# STRATEGIES TO REDUCE REWORK IN SOFTWARE DEVELOPMENT ON AN ORGANISATION IN MAURITIUS

Vimla Devi Ramdoo[1] and Geshwaree Huzooree[2]

[1]Department of Computer Science and Engineering, University of Mauritius
[2]Department of IT, Charles Telfair Institute, Mauritius

***ABSTRACT***

*Rework is a known vicious circle in software development since it plays a central role in the generation of delays, extra costs and diverse risks introduced after software delivery. It eventually triggers a negative impact on the quality of the software developed. In order to cater the rework issue, this paper goes in depth with the notion of rework in software development as it occurs in practice by analysing a development process on an organisation in Mauritius where rework is a major issue. Meticulous strategies to reduce rework are then analysed and discussed. The paper ultimately leads to the recommendation of the best strategy that is software configuration management to reduce the rework problem in software development.*

***KEYWORDS***

*Rework, Software development, Software Quality, Software Configuration Management*

## 1. INTRODUCTION

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 [2]. It usually results from errors, omissions, failures, changes, poor communications and poor coordination. Organisations invest in time, money and effort in order to continuously improve software quality in the evolving business environment [1]. Rework directly impacts the performance and productivity and ultimately the profit margins of the firm. Therefore, it is crucial to identify and eliminate rework that could have been avoided. One famous quote of Total Quality Management (TQM) is "*Do* the *right* things, *right the first time*, every time" implying the complete elimination of rework in the context of software development. However, literature also shows that there is some part of rework that is inevitable when dealing with software engineering.

## 2. RELATED WORK

Boehm's (1987) research shows that rework costs are about 40% to 50% of all software development expenditures and stated that the cost of rework could be reduced by up to 30-50% by finding more faults earlier. According to Charette, 'Studies have shown that software specialists spend about 40 to 50 percent of their time on avoidable rework rather than on what they call value-added work, which is basically work that's done right the first time. Once a piece of software makes it into the field, the cost of fixing an error can be 100 times as high as it would have been during the development stage' (Charette 2005). Measuring and reducing the percentage of avoidable rework should be one objective of most process improvement initiatives.

DOI : 10.5121/ijsea.2015.6502 

Though understood as a major software development activity, rework is often poorly defined and understood. Rework is a common problem in software engineering and an ongoing research area [4]. The importance of studying rework is highlighted in modelling projects where the rework cycle is at the heart of dynamic models. The principle is as follows: when development is done, it is not necessarily correct as it may contain errors that are undetected. The tasks go through testing where the errors are discovered, and eventually lead to rework, hence additional work to be done. This is because when rework increases, both elapsed time and project effort increases [16]. The rework cycle is actually one of the major research and application areas, especially in dynamic modelling. [7].

The cost associated with rework remains one of the main concern in software development since cost is an important parameter defining the success of software projects [5, 20]. It is essentially the monetary cost associated to fix a work that has already been completed. Rework is a prevailing scourge as it consumes up to 40 to 70 % of a project's budget. Project management problems such as communication and work conditions introduce rework in software projects and research shows that rework could be identified and avoided at an early stage. But in this era of software development, not much consideration has been given to studying rework since it is challenging and complex [6]. Research on rework has focussed on minimizing the amount of rework that a software project may acquire, through formal reviews, inspections and tests with the aim to detect and enable the correction of problems as early in the software life cycle as possible. Some researchers mention modelling as a technique to prevent costly rework through prediction and good programming practice [17] while others made use of metrics in order to understand and reduce software rework [18, 19]. Nevertheless, it is generally accepted that rework cannot be eliminated entirely, some are inevitable. However addressing this issue is of utmost importance to keep rework at a minimal level [8,9,10].

## 3. METHODOLOGY

After reviewing relevant literature, a study was carried out in a software organisation in Mauritius in order to determine the root causes of rework in their software development process. Figure 1 depicts the results of the survey through the traditional *Ishikawa* cause and effect diagram (also known as the fish-bone diagram).

