Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 18 Issue 4 Version 1.0 Year 2018
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Rework and Reuse Effects in Software Economy

By Md.Shahadat Hossain

Independent University Bangladesh

Abstract- Software industry supposed to provide software product to their customers at a lower price and right time. Unfortunately, it's true that the industry can't deliver the software at lower a price. Lots of reasons are responsible for this high price of the Software. Such as high wages of stakeholders, the size of software, testing costs, implementation cost and one of the most vital reasons is a rework that increases the cost of software. In this research paper, I focused on rework and reuse, its cost & effect on software economy. How to reduce the rework during the software development life cycle-SDLC. This research found that a long part of the development duration used for rework. This scenario is not only obtainable in a small software firm but also medium and enterprise software companies. Rework issue is one of the big challenges of the software industry. This research explained the problem in a financial point of view and provided needed suggestions to reduce the rework & increase the reuse based on software engineering body of knowledge. The software industry will be profitable if they can reduce the rework and upsurge the reuse of software.

Keywords: software, rework, reuse, economy, quality, time, cost, stakeholders.

GJCST-C Classification: K.6.3

REWORKANDREUSEEFFECTSINSOFTWAREECONOMY

Rework and Reuse Effects in Software Economy

Md.Shahadat Hossain

Abstract- Software industry supposed to provide software product to their customers at a lower price and right time. Unfortunately, it's true that the industry can't deliver the software at lower a price. Lots of reasons are responsible for this high price of the Software. Such as high wages of stakeholders, the size of software, testing costs, implementation cost and one of the most vital reasons is a rework that increases the cost of software. In this research paper, I focused on rework and reuse, its cost & effect on software economy. How to reduce the rework during the software development life cycle-SDLC. This research found that a long part of the development duration used for rework. This scenario is not only obtainable in a small software firm but also medium and enterprise software companies. Rework issue is one of the big challenges of the software industry. This research explained the problem in a financial point of view and provided needed suggestions to reduce the rework & increase the reuse based on software engineering body of knowledge. The software industry will be profitable if they can reduce the rework and upsurge the reuse of software.

Keywords: software, rework, reuse, economy, quality, time, cost, stakeholders.

I. Introduction

Rework is an ongoing problem in the software development process. Rework is generally considered to be potentially avoidable work that is triggered to correct problems or to tune an application (Aaron G. Cass, 2003). Many software firms are confused to isolate the rework. They think we are working to solve the existing problem & it is part of our maintenance, routine work. Now the point is how to differentiate the rework. Rework in software development is the additional effort of redoing a process or activity that was incorrectly implemented in the first instance or due to changes in requirements from clients (Vimla Devi Ramdoo1, 2015). Rework is defined as work measures that have to be completed more than once (Robin McDonald, CCM, LEED G.A., 2013). Peter E.D. Love1 characterized rework as the "unnecessary process of redoing a work activity that was incorrectly carried out the first time." Another definition which emphasizes the essence of rework is "work that is made to conform to the original requirements by completion or correction at least one extra time due to non conformance with requirements". The term rework is clearly defined here. Now the question is what is the source of rework? How can we reduce the rework? What is the cost of rework and what is the effect of rework in the software economy? This research paper not only answering these questions but also explaining the benefit & values of reuse. The term "Reuse" is used for developing the software by using the existing software components. These reusable components are projected assets. This research recommends to software engineers for design and develops software in such a manner so that a software component can be reused in multiple software. This research found that a few software firms are using existing reusable components, but the number of reuse is not satisfactory. Maximum components of the new software are being developing from scratch. Because the existing software components didn't build for reuse. Although some components of previous software were developed for reuse, but all of those components are impossible to reuse due to technology upgrade and requirements changes. The ratio of reuse of already available software components is very limited. It is one of the barriers to the success of the software industry.

This research was performed in a few software companies on multiple projects. A single project is not suitable for this type of research because let's say the first project is fresh it has no reusable component. The first project will develop reusable components. Then in the next project development, reusable components of the first project can be reused. But at the first project, the chance to rework can't ignore. Software reuse is accomplished by creating programs from previously developed software modules (Robert W.Therriault, 1994). Reuse is expected to lead to reduced system development time arid maintenance while increasing reliability by using existing working modules (Robert W.Therriault, 1994).

Author: Department of Software Engineering, School of Engineering & Computer Science, Independent University Bangladesh (IUB).
e-mail: shahadat.se@gmail.com

II. Purpose

The main purpose is to increase existing reuse levels to at least one step upper level and reduce the rework at least one step lower level. Development of project assets for decreasing rework and increasing reuse level of Software, Company. To meet this purpose Software Company must identify.

1. What is their current rework level ?
2. What is their current reuse level ?

First of all, defines the current position of the company where it exists now in rework and reuse level criteria. It is the very important job for software firm and complicated to define the level. A lot of the sensitive issues involved with it. If Software Company can't measure the current level of rework and reuse, then it can't estimate target level.

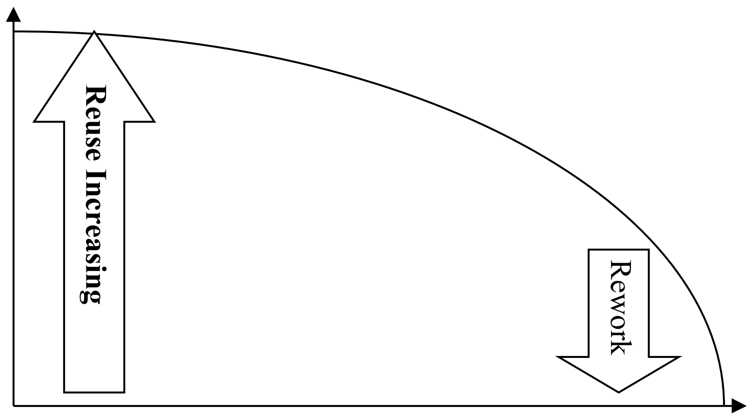


Figure 1: Purpose

Measurement of effect for rework & reuse: The effect of rework and reuse is measured by settings some performance parameters. One parameter is proportionally related to another. Some parameters are upward and some of them are downward. Upward parameters tend to the opposite with downward factors. Such as, if reuse arises, then time will fall, rework will fall, the development cost will drop. If project time rises, then the project cost will increase. The ultimate goal is to save the time, reduce cost and increase the profit margin. Here time and cost will sink parallel way with sinking rework. Profit will increase with the increase of quality & reuse.

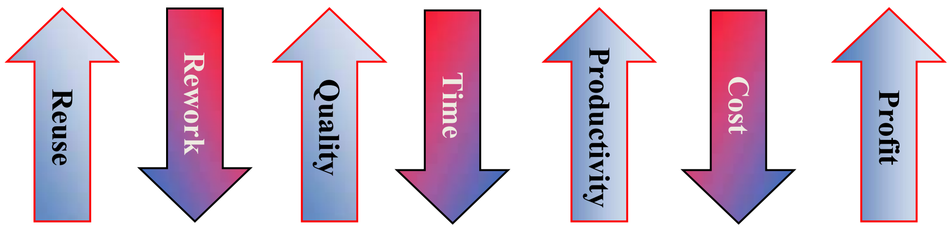


Figure 2: Effect Measurement Parameters

The rework, cost & time are expense related heads. The reuse, quality, productivity & profit is income associated heads. The opposite is happening when downward parameters grow up, and upward constraint goes down. When a rework is arisen in any part of a project such as, in a specific module or, same is happening in a small component, then the time of development is raising instantly. As a result the cost of development rises. The price may be $1 to $1000, depends on the project, its stage of SDLC and on the type of rework. Oppositely when there is no reuse, or the race becomes very limited in a project, then its quality may be having suffering, productivity may collapse and profit margins may plummet.

Actually, every software company want to

- Reduce project time
- Reduce project cost
- Reduce rework
- Reduce customer dissatisfaction
- Increase reuse
- Increase quality
- Increase productivity
- Increase profit

If rework is decreased, then time will cut cost will shrink finally the profit margin of the company will upturn. If the reuse is increased then, the time of development will save, the cost of perfection will diminish ultimately the growth of revenue will upsurge. The major factor is R2. Reduce the Rework and increase the reuse. That is the prime focus of this research paper.

Project Assets: Project assets are the reusable component of a project of a company. Project asset is developing by the predefine process. A process has multiple elements such policy, procedure up to 10th elements based on manner. A procedure that has followed ten elements known as standard method and the component that developed with this ten elements is the reusable component.

