ISSN 2278-3091
Volume 9, No.1.3, 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse0691.32020.pdf
https://doi.org/10.30534/ijatcse/2020/0691.32020

Human Competency Considerations in an Integrated Software Configuration Management Framework

Syahrul Fahmy[1], Amir Ngah[2], Ahmad Faiz[3], Nurul Haslinda[1], Wan Roslina[1]
[1]TATI University College, Malaysia, fahmy@tatiuc.edu.my
[2]Universiti Malaysia Terengganu, Malaysia
[3]SEGi University and Colleges, Malaysia

ABSTRACT

This paper proposes an integrated framework that combines software testing and software quality into standard Software Configuration Management process, whilst highlighting the importance of human competency. Software quality characteristics are embedded in change control procedures whilst software testing approaches provide means for evaluating software artefacts. Existing software configuration management, software quality and software testing standards are referred to in the development of this framework. The importance of competency is highlighted where the minimum competency for implementing each software configuration management process is presented. Advantages over traditional approaches are presented from four different aspects namely *People*, *Process*, *Tools* and *Documentation*. Major challenges in the realization of this integration are discussed including competency assessment; quality characteristics and metrics; test approach; and automation.

Key words: Human Competency, Software Configuration Management, Software Quality, Software Testing.

1. INTRODUCTION

A conceptual framework for integrating software testing and software quality into Software Configuration Management (SCM) was proposed by [1] to address the issues of project delays and low quality software products. The framework was based on existing standards for SCM, software quality and software testing in weaving software quality into SCM and instill standardized testing procedures for evaluating software artefacts.

SCM was first used in software development in the 1950s with the adoption of Configuration Management to manage product changes and has been used since then to ensure timely delivery of software products. In addition, software quality models have been used since the 1990s to ensure conformance (to product requirements and needs). However, the issues of project delays and unfit software products still prevail in software development, prompting the need for a comprehensive approach to address these problems.

This paper extends the work in [1] and proposes an integrated framework that combines software testing and software quality into SCM, whilst highlighting the importance of human competency in the whole process.

The motivation for looking into the competency aspects in SCM is based on the observation that configuration management in software engineering only works to a certain degree. In manufacturing, a same set of input and process would produce the same result every time. However, this is not always true in software engineering as there is another factor involved in the equation: the skill of the person who implements the process, i.e. the human competency. This makes software development, more art than science, compared to manufacturing.

Human competency in software engineering have been extensively studied for example the effects of knowledge and code ownership to software quality [2-3]; the effects of individual decision-making behavior to quality [4], and promoting quality in the development team [5]. However, it has not been the focus of SCM research, apart from project team collaborations [6] and debugging activities [7].

This paper is organized as follows: Section 2 presents a brief overview of SCM, software quality, software testing and human competency. Section 3 presents i-SCM, an integrated SCM framework, combining software quality characteristics and software testing approaches into standard SCM process. Section 4 lists the challenges for this integration and conclusion is presented in Section 5.

2. OVERVIEW

This section presents a brief overview of software configuration management, software quality, software testing and human competency.

2.1 Software Configuration Management

SCM can be loosely defined as “the ability of control and manage changes in a software project”. It is used to control the evolution of software systems [8]. A more formal definition of SCM is “a supporting-software life cycle process that benefits

project management, development and maintenance activities, quality assurance activities, as well as the customers and users of the end product" [9].

Research efforts in SCM has been highly motivated by tackling the problems at hand in software development for example issues in the development of large software systems in the 1980s; object-oriented systems in the 1990s; web services in the 2000s; and late binding systems in the 2010s. Majority of research are technical in nature involving concepts, models and tools. Commercial and proprietary tools are aplenty, and the underlying techniques are no longer confined to SCM, but in other areas such as web services.

Efforts to formalize the process in SCM started as early as the 1960s with the ideas of configuration as a control mechanism in software development [10] and concepts for program specifications [11]. The process underwent further refinement throughout the 1970s with the recognition of software engineering as a new field in computing.

In 1983, IEEE published the first standard for SCM, the IEEE 828 - IEEE Standard for Software Configuration Management Plans, which was revised in 1990, 1998, and 2005. The latest version was released in 2012 [12]. It establishes the minimum requirements for configuration management processes in systems and software engineering.