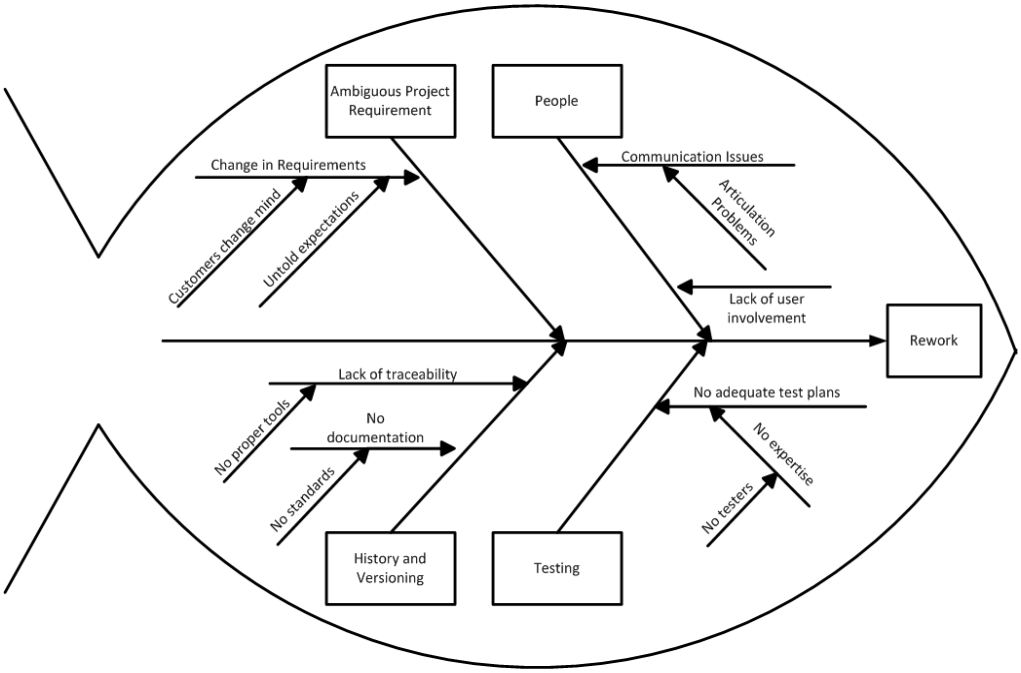


Figure 1. Root cause analysis of rework in software development

The identified root causes of rework are as follows:

## 2.1. Ambiguous Project Requirements

Requirement issues remain a major problem in software development [2] as some requirements do not become apparent until the software exists. There were problems regarding the requirements in the production team since they were uncertain of the exact requirements of the software due to the following reasons: 1) Poorly defined and/or incomplete requirements, 2) Unrealistic expectations of the customers due to communication issues, 3) Conflicting requirements from different members of the production team, 4) Requirement gathering becomes difficult when key members are on leave, 5) Not all members are available at the same time and their degree of involvement differs, 6) Changes in requirements were not documented on a single repository. Information about changes was disseminated on emails, phone conversations and meeting proceedings.

## 2.2. People

Stakeholders' needs and expectations reflect the requirements of the software but they often do not know what they want until they see it. Significant number of studies evaluated the importance of stakeholders in software development. In the organisation, the stakeholders had articulation problems to express their needs and expectations. The developer and the client have different views of the requirements, thus leading to misunderstanding and misconception. People related issues also occurred due to the following reasons: 1) The members of the production team underestimated the importance of the requirement engineering and design phase, 2) The team lacked technical insight and stick to their own opinions, 3) Wrong/Improper coding methodology was eventually applied, 4) The developer was involved in parallel development and rework that resulted into additional schedule pressure in the development process leading to a decrease in code quality; haste makes waste.

### 2.3. Testing

Since rework was a major issue, there were schedule constraints generating pressure on the developers causing a lack of involvement in the testing phase of software development. Moreover, there was a lack of dedicated testing team or automated tool to perform the required testing. More reasons include: 1) There were a limited amount of test cases provided during development for testing and only basic testing were performed, 2) No proper documentation were available to validate the test plan and/or test results, 3) Poor fixing of bugs due to tight deadlines without considerations for reusability of the codes (hard coding / first solution that comes), and 4) Volatility of requirements during implementation or testing.