1. Purpose or Output
2. Performance Parameters
3. Policies
4. Procedures
5. Standards
6. Knowledge, Skills & Environment
7. Tools & Techniques
8. Measurements

9. Control
10. Improvement

For example a Bank account opening form, it is projected assets of the Bank. The predefined format is the standard of the Bank, the printed hard copy of account opening form and pen are the tools. Bank reuses the form information of customers that reduce Bank staff rework and save Banking times. That directly saves the cost of overtime work of the Bank. For the purpose of finding out current rework and reuse level first of all need to figure out five projects.

Following are a list of five projects:

Project Code	Name of the project	Nature of the project	Stakeholders	Remarks
P001	Hospital Management Information Software	Health Care Information System	Renown Hospital and Diagnostic Laboratories	
P002	Eye Care System	Ophthalmic EMR	Renown Eye Hospital and Institute	
P003	Trade Marketing & Distribution system	Manufacturing and Distribution System	Group of Company	
P004	Pharmaceutical ERP	Web Based Interactive ERP	Joint Venture Company	
P005	Digital Health Card System	GPRS Health Tracking	Government Staff	

Figure 3: List of five projects

a) *Advantages of reuse of software components*

Software reuse can cut software development time and costs. The major advantages of software reuse are to:

- ➢ Faster time to market
- ➢ Less effort
- ➢ Time-saving
- ➢ Increase software productivity.
- ➢ Utilize fewer resources
- ➢ Shorten software development time.
- ➢ Improve software system interoperability.
- ➢ Develop software with fewer people.
- ➢ Move personnel more easily from project to project.
- ➢ Reduce the systems development expenditures
- ➢ Reduce the software implementation and maintenance costs.
- ➢ Produce more standardized software.
- ➢ Produce better quality software and provide a powerful competitive Advantage.
- ➢ Leads to better quality software
- ➢ Reduce bugs

Reusable component development phase: In any phases of the software Development life cycle-SDLC, software engineers can develop reusable components. For example requirements specification stage, coding time, documentation segment design part, etc. Let say in requirement specification period, the requirement engineer detected that a requirement is repeated over several systems. So the engineer has to note that this portion of chucks corresponds to the well-defined set of necessities, modules then the engineer can reasonably expect to be able to reuse the requests Module. Similarly, in the coding level when a coder sees the same code is needed to write in multiple blocks, then coder can create a function, or it may be others object depends on the programming language that is used and it can be used, or call at any chunk of code where necessary. The software engineer can reuse the design of the existing subsystem as the design of the new subsystem, the test plan of the existing subsystem as the test plan of The new subsystem as well as others subsystems. Engineers can reuse existing the database schema for new database schema and create new object as well as Modify existing objects as per necessity. Reusable component development is not Phase dependent. Software reusable component can develop in any phase of development life cycle.

Reusable components of a software: Software reuse is accomplished by creating programs from previously developed Software modules. Many different aspects of software can be reused. Some of the constituents that can be reused are as follows:

- Plans
- Software requirement specifications
- Source code
- Software architecture
- Design and user interface
- User manuals
- Software documentation
- Database
- Algorithms
- Test case
- Templates
- Tools
- Procedure
- Plugins
- API
- Queries, reports
- Concepts and domain knowledge
- Implementation & experiences
- Objects and text
- Process
- Library
- Artifacts
- Modules
- Master setup data
- Models
- Themes
- Function
- Package
- Dynamic action
- Template

b) *Reuse percentage of software components*

Several studies into reuse have shown that.

40% to 60% of the code is reusable from one application to another.

60% of design and code are reusable in business applications.

75% of program functions are common to more than one program.

15% of the code found in most systems is unique and new to an application.

15% to 85% rates of actual and potential reuse range (Florinda Imeri, 2012).

Here the maximum number of reused components is the user-defined function program. It is very easy, and friendly to reuse user-delineate function and procedure from one software to another software or from one software module to another module within the same software. Such as current age calculation within date of birth. If a programmer creates a job that will return current age year month day passing the parameter date of birth then it can be used for employee age calculation or same function can be reused for patient age calculation or same function can be reused for customer age calculation. This is suitable for reuse in any module within software as well as in other software. This component will reduce rework and save development time. The next place, of the maximum reused stage is the design part. Here design means software architecture design, database design, user interface design, platform design, and security design, etc. The reusable design saves time and cost. Design phase encourages increasing the reusability.

c) *Reuse's Shortcomings*

Software reuse is hindered by issues. All-time reusable code is not a cure-all for Programmers and does not always provide significant benefits. Quite often maintaining old programs or developing shell scripts for reuse of old code is overlooked. A brief discussion of the important issues is as follows (Robert W.Therriault, 1994).

- Inadequate documentation/training/awareness
- Startup and maintenance costs
- Inflexible design/will cost too much to modify
- Legal problems
- Domain irrelevance
- Technical Difficulty
- Complexity
- Team members conflicts
- Difficult to identify reusable component's
- The technical factor that hinders software reuse is poor conceptualization.
- Additional costs associated with understanding, modifying, certifying, and maintaining the reusable components.

d) *Examples of successful software reuse*

There are many examples of successful software reuse. Several success stories were cited by Charles Lillie at the Second Annual Reuse Education and Training Workshop. These include:

- 312 projects in the aerospace industry, with averages of
  20% increase in productivity.
  20% reduction in customer complaints,
  25% reduced time to repair, and
  25% reduction in time to produce the system.
- A Japanese industry study that noted
  15-50% increases in productivity.
  20-35% reduction in customer complaints,
  20% reduction in training costs, and
  10-50% reduction in time to produce the system.
- A simulator system developed for the US Navy with an increase of nearly 200% in the number of source lines of code produced per hour.
- Tactical protocol software with a return on investment of 400%.

- A NASA report lists reductions of 75% in overall development effort and cost. These are impressive success stories. They clearly indicate that software reuse is a technique that can have a positive impact on software engineering practice in many environments (Ronald J. Leach, 2011).

The above example is the reflection of reuse. It is amazing. 20% to 25% Productivity increase is not a small deal. These examples will influence another company for increase their reusability. Increasing the reuse means reducing the rework that reduces cost and time. As a result profit margin will rise. That is the ultimate goal of this research.

III. Original Work vs. Rework

Software development works in a project typically include the development of new features, changes to existing features, and fixing reported feature defects. The journey from start to finish for these tasks may follow different paths described regarding time spent doing two types of work: original work and rework. Original work in this context is a metric which assesses how much initial time/effort was spent to develop/change/fix/verify a feature. Rework is a metric which assesses how much repeat time/effort was spent to complete a currently-open-and-active, or a previously-closed-and-reopened, feature/change/defect (Segue Technologies, 2014). The time of initial development work and the time of repeat work are clearly identified. Summation of both is the result of total work time.

Causes of rework:

Several avoidable and unavoidable reasons are responsible for rework. Some details can be minimized by seriously focusing on related works. Unavoidable causes that really impossible to ignore. Avoidable reason means those rework aims that can be easily controlled by stakeholders. Following are some spirited explanations.