In addition, ISO published a quality-related standard for SCM in 1995, the ISO 10007 Quality Management - Guidelines for Configuration Management, which was revised in 2003. The latest version was released in 2017 [13]. The ISO standards provide guidance on the use of configuration management within the organization.

IEEE stipulates six crucial process in SCM namely (1) management and planning of the SCM process; (2) software configuration identification; (3) software configuration control; (4) software configuration status accounting; (5) software configuration auditing; and (6) software release management and delivery [12].

The outcome of SCM is the SCM Plan (SCMP), a living document that is used throughout the software life-cycle. SCMP provides a systematic view of the current configuration, supports decision-making activities in processing change request, provides information on the product status, and facilitate future enhancements through detailed product documentation.

2.2 Software Quality

Quality is highly subjective and the term software quality has been referred to "conformance to requirements" [14], "fitness for use" [15], and "capability of software product to satisfy stated and implied needs" [16]. It focuses on the conformance (of software products) to requirements. Quality models are often used to measure quality and understand how quality metrics relate to each other. There are several models for assessing software products including McCall, Boehm, FURPS, Dromey, and ISO 9126. ISO 25010 [16] succeeded ISO 9126 in 2011 and it defines quality from two perspectives namely quality in use and product quality.

ISO 25010 product quality model is composed of eight major characteristics that relate to static properties of software and dynamic properties of the computer system (Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, & Portability). This model is applicable to both computer systems and software products.

Software quality research in SCM are mainly associated with source code defects where focus are directed at reducing or eliminating them. Examples include real-time quality control through the analysis of code change [17]; consecutive changes and software defects [18]; estimating defects and changes in software systems [19]; heterogeneous defect prediction across projects with heterogeneous metric sets [20]; software defect metrics to aid analysis [21]; impact of product development strategy on defects [22]; and a decision support system to predict defects and enhance release management [23].

2.3 Software Testing

Software testing is "the dynamic verification that a program provides expected behaviors on a finite set of test cases, selected from an infinite execution domain" [9]. It is conducted to provide stakeholders with information about the quality of the software product under test [24]. The main standard governing software testing is the ISO 29119 [25-29], a series of five international standards for testing. It is a comprehensive standard, defining concepts and vocabulary; test process descriptions; templates and examples of test documentation; test design techniques; and solution for keyword-driven testing.

Software testing is performed at different levels throughout the software life-cycle, based on the target or the objective of test. The target of the test can vary between a single module (unit testing), a group of modules (integration testing), or an entire system (system testing) [30-31]. The objective of testing is conducted with specific objectives that are quantitatively defined for example acceptance testing, installation testing, alpha/ beta testing, regression testing, and performance testing [30-31].

There is only a few works (if any) on software testing in SCM for example [32], although there are interest on the adoption of SCM in software testing such as [33-34]. One reason would be software testing is being carried out as a separate process in SCM, supported by lack of details on the approach for testing found in SCM standards.

2.4 Human Competency

Competency can be defined as "the combination of knowledge, skills, abilities, and personal attributes that contribute to enhanced employee performance". Successful SCM implementation is dependent on several factors including the competency of the software practitioners. Traditionally, competency was built on personal qualities and developed primarily through experience. Recently, education and training have taken on a greater role in the development of human competency. In SCM, the study of competency has focused on:

- Collaboration activities such as conflict history data [6]; revision history [35]; and the impact of a change made by one developer to other developers [36].
- Learning activities for example the use of version control system in student development projects [37]; implementation of a distributed revision control system as part of the undergraduate and graduate curriculums [38]; and the integration of configuration management into the IT curriculum [39].
- Debugging activities for example the correlation between a commit's social characteristics and bugs [7]; the organization of bug reports into sets for effective management by developers [40]; and the relationship between developers' communication frequency and number of bugs [41].

Although human competency has not been the main focus of SCM research, IEEE has outlined a generic skill set for software practitioners in the Software Competency Model (SWECOM) [42]. SWECOM specifies skill areas, skills within skill areas, and work activities. Activities are specified at five levels of increasing competency namely Technician, Entry Level Practitioner, Practitioner, Technical Leader, and Senior Software Engineer. Software practitioners involved in SCM implementation would have to acquire three specific skill sets namely Plan SCM, Conduct SCM and Managing Software Releases (Table 1).

