International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 05 | May 2019 www.irjet.net p-ISSN: 2395-0072

Code Reuse & Reusability of the Software

BALJEET RAM[1], ANITA DEVI[2]

[1,2]M.Tech (SE) Student, Department of Information Technology at BBAU Lko (U.P.)

Abstract - Code Reuse is currently one of the most active and creative research areas in Computing. Code reuse is the use of existing software to create new software. Reuse and reuse are the two important factors in software development. Reuse involves explicit management of the problems of compilation, packaging, distribution, installation, configuration, implementation, maintenance and updating. The document identifies the purpose, recent trends and the future area of software reuse.

Introduction

In today's world, we are surrounded by a great deal of software, from its development to its use. Everyone wants the quality and performance of the software to be the best, without errors. To measure the quality of the software and its performance, we have the software metrics.

Software metrics play a very important role in software management. Software reuse programs are based on reuse metrics, along with other software parameters, on quality and commitment, to assess their effectiveness and the relationships between the different metrics used in organizations. Methodical gathering of different measurements all through the product advancement lifecycle figures out which practices ought to be connected and which ought to be kept away from to improve efficiency and quality in programming improvement and upkeep. The idea of reusing programming was part of the inheritance of programming from the beginning of the stored program. In the season of emergency programming, reuse proved to be excessively expensive.

Furthermore, its improvement was not verified by the bosses, at that time the reuse of programming reduced the overload. Reuse is important because the cost used in the advance and in the programming support decreases and, in addition, it improves the nature of programming. We evaluate and analyze the Software in light of the fact that reusing the product over and over will improve quality. Reuse of segments does not simply imply reusing code from an application and therefore the next involves reusing the structure, engineering and even different parts of the application [4]. This document is isolated in segments IV. Segment II clarifies the qualities of programming re-use, segment III clarifies the types of re-use, section IV clarifies the re-use measures and the last Vtell area the final and future vote.

2. Literature Review

Today, pcs assume an essential job in pretty much every part of our life. The expanded significance of programming additionally puts more prerequisites on it. Along these lines it is important to have exact, unsurprising, and repeatable authority over the product advancement procedure and item. Programming measures are a device to gauge the nature of programming.

The region of programming estimation or programming building estimation is one of the zones in programming designing where analysts are dynamic since over thirty years. The zone of programming estimation is otherwise called programming measurements. There is a confounding circumstance utilizing the terms programming measures or programming measurements. In writing the terms metric and measure are utilized as equivalent words. A measurement is here not considered in the feeling of a measurement space, it is considered as: estimation is a mapping of observational items to numerical articles. Programming quality ought to be directly identified with a product measure. This is an essential idea of estimation at all and of programming estimation.

An excellent programming reuse process added to improved efficiency, quality and reliability, notwithstanding helping the obtaining proficient in better administration of the timetable, cost and execution of a program or task. An underlying speculation is vital which can be generally paid for itself with the progression of time. Advancement of a reuse program and the reuse procedure utilized in that program can profit both to diminish gambling factors and new framework developments.

Barely any PC programming reuse undertakings include reusing of code. These sorts are as, fundamentally limit some time that software engineers request to achieve down to earth reuse undertakings, increment the odds that software engineers can effectively add up to commonsense reuse assignments, limit some time basic by developers to name infeasible reuse errands, notwithstanding) improve designers' great sense in their ability to control risk such tasks. For the fast improvement, limited expense and diminishing danger factor and best utilization of designers, programming reuse contains high advantages. At the beginning times of creating programming reusing brings about expanding benefits as it enables the past articles to be use in the progressive reuse of later stage objects.

3. Establishing a software reuse program

3.1 Software Reuse

A meaning of software reuse is the way toward making programming frameworks from predefined programming parts.
The upside of software reuse:

The efficient advancement of reusable parts.

The efficient reuse of these parts as structure squares to make new frameworks.

A reusable segment might be code, however the greater advantages of reuse originate from a more extensive and more elevated amount perspective on what can be reused. Programming particulars, structures, tests cases, information, models, plans, documentation, systems, and layouts are on the whole possibility for reuse.

Programming reuse can cut programming advancement time and expenses. The real favorable circumstances for programming reuse are to:

Increment programming efficiency.
Abbreviate programming advancement time.
Improve programming framework interoperability.
Create programming with less individuals.
Move work force all the more effectively from venture to extend.
Diminish programming improvement and support costs.
Produce increasingly institutionalized programming.
Produce better quality programming and give an incredible upper hand.
Select Business Solutions has been helping organizations accomplish programming reuse through both innovation and Component Based Development (CBD) approach for more than 10 years.
Arrangement Breakdown
Structure part and administration arranged frameworks with Select Solution Factory.
Actualize Select Perspective, the main Software Reuse and Component Based Development life-cycle with Select Process Director.

3.2 The reuse process and organization

Among the numerous issues which must be tended to are the product procedures to be pursued, and the structure and the executives of reuse associations. An all around characterized programming advancement process is essential to successful reuse. Since no single programming process model is perfect for all circumstances, diverse programming life-cycles portrays explicit jobs, exercises, techniques, measurements, obligations and expectations of the designers and chiefs. A reuse procedure makes express rules that will guarantee effective reuse. These incorporate unequivocal partition of segment designers (makers) and reusers (utilizers), and express reuse-situated administration and backing. Viable reuse requires huge association changes, normally traversing a few groups inside the organization. Such far reaching developments must be practiced with educated and unequivocal administration initiative.