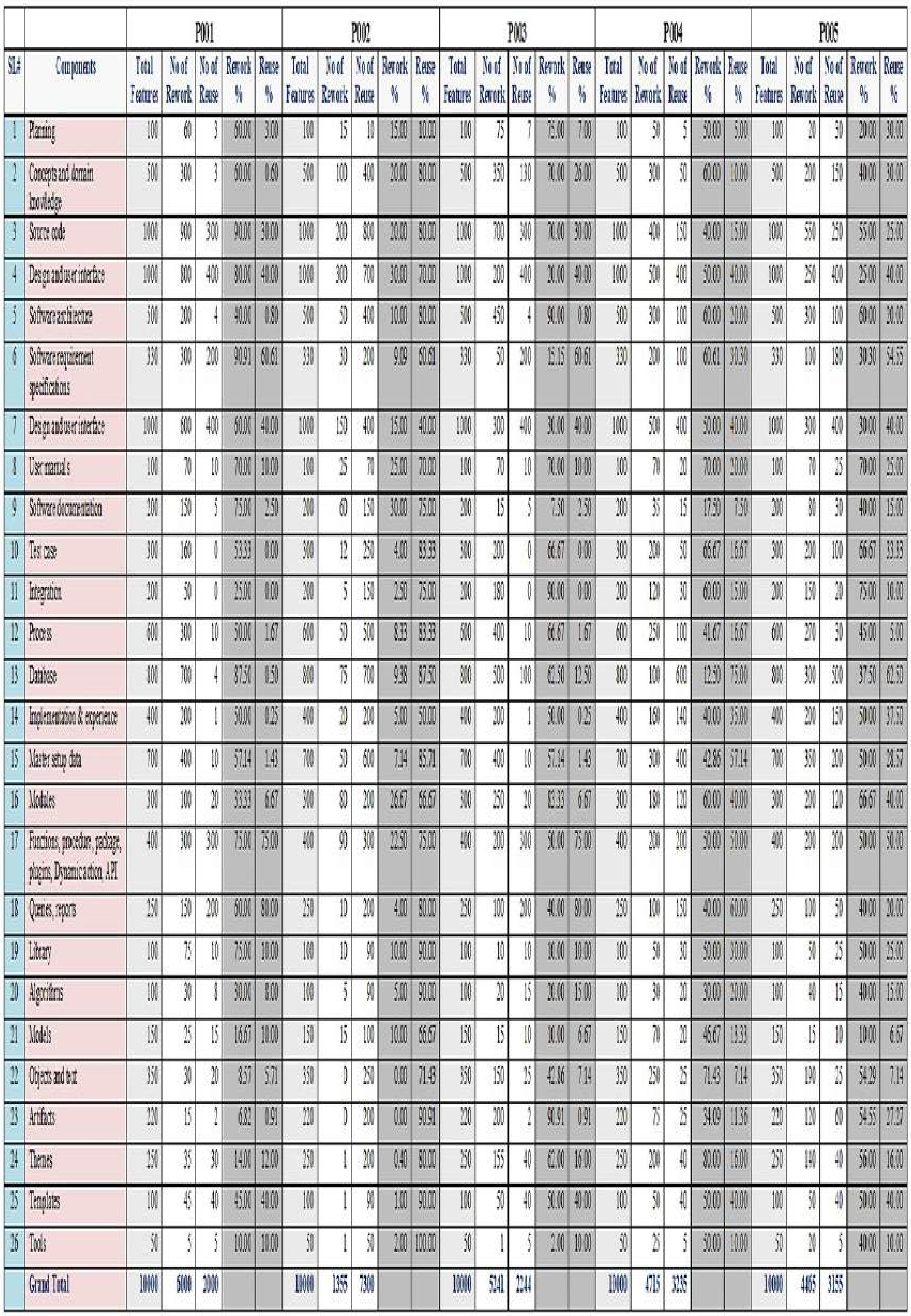
1. The reason for rework is infrequently the result of individuals not doing their jobs well.
2. Improper planning
3. Poor communication
4. Inadequate testing
5. Unstructured programming
6. Poor logic and algorithm
7. Lack of domain knowledge
8. Insufficient time
9. Low-cost budget
10. One reason rework becomes necessary is that the development, design and engineering teams lack visibility into software requirements, which often change throughout the development process.
11. Poor requirements management can have a significant effect throughout the process and on the business itself. In fact requirements, defects account for 70 to 85 percent of rework costs. This is a very high cost.
12. Software errors found late in the development process can cost organizations up to 200 times more than if they were detected during the requirements analysis phase.
13. Late detection of bugs. That happens when teams don't pay enough attention to quality early in the development lifecycle error detection should not be left until the testing stage. According to the Carnegie Mellon Software Engineering Institute, "Data indicate that 60-80% of the cost of software development is in rework (IBM, 2009)."
14. The cost of rework can approach or exceed 50% of total project cost.
15. Rework cost rises dramatically the longer the delay relative to the life cycle, between the occurrence of a problem and its remediation. For example, for a problem that occurs in requirements analysis, the cost of repair may rise by one to two orders of magnitude depending on how late in the life cycle it is caught (Aaron G. Cass, 2017).
16. One incident where the cost of software rework was actually calculated. A bank totaled the cost of automated re-testing at $1 million to $1.5 million per month. This number doesn't even include the cost of manual testing, design time, meetings, coding, or unit testing (David McAllister, 2017).

Rework cost may fluctuate from organization to organization. Above history of rework cost is asking us why we do allow rework, why we do not reduce rework yet.

Risk level of rework: The risk level of rework differs over risk to risk, project to project. High-level risk of rework can lead to project failure. Risk management techniques would allow the project risk management team to identify, classify & prioritize the risk level, risky modules or components. In reality, it is very difficult to point out the risk level of rework in a large system. Rework become obligatory for a specific issue until the problem resolve. If the matter repeats several times and continuously works for the same matter, it not only waste time & money but also Damage Company good well. If the rework risk is too large for a firm to be willing to accept, the firm can avoid the risk by changing project strategies and tactics to choose a less risky alternate or may decide not to do the project at all. For example, if a project has a tight schedule constraint and includes state of the art technology.

Rework and reuse data of above projects: Data for this research were collected from small, medium and enterprise software firms. There are maximum local software development firms of Bangladesh & a few international firms. This paper does not mention the firm's name, project & client's name in respect of the privacy policy. Software firms provided valuable data about their current and past projects. Many firms willingly told me that they had rework concerns but, they will try to overcome the disputes and they also agreed with me to develop reusable software components for their future projects.

Table 1: Rework and reuse data of above projects



		P001					P002					P003					P004					P005				
SL#	Components	Total Features	No of Rework	No of Reuse	Rework %	Reuse %	Total Features	No of Rework	No of Reuse	Rework %	Reuse %	Total Features	No of Rework	No of Reuse	Rework %	Reuse %	Total Features	No of Rework	No of Reuse	Rework %	Reuse %	Total Features	No of Rework	No of Reuse	Rework %	Reuse %
1	Planning	100	60	3	60.00	3.00	100	15	10	15.00	10.00	100	75	7	75.00	7.00	100	50	5	50.00	5.00	100	20	30	20.00	30.00
2	Concepts and domain knowledge	500	300	3	60.00	0.60	500	100	400	20.00	80.00	500	350	130	70.00	26.00	500	300	50	60.00	10.00	500	200	150	40.00	30.00
3	Source code	1000	900	300	90.00	30.00	1000	200	800	20.00	80.00	1000	700	300	70.00	30.00	1000	400	150	40.00	15.00	1000	550	250	55.00	25.00
4	Design and user interface	1000	800	400	80.00	40.00	1000	300	700	30.00	70.00	1000	200	400	20.00	40.00	1000	500	400	50.00	40.00	1000	250	400	25.00	40.00
5	Software architecture	500	200	4	40.00	0.80	500	50	400	10.00	80.00	500	450	4	90.00	0.80	500	300	100	60.00	20.00	500	300	100	60.00	20.00
6	Software requirement specifications	330	300	200	90.91	60.61	330	30	200	9.09	60.61	330	50	200	15.15	60.61	330	200	100	60.61	30.30	330	100	180	30.30	54.55
7	Design and user interface	1000	600	400	60.00	40.00	1000	150	400	15.00	40.00	1000	300	400	30.00	40.00	1000	500	400	50.00	40.00	1000	300	400	30.00	40.00
8	User manuals	100	70	10	70.00	10.00	100	25	70	25.00	70.00	100	70	10	70.00	10.00	100	70	20	70.00	20.00	100	70	25	70.00	25.00
9	Software documentation	200	150	5	75.00	2.50	200	60	150	30.00	75.00	200	15	5	7.50	2.50	200	35	15	17.50	7.50	200	80	30	40.00	15.00
10	Test case	300	160	0	53.33	0.00	300	12	250	4.00	83.33	300	200	0	66.67	0.00	300	200	50	66.67	16.67	300	200	100	66.67	33.33
11	Integration	200	50	0	25.00	0.00	200	5	150	2.50	75.00	200	180	0	90.00	0.00	200	120	30	60.00	15.00	200	150	20	75.00	10.00
12	Process	600	300	10	50.00	1.67	600	50	500	8.33	83.33	600	400	10	66.67	1.67	600	250	100	41.67	16.67	600	270	30	45.00	5.00
13	Database	800	700	4	87.50	0.50	800	75	700	9.38	87.50	800	500	100	62.50	12.50	800	100	600	12.50	75.00	800	300	500	37.50	62.50
14	Implementation & experience	400	200	1	50.00	0.25	400	20	200	5.00	50.00	400	200	1	50.00	0.25	400	160	140	40.00	35.00	400	200	150	50.00	37.50
15	Master setup data	700	400	10	57.14	1.43	700	50	600	7.14	85.71	700	400	10	57.14	1.43	700	300	400	42.86	57.14	700	350	200	50.00	28.57
16	Modules	300	100	20	33.33	6.67	300	80	200	26.67	66.67	300	250	20	83.33	6.67	300	180	120	60.00	40.00	300	200	120	66.67	40.00
17	Functions, procedure, package, plugins, Dynamicaction, API	400	300	300	75.00	75.00	400	90	300	22.50	75.00	400	200	300	50.00	75.00	400	200	200	50.00	50.00	400	200	200	50.00	50.00
18	Queries, reports	250	150	200	60.00	80.00	250	10	200	4.00	80.00	250	100	200	40.00	80.00	250	100	150	40.00	60.00	250	100	50	40.00	20.00
19	Library	100	75	10	75.00	10.00	100	10	90	10.00	90.00	100	10	10	10.00	10.00	100	50	30	50.00	30.00	100	50	25	50.00	25.00
20	Algorithms	100	30	8	30.00	8.00	100	5	90	5.00	90.00	100	20	15	20.00	15.00	100	30	20	30.00	20.00	100	40	15	40.00	15.00
21	Models	150	25	15	16.67	10.00	150	15	100	10.00	66.67	150	15	10	10.00	6.67	150	70	20	46.67	13.33	150	15	10	10.00	6.67
22	Objects and text	350	30	20	8.57	5.71	350	0	250	0.00	71.43	350	150	25	42.86	7.14	350	250	25	71.43	7.14	350	190	25	54.29	7.14
23	Artifacts	220	15	2	6.82	0.91	220	0	200	0.00	90.91	220	200	2	90.91	0.91	220	75	25	34.09	11.36	220	120	60	54.55	27.27
24	Themes	250	35	30	14.00	12.00	250	1	200	0.40	80.00	250	155	40	62.00	16.00	250	200	40	80.00	16.00	250	140	40	56.00	16.00
25	Templates	100	45	40	45.00	40.00	100	1	90	1.00	90.00	100	50	40	50.00	40.00	100	50	40	50.00	40.00	100	50	40	50.00	40.00
26	Tools	50	5	5	10.00	10.00	50	1	50	2.00	100.00	50	1	5	2.00	10.00	50	25	5	50.00	10.00	50	20	5	40.00	10.00
	Grand Total	10000	6000	2000			10000	1355	7300			10000	5241	2244			10000	4715	3235			10000	4465	3155		

