation but not as per standard and using requires effort	2
c.	Complete Documentation	Complete documentation as per IEEE formats	3

4.3 Modularity And Complexity

The degree of segregation and containment of software or components is called modularity.Two types of modularity – functional modularity and architectural or design modularity can be considered. Considerable amount of work has been done in defining the measures of modularity as static cohesion and dynamic cohesion metrics for structured and object oriented methods.

Complexity is a well-studied concept. The McCabe's complexity measure and nesting depths are de-facto methods of measuring code complexity.

Definition of the complexity and modularity measuresare debatable with respect to the context. The academic intent of measuring these is for effort estimation, maintenance and to some extent for reuse. This paper will not go into the details and the research being carried out in these areas.Without getting into the mathematical definitions and details of these measures, modularity and complexity as a parameterwhich affects reuse, levels are defined as following:

Table 3 – Modularity and Complexity Levels

S No	Level	Explanation	Level Assigned
a.	Highly Unstructured	High Intermodule Coupling High Intramodule Coupling Low Cohesion Nesting Depth More than 6. McCabe's Complexity more than 10.	1
b.	Medium Level of Modularity	High Intermodule Coupling Low Intramodule Coupling High Cohesion Nesting Depth Less Than 6 McCabe's Complexity More Than 10.	2
c.	Good Level of Modularity	Low Intermodule Coupling Low Intramodule Coupling High Cohesion Nesting Depth Less Than 6 McCabe's Complexity Less Than 10	3

It is possible to use the mathematical formulas and computations for specific domains and arrive at specific numbers to define these levels. But that is another branch of research. For now, industry standard values of Nesting Depth and McAbe's complexity values are used and other aspects are left to the judgement of the practitioner.

4.4 Installation and Packaging

The installation and packaging is ensured before the software is re-used. In case direct code is being copied and used, the dependencies need to bewell defined. In case a direct install is required, the installation procedures are documented. The various levels defined for this parameter are given in Table 5.

Table 4 – Definition of Installation Levels

S No	Level	Explanation	Level Assigned
a.	Manual Usage	The source code is inserted manually, library is linked manually or an executable and its dependencies are configured manually.	1
b.	Semi-automatic	The instructions for using and dependencies are documented as step by step procedures and can be executed without much effort.	2
c.	Automatic	The reusable module comes as an install package with built in configurations and module can be adopted easily with supported documentation.	3

4.5 *Reuse Rights*

A clear statement of copyright for software helps reuse community regarding the legal and IPR implications of using the identified software. The reuse by external world should be accompanied with the copyright statement duly mentioning the ownership of the generating entity within organisation or the parent organisation itself.

Table 6 - Definition of Reuse Rights Levels

S No	Level	Explanation	Level Assigned
a.	Undefined	Reuse aspects and rights are not defined.	1
b.	Internally Defined	Reuse and IPR rights are addressed for use within the organisation by different entities.	2
c.	Externally Defined	Reuse and IPR rights are addressed for use by entities outside the organisation.	3

4.6 Testing

This criterion signifies that the extent of testing the software has been subjected to. In this case we define extensive levels in order to signify various testing scenarios under operation.

Table 5 – Definition of Testing Levels

S No	Level	Explanation	Level Assigned
a.	Prototyping	The application is developed like a prototype without any test document.	1
b.	Functional Testing	Functional Testing Document is available with test plan and test cases and results.	2
c.	Static Testing	The code has gone through identified static testing tools	3
d.	Inspections	Third party has inspected the functional test cases and submitted a report.	4
e.	Unit Testing	The systematic unit testing documentation is available with execution results.	5
f.	Performance Testing	The run time execution traces and time are measured and validated against the documented performance specifications.	6
g.	Formal Verification	Advanced formal techniques of model checking or theorem proving are applied and certified as per these techniques	7

4.7 Certification

In critical application areas, software is certified by a third-partyor an identified independent Software Quality Assurance or Independent Verification and Validation agency based on the maturity of the compliance to the identified process and coding standards.The agency is involved throughout the development process and all staged certification aspects are audited by the agency. The agency gives a clearance certificate for the usage of the software identified by a checksum or identifiable signature produced during the application execution of the module. This adds formalism to the qualification process and increases the assurance of software through transparency.