Figure 1 demonstrates an essential reuse process model. To expressly bolster reuse-empowering rules, the model has four unmistakable procedure components:

Create: This procedure gives reusable resources suitable to the use procedure. Resources might be new, reengineered or bought, and of different sorts, for example, code, interfaces, designs, tests, and instruments. Exercises incorporate stock

and area examination of existing applications and resources, appraisal of use needs, market and innovation patterns, and design and part definition and development.

Utilize: This procedure reuses advantages for produce items (applications or frameworks). Exercises incorporate the examination of area models and resources, the investigation of item prerequisites, the adjustment of advantages, the advancement of items, and the particular of proposed segments.

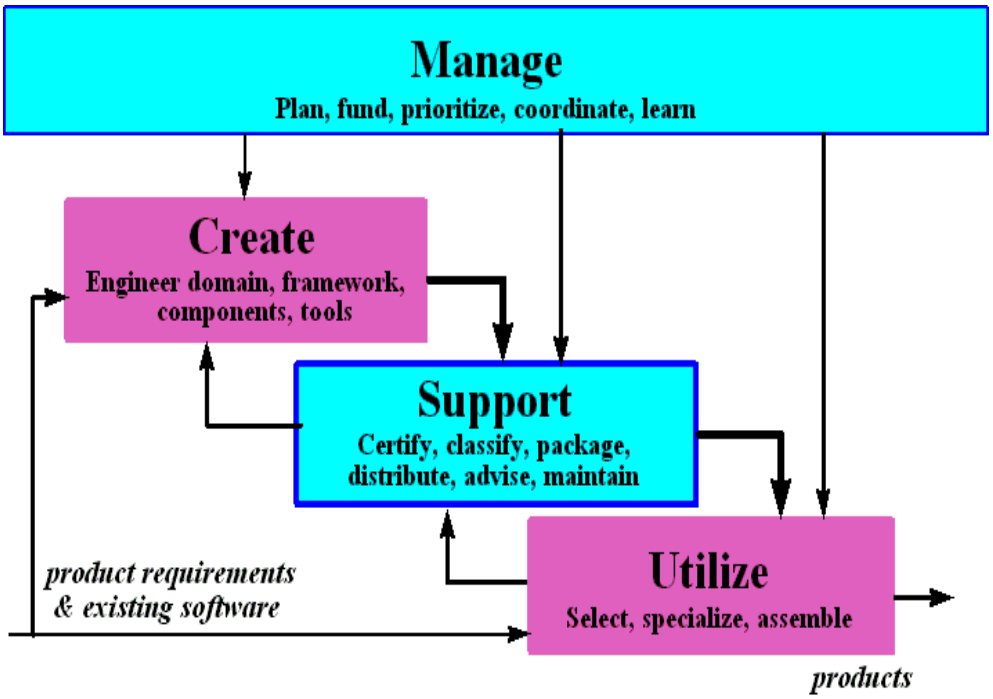


Figure 1: Ideal reuse process and organization

•Support: This procedure bolsters the general reuse process, overseeing and keeping up the benefit accumulation. Exercises incorporate the confirmation of new resources, grouping for library stockpiling, coordinating use needs with the benefits, giving utilization backing, and gathering criticism and deformity reports.

•Manage: This procedure obliges and controls different components. It must arrangement, start, asset, track, organize, facilitate and improve the reuse procedure. Exercises incorporate setting needs and timetables for new resource development, settling clashes when required resources are not accessible, building up preparing, and setting course.
Explicit measurements are expected to deal with the creation, backing and use of reusable programming, helping uses comprehend the estimation of the advantages, and helping makers organize speculations.

4. Methodology

A study led to check the results of new methodology created by stirring up the past various methodologies as specially appointed, part reuse and model based reuse and so on. The impact of this reuse approach checked through an overview. Quality properties like necessities and plan determination checked. Confirmation and Validation strategy apply that checks the quality and rightness amid the product life cycle. The study is done to guarantee programming Quality Attributes, while utilizing gauges for security basic programming frameworks and chose Verification and Validation procedures show the figure 2.