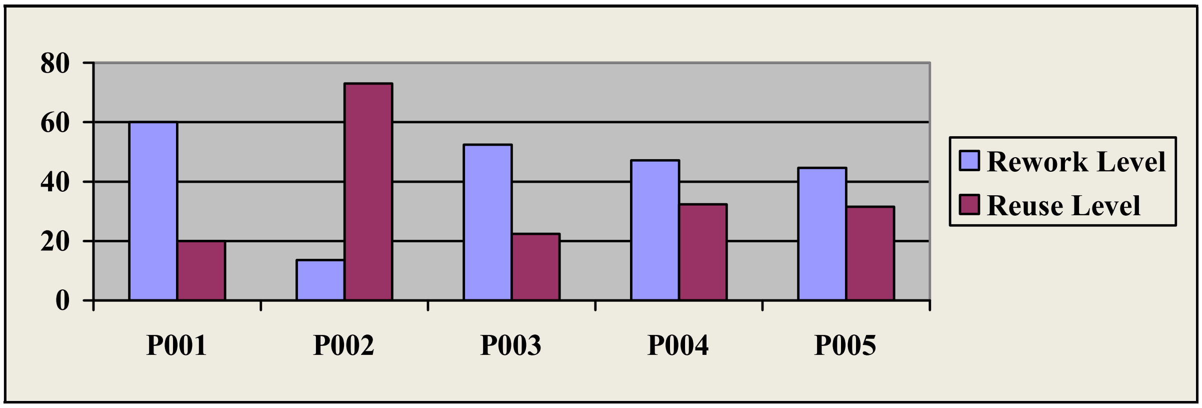


Figure 4: Rework and reuse summary of projects

Current rework & reuse level: Activity-based cost analysis helps to identify the current rework and reuse level. To figure out the percentage of present rework and reuse position, we used following formula for that calculation.

Rework Level = (Number of functionalities that needs rework (defect found) / Total number of components in an application) X 100

$$\text{Rework Level} = \frac{\text{Number of functionalities that needs rework (defect found)}}{\text{Total number of components in an application}} \times 100$$

$$\text{Reuse Level} = \frac{\text{Number of functionalities that reused from projects assets}}{\text{Total number of components in an application}} \times 100$$

Current rework and reuse level of projects:

Table 5: Current rework and reuse level of projects

Project Code	Average Rework	Average Reuse	Current Rework Level	Current Reuse Level
P001	0.6	0.2	60%	20 %
P002	0.1355	0.73	13.55 %	73 %
P003	0.5241	0.2244	52.41 %	22.44 %
P004	0.4715	0.3235	47.15 %	32.35 %
P005	0.4465	0.3155	44. 65 %	31.55 %

Cost analysis for rework of the project P001:

Our enlisted P001 project will come in at a 60 percent rework rate:

Rework Cost = 0.60 * Project Dev Cost

Managing Software Requirements indicate that about half of all software defects are due to missing or bad requirements, but the cost of finding and fixing requirement defects is higher than that for other kinds of defects. In fact, they indicate that 70 to 85 percent of the rework cost is due to requirements defects. For our example, we could use 75 percent (or use own measurement):

Rework Cost of Requirements Defects = 0.75 * Rework Cost

Requirements defects range is higher. If the defect detects at an early stage then, the cost is lower but at production stage cost may be 100 or 1000 times higher. Following are the ratio of bug fixing in four stage.

- To fix a problem at the requirements stage costs is 1.
- To fix a problem at the development stage costs is 10.
- To fix a problem at the testing stage costs is 100
- To fix a problem at the production stage costs is 1000(Shahadat, 2018).

We estimate that the combined total objects of screens plus reports will be 300. Multiply by four person weeks to get twelve hundred person-weeks as development effort. We now expect to have 1200 hundred defects at system test or user acceptance test. Estimate that half of those, fifty, will be due to requirements problems.

Now estimate rework cost at 60% of the initial development cost. Average wages paid $1,000 per person-week ($48,000 per year salary).

So the initial development cost will be 1200 PW * $1000 = $12,00,000.

Rework cost will add 60 percent to that:

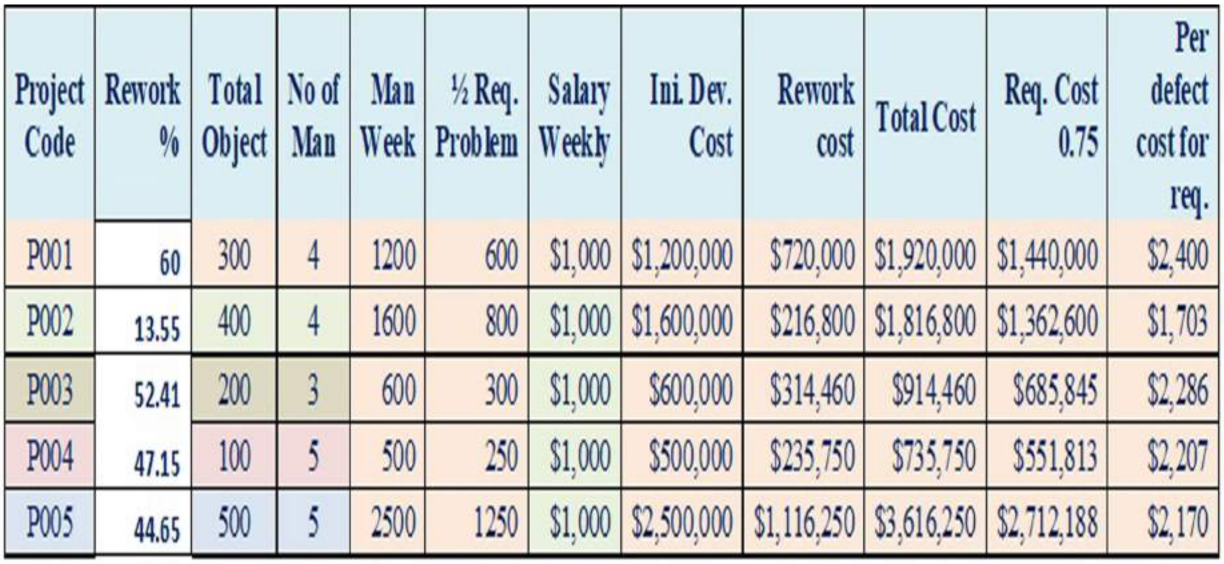
Rework costs = 0.60 * $1200, 000= $720,000

Software development cost = initial development cost + rework cost
Total development cost = $12,00,000 + $7,20,000
Total development cost = $19,20,000
Next, you estimate the amount of rework that will be due to the requirements errors: 0.75 * $19,20,000= $14,40,000

Finally, you divide this rework cost by the number of requirements defects to determine the cost per requirement defect: $2,400 = $14,40,000/600 (1200/2 requirements defects).