Table 1: Software Configuration Management Skill Sets and Activities based on the IEEE Software Engineering Competency Model

Software Configuration Management Skill Set	Software Configuration Management Activities
Plan SCM	▪ Determine organizational context for and constraints on SCM ▪ Identify software components to be controlled by SCM ▪ Design data and code repositories ▪ Plan versioning procedures for path branching and path integration ▪ Develop/adopt a change control process ▪ Identify and procure SCM tools ▪ Establish SCM library ▪ Develop SCMP
Conduct SCM	▪ Follow SCMP ▪ Use SCM tools ▪ Control path branching and path integration during development ▪ Generate, classify, and manage problem reports ▪ Maintain and update SCM baselines ▪ Prepare SCM reports ▪ Conduct SCM audits
Manage Software Releases	▪ Develop software release plan ▪ Identify and procure software release tools ▪ Use software release tools ▪ Produce software releases ▪ Design and implement tools and procedures for generating patches to be delivered

3. CONCEPTUAL FRAMEWORK

Figure 1 illustrates the integrated SCM framework (i-SCM). Standard SCM process serves as the backbone in this framework and can be adopted from existing standards such as IEEE 828 [12], CMMi [43], ISO 10007 [13], or other proprietary process as practiced by the organization. Software quality are incorporated through pre-defined, project-specific quality characteristics. These characteristics can be based on existing standards such including ISO 25020. Software testing contributes through procedures for evaluating changes made to the software product. Existing standards including ISO 29119 can be referred to in determining the approach for testing.

Human competency, based on SWECOM, can be adopted to determine the minimum set of skills required by software practitioners to successfully implement the SCM process. The final outcome of SCM implementation is the SCMP, that serve not only as proof of conformance, but also as reference for future enhancement of the software product.

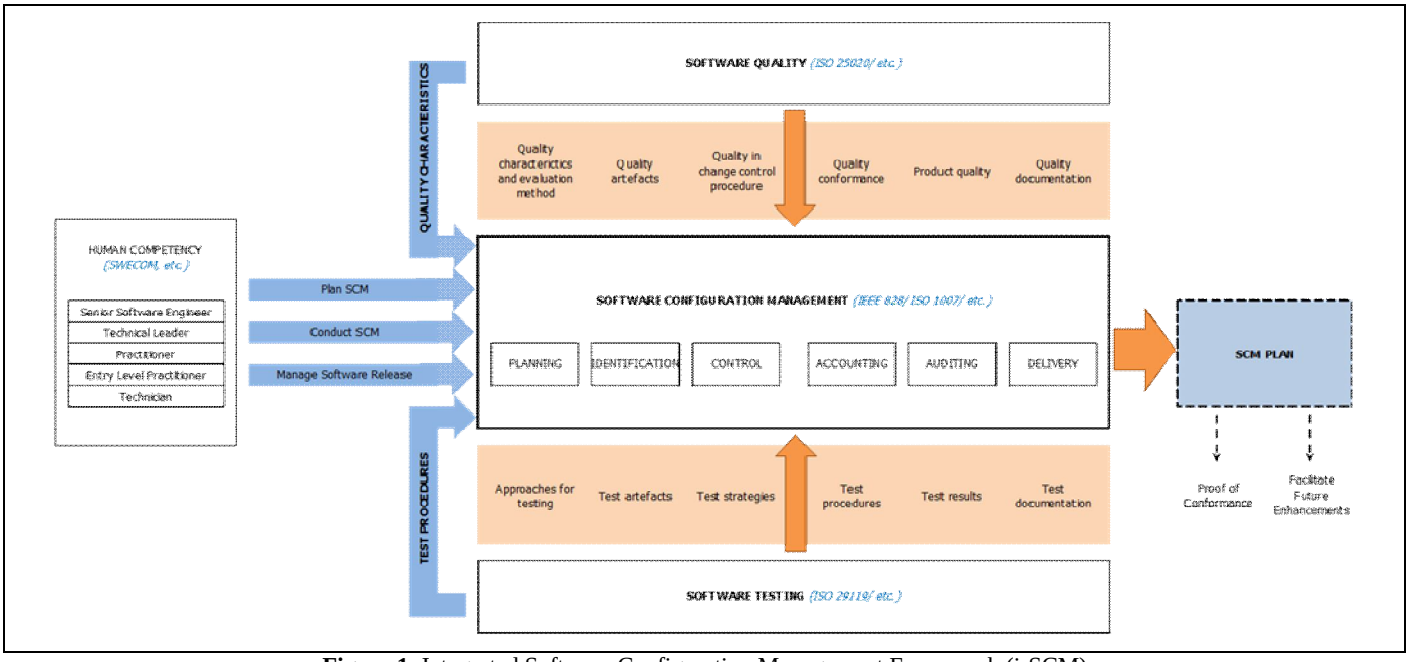

Figure 1: Integrated Software Configuration Management Framework (i-SCM)