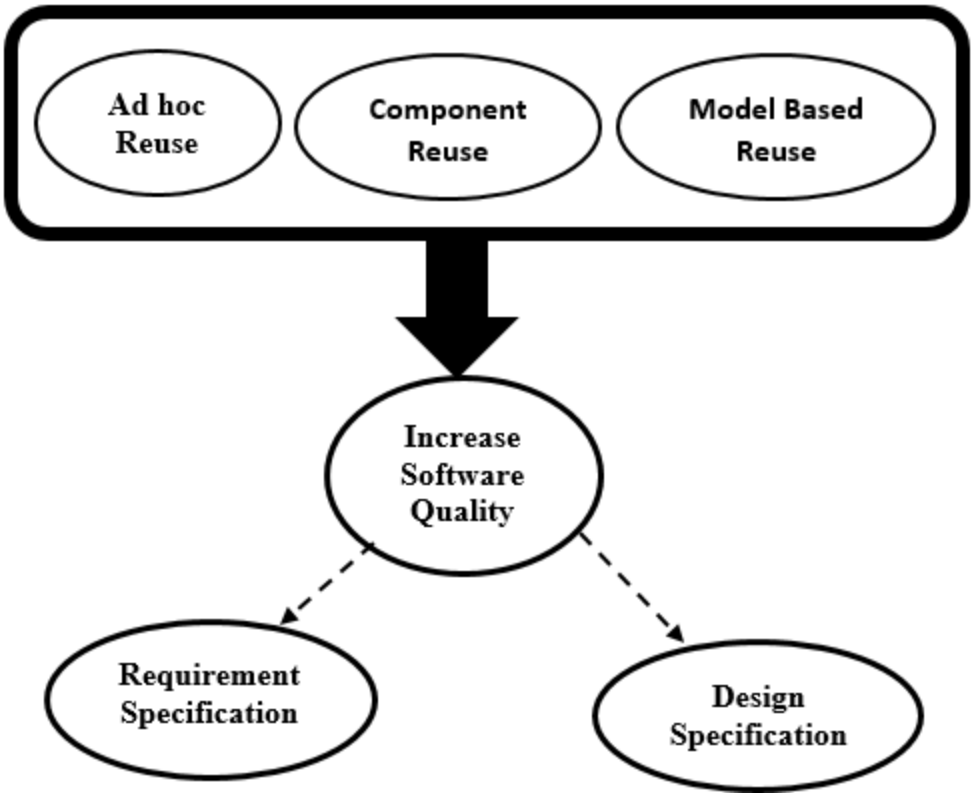


Fig. 2

To build up the proposed methodology following advances performed.
Review diverse old methodologies
Understanding what is programming reuse
Understanding the connection between quality SR
Select the quality ascribes to investigating
Purpose another methodology
Define the means of new methodology
Other procedures looked at
Survey directed
Risk factor examined
Result investigated

Programming quality accomplishment is the fundamental reason for this examination. As indicated by client quality is to satisfy his prerequisite at a worthy dimension of expense. As indicated by client quality is anything but difficult to utilize and work with programming. As per engineer quality is anything but difficult to configuration, keep up and reuse programming's parts and as per designer quality is low improvement cost just as high consumer loyalty. To satisfy these requests our methodology will utilize a poll that filled by various programming related people. Here spoke to some quality parameters.

4.1. Requirement specification

The proposed model concentrated on the imperfections in the necessities detail stage and further connection between the shortcoming in prerequisite particular and its underlying foundations to guarantee the nature of the SRS report. The method helps in preparing the SRS creators, and supports making progressively careful and right flaw anticipation and issue location procedures. The necessities may influenced from lack and the nearness of different and unpredictable words exists in SRS due to the definitive, multicultural and particular essentials. The rate of SRS in light of class properties masterminds the challenge strategies as repeated clarifications.

4.2 Design specification

The design stage executes a critical part in connecting the necessities stage and the usage stage, amid software development phases and it is a method of converting necessities of software

Inspired by programming social orders. The system reserved of a product technique depiction practice and a strategy for programming plan thought. Dependable with the said structure geographies, a plan examination technique has been arranged. Perceived properties, for example, accuracy, consistency, breadth and reusability could be affirmed in configuration level.

Check and approval are two significant terms that are utilized in the business of testing or quality protection. The two of them appear to something very similar. In any case, the two of them are minimal diverse with regards to the universe of programming item. Programming or any related item needs to experience the total cycle of improvement and testing before being propelled in the market. Confirmation assesses all the fundamental things identified with the item being created. It is imperative to make reference to that amid confirmation, one isn't trying the real item.

This is new investigation for all association structures and programming creators to choose distinctive experiences they had with different sorts of reuse. This information can be used to look at and relate best applies for the reuse. It can in like manner be used as data toward fuse as a piece of suggestions, for research papers and specialists proposition. Following inspirations exists behind this audit:

A philosophy is the choice of methodology or blend of strategies the program imprints to use reuse
A technique is the interesting from the improvement procedures that considers reuse
Component-developed reuse is arranged in light of successfully made parts or planned for reuse on a fragment premise.
Model-based reuse can't avoid being reuse that relies upon reusing models produced using diverse tasks or sections.
Product line reuse can't avoid being reuse in light of a systematized yet tailorable item advertising.
Ad Hoc reuse that the expert thinks about, that happens to meet an essential yet was not expected for reuse.

5. Use of Metrics to Indicate Reuse Effectiveness

5.1 Why Metrics

There are four purposes behind estimating programming procedures, items and assets:
To characterize
To evaluate
To predict
To improve

Software reuse is not unlike any other software development process in that in order to correctly establish its usefulness we must have methods to quantify it. Quantifying reuses effectiveness in order to justify the initial cost of implementing a program may very well be the greatest factor as to why it is not more widely put into practice. The most important software reuse measurement problem at present is the lack of data collection. This statement should not be taken out of context, since measurement data should not be collected without first defining the purpose of measurement.