Here one project P001 cost analysis has been described. Note that this is not total project cost or even your total development cost, it does not include project Management, QA time, analyst time, and so on. It only covers pure development effort and rework cost of a week. Cost analysis of the rest of the project is the same. Following table showing the calculation of cost of rework for above five projects.

Table -6 shows how the cost of rework is changing. Project P001 reworks percentage increased to 65%, simultaneously rewrite cost enlarged to $7,80,000 & total development cost increased to $1,980,000. Alternatively, if the rework fall to 55%, then rework cost fall to $6, 60,000, and total development cost falls to $1,860,000. This is just one variable effect on software economy. To complete this task quickly, if include another person then time will reduce to 5.6 days, but the development cost will increase to $3,200,000. Again, if remove one person to reduce the cost, then some cost will reduce, and the development cost will be $1,920,000, but time will increase to 9.33 days. Here total objects, number of man & salary are variable. Cost of development is varying with rising and falling off any of this variable values. Software companies want to reduce the rework.



Project Code	Rework %	Total Object	No of Man	Man Week	½ Req. Problem	Salary Weekly	Ini. Dev. Cost	Rework cost	Total Cost	Req. Cost 0.75	Per defect cost for req.
P001	60	300	4	1200	600	$1,000	$1,200,000	$720,000	$1,920,000	$1,440,000	$2,400
P002	13.55	400	4	1600	800	$1,000	$1,600,000	$216,800	$1,816,800	$1,362,600	$1,703
P003	52.41	200	3	600	300	$1,000	$600,000	$314,460	$914,460	$685,845	$2,286
P004	47.15	100	5	500	250	$1,000	$500,000	$235,750	$735,750	$551,813	$2,207
P005	44.65	500	5	2500	1250	$1,000	$2,500,000	$1,116,250	$3,616,250	$2,712,188	$2,170

Figure 6: Cost of rework

Now if the project P001 comes up with 20% reuse, then it's time will save 20% and the cost will save $240,000, and total development cost will be $960,000. The massive amount of cost is kept for reuse. I have successful records of cost & time saving by reusing one project objects to multiple projects and made a handsome profit. Here if comparison table 6 to table 7, the variation of development cost will be realized. Figure-7 shows the effect of reuse in the software economy. Here the percentage of reuse, total screen, number of men, salary are variables. Changes in this variable's value might change the cost of development.

For example, if P001 reuse, increase to 25% than its rate, reduce from $240,000 to $300,000 save for reuse is $60,000. Dramatically cost is falling by the rising of reuse. So Software companies must try to increase their reuse level.

Project Code	Reuse %	Total Screen	No of Man	Man Week	Salary Weekly	Initial Development Cost	Cost Save for Reuse	Total Cost
P001	20	300	4	1200	$1,000	$1,200,000	$240,000	$960,000
P002	73	400	4	1600	$1,000	$1,600,000	$1,168,000	$432,000
P003	22.44	200	3	600	$1,000	$600,000	$134,640	$465,360
P004	32.35	100	5	500	$1,000	$500,000	$161,750	$338,250
P005	31.55	500	5	2500	$1,000	$2,500,000	$788,750	$1,711,250

Figure 7: Cost save for reuse

Rework and reuse effects in software economy are opposite of each other's. If rework rises, then development cost rise, if rework fall, then development cost falls. Oppositely if reuse rises, then development cost fall, if reuse fall development cost rise. Both rework and reuse directly hit to cost. Rework rise cost rise, reuse rise cost fall. Target to upswing reuse & decrease rework. Bellow figure -8 shows that initial development cost exceed due to rework cost.

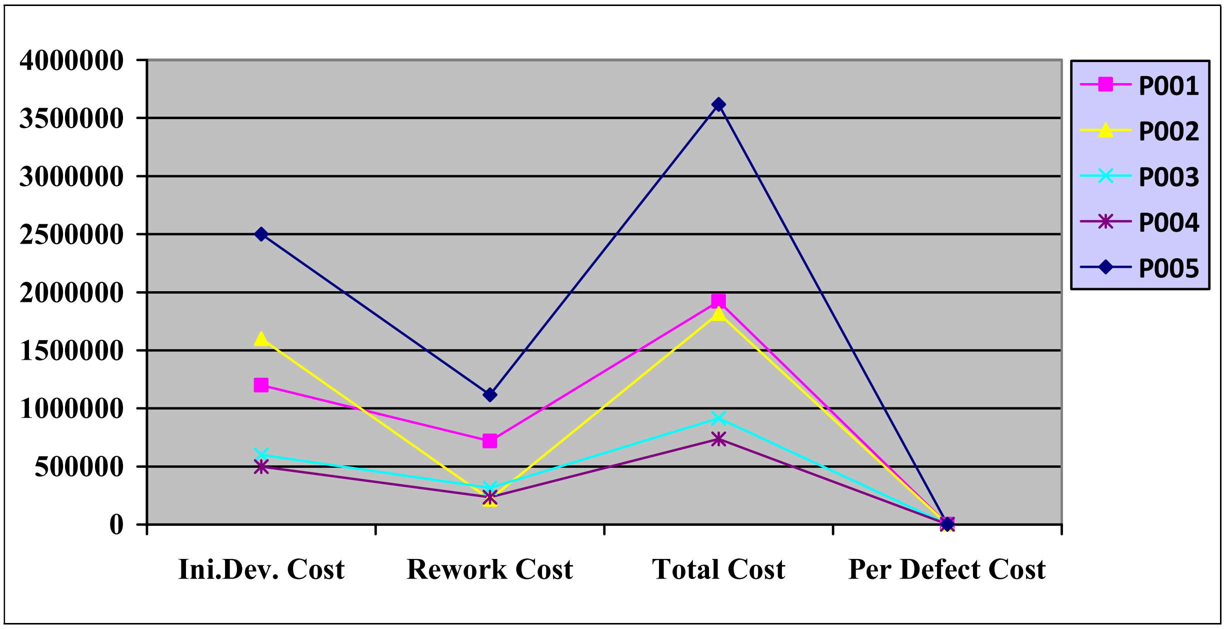


Figure 8: Cost of rework

In figure-9 we see total development cost exceed initial development cost for rework Cost. Total development cost becomes lower than the initial development cost for reuse. Development cost is falling and rising magically with increasing and decreasing of Rework and reuse charge. This figure shows that total development cost is bellow of initial development cost for reuse. Those projects have more reused its total Development cost is lower than the initial budget. Those projects have high rework its total development cost is higher than the original development charge. All project total development cost become higher than the opening development cost due to rework cost.