### 2.4. History and Versioning

It was time consuming to search for histories and versions of documentations and codes due to lack of traceability. Backups were done locally at the developer's workplace and on a remote server. However since there are no proper versioning and backups procedures, there were inconsistencies between the remote server and the local version resulting into an ad-hoc backup procedure. Moreover changes in the requirements and specifications were not documented properly as discussed in Section 2.1.

## 4. ANALYSIS OF ALTERNATIVES TO REDUCE REWORK

This section evaluates the alternatives proposed in literature with the aim to reduce rework in the software development process of the software organisation. Their respective advantages and disadvantages are evaluated to determine their feasibility and appropriateness in reducing rework in software development.

### 4.1. Alternative I: Standards and Procedures

Standards and procedures are the key to effective quality management that may be organisational or project standards as proven in many research. There are 2 types of standards namely product standards (define characteristics that all components should exhibit e.g. a common programming style), and process standards (define how the software process should be enacted). Standards and procedures encapsulate best practices thus avoiding repetition of past mistakes thereby reducing rework such as Capability Maturity Model (CMM) and International Organization for Standardization (ISO) that are based on the well-documented facts to produce a better product through continuous improvement. The commonly mentioned standards in literature are Information Technology Infrastructure Library (ITIL), Control Objectives for Information and Related Technology (CobiT), Capability Maturity Model Integration (CMMI), and ISO 9001 [12, 13, 14]. Another principle of building quality products through a quality culture is TQM by doing the right thing at the right time. Other frameworks gaining recent awareness are Balanced Scorecard, ISO 17799 (IT security techniques), Project Management Book of Knowledge (PMBOK) and PRojects IN Controlled Environments (Prince) 2 [14]. Rework is also considered as a measure in CMM, along with quality and productivity [8]. The option of adopting standards and procedures in order to reduce rework in software development is evaluated in Table 1.

Table 1. Analysis of standards and procedures

| **Advantages** | **Disadvantages** |
|---|---|
| Help to reduce ad-hoc processes and procedures in the organisation. | Difficult to set up and maintain. It takes time to inculcate standards and procedures into the culture of an organisation. |
| Develop and maintain best practices. | Expertise is needed to train resources. |
| Provide coding and module naming conventions. | Certification costs are high. |
| Identification of specifications, test plans and procedures, programming manuals, and other documents. | Dependant on external auditors leading to more labour cost. |
| Appropriate procedures are provided to be used for the identification and resolution of bugs. | Employees tend to check compliance to standards only when auditing is near. |

Standards and procedures will help in reducing rework by:

1) Verification of compliance with established standards and procedures,
2) Defined and structured standards to follow when carrying out the rework process,
3) Help reduce rework process time by providing appropriate guidelines.

However, the high cost factor remains a major consideration when integrating frameworks [14, 15]. Their concepts can however be inspired, such as the configuration management from ITIL, managing human resources and quality from CobiT, quality assurance from CMMI and quality audits from ISO 9001.

## 4.2. Alternative II: Audits and Reviews

Audits and reviews are principal methods of validating the quality of a process or of a product. It consists of entry criteria, procedures and exit criteria [21]. During the development process, auditors and reviewers examine part or all of a process or system and its potential documentation to find potential problems to ensure developed projects meet customer's expectation effectively and efficiently, and to drive continuous improvement in software quality [22]. For reducing rework, audits and reviews help in the discovery of system defects and inconsistencies.

Some examples of such processes are: 1) software process assessment that evaluates software development against benchmarks, 2) technical review that checks for correctness and 3) post mortem report that evaluates a project at the end similarly as an audit but less formal [22].