5.2 Software Reuse Metrics and Models

As an association executes the product reuse program to improve profitability and quality. Commonly for complex IT associations and organizations it is extremely hard to quantify the product's profitability and guarantee the nature of the product. In this way, we have programming measurements to quantify the application detail to give objective and any repeatable information for making an improvement in the appropriate regions. Associations executing efficient reuse must almost certainly measure

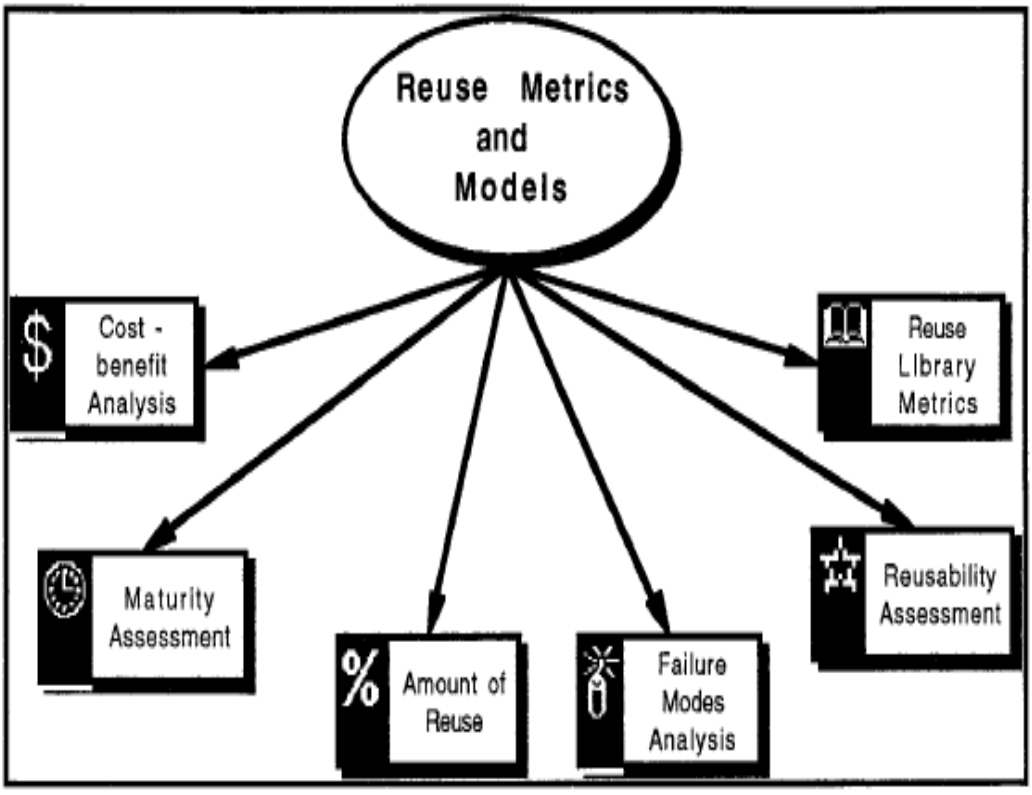
•Quantify their advancement
•Identify the best reuse systems
A measurement is a quantitative marker of a property of a thing (programming, process, and so forth). A model determines connections among measurements (and outside characteristics).

6. Types of reuse metrics

For manufacture any software reusable, at the time of design only we have to think of the reusable properties so that design of that software is fit for major variation in the requirements. Once any variation is made to any segment of that software then all the segments of that software should familiarize to that changes. This is seamless system for the reuse.
We have 6 types of metrics to quantity the excellence-
Reuse Cost- Benefits
Maturity Assessment
Amount of reuse
Reuse Library metric
Failure mode
Reusability Assessment

Reuse cost-benefits simulations include financial cost/benefit analysis as well as quality and output payoff. *Maturity assessment* models classify reuse programs by how forward-thinking they are in executing systematic reuse. *Amount of reuse* metrics are used to assess and monitor a reuse improvement effort by tracking parts of reuse for life cycle objects. *Failure modes analysis* is used to detect and order the disorders to reuse in a given institute. *Reusability* metrics designate the probability that an thing is reusable. *Reuse library metrics* are charity to manage and road usage of a reuse repository.

Governments often chance meeting the need for these metrics and models in the direction obtainable. Software reuse can smear to any life cycle product, not only to wreckages of source code. This means that developers can follow reuse of requirements documents, system stipulations, design structures, and some other development object



6.1 Reuse Cost-Benefits

As organizations expect systematic software reuse, the first inquiry that will arise will possibly concern budgets and benefits. Establishments will want to explain the cost and time elaborate in systematic reuse by reckoning these costs and budding payoffs. Cost benefit analysis models include economic cost-benefit models and quality and production payout analyses. Numerous reuse cost-benefit models have been stated. None of these models are resulting from data, nor have they been authenticated with data. Instead, the reproductions allow a user to pretend the tradeoffs between important

economic bounds such as cost and output. These are projected by setting arbitrary values for cost and output measures of systems deprived of reuse, and then approximating these parameters for systems with reuse.

This method involves addition up the welfares of the course of action we do and relate these with the cost allied with it. One example- suppose we make a health tender in which we create some new modules and reuse some preceding modules. If the combined application is cost effectual than the normal application then we will use that application only. I estimate this example to explain u that if reusing any module stretches u benefit in financial term then only we will reuse that part. Rate of any reuse program be contingent upon small and large application domain. If we have small application domain in which only few mechanisms are reused then cost connected with it is less and cunning of the cost is also easier. There is considerable cohesion among the models, as labeled in the following:

Cost/Productivity Models
Quality of Investment
Business Reuse Metrics

6.1.1 Cost/Productivity Models

Gaffney and Durek [1989] propose two cost and productivity models for software reuse. The simple model demonstrations the cost of reusing software mechanisms. The cost-of-development model shapes upon the simple model by expressive the cost of developing recyclable components.