3.1 Planning

Planning is the first process in SCM and should be consistent with the organizational context, constraints, and nature of the project. Effective planning coordinates activities throughout the software product life-cycle. The output of this process is the SCMP. SCMP is subjected to SQA review; documented and approved; and controlled. Software quality requirements are identified in this phase including strategies for achieving them. In addition, approach for testing, including the levels and objectives of testing, is also identified and documented in this process.

3.2 Configuration Identification

The next process is the identification of Configuration Items (CIs) to be controlled. CIs are selected using established criteria early in the software product life-cycle and reviewed as the product evolves. Typical CIs include requirements, designs, and source codes. Software quality and testing artefacts such as test specification and supporting tools are identified in this process. CIs may consist of multiple related artefacts that form a baseline. These baselines including approved changes, represent the current approved configuration.

3.3 Control

After the initial release of configuration information, all changes are controlled and documented. This covers the process of determining what changes to make; the authority for approving changes; and the implementation of changes. Software Trouble Report, Software Change Request and similar documents should explicitly state the quality characteristics that will be affected by the proposed change and how these changes will be tested.

3.4 Accounting

Accounting is the recording and reporting of information for managing a configuration effectively. It is performed throughout the software product life-cycle. Types of information recorded include approved CIs and baselines; current implementation of changes; and status of release. Conformance to defined quality characteristics are also recorded along with test objectives and procedures used.

3.5 Auditing

Auditing is performed in accordance with documented procedures to determine whether a product conforms to its requirements and configuration information. Informal audits can be conducted as necessary whilst formal audits can be carried out as scheduled. Here software testing can play a more significant role through the adopted approach and testing tools used.

3.6 Delivery

This final process involves the identification, packaging, and delivery of artefacts such as an executable program. Quality and test documentation are included in the conventional SCM Plan, underlining initial quality characteristics of the product, quality characteristics after the implementation of change, test approach adopted, test tools used, and results obtained.

The minimum competency for implementing each process is determined by SWECOM's SCM skill set:

- Minimum competency for managing the Planning process is Practitioner, as a practitioner "participates in determining impact of constraints on SCM imposed by policies, contract, and SDLC".

- Minimum competency for managing the Identification process is Practitioner, as a practitioner "participates in identifying SCIs and the relationships among them" and "participates in developing software release plans".
- Minimum competency for managing the Control process is Technical Leader, as a technical leader "appoints members and convenes the CCB" and "tailors and adopts mechanisms for requesting, evaluating, and approving software changes".
- Minimum competency for managing the Accounting process is Technical Leader, as a technical leader "leads the CCB in making yes/no decisions on change requests" and "ensures that approved changes are made and documented".
- Minimum competency for managing the Audit process is Practitioner, as a practitioner "leads the building and verifying of software releases".
- Minimum competency level for carrying out the Delivery process is Practitioner, as a practitioner "leads the building and verifying of software releases".

The implementation of these processes contribute to the development of the SCM Plan (Figure 2). The SCM Plan takes shape during Planning where the Contractual, Organizational, Project and Software Quality and Test Requirements are identified. During the Identification process, contractual and organizational requirements dictate the type of artefacts that are going to be controlled. List of controlled artefacts and approved baselines are added to Project Documentation.