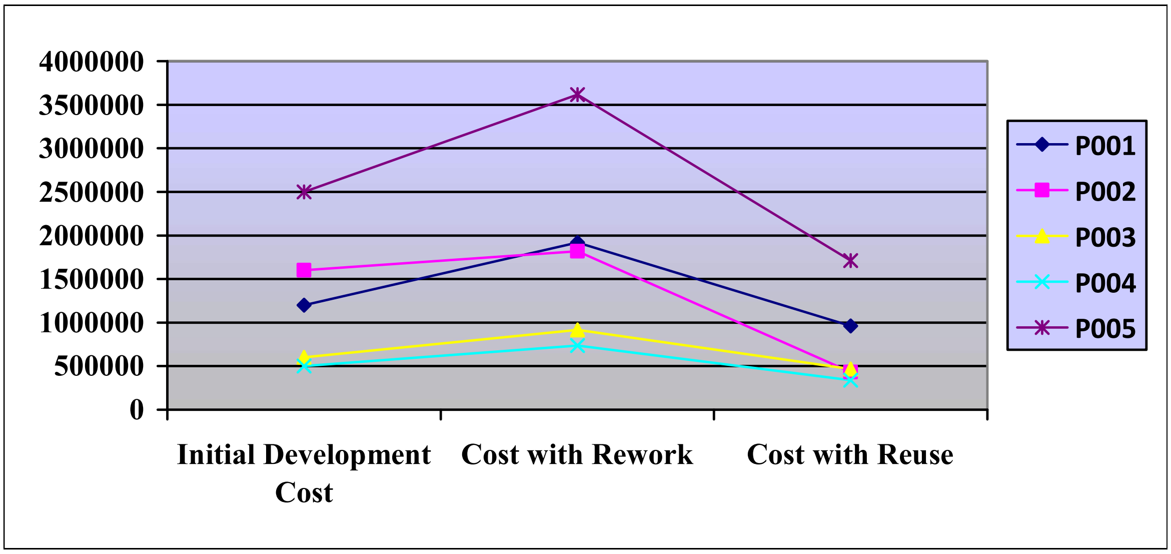
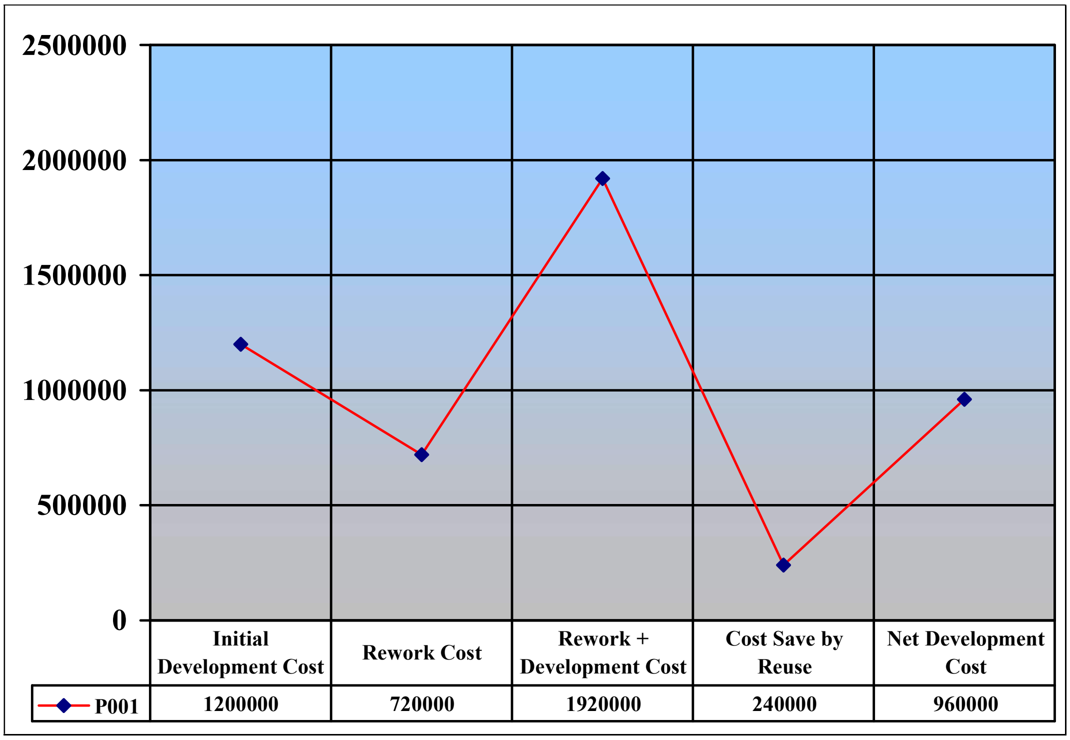


Figure 9: Rework cost and Reuse cost effect

Variation of cost: A project having 60% rework rate and 20% reuse rate frequently up and down of its development cost. Although the project development cost rise to the top than the initial development costs due to rework cost but it fluctuates with reuse effects. When the costs of development arises in this situation reuse is the best solution to reduce the cost. Figure-10 is explaining the combination effect of rework and reuse of a particular Project. Reuse is very effective, to reducing the development cost instantly, to the below than the initial development cost. Reuse avoids the vast amount of cost.



	Initial Development Cost	Rework Cost	Rework + Development Cost	Cost Save by Reuse	Net Development Cost
P001	1200000	720000	1920000	240000	960000

Figure 10: Up and down of a project cost for rework and reuse effect

How much cost saves from reuse of software: It is a vigorous question of the Software industry. The software company expects to save some cost by reuse. Because they are reusing some portion of a system and not developing the particular component from scratch. The amount of cost saving depends on the number of reusable components they used. There are several reasons for this discrepancy. The amount saved depends upon many factors. The most important factors are the following (Ronald J. Leach, 2011):

- The life cycle model used in the software development process.
- The development history of the software system of which the artifact is a substantial portion.
- The cost of beginning a policy of software reuse.
- The cost of creating and maintaining a reuse library of software artifacts.
- The percentage of the system that is made using existing software artifacts.
- The ratio of change in each software artifact that is being reused.
- Different levels of an organization have different goals for the reuse programs.

Figure-11 shows a project cost saving scenario from the reuse of previous software components, the extra cost paid for the rework and the difference between the original development budget & the net price of the development. How much cost will save from the reuse, will define by the management policy and planning.

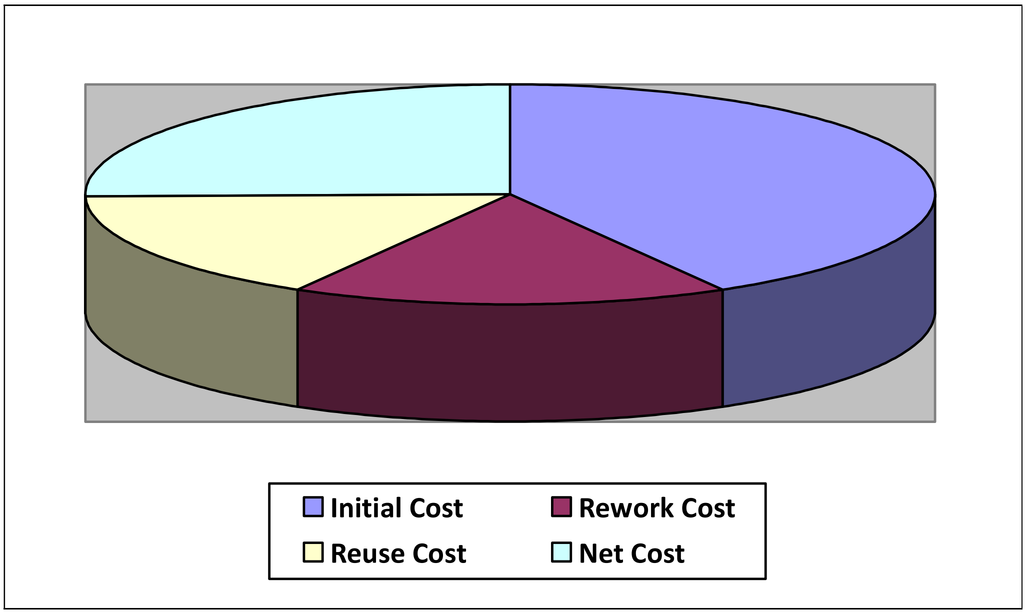


Figure 11: Cost saving from reuse of software

Time & cost effect: Time and cost is correlated with each other. Both are leading parameters of the project. Following example is more than enough to understand the time, and the cost effect on a project. An 8.7 kilometres Moghbazar-Mouchak flyover project was taken up in January 2011 and was supposed to be completed by December 2015. But, the authorities concerned went for a one-and-a-half-year time extension until mid-2017 with the construction still in progress. The construction cost of the flyover has increased to almost 72 percent as the construction agency was unable to complete the work in time and extended the project tenure several times. The authorities extended the deadline for completion three times, responding to the request from the builder. The construction costs has jumped to Taka 1327.4 crore from the original estimation of Taka 772 crore (Rick Haque, 2017).