The humble model works as follows:

Let C be the cost of software growth for a given product comparative to all new code (for which C =1). R is the amount of reused code in the product (R <= 1). B is the cost relative to that for all new code, of integrating the reused code into the new products (b=1 for all new code).
The relative cost for software development is:
[(relative cost of all new code) * (proportion of new code)] +
[(relative cost of reused software) *(proportion of reused software)]

The equation for this is: $C = (1)(1- R) +(b)(R)$
= [(b-1)R]+1
And the agreeing relative productivity is: P =1/C
= 1/((b-1)R+1)

6.1.2 Quality of Investment

Barnes and Bollinger [1991] observed the cost and risk features of software reuse and advised an analytical method for making good reuse savings. Reuse actions are divided into Producer events are reuse savings, or costs experienced while making one or more work products easier to reuse by others. Customer activities are reuse assistances, or measures in dollar bill of how much the past reuse outlay helped or hurt the efficiency of an activity. The *total reuse benefit* can then be found by estimating the reuse benefit for all consequent activities that income from the reuse investment.

The *quality of investment* (Q) is the ratio of reuse profits (B) to reuse investments (R):
$Q = B/R$
If Q is less than one for a reuse power, then that effort resulted in a net financial loss. If Q is greater than one, then the investment providing a good return. Three major plans are identified for increasing Q:
Increase the level of reuse
Reduce the average cost of reuse and
Reduce the investment needed to achieve a given reuse benefit

6.1.3 Business Reuse Metrics

Poulin et al. [1993] present a set of metrics used by IBM to estimate the effort saved from reuse. The study assesses the potential benefits compared to the time and resources needed to identify and integrate reusable software into a product. Although the measures used are similar to the already discussed cost / productivity models [Gaffney and Durek 1989], the metrics are named from a business perspective and provide a more accurate breakdown for some calculations. For

example, the cost is divided into development and maintenance costs. Metrics are derived from a set of data elements defined in Table 1

Data Element	Symbol
Shipped source instructions	SSI
Changed source instructions	CSI
Reused source instructions	RSI
Source instructions reused by others	SIRBO
Software development cost	Cost per LOC
Software development error rate	Error rate
Software error repair cost	Cost per error

Table 1: Observable Data [Poulin et al. 1993

Taking into account the above data, the following metrics are defined:

• Percentage of re-use: reflects the quantity of product that can be attributed to re-use.
(This is comparable to R in the Gaffney and Durek model). Poulin et al. Distinguish the percentage of re-use of a Product, the percentage of reuse of a product version and the percentage of re-use for the entire organization.
Percentage of product re-use = RSI / (RSI + SSI) * 100 percent
• Avoids cost reuse: measures reduced total product costs as a result of reuse. (This is equivalent to 1-RC in the Gaffney and Durek model). Poulin et al. Calculate that the financial benefit attributable to reuse during the development phase is 80% of the cost of developing a new code, derived
Of studies showing that the cost of integrating an existing software is equal to 20% of the cost of a new development (b in the Gaffney and Durek model). This study also recognizes the savings made during the maintenance phase, since the reused software usually contains fewer errors; The total elimination of re-use costs is calculated as the sum of the avoidance of costs in development and maintenance activities.
Development cost prevention = RSI * 0.8 * (cost of the new code)
Service cost avoidance= RSI*(error rate)*(new code cost)
Reuse cost avoidance= Development cost avoidance + Service cost avoidance
• Added reuse value: a productivity index that differs from previous definitions of relative productivity by including it in the definition of the source code of the reused code that is reused within the product and the source code that other users reuse.
Added reuse value = (SSI + RSI + SIRBO) / SSI
• Additional development costs: increased product costs as a result of reusable software development (such as E in the Barnes and Bollinger model). This study estimates the cost of the additional effort at 50% of the cost of the new development.
Additional development cost = (relative re-use cost-1) * code written to be reused by others * cost of the new code.
The relative cost of writing for reuse is the cost of writing a reusable code in relation to the cost of writing a code for a single use. The code written to be reused by others is the kloc code written to be reused by the startup project.

6.2.1 Use of Cost Benefit Tool

Brainstorm costs and Benefits:

In this we consider the overall project requirements, its needs and the resources e.g.-the labor required in the project, the type of hardware and software required etc. We should consider all the cost associated with the project and the benefits that we achieved with that resources.

Assign a monetary value to Cost:

This includes cost of physical resources needed, as well as cost of human effort that is involved in every phase of project. Sometime if the technology is new or the project is higher than the training cost is also considered.

Assign a monetary value to the Benefits:

In this we quantify in monetary terms the benefits arising out of the projects.so that if the quantity is not according to the cost we invested in the project then cost-benefit analysis gives no benefit. In software world documentation and quantification of benefits is very hard and time consuming.

Compare Cost and Benefits:

In this we match the cost of our costs to the value of our benefits and then use this analysis to decide our course of action. Calculate the total costs and the total benefits and then compare the two of them to determine whether our benefits outweigh our costs or not. In many projects we can't see the benefits after the completion of the project it takes time so to we use the term Payback period- how long would it take to reach to break point means the point at which benefits have started repaying the cost. This is a very good tool to decide whether to pursue a project or not.