Table 2. Analysis of audits and reviews

| **Advantages** | **Disadvantages** |
|---|---|
| Discover defects and inconsistencies in the process. | Time and resource consuming process to carry out audits and reviews. |
| Provides quality reviews by validating the quality of the process. | If review result is classified as "No Action", the review effort will not have any significant contribution to the rework issue. |
| Help to examine the documentation to find potential inconsistencies leading to rework. | If review result is classified as "Reconsider overall design", top management may be unwilling to adopt this approach due to costing, resources and schedule constraints. |
| Consider whether necessary actions are being taken promptly to remedy any significant failings or weaknesses. | Need expertise and dedicated employees. |

Audits and reviews will help in reducing rework by:

1) Carrying out technical analysis of the software and documentation to find possibilities of rework between the specification and the design, code or documentation,
2) Discovering defects and inconsistencies at an earlier stage to prevent rework.

## 4.3. Alternative III: Configuration Management

Software configuration management (SCM) is the process of tracking and controlling changes to software and hardware. General SCM systems provide tracing information within the development environment [23]. Configuration management practices include revision control and the establishment of baselines. The necessity of SCM tools have increased in software development organisations since software is continuously growing in size and complexity, and standards such as ISO emphasizes traceability management as a core factor in determining software quality and customer satisfaction [23, 24]. Interdependencies between software project parameters and changes are often not explicitly documented leading to rework [24]. SCM is a proven practice which can be adopted to reduce rework and thus bring quality to the software development process [25].

There are software tools available for performing software configuration management, notably history and versioning. They provide an automated support for the change process in both software and documentations. Some of most widely used tools in the software industry are Redmine and Apache Subversion [27], as well as others such as Git/Github that has recently started gaining popularity by means of the open source developer's community.

**Redmine** is a free and open source, web-based project management and bug-tracking tool [26, 29]. It includes calendar and Gantt charts to aid visual representation of projects and their deadlines. It supports multiple projects. Redmine provides integrated project management features, issue tracking and support for multiple version control options. Moreover, it has the productivity underlying feature of software configuration management. Further characteristics of Redmine are:

1) It is a flexible project management web application,
2) It integrates Subversion (Software Configuration Management),

3) It supports multiple projects,
4) It enables flexible role based access control (multiple users),
5) It is a flexible bug tracking system,
6) Gantt chart (time tracking) and calendar facilities are included,
7) News, documents and files management can be performed,
8) It can send feeds and email notifications,
9) It provides a wiki per project that contains valuable project information e.g. database details,
10) It provides multiple LDAP authentication support (use of business email and password to login directly),
11) It has multi-language support,
12) Multiple databases support is accessible.

**Apache Subversion** is an open source software versioning and a revision control system used to track the changes to directories of files under version control [30]. Developers use Subversion to maintain current and historical versions of files such as source code, web pages and documentation. Its goal is to be a mostly-compatible successor to the widely used Concurrent Versions System (CVS) in software development [28]. It is simple to use and it supports the needs of a wide variety of users and projects, from individuals to large-scale enterprise operations. Ben Collins-Sussman stated that the Subversion project is a kinder and gentler versioning system [27] and it is widely used in software organisations.

Table 3. Analysis of configuration management

| **Advantages** | **Disadvantages** |
|---|---|
| Facilitate prioritisation. | Time consuming to use. |
| Control and monitor rework and help to track the evolution of the project. | Employees might be unwilling to accept the new system. |
| Centralised system that motivates team work. | Training is required. |
| Provide accurate versioning of scripts promoting traceability. | Employees may fail to keep the system up-to-date on a long term basis. |
| Reduced time consumption of rework by providing history of changes. | |
| Promote security by assigning ownership to tickets. | |
| Eliminate ad-hoc working environment. | |

A system of configuration management will help in reducing rework by:

1) By providing versioning of all rework, the developer will not waste additional time in searching archives/ backups,
2) History will facilitate tracking of all user access and updates done on the scripts via ticketing,
3) Each advantage will help in building quality in the software development process,
4) Enable developer to work in a controlled manner (instead of ad-hoc as currently the case).

## 5. RESULTS & RECOMMENDATION

Multiple criteria were chosen for evaluation of the alternatives in order to come up with the best recommendation to resolve the rework issue from software development in the organisation. The criteria are:

1) Production, as it is the software production environment which is flecked by rework issues,
2) Cost, as rework causes an increase in the cost of the software project in terms of scope, time and resources,
3) Time, as rework involve time consumption causing delays and schedule pressure in the software project, and
4) Resources, such as human resources as rework involves many people from the production team to re-do tasks that they had already done in the past.