Investment for reuse: Reuse is the robust components of the development process. First and foremost, we must recognize that reuse has the same cost and risk Characteristics as any financial investment (BH Barens, 1991). To get the benefit from software reuse, it is expected that the company should invest in the development of reusable software components. ROI of reusability depends on the efficient investment of reusable components. There are additional costs associated with understanding, Modifying, certifying, and maintaining it (Ronald J. Leach, 2011). It is clear that price is involved in the development of reusable software components as well as cost is involved in the uses of existing reusable software components. Barry Boehm explained the cost calculation as followed (Barry Boehm, 1997):

RCWR: Relative Cost of Writing for Reuse

$$\text{RCWR} = \frac{\text{The cost of developing reusable asset}}{\text{The cost of developing single-use asset}}$$

Investment for the development of reusable software component is calculated by the above formula.

RCR: Relative Cost of Reuse

$$\text{RCR} = \frac{\text{The Cost to reuse asset}}{\text{The cost to develop the asset from scratch}}$$

Investment of reuse of already developed existing reusable software component calculated by the above formula.

ROI of reuse: Return on investment for reuse is a widely used measure to compare the effectiveness of reusable components development investments. It is commonly used to justify software projects. The plain ROI calculation is to divide the net return from an investment by the cost of the investment and express this as a percentage. ROI, while a simple and extremely popular metric, it may be easily modified for different situations.

The ROI formula is:

$$\textbf{ROI}\ \% = \frac{\text{Return from reuse - Cost of Investment}}{\text{Cost of Investment}}\ \text{X 100}$$

There is another term called ROR (rate of return), Rate of profit. The return is also known as money gained or lost on investment, profit or loss, gain or loss, net income or loss. The cost of investment is also known as investment, capital, principal, costs. A project is more likely to proceed if its ROI is higher because higher the better. For example, a 200% ROI over five years indicates a return of double the project Investment, over a five year period. Financially, it makes sense to choose projects with the highest ROI first, then those with lower ROI's. While there are exceptions, if a project has a negative ROI, it is questionable if it should be authorized to proceed.

Let's say a project P001 developed a reusable component in six month durations. P001 invested $6,000 for this reusable software component development. Later this reusable component was used in five projects. This reuse saved $24,000 development cost that is the return from reuse for the first year. So the ROI is

ROI = ($24,000 - $6,000)/$6,000 * 100

ROI = 300% for the first year

2nd years to 5th years the project will get return $30,000 per year if there is no exceptional investment require for this reusable component. The return will come four Times i.e.400% higher than investment continuously for the next four years. The ROI for the next four years will be ROI = 400%*4 =1,600%. A Net ROI of five years projects is 1,600% + 300%=1,900%. It is a successful investment for the development of reusable components and the successful reuse benefit. Here we see the return of reusable components is four times higher than the investment. The return increases 300% to 1900% within five years of project duration. That makes sense to decide for investment for reusable software components.

Reuse effect on software product line: It is essential to define the product line for produce new product by reusing an existing software product. Software product line architecture recycles standing product for productivity, quality, and profitability. Software product line practice carefully elicits, specify, analyze, and manage software requirements. This approach based on the systematic creation and reuse of existing assets in support of new product development (Emilio Insfran, 2014).

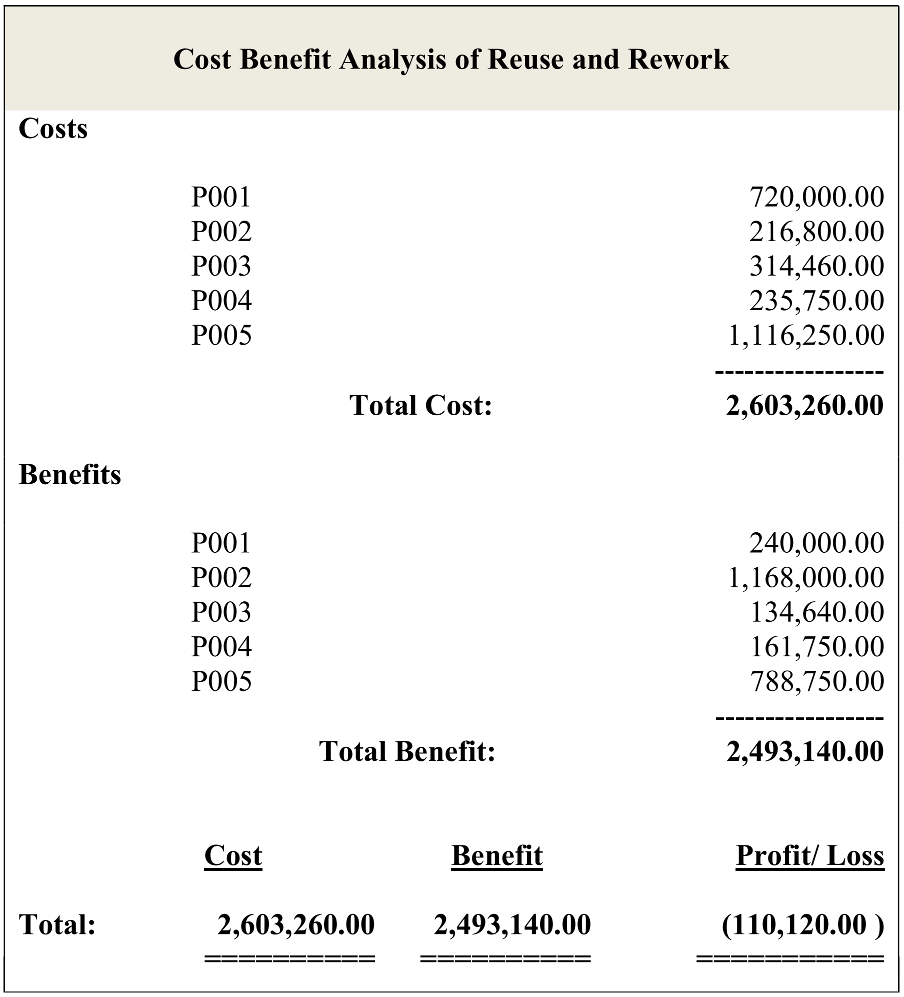
The Apple iOS is the best example of software reuse. When the original iPhone launched, the OS was called "iPhone OS", and it kept that name for four years, only changing to iOS with the release of iOS 4 in June of 2010. iOS is the name of the operating system that runs the iPhone, iPod touch, and iPad. It's the core software that comes loaded on all devices to allow them to run and support other apps. One year after the iPhone became a bigger hit than almost anyone projected, Apple released iOS 2.2.1. It was released on January 27, 2009 (then called iPhone OS 2.0) to coincide with the release of the iPhone 3G. The 1st generation iPad was released in June 17, 2009, and version 3.2 of the software came with it. It added features including copy and paste, Spotlight search, MMS support in the Messages app, and the ability to record videos using the Camera app. Many aspects of the modern iOS began to take shape in iOS 4. Features that are now widely used debuted in various updates to this version, including FaceTime, multitasking, iBook's, organizing apps into folders, Personal Hotspot, AirPlay, and AirPrint. Another important change introduced with iOS 4 was the name "iOS" itself. iOS 4 It was released on July 25, 2011. iOS 5 was released on May 7, 2012 with wirelessness, and cloud computing features. A Controversy was one of the dominant themes of iOS 6 was released on Feb. 21, 2014. Like iOS 6, iOS 7 was met with substantial resistance upon its release on June 30, 2014. Unlike iOS 6, though, the cause of unhappiness among iOS 7 users wasn't that things didn't work. Rather, it was because things had changed. After the firing of Scott Forstall, iOS development was overseen by Jony Ive, Apple's head of design, who had previously only worked on hardware. In this version of the iOS, Ive ushered in a major overhaul of the user interface, designed to make it more modern. More consistent and stable operation returned to the iOS in version 8.0 was released on August. 13, 2015. iOS 9 was released on August. 25, 2016 with major improvements were delivered in speed and responsiveness, stability, and performance on older devices. iOS 10 was released on July 19, 2017. iOS 11 was released on May 29, 2018, contains lots of improvements for the iPhone, but its major focus is turning the iPad Pro series models into legitimate laptop replacements for some users.

Apple was continuously updating iOS and releasing one after one product by reusing the previous iOS. Apple becomes world's first trillion-dollar public company as on Thursday 2nd August, 2018. Apple Is Worth $1,000,000,000,000. Two decades ago, it was almost Bankrupt.