Table 6 – Definition of Certification Levels

S No	Level	Explanation	Level Assigned
a.	Not Certified	The software has not gone for any SQA/ IV&V/ certification.	0
b.	Internally Certified	All stages and artefacts are inspected by an independent SQA / IV&V agency. Based on this a certificate is given by the agency specifying the details of the processes and artefacts.	1
c.	Externally Certified	All stages and artefacts are inspected by an external certification agency in coordination with independent SQA/IV&V. A certificate is given by the agency specifying the details of the processes and artefacts.	2

4.8 Portable/Extensible

The software is developed for porting on multiple platforms and operating systems. The portability can be built as simple configurations during the installation. Code level portability allows codes to be used with various compilers without any compile specific risks, thus ensuring portability and reuse. The three levels defined with respect to this aspect are given in the following table:

Table 7 – Definition of Portability Levels

S No	Level	Explanation	Level Assigned
a.	Portability Not Defined	The portability hasn't been studied and no conscious selection of portability compliant coding standards. Manual effort can help reuse.	1
b.	Specific Portability	The software is portable with respect to identified specific platforms or family of platforms.	2
c.	Generically Portable	The software is designed initially for multiple platforms and well defined dependencies are built with configuration details.	3

5. REUSABILITY FACTOR

The software assets developed by various entities in the organisation may have different levels for above parameters. To estimate a factor which indicates the overall reuse maturity a parameter called 'Reusability Factor' is proposed. The reusability factor is computed based on the levels defined for each of the parameters defined in section 6. The simple computation mechanism is inspired from sum of weightages mechanism used for multi criteria decision making paradigm. Popularly MCDM is used for analysis of alternatives. It is appropriate to use it here for initial estimation of reusability factor to assess the reuse readiness level. The following equation is proposed to calculate the reusability factor:

$$RF = \sum_{i=1}^{i=8} Pi * Wi$$

Where Pi is the parameter as defined from section 4.1 to 4.8 respectively and Wi is the weight given to each parameter based on the relevance of the parameter in the reuse framework.The parameters defined in section 4 are in the order of their importance in implementing reuse framework. The multiplication factor used as weightage is assigned as per this order of importance. Following table summarises the level range from section 4 and weight of each parameter:

Table 8 – Ranges and Weightage for Reusability Parameters

S No	Para meter	Parameter Name	Range Value	Wt
1.	P1	Support	1-3	1
2.	P2	Documentation	1-3	2
3.	P3	Modularity	1-3	3
4.	P4	Install Package	1-3	4
5.	P5	Reuse Rights	1-3	5
6.	P6	Testing	1-7	6
7.	P7	Certification	0-2	7
8.	P8	Portable/Extensible	1-3	8

It can be seen that by multiplying the max range of each parameter by the respective weight gives a max value as 125. We call this RFmax.

For every software module, this RF is computed. The computed RF for software modules is used to define the RRL.

6. REUSE READINESS LEVELS

The Reuse Readiness Levelssignify the increasing maturity of the software and its reusability. There are nine levels defined qualitatively in NASA paper [18]. In this paper, RRLs are correlated with the RF value computed in section 5. The Max RF is divided by nine and equal amount is quantitatively assigned to each level in increasing steps. The RRLs nomenclature, the quantitative RF assignment and corresponding definition based on the maturity criteria in given in Table 11.

The point to note is that there can be considerable difference in values of individual parameters within the same RRL. This is attributed to the organisational factors and policies. The parameters which are assigned low levels can be stressed while attempting to reach the next RRL level. The maturity of the identified RRL will be considered as being built since all the affecting parameters are not at the highest level.