6.3 MATURITY ASSESSMENT

The maturity models of reuse support an assessment of how advanced reuse programs are in implementing systematic re-use, using an ordinal scale of re-use phases. They are similar to the capacity maturity model developed at the Software Engineering Institute (SEI) at Carnegie Mellon University [Humphrey 1989]. A maturity model is the core of planned reuse, which helps organizations understand their past, current and future goals for re-use activities. Different models of maturity for reuse have been developed and used, although they have not been validated.

Koltun and Hudson Reuse Maturity Model

Koltun and Hudson [1991] developed the model of reuse of maturity indicated in the table 9 columns indicate stages of maturity of reuse, which should improve on an ordinary scale of 1 (Initial / Chaotic) 5 (Integrated). The rows correspond to the dimensions of the re-use maturity as motivation / culture and planning for reuse. For each of the ten dimensions of reuse, the amount of commitment and commitment of the organization increases as we move from the first organization / chaotic rooted following recycling. Integrated reuse incorporates fully automated support tools and accurate reuse measures to monitor progress.

To use this model, an organization will evaluate its maturity reuse before starting a program to improve reuse by identifying its location in each dimension. (In our experience, most organizations are between Initial / Chaotic and Supervised at the beginning of the program). The organization will use the model to guide the activities. This must be done to reach higher levels of maturity of re-use. Once the organization realizes integrated reuse, reuse becomes part of the business routine and will no longer be recognized as a separate discipline.

6.3.2 SPC Reuse Capability Model

Reuse capability model developed by the Software Productivity Consortium [Davis 1993] has two components: an The assessment model and an implementation model.
The assessment model consists of a set of categorized critical success factors (stated as goals) that an organization can use to assess the present state of its reuse practical

	1 Initial/ Chaotic	2 Monitored	3 Coordinated	4 Planned	5 Ingrained
Motivation/ Culture	Reuse discouraged	Reuse encouraged	Reuse incentivized re-enforced rewarded	Reuse indoctrinated	Reuse is the way we do business
Planning for reuse	None	Grassroots activity	Targets of opportunity	Business imperative	Part of strategic plan
Breadth of reuse	Individual	Work group	Department	Division	Enterprise wide
Responsible for making reuse happen	Individual initiative	Shared initiative	Dedicated individual	Dedicated group	Corporate group with division liaisons
Process by which reuse is leveraged	Reuse process chaotic; unclear how reuse comes in.	Reuse questions raised at design reviews (after the fact)	Design emphasis placed on off the shelf parts	Focus on developing families of products	All software products are genericized for future reuse
Reuse assets	Salvage yard (no apparent structure to collection)	Catalog identifies language and platform specific parts	Catalog organized along application specific lines	Catalog includes generic data processing functions	Planned activity to acquire or develop missing pieces in catalog
Classification activity	Informal, individualized	Multiple independent schemes for classifying parts	Single scheme catalog published periodically	Some domain analyses done to determine categories	Formal, complete, consistent timely classification
Technology support	Personal tools, if any	Many tools, but not specialized for reuse	Classification aids and synthesis aids	Electronic library separate from development environment	Automated support integrated with development environment
Metrics	No metrics on reuse level, pay-off, or costs	Number of lines of code used in cost models	Manual tracking of reuse occurrences of catalog parts	Analyses done to identify expected payoffs from developing reusable parts	All system utilities, software tools and accounting mechanisms instrumented to track reuse
Legal, contractual, accounting considerations	Inhibitor to getting started	Internal accounting scheme for sharing costs and allocating benefits	Data rights and compensation issues resolved with customer	Royalty scheme for all suppliers and customers	Software treated as key capital asset

Table 2: Hudson and Koltun Reuse Maturity Model

6.4 Amount of Reuse

The number of reuse metrics is used to evaluate and monitor an effort to improve reuse by monitoring the re-use percentages of life-cycle objects over time. In general, the metric is:

Amount of life cycle object reused.
Total size of life cycle object

A common form of this metric is based on the lines of code as follows:

Lines of reused code in system or module.
Total lines of code in system or module

Frakes [1990] Terry [1993] and Frakes and Terry [1994] extend the "quantity of reuse" metrics define base reuse level including factors such as the level of abstraction of life-cycle objects and formal definitions from within and from external. Reuse. Work is also being done [Bieman and Karunanithi 1993; Chidamber and Kemerer 1994] to define specific metrics for object-oriented systems. The following sections discuss this work in detail. To date, he has done little work in the amount of metrics for re-using generative reuse, although Bigger staff [1992] implicitly eliminates this metric reporting a specific effort-generated source code report for the Genesisenesis40 KSLOC system during 30 minutes of effort specifications.

5.3.1 Reuse Level

The basic dependent variable in efforts to improve software reuse is the level of reuse [Frakes 1993]. Measuring the level of reuse assumes that a system is composed of parts at different levels of abstraction. The levels of abstraction must be defined to measure reuse. For example, a C-based system consists of modules (.c files) that contain functions and functions that contain lines of code. The level of reuse of a C-based system, therefore, can be expressed in terms of modules, functions or lines of code. A software component (lower level element) can be internal or external. A lower level internal component is one developed for the higher level component. The top level component uses a lower level external component, but was created for a different element or for general use.

The following quantities can be calculated considering a higher level element composed of lower level elements:

L = the total number of lower level items in the upper level entry.

E = the number of lower-level elements in an external repository in the top-level element.