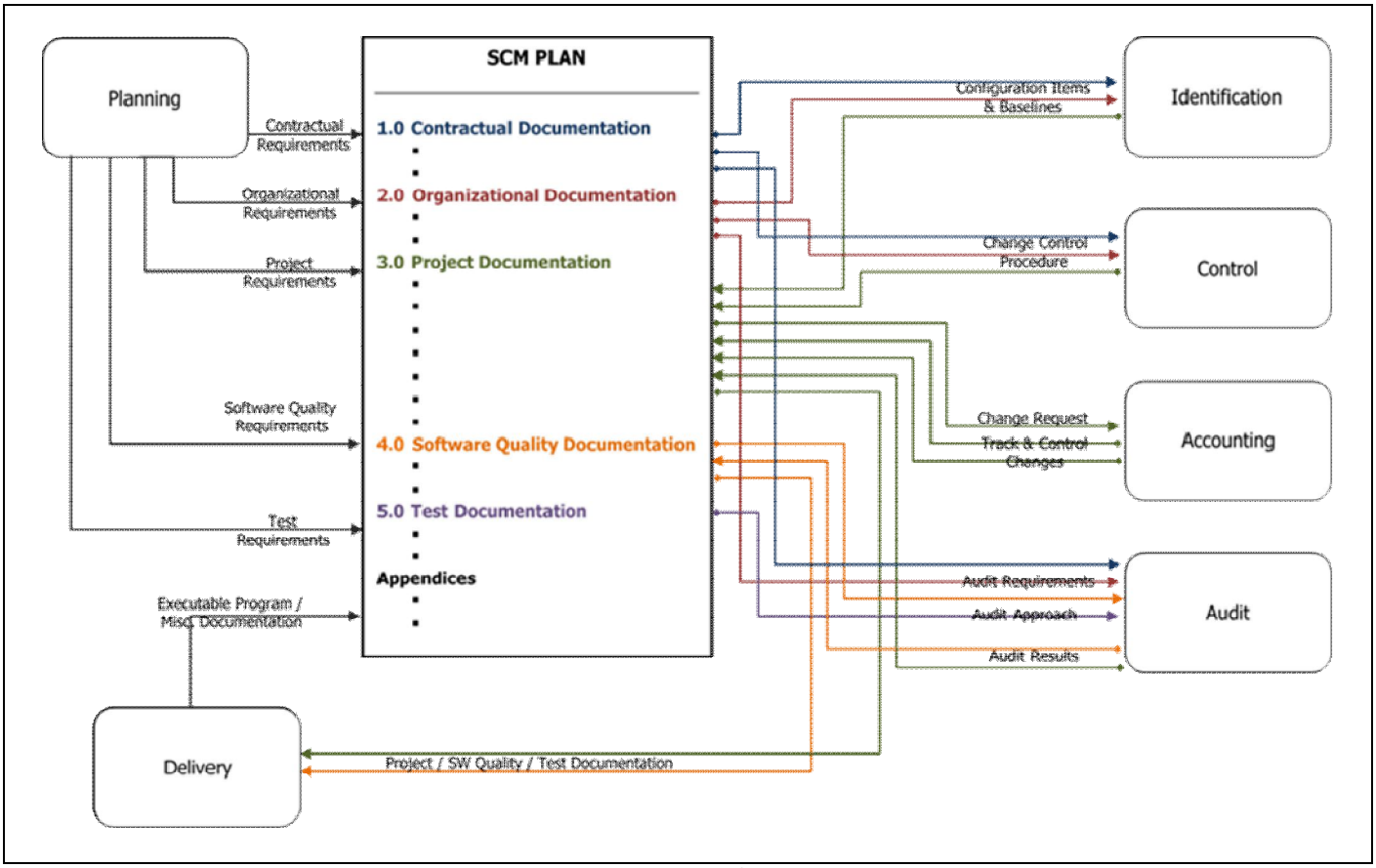

Figure 2: Development of the SCM Plan

In Control, contractual and organizational requirements characterize the change control and change request procedures. These information are also appended to Project Documentation. In Accounting, change requests and artefacts approval are processed. Change requests, CIs and baselines approvals are appended to Project Documentation.

In Audit, the contractual, organizational, project and software quality requirements are taken into consideration and referred to in determining the test approach. Test results are appended to the Project and Software Quality Documentation. In Delivery, the Project, Software Quality and Test Documentation are included in the software package.

Finally, the software product and other documentation are attached to the SCMP as appendices. The advantages of i-SCM over traditional SCM approach can be viewed from four different aspects namely People, Process, Tools and Documentation (Table 2).