Table 9 – Reuse Readiness Levels

Level	Name	Definition
RRL1	Initial RF = 1-14	Software was developed without reusability consideration as nothing is available except source code and executable binaries. Though a few of the parameters may have some value but overall reuse maturity is very low. Somehow reuse is done with extreme efforts but reuse maturity is extremely low.
RRL2	Possible RF = 15-28	The basic documentation is available for software development life cycle, installation and use. It is possible to use with the help of documents and support from the originator. Some of the advanced parameters like testing, portability and certification are missing in the design.
RRL3	Easy RF = 29-42	The software is developed in highly structured manner or using object oriented manner which makes it easy to understand. Extensive documentation is available for reference and usage.
RRL4	Packaged RF = 43-58	To ensure reusability, the software is designed as bundle/ collection of modules, artefacts along with corresponding libraries with configuration possible on various environments. The software is designed for easy install and execute with configuration settings. The software is also supported with complete build environment and scripts for rebuilding and
RRL5	Demonstrated RF = 59-72	The utility of software is demonstrated in various environments. The software provides auto build installation on many platforms. The functionality is fully documented and demonstrated. The utility of the functionality is also understood by all the stakeholders so that it is easy to
RRL6	Certified RF = 73-86	The software is tested and certified for compliance of standards and for reuse on limited number of target environments with standard given set of libraries. The software is tested and demonstrated for its functionality in a lab on one or more platforms. The software is also certified to be extended further in a manner as specified by developer.
RRL7	Practical RF = 87-100	Software is designed for reuse and can be applied without much difficulty in the new scenarios. The development environment is easily available without licensing difficulties. The software is portable and extensible to the environment of choice with minimum effort.
RRL8	Prized RF = 101-114	Software has been reused in a variety of contexts and platforms. Its reuse has allowed users for significant cost, time and risk reduction. Has gained wide popularity as de-facto choice for it reuse.
RRL9	Established RF = More than 115	The software is highly modular, well documented & supported, extensible, portable, standards compliant and tested for its reuse on a variety of environments. The reuse of software has been kept open with no restrictions on modification, customization and redistribution. Many successful software products have made reuse of it.

7. CONCLUSION

A quantified approach is proposed to assess the parameters affecting reuse and then estimating the reusability factor, which in turn is used to arrive at the reusability readiness level. These RRLs can be used for assessing the maturity of software before the software is identified for reuse. A discussion on possible uses of RRLs is given in [21] and use cases for the application of RRLs is defined in the paper.The RRLs are useful for continuous improvement and optimization in the software development reuse. There has to be consistent effort by the organisations for improving the RRL level continuously. The parameters for reusability, reusability factor and RRL assessment will help any organisation in achieving the maturity of software reuse in a quantifiable manner. The initial estimate of reusability factor proposed here is for the purpose of identifying the RRLs. This factor can be more accurately modelled by using the actual data once the reuse framework is

implemented, classification of reuse is established and is quantifiably measured. It is worthwhile to mention that once the reusability framework is established, the reusability measurement metrics as discussed in [24] and other industry references should be applied to refine and improve the RRLs. A separate study can be done to correlate the reusability metrics with the reusability parameters defined here. Some of the reusability parameters can be attempted to be quantified directly from standard metrics.