Table 4 shows an evaluation of the alternatives proposed in Section 4 based on the selected criteria.

Table 4. Evaluation of alternatives

| **Criteria** | **Standards and Procedures** | **Audits and Reviews** | **Configuration Management** |
|---|---|---|---|
| Production | Bring in quality and control by ensuring established standards and procedures.<br><br>Provide appropriate guidelines to follow. | Ensure consistencies between specification and design, code or documentation. | Bring in quality and traceability.<br><br>Helps in maintaining accurate history and documentation.<br><br>Enable monitoring and control with user-friendly interface. |
| Cost | Costly to recruit experts for setting up standards.<br><br>Certification costs. | Certification costs.<br><br>Costly to make changes to the system after the reviews.<br><br>Costly to invest in external audits and reviews. | Open-source software such as Redmine and Subversion are available for free. |
| Time | Time consuming to set up and carry out. | Time consuming to carry out auditing and review process.<br><br>“No action” result after reviews render the audit time consumption unworthy. | The installation is done only once and need to be maintained during software projects development. |

| | | | |
|---|---|---|---|
| Resources | No available resource person with expertise.<br><br>Dependant on auditors. | Currently no expertise to perform audit and reviews in the organisation. | No external expertise required.<br><br>Tools are well-documented, and can be used for installation, configuration and training. Thus, it is considered to be a plug and play option for a developer. |

Based on the root-cause analysis shown in the *Ishikawa* diagram of Figure 1, the following criteria was selected in order to reduce rework:

1) Traceability / Versioning
2) Quality
3) Documentation
4) History of rework
5) Cost Efective

Figure 2 show the evaluation of the most feasible option for the organisation based on literature of the alternatives.

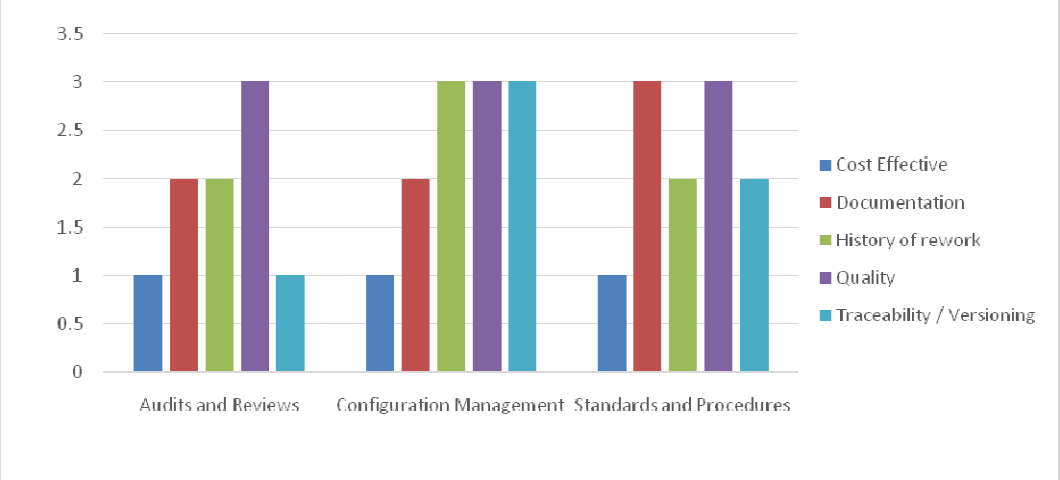


Figure 2. Feasibility of alternatives

Configuration management, that is a proper tracking and versioning system proved to be the most viable option as it will resolve most of the root causes of rework as compared to the other two options. One example of such a system is Redmine with Apache Subversion (SVN) that comes with a good documentation on the official Redmine website [31]. The recommended option was implemented at the start of one particular software project. Table 5 shows the test cases needed to be executed to check if the versioning system is up and running. Table 6 shows the test cases needed to be executed to check if the tracking system (Redmine) was up and running.