Table 2: Advantages of i-SCM

Component	Traditional SCM	i-SCM
People	Mainly tasked with the operation of tools	Competency is a dominant factor in SCM implementation
Process Software Quality	×	Focal in the Planning, Control, Accounting, Auditing, and Delivery
Software Testing	×	Focal in the Control, Auditing, and Delivery
Tools	Dominant factor in SCM implementation	**Supports** SCM implementation
Documentation	Mainly acts as project documentation	**Guides** SCM implementation and passed over to facilitate future product enhancements

i-SCM emphasizes on the competency of People in the implementation of process, utilization of tools, and generation of documentation. Traditional SCM approach relies heavily on the use of tools for implementation and People are mainly tasked to operate these tools. In addition, i-SCM promotes the use of SWECOM SCM skill set to successfully implement SCM. Traditional SCM does not make any distinctions regarding these skills.

i-SCM focuses on the importance of software quality where it is identified in the Planning process, explicitly stated in Control, effects of a proposed change to quality is taken into consideration in Accounting, quality audits are carried out in addition to project audits, and quality documentation is included in Delivery. Traditional SCM implementation gives little emphasis (if any) to software quality as quality assessment is usually carried out in other software engineering process. Test approaches are also the focus of i-SCM where test strategies are explicitly stated in the Control process, test procedures are specified in Accounting and Auditing, and test results are included in Delivery.

4. CHALLENGES

There are four major challenges in the realization of this integration namely competency assessment; quality characteristics and metrics; test approach; and automation.

4.1 Competency Assessment

The first challenge is the identification of assessment methods and approach. Although many software organizations use proprietary competency models to assess the performance of their employees, general (software engineering) competency models can also be adopted including INCOSE [44]; ENG Competency Model [45]; NASA APPEL Competency Model [46]; MITRE Competency Model [47]; and CMMI Competency Model [48]. Approach for assessment will also need to be identified for example self-estimation, interviews and/ or work product audit based on the assessment needs.

4.2 Quality Characteristics and Metrics

The second challenge is the identification of applicable quality characteristics for SCM. Previous findings such as quality factors for certification [49] and refinement of existing standards to evaluate quality [50] could be adopted for this purpose. Once the characteristics have been identified, related metrics would then be formulated for each characteristics. It is crucial that the relationship between quality and SCM process is preserved as outlined by relevant standards.

4.3 Test Approach

The third challenge is the identification of suitable approach for testing the various software artefacts. Previous approaches including program slicing [51] can be used. In addition, the use of suitable support tools for testing would also need to be identified. Not only testing needs to evaluate CIs and baselines to configurations, but it also need ensure conformance to predefined quality requirements [52].

4.4 Automation

Perhaps the most interesting challenge is the automation of testing in SCM. The advantages of automated testing has been widely reported. With regards to SCM, automated testing would enable early defect detection, thus increasing development speed; lead to improved efficiency, as tests can be run unattended; and test automation provides a larger coverage than manual test, hence assuring a higher product quality is obtained.

5. CONCLUSION AND FUTURE WORK

This paper proposes i-SCM, an integrated framework that combines software testing and software quality into standard SCM process, whilst highlighting the importance of human competency in the whole process. Software quality characteristics are embedded in the change control procedure and software testing provides means for testing software artefacts to ensure conformance. It highlights the competency of people in the implementation of process, utilization of tools, and generation of documentation. I-SCM promotes the

delivery of project-specific and quality documentation to support future enhancements to the software product. Future work include the identification of suitable quality characteristics for SCM and existing tools for testing them.