I = the number of lower level elements in the upper level element that do not come from an external repository.

M = number of items that do not come from an external repository that are used more than once.

These counts are of unique components (types), not tokens (references). Given these quantities, the following reuse level metrics are defined [Frakes 1990]:

External reuse level: E / L

Internal level of reuse: M / L

Total re-use level = External re-use level + Internal re-use level.

Internal, external and total reuse levels will take values between 0 and 1. More re-use occurs when the value of the re-use level approaches 1. A re-use level of 0 indicates that it cannot be reused.

The software tool calculates the level of reuse and frequency for the C code. Given a set of C files, the instrument reports the following information:

1. Level of internal re-use;

2. Level of external re-use;

3. Level of total re-use;

4. Frequency of internal re-use;

5. Frequency of external re-use;

6. Total reuse frequency

5.3.2 Reuse Metrics for Object-Oriented Systems

Bieman [1992] and Bieman and Karunanithi [1993] proposed the re-use of metrics for object-oriented systems. Bieman 1992] identifies three points of view from which to see reuse: server, client and system. The server perspective is the perspective of the library or a particular component of the library. The analysis focuses on how customers reuse the entity.

From the customer's point of view, the goal is to know how a particular program entity reuses other program entities. The system perspective is a view of reuse in the system in general, including servers and clients. A server's class reuse profile characterizes the way client classes reuse the class. The textual reuse of the server in an object-oriented system is basically the same as for procedural systems, using object-oriented terminology. Reuse of the leveraged server is supported by inheritance. A client can reuse the server by extension, add methods to the server, or overload, redefine methods. (Note that mcgregor and Sykes [1992] offer good definitions of object-oriented terminology.

Chidamber and Kemerer [1994] propose a set of metrics for object-oriented design. The article defines the following metrics based on measurement theory:

A) class-weighted methods;

B) depth of the hereditary tree;

C) number of children;

D) Coupling between classes of objects;

E) answers for a class.

5.4 Software Reuse Failure Modes Model

Implementing systematic re-use is difficult, as it involves technical and non-technical factors. Error mode analysis provides an approach to measure and improve a reuse process based on a model of the ways in which a reuse process can fail. The model of the reuse mode of reuse reported by Frakes and Fox [1996] can be used to evaluate the quality of a systematic re-use program, to determine the impediments to reuse in an organization and to design an improvement strategy for a program of reuse. Systematic reuse.

Given the many factors that can affect reuse success, how does an organization decide what to address in its reuse improvement program? This question can be answered by finding out why reuse is not performed in the organization. This can be done by considering the methods of reuse, ie the ways in which reuse can fail.

The reuse error mode model has seven error modes corresponding to the steps a software engineer must complete to reuse a component. The error modes are:

♣ The part does not exist

♣ The part is not available

Parte part missing

♣ Part is not included

♣ The part is invalid

♣ The part cannot be integrated

Each error mode has error causes associated with it. ª No attempt to reuse, º has among its causes of failure, for example, resource limitations, no intention of re-use and lack of education. To use the model, an organization collects data on the methods and causes of non-reuse and then uses this information to prioritize reuse improvement activities.

5.5 Reusability Assessment

Another important area of measurement of reuse relates to the estimation of reuse of a component. These parameters are potentially useful in two key areas of re-use: re-use of projects and re-engineering for re-use. The essential question is: are there any measurable attributes of a component that indicate its possibility of reuse? If so, then these attributes will be objective to reuse design and re-engineering. One of the difficulties in this area is that reuse attributes are often specific to certain types of reusable components and to the languages in which they are implemented. In this section we examine the work in this area.

In a study by NASA software, Selby [1989] identified several attacks on the module that distinguished the black box reuse modules from others in his sample. Attributes include:

♣ Less calls to the module by source line;

♣ Less I / O parameters per source line;

♣ Less reading / writing instructions per line;

Commento More comment on the proportions of the code;

♣ More calls to the utility function by source line;

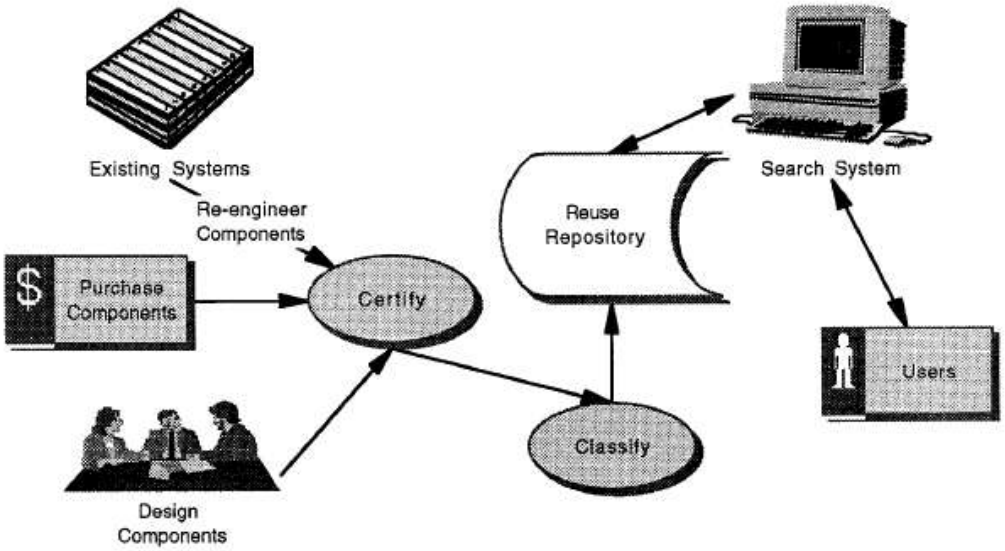
♣ Less origin lines.