References

[1] Frakes, W. B. and Isoda, S. Success Factors of Systematic Reuse. IEEE Software 11, 5(Sept 1994), 14-19

[2] Mcllroy, M., Mass produced software components: Software engineering concepts and techniques. In Proceedings of NATO Conference on Software Engineering (1969), 88-98

[3] Williams, R.D., Pyster, A.B., Stuckle, E.D., Penedo, M.H., Boehm, B.W., A Software Development Environment For Improving Productivity. IEEE Computer 06. 17(June 1984), 30-44

[4] Kang, K.C., Cohen, S. &Holibaug, H.R., Reuse Based Software Development Methodology, Application OfReusable Software Component Project, Report No SEI-92-SR-4

[5] Prieto-Diaz, R., Freeman, P., Classifying Software For Reusability, IEEE Software 4, 1 (Jan 1987), 6-16

[6] Bellinzona, R., Fugini, M.G., Pernici, B., "Reusing Specifications in OO Applications", IEEE Software 12, 2 (Mar 1995), 65-75

[7] Ostertag, E., Hendler, J., Prieto-Diaz, R., Braun, C., Computinf similarity in a reuse library System: An Ai-Based Approach, ACM Transactions on Software Engineering and Methodology 1, 3 (1992), 205-228

[8] Aarthi Prasad, Park, E.K. , AI Based Classification and Retrieval of Reusable Software Components.

[9] Frakes, W.B., Pole, T.P., An Empirical Study of Representation Methods for Reusable Software, IEEE Transactions on Software Engineering 20, 8 (August 1994), 617-630

[10] Emmerich, W., Kaveh, M., Component Technologies: Java Bean, COM, CORBA, RMI, EJB and the CORBA Component Model, Proceedings of the 24th International Conference on Software Engineering, (May19-25, 2002), 691-692

[11] Soora, S. K., A framework for software reuse and research challenges. International Journal of Advanced Research in Computer Science and Software Engineering, 4, 10 (Oct 2014), 441-448

[12] Frakes, W. B. and Kang, K. C. Software reuse research: Status and Future. IEEE Transactions on Software Engineering 31, 7 (July 2005), 529-536

[13] Sugumaran, V., Park, S. and Kang, K. C. Software product line engineering. Communications Of The ACM, 49, 12 (Dec 2006], 29-32

[14] Kiczales, G., Lamping, j., Mendhekar, A., Maeda, C., et al. Aspect-Oriented Programming. In 11th European Conference on Object Oriented Programming (ECOOP'97), Lecture Notes in Computer Science 1241, 220-242, Helsinki, Finland, June 1997. Springer-Verlag

[15] Cazolla, W., Pini, S., Ancona, M., AOP for software evolution: A design oriented approach. ACM Symposium on Applied Computing, (2005) 1346-1350

[16] Kishi, T., Noda, N., Formal Verification and Software Product Lines, ACM Communications 49, 12 (2006) 73-77

[17] Antovski, L., Imeri, F., Review of Software Reuse Processes, International Journal of Computer Science 16, 6 (Nov 2013), 83-88

[18] Reuse Readiness Levels (RRLs), Software Reuse Working Group – NASA Earth Science Data Systems, (Apr 2010), Version 1.0

[19] Tamblyn, S., Henry, J., King, E., A Model-Based Design and Testing Approach for Orion GN&C Flight Software Development, IEEAC Paper#1491, Version 3 2010

[20] Frakes, W. B. and Terry, C. Software Reuse: Metrics and Models. ACM Computing Surveys 28, 2 (June 1996), 415-435

[21] Dowsn, R.R., Marshall, J.J., A Proposal on Using Reuse Readiness Levels to Measure Software Reusability, Data Science Journal 9 (July 2010), 73-88

[22] Poulin J.S., Caruso J.M., Determining the Value of Corporate Reuse Program, Proceedings of the IEEE Computer Society International Sofware Metrics Symposium, (21-22 May 1993), pp16-27

[23] Karlson, Even-Andre, GuttormSindre, and Tor Stalhane, "Techniques for Making More Reusable Components," REBOOT Technical Report #41, 7 June 1992.

[24] Poulin J.S., Measuring Software Reusability, Proceedings of The Third Conference on Software Reuse, (1-4 November), 1994

[25] Pahal A., Chillar R.S., "Code Readability: A Review of Metrics for Software Quality", IJCTT, Vol 46, Number 1 – April 2017