Table 5. Test case of versioning system

| | **Test Cases for checking SVN** | **Expected Result** |
|---|---|---|
| 1 | Developer update from server and opens test.aspx | Version of developer is the latest one **(V 1.0)** |
| 2 | Team Leader update from server and opens test.aspx | Version of Team Leader is the latest one **(V 1.0)** |
| 3 | Developer modifies the file, update from server and commit to server | Version on server is updated **(V 1.1)** |
| 4 | Team Leader modifies the file, update from server and commit to server | Version on server is updated **(V 1.2)** |

Table 6. Test case of tracking system

| | **Test Cases for checking Redmine** | **Expected Result** |
|---|---|---|
| 1 | Team leader logs in a ticket **(#0001)** and assigns the ticket to developer | The ticket is recorded in Redmine as status "new" |
| 2 | Developer receives an email that ticket **#0001** has been assigned to him/her | The ticket is updated to status "assigned" |
| 3 | Developer takes updates from server, work the ticket, update to server and commit the changes **(V 1.5).**<br>Developer records the version number on Redmine and assigned the ticket to Team Leader as Resolved | The link to codes is made automatically by Remine **(V 1.5)** includes modified files X.aspx, Y.aspx and Z.aspx, with specified line modified. The ticket changes status to "Resolved" |
| 4 | Team leader tests the modifications made and closes the ticket. History of versions worked and descriptions are archived in the ticket | The ticket changes status to "Closed" and is archived |

In order to determine the gain in rework, several key performance indicators (KPI) may be monitored within the organisation after the implementation of the configuration management system of tracking and versioning as a future work. Such a KPI measure is annual costing, used in the organisation to have an overall view of additional costs associated with software development. Similarly the number of requirements and tickets are used to account the amount of tasks that was present on a similar previous software project, compared with the amount of tasks in the software project after implementation of the software configuration management system. These indicators can be used as a measure to monitor the effect of rework reduction on the organisation.

## 6. CONCLUSION

Despite successes in reducing rework, we acknowledge that rework cannot be eliminated entirely as it is inevitable due to many factors mentioned in the paper. Also, not doing a task 'right the

first time' eventually creates more work and lead to rework as a consequence. In addition, not all rework-inducing problems can be detected as soon as they occur; some problems will only be caught at some distance downstream. Nevertheless, the rework process need not be as difficult as many make it. The software organisation considered in this paper was having serious problems regarding rework, and a major part of it was caused by a lack of software configuration management, among other reasons such as traditional requirements problems. This particular area of deficiency was improved by introducing a proper history and version control system by introducing Redmine and Apache Subversion in the organisation. Preliminary results showed a gain in productivity as rework was greatly reduced. As a future work, metrics such as key performance indicators could be used to measure the effect of rework reduction and gain in productivity through software configuration management tools in the organisation. Regular audits and reviews can also be performed to ensure continuous improvement of the development process within the organisation at a later stage. This will facilitate improvement of other deficiencies related to causing rework to occur. To conclude, rework is not 'bad luck' and is absolutely manageable if given due considerations to the causes of rework.

## REFERENCES


[1] Vimla Devi Ramdoo, Geshwaree Huzooree, and Oomesh Gukhool, The study of the complexity of software quality to promote sustainable growth in business through the delivery of quality software, Uom Research Week 2012, University of Mauritius 2012.

[2] Geshwaree Huzooree and Vimla Devi Ramdoo. A Systematic Study on Requirement Engineering Processes and Practices in Mauritius. International Journal of Advanced Research in Computer Science and Software Engineering, v5.2, 40-46, 2015.

[3] Helio Yang, Y., Software quality management and ISO 9000 implementation, Industrial Management & Data Systems 101.7: 329-338, 2001.

[4] Rúbio, Thiago RPM, and Carlos ASJ Gulo. Characterizing Developers' Rework on GitHub Open Source Projects. Doctoral Symposium in Informatics Engineering. 2015.

[5] Raja, Uzma, and Marietta J. Tretter. Defining and evaluating a measure of open source project survivability. Software Engineering, IEEE Transactions on38.1: 163-174, 2012.