These data suggest that the modules that have these attributes will be more reusable. Basili et al. [1990] reported on two systems of reuse of studies written in Ada. The first study defined data link measurements to characterize and identify reusable components. Data linking is the exchange of global data between program elements [Hutchins and Basili 1985]. Basili et al. [1990] proposed a method to identify data links within a program. After the identification of the connections, a cluster analysis is performed and the coupling forces are weighed. A pairing is based on references to variables and parameters (data links). Aliasing, or reference, is not taken into consideration; only one level of data links is considered. Group analysis identifies which modules are strongly coupled and may not be valid candidates for reuse and which modules are considered independent of the others and potentially reusable.

The second study defines an abstract measure of the reuse of Ada components. Potentially reusable software is identified and a method is defined for measuring distances from that ideal.

5.6 Reuse Library Metrics

As can be seen in Figure 3, a reuse library is a repository for storing reusable resources, as well as an interface for searching the repository. Library resources can be obtained from existing systems through redesign, designed and built from scratch or purchased. Thus, the components are usually certified, a process to ensure that they have desired attributes, such as tests of a certain type. The components are classified so that users can search for them effectively. The most common classification schemes are enumeration, facets and indexing of free text [Frakes and Gandel 1990]. The evaluation criteria for indexing schemes of re-use libraries are: costs, search for effectiveness, support for understanding and efficiency.



Indexing costs include the cost of creating a classification scheme, maintaining the classification scheme, and up-dating the database for the scheme. These concerns are negligible for a small collection, but become more important as the assembly

grows. Each of these costs can be measured in terms of dollars or effort. These costs can be normalized by dividing total costs by the number of components handled by the library.

Quality of assets is alternative important facet of a reuse library. There is considerable anecdotal evidence that this is the most important factor in defining successful use of a reuse library. Frakes and Nejmeh [1987] proposed the following metrics as indicators of the quality of assets in a reuse library.

Time in Use: the module should have been used in one or more systems that have been released to the field for a period of three months.

Reuse Statistics: the extent to which the module has been successfully re-used by others is perhaps the best indicator of module quality.

Reuse Reviews: favorable reviews from those that have used the module are a good indication that the module is of higher quality.

Complexity: overly complex modules may not be easy to modify or maintain
Inspection: the module should have been inspected.
Testing: the modules should have been thoroughly tested at the unit level with statement coverage of 100 percent and branch coverage of at least 80 percent.
Time on-line (system availability): this is a measure of the number of hours the system is available for use.
These metrics can provide good management information for a library system. They can be used to demonstrate the value of the library for administration and to provide information for continuous quality improvement.

Conclusion and Future Scope

N this document, we present a survey on reuse and types of reuse that reduces development time and effort and the most important cost of the application, since we don't have to do all the parts of that application from scratch. Some parts are reused. A re-use program must always be planned, done carefully and must be systematic to give maximum benefit. Software reuse strategies are implemented in an organization to improve software quality and productivity. To measure the quality and productivity of software metrics are used. The metric is a quantitative indicator of an attribute. For reuse, knowledge of the domain is very important. Without adequate knowledge of what to use and where to use the application, it is not possible to provide us with a quality product with efficient time. So domain engineering plays an important role in reuse. Knowledge of the domain is the key concept of systematic reuse.

7. References

[1] T.Karthikeyan, J.Geetha, “A Study and Crtical Survey on Service Reusability Metrics”,I.J. Information Technology and Computer science”,2012,5,25-31.
[2] Arun Sharma, Rajesh Kumar & P.S Grover. “A Critical Survey of Reusability Aspects for Component Based”.
[3] William Frakes Carol Terry, “Software Reuse and Reusability Metrics and Models”, Virginia techcomputer Science Department2990 Telestar Ct. And Applied Expertise, Inc.1925 N. Lynn St. Suite 802Arlington, VA 22209.
[4] Jeffrey S. Poulin, “Measuring Software Reusability”, Loral Federal Systems–Owego.
[5] Jorge Cláudio Cordeiro Pires Mascena, “A Comparative Study on Software Reuse Metrics and Economic Models from a Traceability Perspective”, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira Federal University of Pernambuco.
[6] AGRESTI, W. AND EVANCO, W. 1992. Projecting software defects in analyzing Ada designs.
[7] Tutorial: Software Reuse—Emerging Technology, W. Tracz, Ed. IEEE Computer Society Press, Washiington, D.C.
[8] BARNES, B. AND BOLLINGER, T. 1991. Making software reuse cost effective. *IEEE* Softw. 1,
13–24.
[9] BASILI, V. R., ROMBACH, H. D., BAILEY, J., AND DELIS, A. 1990. Ada reusability and measurement. Computer Science Tech. Rep. Series,
University of Maryland, May.
[10] W. Frakes and C. Terry, “Software Reuse: Metrics and Models,” ACM Computing Surveys, vol. 28, pp. 415-435, 1996.