REFERENCES

1. Fahmy, S., Deraman A., & Ngah, A. (2018). A Conceptual Framework for Integrating Software Quality and Testing into Software Configuration Management. The Postgraduate Workshop@SOFTEC Asia 2018, Sept 02nd, Kuala Lumpur, Malaysia. https://doi.org/10.1145/3185089.3185117
2. Orru, M. and Marchesi, M. (2016). A Case Study on the Relationship Between Code Ownership and Refactoring Activities in a Java Software System. In Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics, Austin, USA, 43-49. https://doi.org/10.1145/2897695.2897702
3. Ribeiro, D.M., da Silva, F.Q.B., Valença, D., Freitas, E.L.S.X., and França, C. (2016). Advantages and Disadvantages of Using Shared Code from the Developers Perspective: A Qualitative Study. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Ciudad Real, Spain, Article 33, 6 pages.
4. Jia, J., Zhang, P., and Capretz, L.C. (2016). Environmental Factors Influencing Individual Decision-Making Behaviour in Software Projects: A Systematic Literature Review. In Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering, Austin, USA, 86-92. https://doi.org/10.1145/2897586.2897589
5. Prikladnicki, R., Perin, M., and Marczak, S. (2016). Virtual Team Configurations that Promote Better Product Quality. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Ciudad Real, Spain, Article 18, 5 pages.
6. North, K.J., Bolan, S., Sarma, A., and Cohen, M.B. (2015). GitSonifier: Using Sound to Portray Developer Conflict History. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 886-889.
7. Eyolfson, J., Tan, L., and Lam. P. (2011). Do Time of Day and Developer Experience Affect Commit Bugginess?. In Proceedings of the 8th Working Conference on Mining Software Repositories, Waikiki, USA, 153-162. https://doi.org/10.1145/1985441.1985464
8. Babich, W.A. (1986). Software Configuration Management, Coordination for Team Productivity. Addison-Wesley.
9. SWEBOK. (2014). Guide to the Software Engineering Body of Knowledge. IEEE Computer Society Press, Los Alamitos, CA, USA.
10. Oettinger, A.G. (1964). A Bull's Eye View of Management and Engineering Information Systems. In Proceedings of the 1964 19th ACM National Conference. New York, USA, 21.1-21.14. https://doi.org/10.1145/800257.808892
11. Liebowitz, B.H. (1967). The Technical Specification: Key to Management Control of Computer Programming. In Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, Atlantic City, USA, 51-59.
12. IEEE 828. (2012). IEEE Standard for Configuration Management in Systems and Software Engineering. The Institute of Electrical and Electronics Engineers. (71 pages).
13. ISO 10007. (2017). Quality Management Systems - Guidelines for Configuration Management. International Organization for Standardization. (10 pages).
14. Crosby, P.B. (1979). Quality is Free: The Art of Making Quality Certain. McGraw-Hill.
15. Humphrey, W.S. (1989). Managing the Software Process. Addison-Wesley.
16. ISO 25010. (2011). Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models. International Organization for Standardization. (34 pages).
17. Heinemann, L., Hummel, B., and Steidl, D. (2014). Teamscale: Software Quality Control in Real-Time. In Companion Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 592-595. https://doi.org/10.1145/2591062.2591068
18. Dai, M., Shen, B., Zhang, T., and Zhao, M. (2014). Impact of Consecutive Changes on Later File Versions. In Proceedings of the 3rd International Workshop on Evidential Assessment of Software Technologies, Nanjing, China, 17-24. https://doi.org/10.1145/2627508.2627512
19. Malhotra, R., and Agrawal, A. (2014). CMS Tool: Calculating Defect and Change Data from Software Project Repositories. SIGSOFT Softw. Eng. Notes 39(1): 1-5.
20. Nam, J., and Kim, S. (2015). Heterogeneous Defect Prediction. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 508-519.
21. Henderson, C. (2008). Managing Software Defects: Defect Analysis and Traceability. SIGSOFT Softw. Eng. Notes 33, 4, Article 2, 3 pages.
22. Ramler, R. (2008). The Impact of Product Development on the Lifecycle of Defects. In Proceedings of the 2008 Workshop on Defects in Large Software Systems, Seattle, USA, 21-25. https://doi.org/10.1145/1390817.1390823
23. Tosun, A., Turhan, B., and Bener, A. (2009). Practical Considerations in Deploying AI for Defect Prediction: A Case Study within the Turkish Telecommunication Industry. In Proceedings of the 5th International Conference on Predictor Models in Software Engineering, Vancouver, Canada, Article 11, 9 pages.
24. Kaner, C. (2006). Exploratory Testing (Keynote Address). Quality Assurance Institute Worldwide