[6] Zahra, Sobia, et al. Performing Inquisitive Study of PM Traits Desirable for Project Progress. International Journal of Modern Education and Computer Science (IJMECS) 6.2: 41, 2014.

[7] Rahmandad, Hazhir, and Kun Hu. Modeling the rework cycle: capturing multiple defects per task. System Dynamics Review 26.4: 291-315, 2010.

[8] Diaz, Michael, and Jeff King. How CMM impacts quality, productivity, rework, and the bottom line. CrossTalk 15.3: 9-14, 2002.

[9] Cass, Aaron G., Stanley M. Sutton Jr, and Leon J. Osterweil. Formalizing rework in software processes. Software Process Technology. Springer Berlin Heidelberg, 16-31, 2003.

[10] Haley, Tom, et al., Raytheon Electronic Systems Experience in Software Process Improvement. No. CMU/SEI-95-TR-017. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1995.

[11] Charette, R.N., Why software fails. IEEE spectrum, 42.9 (36), 2005.

[12] Vidal, H.,Wan, J And Han, X.. Capability Models: ISO and CMM. Kansas: Department of Computing and Information Sciences, Kansas State University, 1998.

[13] Gerke, L., And Ridley G. Towards an abbreviated COBIT framework for use in an Australian State Public Sector, 2006.

[14] Cater-Steel, A., Wui-Gee, T., And Toleman M. Challenge of adopting multiple process improvement frameworks. Proceedings of 14th European conference on information systems, 2006.

[15] Paulk, M. C.. Surviving the quagmire of process models, integrated models, and standards, 2004.

[16] Gopal, Anandasivam, Tridas Mukhopadhyay, and Mayuram S. Krishnan. The role of software processes and communication in offshore software development. Communications of the ACM 45.4: 193-200, 2002.

[17] Basili, Victor R., et al. Characterizing and modeling the cost of rework in a library of reusable software components. Proceedings of the 19th international conference on Software engineering. ACM, 1997.

[18] Morozoff, Edmund P. Using a line of code metric to understand software rework. Software, IEEE 27.1: 72-77, 2010.
[19] Conroy, Patrick, and Philippe Kruchten. Performance norms: An approach to rework reduction in software development. Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on. IEEE, 2012.
[20] Basili, Victor R., et al. Characterizing and modeling the cost of rework in a library of reusable software components. Proceedings of the 19th international conference on Software engineering. ACM, 1997.
[21] IEEE Computer Society, IEEE Standard for Software Reviews and Audits, 2008.
[22] Fallan, M. Hosein, et al. Development process audits and reviews. AT&T Technical Journal 70.2: 99-108, 1991.
[23] Kim, Dae-Yeob, and Cheong Youn. Traceability enhancement technique through the integration of software configuration management and individual working environment. Secure Software Integration and Reliability Improvement (SSIRI), Fourth International Conference on. IEEE, 2010.
[24] Vanbrabant, Bart, and Wouter Joosen. A framework for integrated configuration management tools. Integrated Network Management (IM 2013), IFIP/IEEE International Symposium on IEEE, 2013.
[25] Khan, Chaudry Bilal Ahmad, and Ali Ahsan. Recommended configuration management practices for freelance software developers. Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on. IEEE, 2014.
[26] Lang, J., and E. Davis. Redmine-open source project management web-application, 2010.
[27] Collins-Sussman, Ben, Brian Fitzpatrick, and Michael Pilato. Version control with subversion. O'Reilly Media, Inc., 2004.
[28] Collins-Sussman, Ben. The subversion project: building a better CVS. Linux Journal 2002.94: 3, 2002.
[29] Redmine, available on http://www.redmine.org/ (accessed on 18/09/2015).
[30] Apache Subversion, available on https://subversion.apache.org/ (accessed on 18/09/2015).
[31] HowTo configure Redmine for advanced Subversion integration, available on http://www.redmine.org/projects/redmine/wiki/HowTo_configure_Redmine_for_advanced_Subversion_integration (accessed on 18/09/2015).