Microsoft is an unusual company for the sheer number of product lines. Microsoft has revolutionary reuse records since its start on November 10, 1983, to till now. Microsoft's bread and butter are Windows, the OS is doing quite well. Microsoft revealed that it had sold 400 million copies of its latest version, Windows 7. Microsoft's big sales pitch with Windows 10 is that it's one platform, with one consistent experience and one app store to get software from. There are seven different versions of Windows 10. Anyone who knows anything about Microsoft is aware of how essential its Office franchise is to the company. In every product, Microsoft reused the existing product, added new features and released a new product in the same product line.

Life cycle affected by rework & reuse: Software development life cycle different model are affected by rework & reuse. Such as waterfall model, spiral model, rapid prototyping model, agile model, etc. The cost saving from reuse can be started earlier in the life cycle model and can be realized at any phase (design phase, coding phase, test phase, etc.) of life cycle subsequent to the point at which the system is reused. When programmers took any component from the reusable library and used it as is without any change, then the element need not be tested because it was tested as a module earlier. Programmers only need to perform unit testing and integration testing in which the reusable component is engaged. No additional test cases, test plans or documentation need to write for this reusable component. At any segment of the life cycle if a bug is generated, or a scope for rework is produced, and it inherits to next part then its cost become several times higher than the earlier chapter. Early detection and prevention are cost-effective. If a scope of rework is formed in requirement specification phase, but it realized later in testing stage than the rework cost become higher than the requirement specification phase. The cost of rework varies from one phase to another phase of the software development life cycle.

Cost-benefit analysis of software reuse & rework: Rework is cost heads. Although rework has no financial benefit, but this research found a potential benefit of rework. Reworks help to find out the undiscovered bug, logical and exceptional issues. At some point of view, rework is re-check, re-testing of an existing system during next work. If an effort is done twice the result of the second labor is better than the result of first work. During the rework, some additional modification and the necessity to include new features may grow. The existing issues are cleaning by the Rework. Rework has some positive benefit. Rework cost is high, the cost vary from project to project. Reuse is revenue heads that save development time investment cost and improve quality. Figure- 12 is showing total cost and benefit of all project. Here particular cost heads didn't mention. Project to project the values of profit and loss may be varying. Some project may have a standard amount of turnover for remarkable reuse. Alternatively, project must count loss for oversize rework. Software reuse does not come free. We anticipate that developing reusable software on AAS will cost twice as much as developing nonreusable software. This alone could have deterred the AAS management from implementing a reuse program (Johan Margono, 1992).



Cost Benefit Analysis of Reuse and Rework

Costs

	P001	720,000.00
	P002	216,800.00
	P003	314,460.00
	P004	235,750.00
	P005	1,116,250.00

	Total Cost:	**2,603,260.00**

Benefits

	P001	240,000.00
	P002	1,168,000.00
	P003	134,640.00
	P004	161,750.00
	P005	788,750.00

	Total Benefit:	**2,493,140.00**

	Cost	**Benefit**	**Profit/ Loss**
Total:	**2,603,260.00**	**2,493,140.00**	**(110,120.00)**
	==========	==========	===========

Figure 12: Cost Benefit Analysis of reuse and rework

Anomaly Metrics Model for Software Rework Reduction: Majority of the reported anomalies belong to this category of real faults in the software or documentation delivered together with the software. Reproducible anomaly is an observed failure during testing that cannot be reproduced by the developer that is assigned to fix it. Getting many such failures might be due to the existence of many intermittent faults in the product. This indicates a robustness problem that probably requires improvements to the product architecture. Insufficient debugging environments are other common reasons for not being able to reproduce the failures. Anomalies occur when the requirements documentation is vague or incomplete. For example, when a test engineer, and a developer interpret a requirement differently, the tester is likely to submit an abnormality report. In these cases, the inconsistency report is defined as an opinion for function report which might also result in a correction. When an organization reports many anomalies of this type, it indicates that the requirements are not pure enough. (Lars-Ola Damm, 2008). According to Ola Damm above statement, it is important to cure all types of anomalies of every stage of SDLC whenever the anomalies introduce. It will reduce the rework otherwise in later stage rework will be increased and the rate of rework will also increase. The best way of anomalies & bug's rework reduction is killed it before born.

Prevention is better than a cure: This popular saying most definitely holds true when it comes to bugs or security issues identified within the SDLC. During the development process, it is more cost-effective and efficient to fix bugs in the early stages rather than later ones. The fee increases exponentially as the software moves forward in the SDLC. This research focuses on prevention the cure because prevention reduces rework that save time & cost of software development.

IV. Results

Bug fixing and rework are not the same things, but both are cost heads. Rework cost exceeds the project budget. Reuses saves both time and cost. This paper is influencing to reduce rework and increase reuse of software components to ensure the successive economic growth of Software Company. The prime goal of this project is to develop process assets that will be used to reduce rework & increase reuse levels of the software company. Here I didn't find any benefit for which Software Company can neglect to develop reusable components. This research found that rework is harmful to the software economy. This research suggesting for early detection and prevention of bugs which is more cost-effective than testing & implementation phase.

V. Conclusion

This research found that economic growth of software companies falling for rework. All sizes of firms have more or less the rework problem. It is now one of the enormous challenges of the software industry. Rework is the barrier to the continuous achievement of financial improvement. This research focused on reuse to reduce the financial losses. The R2 is influential elements that move the economy. This paper is significant for modern software companies for high quality software development, as industries of all types utilize software applications to varying degrees. Unfortunately for startups, small businesses, and even multimillion dollar companies, tightening costs and rising competition mean a desperate scramble to find areas in which to slash expenses. Reduce your software development costs without sacrificing the quality of your product by following this paper cost saving strategy of reducing rework and increasing reuse.

Acknowledgements

I was inspired for this research by my professor of Independent University Bangladesh. I am grateful to my department for their cooperation. I would like to express my thanks to all software firms for their help. My wife Mrs Hosneara Shahadat and my Mother encourage me cooperated with me to perform the research.

References Références Referencias

1. Aaron G. Cass, Formalizing Rework in Software Processes (2003). Department of Computer Science, University of Massachusetts.
2. Vimla Devi Ramdoo, Strategies to Reduce Rework in Software Development on an Organization in Mauritius (2015). Department of Computer Science and Engineering, University of Mauritius.
3. Robin McDonald, CCM, LEED G.A., Root Causes & Consequential Cost of Rework (2013). Insurance North America Construction.
4. Robert W.Therriault Industry Versus DOD: A Comparative Study of Software Reuse(1994).
5. Segue Technologies Use Test Track Metrics to Measure and Manage Software Project Rework (2014).
6. Ronald J. Leach Software Reuse Methods, Models, and Costs (2011).
7. Florinda Imeri an Analytical View on the Software Reuse (2012).
8. IBM Reducing rework through effective requirements management (2009).
9. Aaron G. Cass Formalizing Rework in Software Processes (2017).
10. David McAllister Software Waste and the Cost of Rework (2017).
11. Shahadat Challenges of software quality assurance and testing (2018). Department of Software Engineering, School of Engineering & Computer Science, Independent University Bangladesh.
12. BH Barnes Making reuse cost effective (1991).
13. Johan Margono Software reuse economics: cost-benefit analysis on a large-scale ADA project (1992).
14. Lars-Ola Damm A Model for Software Rework Reduction through a Combination of Anomaly Metrics (2008).
15. Barry Boehm Software Reuse Economics (1997).
16. John Managing Software Deliverables (2004): A Software Development Management Methodology Managing John W. Rittinghouse. ISBN: 1-55558-313-X.
17. Jones A short history of the cost per defect metric (2012). 2-2. Capers Jones, President, Capers Jones & Associates LLC.
18. Rex, Investing in software testing: the cost of software quality (2000). Rex Black is President and Principal Consultant of RBCS, Inc. R. Black, Managing the Testing Process, Second Edition. Wiley, New York, 2002.
19. Ricardo, Testability of dependency injection (2007) University of Amsterdam Faculty of science, master research software engineering.
20. Ross, The secrets to high customer satisfaction (2013).
21. Sommerville, Requirements engineering challenges, Ian Sommerville (2013).
22. Linda Westfall Software risk management (2001).
23. Rick Haque Builder gets time extension, cost shoots up 72pc (2017).
24. Emilio Insfran Requirements engineering in software product line engineering (2014).