Annual Software Testing Conference, Orlando, FL, USA.
25. ISO 29119-1 (2013). Part 1: Concepts and Definitions. International Organization for Standardization. (56 pages).
26. ISO 29119-2 (2013). Part 2: Test Processes. International Organization for Standardization. (59 pages).
27. ISO 29119-3 (2013). Part 3: Test Documentation. International Organization for Standardization. (127 pages).
28. ISO 29119-4. (2015). Part 4: Test Techniques. International Organization for Standardization. (139 pages).
29. ISO 29119-5. (2016). Part 5: Keyword-Driven Testing. International Organization for Standardization. (54 pages).
30. Naik, K., & Tripathy, P. (2008). Software Testing and Quality Assurance: Theory and Practice, Wiley-Spektrum. https://doi.org/10.1002/9780470382844
31. Sommerville, I. (2015). Software Engineering, 10th ed., Pearson.
32. Appleton, B., Berczuk, S., adn Cowham, R. (2007). Testing's Role in the Software Configuration Management Process. https://www.cmcrossroads.com/article/testings-role-software-configuration-management-process. Retrieved July 2018.
33. Jindal, N. (2016). Role of Software Configuration Management (SCM) in Software Testing. https://www.linkedin.com/pulse/role-software-configuration-managementscm-testing-nitish-jindal. Retrieved July 2018.
34. Software Testing Solution. (2017). Configuration Management for Software Testing. http://softwaretestingsolution.com/blog/configuration-management-software-testing-much-meets-eye/. Retrieved July 2018.
35. Huang, S-K., and Liu, K-M. (2005). Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants. THE 2005 International Workshop on Mining Software Repositories, Saint Louis, USA, 1-5. https://doi.org/10.1145/1083142.1083158
36. Sarma, S., Branchaud, J., Dwyer, M.B., Person, S., and Rungta, N. (2014). Development Context Driven Change Awareness and Analysis Framework. In Companion Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 404-407.
37. Makiaho, P., Poranen, T., and Seppi, A. (2014). Version Control Usage in Students' Software Development Projects. The 15th International Conference on Computer Systems and Technologies, Ruse, Bulgaria, 452-459. https://doi.org/10.1145/2659532.2659646
38. Gowtham, S. (2014). Revision Control System (RCS) in Computational Sciences and Engineering Curriculum. In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment, New York, USA, Article 76, 3 pages.
39. Jiang, K., and Kamali, R. (2008). Integration of Configuration Management into the IT Curriculum. In Proceedings of the 9th ACM SIGITE conference on Information Technology Education, Cincinnati, USA, 183-186. https://doi.org/10.1145/1414558.1414606
40. Bortis, G., and van der Hoek, A. (2013). PorchLight: A Tag-Based Approach to Bug Triaging. In Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, Piscataway, USA, 342-351.
41. Abreu, R., and Premraj, R. (2009). How Developer Communication Frequency Relates to Bug Introducing Changes. In Proceedings of the Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution, Szeged, Hungary, 153-157. https://doi.org/10.1145/1595808.1595835
42. IEEE Software Engineering Competency Model (SWECOM). IEEE Computer Society Press, https://www.computer.org/web/peb/swecom-download, retrieved Sept 2018.
43. CMMI Product Team. (2010). CMMI for Development Version 1.3. Software Engineering Institute.
44. INCOSE. (2010). Systems Engineering Competencies Framework 2010-0205. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2010-003.
45. ENG Competency Model. (2013). Defense Acquisition University (DAU)/ U.S. Department of Defense Database https://dap.dau.mil/workforce/Documents/Comp/ENG%20Competency%20Model%2020130612_Final.pdf Accessed July 2018.
46. NASA Project Management and Systems Engineering Competency Model. (2009). Academy of Program/Project & Engineering Leadership (APPEL). Washington, DC, USA: US National Aeronautics and Space Administration (NASA). http://appel.nasa.gov/competency-model/ Accessed July 2018.
47. MITRE Systems Engineering Competency Model. (2007). http://www.mitre.org/publications/technical-papers/systems-engineering-competency-model Accessed July 2018.
48. CMMI-Based Professional Certifications: The Competency Lifecycle Framework (2004). Software Engineering Institute, CMU/SEI-2004-SR-013. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=6833 Accessed July 2018.
49. Deraman, A., Yahaya, J., Baharom, F., & Hamdan, A.R. (2010). User-Centred Software Product Certification: Theory and Practices. International Journal of Digital Society (IJDS), Vol.1 (4), Dec 2010, pp. 281-288.
50. Fahmy, S., Haslinda, N., Roslina, W., & Fariha, Z. (2012). Evaluating the Quality of Software in e-Book Using the ISO 9126 Model. The 7th Asia Pacific International Conference on Information Science and Technology, July 4-7, Jeju Island, Korea.

51. Ngah, A., Munro, M., and Gallagher, K. (2012). Regression Test Selection Model Using Decomposition Slicing. In the Proceedings of the IASTED International Conference on Software Engineering. https://doi.org/10.2316/P.2012.780-020
52. Nishad Nawaz. Artificial Intelligence interchange human intervention in the recruitment process in Indian Software Industry. International Journal of Advanced Trends in Computer Science and Engineering. Volume 8, No.4, July – August 2019 https://doi.org/10.30534/ijatcse/2019/62842019